Technical University of Crete
School of Electrical and Computer Engineering

Diploma Thesis

A Wireless Haptic System for Multisensory Texture Data Capture and Vibrotactile Feedback Representation in Virtual Reality

Anna Minadaki

Thesis Committee
Professor Katerina Mania (supervisor)
Professor Michail G. Lagoudakis
Professor Michail Zervakis

Chania | Crete, January 2025

ΠΟΛΥΤΕΧΝΕΙΟ ΚΡΗΤΗΣ

Σχολη Ηλεκτρολογων Μηχανικων & Μηχανικων Υπολογιστων

Διπλωματική Εργασία

Ασύρματο Σύστημα για Πολυαισθητηριακή Καταγραφή Δεδομένων Υφής και Απτική Αναπαράσταση μέσω Δονήσεων σε Εικονική Πραγματικότητα

Άννα Μηναδάκη

Εξεταστική επιτροπή
Καθηγήτρια Αικατερίνη Μανιά (Επιβλέπουσα)
Καθηγητής Μιχαήλ Γ. Λαγουδάκης
Καθηγητής Μιχαήλ Ζερβάκης

Χανιά | Κρήτη, Ιανουάριος 2025

Abstract

The sense of touch is a fundamental aspect of human interaction with the environment, providing rich perceptual information about surface textures and material properties. As virtual environments increasingly aim to replicate real-world experiences, haptic feedback systems play a critical role in enhancing immersion. However, realistic texture representation remains a significant challenge, due to the complexity of tactile perception and the multi-dimensional nature of textures. This thesis is focused on developing a wireless system capable of capturing, processing and representing real-world textures, by simulating them through vibrotactile feedback.

The system leverages a dual-sensor approach, using a MEMS microphone and a three-axis accelerometer to record synchronized audio and vibrational data during bare-fingertip user interactions with textures. Data preprocessing techniques, including filtering and denoising, isolate meaningful signal components, while adaptive peak detection identifies consistent interaction regions. Features extracted from the processed signals are mapped to vibration frequency and amplitude to drive a Linear Resonant Actuator (LRA), creating perceptually realistic texture representations. The system supports audio-only, accelerometer-only and fused feedback modes, as well as dynamic hand velocity modulation for enhanced realism.

A user evaluation in a custom Unity-based virtual environment with Leap Motion integration for hand tracking demonstrated the system's effectiveness in representing three distinct textures: combs with 1 mm and 2 mm tooth spacing and 180-grit sandpaper. Participants achieved texture identification rates of 80%–90%, with the fusion mode providing the most realistic feedback across all textures. Audio features excelled in representing fine textures, while accelerometer features captured coarser patterns more effectively. Although dedicated feature pairs tailored to individual textures offered marginally improved realism, common feature pairs proved sufficient for most applications, balancing simplicity and performance.

The findings highlight the potential of combining audio and vibrational data for multimodal texture representation, while also identifying areas for improvement, such as addressing minor looping artifacts and scaling the feature mapping process. This work provides insights into achieving perceptually meaningful and immersive texture experiences, with applications in virtual reality, telepresence and tactile simulation.

Περίληψη

Η αίσθηση της αφής αποτελεί θεμελιώδη πτυχή της ανθρώπινης αλληλεπίδρασης με το περιβάλλον, παρέχοντας λεπτομερείς πληροφορίες για την υφή και τις ιδιότητες των υλικών. Καθώς τα εικονικά περιβάλλοντα στοχεύουν ολοένα και περισσότερο στην αναπαραγωγή εμπειριών του πραγματικού κόσμου, τα συστήματα απτικής ανάδρασης παίζουν καθοριστικό ρόλο στην ενίσχυση της εμβύθισης. Ωστόσο, η ρεαλιστική αναπαράσταση των υφών αποτελεί μια σημαντική πρόκληση, λόγω της πολυπλοκότητας της απτικής αντίληψης και της πολυδιάστατης φύσης των υφών. Η παρούσα διπλωματική εργασία επικεντρώνεται στην ανάπτυξη ενός ασύρματου συστήματος ικανού να καταγράφει, να επεξεργάζεται και να αναπαριστα υφές του πραγματικού κόσμου, προσομοιώνοντάς τις με ανάδραση μέσω δονήσεων.

Το σύστημα χρησιμοποιεί δύο αισθητήρες, συνδυάζοντας ένα MEMS μικρόφωνο και ένα επιταχυνσιόμετρο τριών αξόνων για την καταγραφή συγχρονισμένων δεδομένων ήχου και δονήσεων κατά τη διάρκεια αλληλεπιδράσεων χρηστών με υφές μέσω άμεσης επαφής με το δάχτυλο. Τεχνικές προεπεξεργασίας δεδομένων, όπως το φιλτράρισμα και η μείωση θορύβου, απομονώνουν τα σημαντικά στοιχεία των σημάτων, ενώ μέσω προσαρμοστικής ανίχνευσης κορυφών εντοπίζονται σταθερές περιοχές αλληλεπίδρασης. Από τα επεξεργασμένα σήματα εξάγονται χαρακτηριστικά, τα οποία αντιστοιχίζονται σε συχνότητα και πλάτος δόνησης, ενεργοποιώντας έναν Γραμμικό Δονητή Συντονισμού (LRA) και δημιουργώντας ρεαλιστικές αναπαραστάσεις υφών. Το σύστημα υποστηρίζει λειτουργίες ανάδρασης βασισμένες αποκλειστικά στον ήχο, αποκλειστικά στο επιταχυνσιόμετρο ή στον συνδυασμό των δύο, καθώς και δυναμική προσαρμογή της ανάδρασης με βάση την ταχύτητα του χεριού, ενισχύοντας το ρεαλισμό. Μια αξιολόγηση χρηστών σε ένα εξατομικευμένο εικονικό περιβάλλον που αναπτύχθηκε με τη χρήση της πλατφόρμας Unity, με ενσωμάτωση του Leap Motion για την ανίχνευση και παρακολούθηση της κίνησης του χεριού, ανέδειξε την αποτελεσματικότητα του συστήματος στην αναπαράσταση τριών υφών: χτενών με απόσταση δοντιών 1 mm και 2 mm και γυαλόχαρτου 180 grit. Οι συμμετέχοντες πέτυχαν ποσοστά αναγνώρισης υφών της τάξης του 80%–90%, με τη λειτουργία συνδυασμού να προσφέρει την πιο ρεαλιστική ανάδραση για όλες τις υφές. Τα χαρακτηριστικά του ήχου αποδείχθηκαν πιο αποτελεσματικά στην αναπαράσταση υφών με μικρές λεπτομέρειες, όπως το γυαλόχαρτο, ενώ τα χαρακτηριστικά του επιταχυνσιόμετρου αποτύπωσαν με μεγαλύτερη ακρίβεια τις τραχύτερες επιφάνειες. Παρόλο που τα εξειδικευμένα ζεύγη χαρακτηριστικών, προσαρμοσμένα σε μεμονωμένες υφές, προσέφεραν μια μικρή βελτίωση στον ρεαλισμό, τα κοινά ζεύγη χαρακτηριστικών αποδείχθηκαν επαρκή για τις περισσότερες εφαρμογές, εξασφαλίζοντας μια καλή ισορροπία μεταξύ απλότητας και απόδοσης. Τα ευρήματα αναδεικνύουν την προοπτική του συνδυασμού δεδομένων ήχου και δονήσεων για πολυτροπική αναπαράσταση υφών, ενώ εντοπίζουν σημεία προς βελτίωση, όπως η αντιμετώπιση μικρών επαναληπτικών ανωμαλιών και η ανάπτυξη της διαδικασίας αντιστοίχισης των χαρακτηριστικών. Η εργασία αυτή παρέχει κατευθύνσεις για την επίτευξη ρεαλιστικών εμπειριών υφής, με εφαρμογές στην εικονική πραγματικότητα, την τηλεπαρουσία και την απτική προσομοίωση.

Acknowledgements

I would like to express my gratitude to my supervisor, Professor Katerina Mania, for giving me the opportunity to pursue this thesis in the field of haptics. Her enthusiasm and support throughout the development of this work have been truly inspiring.

I am also deeply thankful to the members of the SURREAL Lab for their help, guidance, and the warm welcome they extended to me as part of their team.

I want to express my love and deepest gratitude to my dear friends Giorgos, Konstantina, Michalis and Christos who have become family to me over all these years, supporting me through thick and thin.

I am equally thankful to my girls, Christina, Elisavet, Irka and Eirini, for always being there for me.

Lastly, I want to thank my brother Giorgos, for playing a key role in motivating me to successfully complete this thesis and my parents for their unconditional love and support.

Contents

List of Figures

LIST OF FIGURES

Chapter 1

Introduction

This chapter introduces the concept of texture representation through haptic feedback. Additionally, a summary of the whole idea of this thesis is presented including the implementation, evaluation and results. Finally an outline of the structure of the entire thesis is also included.

1.1 Brief Description

The sense of touch plays a fundamental role in how humans perceive and interact with their environment. The skin is the largest organ of our body, with a complex somatosensory system that provides rich perceptual experiences. Manipulating objects and surfaces with our hands gives us discrete information about the physical properties of the object such as shape, texture, temperature and size. As technology evolves, we are observing an increasing tendency towards simulating the real world in virtual environments. While virtual reality technologies have made significant advances in simulating visual and auditory experiences, replicating tactile feedback remains a complex challenge.

Specifically, texture is a critical component of tactile perception that can greatly enhance realism in virtual environments. Texture encompasses a range of perceptual qualities including roughness, smoothness and pattern variations, which are inherently complex and require precise recreation for effective virtual representation. Unlike visual or auditory feedback, which can rely on established display technologies, haptic feedback demands physical interaction and precise actuation to simulate the nuances of textures accurately.

The field of texture representation in haptics, while promising, remains less explored compared to visual and auditory domains. This is largely due to the complexity of texture perception, which involves dynamic and multi-dimensional properties that challenge both hardware and software capabilities. Additionally, the lack of standardized evaluation metrics and the high sensitivity and variability of the human somatosensory system make achieving realistic texture rendering challenging. However, recent advancements in

actuator technology, machine learning-driven simulations and sensor precision are beginning to address these challenges, driving the field towards more immersive and authentic haptic experiences. Applications of texture recreation extend to virtual reality, telepresence and medical training, where the ability to simulate real-world tactile sensations enhances immersion and utility.

This thesis focuses on the development, implementation, and evaluation of a wireless multisensory system designed to capture, process and represent the tactile characteristics of three real-world textures through vibrotactile feedback. By recording synchronized signals from a MEMS microphone and a three-axis accelerometer during bare-fingertip interactions with textures, the system aims to create perceptually realistic and accurate representations of these textures in virtual environments. The main objectives of this work are to achieve accurate texture representation and to evaluate which feedback modes, feature mappings and modulation strategies yield the best results for perceptually realistic haptic feedback.

The hardware architecture of the system is built around the ESP32-S3 microcontroller, which manages the simultaneous recording and processing of audio and vibrational data using the SPH0645LM4H MEMS microphone and ADXL345 accelerometer. The whole system is powered by a 18650 Battery Shield Mobile Power Bank in order to ensure proper handling of the power needs of each component and also to make the system wireless. The microphone records audio signals at a high sampling rate of 48 kHz via the I2S protocol, capturing the spectral richness of texture interactions. The accelerometer records at 800 Hz rate using the I2C protocol, capturing the dynamic vibrational patterns generated during interactions with surface textures. These data streams are synchronized using microsecond-level timestamps provided by the ESP32-S3, ensuring precise temporal alignment, which is critical for accurate signal processing and feature extraction.

To handle these tasks efficiently, the system leverages FreeRTOS, a real-time operating system that allows the ESP32-S3's dual-core architecture to manage multiple concurrent processes. The audio and accelerometer tasks are run on separate cores to minimize interference and ensure simultaneous data acquisition. Dedicated tasks handle data collection and storage while inter-core communication ensures synchronization between the two sensors.

A server-client system is implemented using the ESP32-S3's integrated Wi-Fi capabilities, allowing for wireless control of the recordings. The ESP32-S3 operates as a server, receiving commands from a client application to initiate or stop recordings. Once a recording is completed, the audio and accelerometer data which are saved within the microcontroller's memory, are available for download via HTTP.

Three textures were chosen for this thesis, representing a range of tactile properties: a comb 2 mm spacing between its teeth, a comb 1 mm spacing and sandpaper with 180 grit. The comb textures provide structured, periodic vibrations due to their regular patterns, with the 1 mm comb generating lower-frequency vibrations and the 2 mm comb producing higher-frequency vibrations. In contrast, the sandpaper introduces an irregular and continuous microstructure, creating rich spectral content and higher-frequency

vibrations. This diversity ensures that the system can be thoroughly evaluated across different texture types, testing its ability to capture and represent both structured and unstructured surface properties.

(a) Side View of the Data Capture System (b) Top View of the Data Capture System

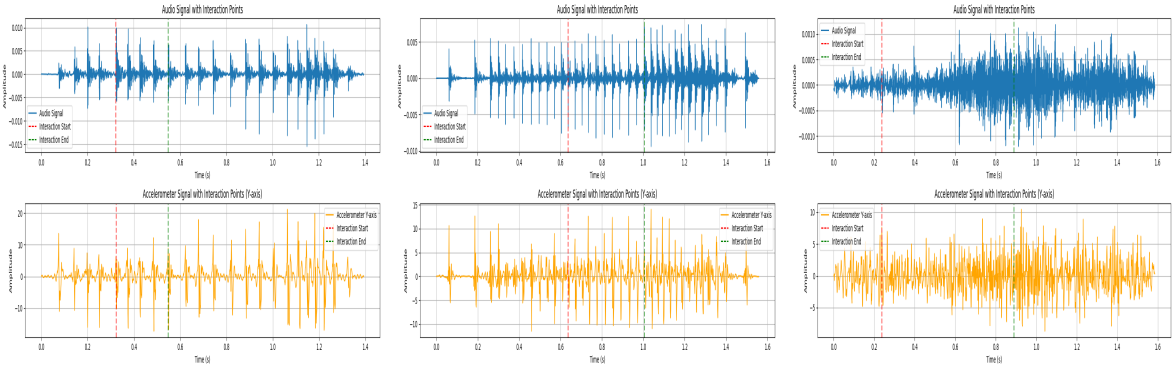
Figure 1.1: Overview of the Data Capture System.

After capturing the raw data from the ESP32-S3, both audio and accelerometer signals are loaded from their respective files. Each recording includes timestamps that allow precise alignment of the two signals. However, to ensure that all subsequent steps focus solely on the portion of the signal representing actual texture interaction, the system first identifies the interaction region in the accelerometer data based on energy thresholds. The audio signal is then cropped to the same temporal window. This step eliminates idle segments before any denoising or filtering takes place, allowing preprocessing to concentrate on the meaningful portion of the interaction.

Within this interaction region, the accelerometer signal undergoes DC offset removal to center the data, followed by notch filtering to suppress narrow-band noise such as electrical interference. A second-order Butterworth bandpass filter then isolates the 30–350 Hz range, capturing the vibrational frequencies most relevant to texture perception. Finally, wavelet denoising is applied to preserve sharp transients, which are critical for representing the sudden changes associated with fingertip movements on textured surfaces, while reducing residual noise. By contrast, the audio signal is processed using a fifth-order Butterworth bandpass filter in the range of 30–1,000 Hz to capture the acoustic details of texture contact. Like the accelerometer data, it also undergoes DC offset removal and notch filtering for electrical interference. However, it employs both adaptive Wiener filtering and wavelet denoising to deal with complex ambient noise that can arise in audio recordings. These methods remove unwanted sound artifacts while preserving subtle spectral details.

Once both signals have been cleaned and trimmed to the interaction region, adaptive peak detection is performed to isolate the largest, most consistent portion of the interaction. Because the fingertip interactions during recordings are unconstrained, variations in speed or pressure naturally occur, which can introduce irregularities in both the ac-

celerometer and audio signals. By applying peak detection criteria tailored to each signal type this step identifies the segment of the interaction that remains stable under such variations.



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 1.2: Final Interaction Region in both Signals using Adaptive Peak Detection.

A central focus of this thesis is the mapping of extracted features to vibration parameters of frequency and amplitude that drive a Linear Resonant Actuator (LRA) to recreate the textures. Two mapping strategies are explored: dedicated feature pairs, which are optimized for individual textures and common feature pairs, designed to generalize across all textures. Additionally, the system supports three feedback modes: audio-only, accelerometer-only and fusion, where the fusion mode combines audio-derived frequency features with accelerometer-derived amplitude features, leveraging the strengths of both modalities.

In this work, feature extraction and mapping serve as the link between the recorded signals and the haptic representation of textures. After the signals are preprocessed and trimmed to their most consistent segments, they are loaded from the generated WAV audio and CSV accelerometer files. To allow loopable playback, each signal is extended by a concatenation process that repeats the trimmed segment multiple times, applying a brief crossfade to ensure seamless transitions. The signals are then split into short overlapping segments to capture the temporal variations and preserve the relevant frequency content of the texture interactions.

Within each segment, a set of temporal, spectral, statistical and for audio perceptual features is extracted, reflecting the intensity, energy distribution, and dynamics of the textures. These features are aggregated across all recordings to compute global ranges, which provide a consistent basis for normalizing feature values.

To evaluate which combination of features yields the most realistic haptic feedback, three output modes were created for the mapping algorithm: audio-only, accelerometer-only and fused. In audio-only mode, both frequency and amplitude are derived from audio-based features and in accelerometer-only, they come exclusively from accelerometer

data. Meanwhile, fused mode combines one signal source for frequency and the other for amplitude. This design produces time-varying CSV outputs that capture the distinct vibrational signatures of each texture.

After the data capture, preprocessing and feature extraction stages are completed, the final step merges all system components into a virtual environment built with Unity, with integration of the Leap Motion for precise hand and finger tracking. Users interact with 3D models corresponding to the real-world textured objects, while their fingertip collisions trigger haptic responses from a Linear Resonant Actuator (LRA), wirelessly driven by the ESP32-S3 microcontroller. The CSV files from the feature mapping stage are pre-uploaded to the ESP32-S3's onboard file system. During interaction in the Unity environment, these CSV files are selectively accessed depending on which virtual texture the user's fingertip collides with.

A user interface (UI) allows switching between audio-only, accelerometer-only or fused feature and for each of these modes to chose between common or dedicated feature pair representation, each corresponding to a different CSV file of vibration parameters sets, making it straightforward to compare different texture representations. The UI also features a velocity modulation toggle to dynamically adjust the LRA's vibration output in real time. When velocity modulation is enabled, the fingertip's speed detected by the Leap Motion directly scales the LRA's amplitude and frequency, resulting in stronger, more intense feedback for faster movements and subtler feedback for slower ones, adding an extra layer of realism to texture exploration.

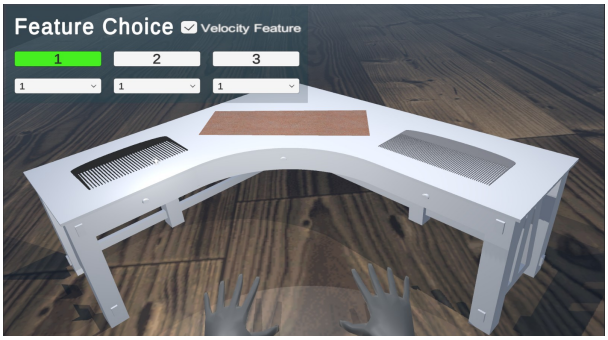


Figure 1.3: Unity Scene Overview.

For unbiased evaluation, the environment also features a simplified view with identical 3D cubes each corresponding to one of the recorded textures, without any visual cues so that participants rely solely on haptic feedback to recognize each surface. Additionally, the UI menu conceals the names of the feedback modes and feature types, labeling them only as 1, 2, 3 for modes and 1, 2 for feature types, preventing users from knowing which configuration is active, avoiding any bias in their assessments.

The evaluation was designed to assess how effectively the developed haptic feedback system replicates recorded textures through direct mapping of extracted features to vibrotactile parameters, consisting of two main stages. In the first stage a feature selection

process was performed where a broad set of features from both signal types, initially chosen for their theoretical relevance to texture representation, was examined to find those that most accurately map to vibration frequency and amplitude. The process began with visual elimination, discarding features that produced flat or implausible outputs, followed by perceptual testing in which each candidate feature pairing was actually driven through the Linear Resonant Actuator (LRA) for fingertip evaluation. During this phase, a think-aloud methodology was employed, where participants expressed their real-time opinions on the vibrational realism. By splitting features into categories of audio-only, accelerometer-only and fusion where one sensor provided frequency and the other amplitude, the evaluation concluded to the most perceptually meaningful pairs for each mode. The process yielded for each mode both dedicated feature pairs, tailored to each specific texture and common feature pairs, which perform adequately across all textures.

Next, in the user study, ten participants tested the final feature sets derived from the feature selection phase, across three feedback modes: audio-only, accelerometer-only, and fusion. They interacted with each one of the three textures within the Unity environment. To eliminate external auditory cues, participants wore noise-canceling headphones. The study not only compared common versus dedicated feature pairs, but also incorporated velocity modulation. During the tests, each participant rated realism, consistency and responsiveness, enabling a multi-dimensional assessment of which combinations most convincingly replicated the recorded textures. This user evaluation phase provided the basis for determining how effectively different modes and feature types could yield immersive, accurate haptic feedback.

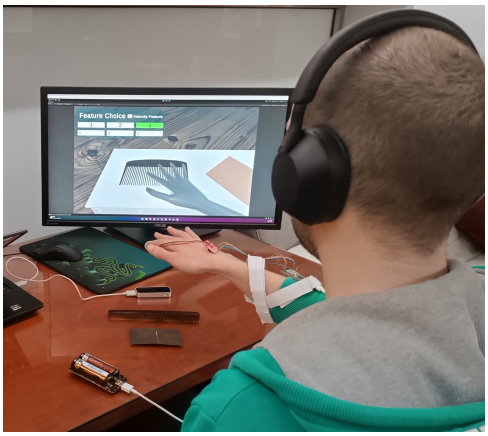


Figure 1.4: User Evaluation process where a user is interacting with a virtual texture wearing noise-canceling headphones.

Overall, participants demonstrated 80–90% accuracy in identifying each texture purely through vibrotactile feedback, highlighting the system's capability to represent textures convincingly. Among the three feedback modes tested, fusion mode employing audio-

derived frequency with accelerometer-derived amplitude, consistently delivered the most realistic feedback, achieving the highest realism ratings across all textures. Although dedicated feature pairs offered marginally greater realism, particularly for sandpaper's fine-grained irregularities, common feature pairs proved sufficient for the combs, suggesting that a universal approach can balance ease of implementation with perceptual fidelity. Additionally, participants reported that velocity modulation significantly enhanced realism during interactions. While minor perceptual artifacts, such as looping effects, were noted by some participants during continuous feedback, they were not found to be distracting.

1.2 Structure of the Thesis

- Chapter 1: Introduction
  This chapter introduces the concept of texture representation through haptic feedback and provides a summary of this thesis subject, implementation and results.

- Chapter 2: Research Overview
  This chapter presents the research that was conducted for this thesis. This includes the human senses with focus on the sense of touch and tactile perception. Also, the haptic technology field is discussed introducing the concept of haptic textures. It provides an overview of related works, highlighting key technologies and methodologies that influenced the research.

- Chapter 3: Theoretical and Technological Background
  This chapter introduces the hardware and software components and frameworks as well as communication protocols that were used in the implementation of this thesis.

- Chapter 4: Implementation
  This chapter focuses on the implementation details of the proposed system. It explains the data capture process, preprocessing steps, feature extraction techniques, and mapping strategies. Integration with a Unity-based virtual environment and the actuator driving mechanism are also described in detail.

- Chapter 5: Evaluation
  This chapter presents the evaluation of the system, consisting of a perception testing using think-aloud methodology followed by a user evaluation. The results of the evaluation are also presented and discussed in detail.

- Chapter 6: Conclusions & Future Work
  The thesis concludes with a summary of findings, highlighting the system's achievements and limitations. Also a discussion for future directions for enhancements of the system is included.

Chapter 2

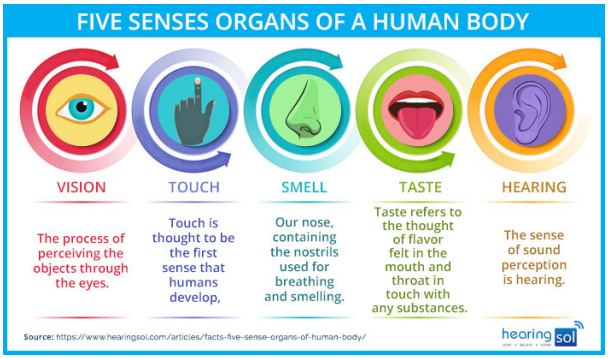
Research Overview

In this chapter, the literature review that was conducted will be discussed, in the scope of understanding in a deeper level the theoretical background of human perception and the current state of technology in the field of texture haptic feedback. First the human perception of the real world through the basic senses is presented, emphasizing on the sense of touch. Then, an analysis of the perception of textures and the factors that contribute to our ability to recognize and discriminate between different textured objects and surfaces is discussed, which is one of this thesis main focuses. After understanding the theoretical background of this field, the current technological methods that have been deployed to deliver haptic feedback will be examined. Finally some related works focusing on texture feedback through different methods are presented, including audio-to-haptic, which is one of the main methods that was chosen for this thesis.

2.1 Human Senses

Our perception of the world around us is based on a system of exchanging information with our environment. This system enables us to perform everyday tasks such as cooking, driving a car, or having social interactions, as well as more complex operations like surgery or handling machinery. Our sensory systems work in harmony to guide our actions, with the most well-known and widely accepted categorization of human senses being vision, audition(hearing), gustation (taste), olfaction (smell) and somasthesis (touch).

However, beyond the above primary sensory modalities, there are additional senses that contribute to our understanding and interaction with the environment. For example, proprioception refers to the sense of body position and movement of body parts relative to each other while the vestibular sense detects changes in the head position and body movement, contributing to balance and spatial orientation. While the peripheral components of the sensory systems vary, they significantly influence each other in shaping our perception. Vision, is considered the most important and complex sense, as around 30% of the human brain's cortical surface is dedicated to processing visual information.



Source: https://inspiritvr.com/senses-study-guide/

Figure 2.1: The five primary human senses.

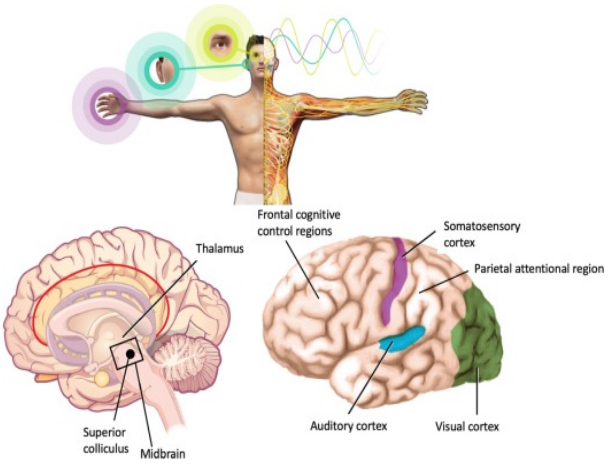
2.1.1 Sensory System

Our sensory systems extract information about four key characteristics of stimuli: sensory modality and submodalities, intensity, duration, and location. Receptors in these systems vary in their sensitivity to different types of physical stimuli that they are specialized to detect, such as light, sound waves or pressure. This sensitivity plays a crucial role in determining the detection threshold and responsiveness of the receptors as well as to how effectively they can convert stimuli into electrochemical signals through sensory transduction. Following sensory transduction, sensory information is encoded into neuronal activity.

Sensory information travels to the brain via specialized neural pathways. Neurons within sensory pathways exhibit a characteristic structure, comprising a cell body, dendrites for receiving inputs, and an axon that transmits output in the form of action potentials. These action potentials travel along the axon at speeds ranging from 10 to 100 meters per second. Neurons typically maintain a baseline firing rate of 5-10 spikes per second in the absence of stimulation. Changes in sensory input intensity or other stimulus characteristics cause deviations from this baseline, signaling information to the brain. The period following an action potential limits neuronal firing rates to about 1000 spikes per second, influencing how information is processed and transmitted.

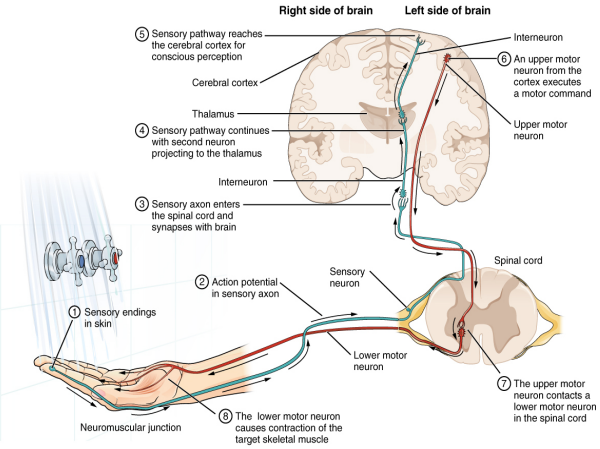
Sensory pathways to the brain are organized into specialized, structured networks of neurons. Most sensory modalities operate through multiple parallel pathways that analyze and convey different aspects of the sensory signal. These pathways project to primary receiving areas in the cerebral cortex, often after passing through relay areas in the thalamus. The thalamus acts as a critical relay station, routing sensory information to the appropriate cortical areas for further processing and integration into perception and cognition [1].

In conclusion, human sensory systems are diverse and interconnected, facilitating our perception and interaction with the environment. This thesis emphasizes the importance



Source: https://www.sciencedirect.com/topics/computer-science/sensory-stimulus

(a) Sensory Brain Regions



Source: https://en.wikipedia.org/wiki/Sensory_nervous_system

(b) Sensory Nervous System

Figure 2.2: Sensory System

of understanding their integration, particularly how tactile feedback complements auditory and visual inputs. Such insights are crucial for enhancing human sensory capabilities and advancing innovative technologies.

2.1.2 Vision and Touch

Vision and touch are two intuitively distinct sensory domains; however, they share substantial similarities. Research demonstrates that vision and touch work together to enhance our perception of object properties such as orientation, shape, and surface texture.

Vision and touch interact in orientation perception, as seen in the tilt illusion, where a tactile pattern can enhance the visual effect of perceived tilt.[2] Another example is the binocular rivalry, where it is found that a tactile pattern matching one visual stimulus can make that stimulus more dominant.[3] These interactions suggest shared neural representations for orientation in both senses.

Neuroimaging studies show that visual and tactile stimuli activate brain regions, such as the parietal cortex, suggesting a shared neural basis for integrating these sensory inputs. This indicates that visual cortical areas process information from multiple senses, highlighting the overlap between vision and touch. [4] , [5] This integration is essential for precise texture discrimination, as it enhances the accuracy and immediacy of tactile perception [6]. For example, visual input helps in identifying the overall form of an object, while tactile input provides detailed information about its surface texture [7].In addition, research shows that visual stimuli can prepare the brain to expect certain textures, improving tactile discrimination [8].

Vision is effective at recognizing shapes and sizes as it provides detailed spatial in-

formation, while touch assesses textures and fine details through direct physical contact. When focusing on textures, tactile sensations can affect how textures are perceived visually, especially for smoother surfaces. This observation shows how important touch is in our perception of textures, especially for everyday materials like fabrics. In these situations, our brain may often prioritize touch, even though vision could theoretically provide better discrimination in certain cases, such as discriminating rougher surfaces. This challenges the idea that our brain always uses the most effective sense, showing instead that it considers the sense most relevant to our daily experiences. Additionally, it is found that there is an asymmetry between visual and tactile stimuli, indicating that touch tends to influence how humans see more than vice versa in certain contexts, like the rapid discrimination of rough textile samples [9] .

2.1.3 Audition and Touch

Research has revealed a close relationship between the auditory and tactile systems, indicating an overlap in information processing between these senses. fMRI (functional magnetic resonance imaging) studies have found processing centers within the human somatosensory cortex for auditory frequencies, particularly within the 100–300 Hz range. This suggests that these sensory modalities share neural pathways, that allow auditory and tactile stimuli to influence each other and enhance our perception of textures and other tactile properties of the stimulus. [10]. Furthermore, fMRI research comparing responses to somatosensory and auditory stimuli has found overlapping activation patterns in the secondary somatosensory cortex (SII) and auditory belt areas, indicating that these regions are crucial for complex sensory integration [11]. In practical applications, studies on electro-haptic stimulation have proved that synchronized tactile feedback significantly improves speech recognition for cochlear implant users in noisy environments, indicating the effective integration of tactile and auditory information to enhance auditory perception. [12] Also, hearing and touch are both very sensitive to temporal delays, and can detect even vert low latency differences relative to each other.

The connection between hearing and tactile perception can be demonstrated in numerous ways, such as auditory-induced tactile illusions and the modulation of tactile perception by auditory stimuli. These interactions show robust neural connections between these senses, with shared pathways in the somatosensory cortex, insula, and auditory cortex [13]. Temporal summation, the process where stimuli that happen close together are combined to create a stronger perception, occurs in both the auditory and tactile systems, though they do so in different ways. The auditory system shows temporal summation over a broad frequency range (20 Hz to 20 kHz), while the tactile system produces summation at higher frequencies[14]. Temporal frequency channels are linked across the two modalities of audition and touch, particularly in the 20-300 Hz range. This connection is highlighted in tasks that require fine temporal discrimination, where auditory stimuli can interfere with tactile frequency perception, indicating specific cross-modal interactions [15]. Other studies have shown that aspects of touch, such as texture

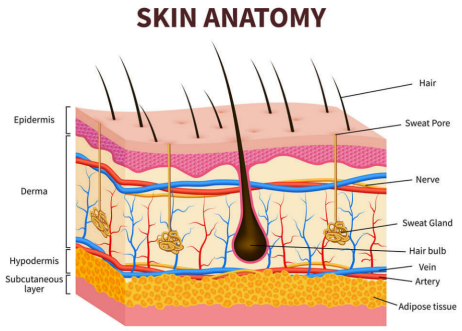
and vibration in the range of 10-500 Hz, correspond to auditory properties like pitch and rhythm. Therefore, it is evident that these senses work together to process complex sensory information resulting to a unified perceptual experience [16]. Historically, there have been early hints at the connection between touch and hearing, such as when Edison's phonograph produced unintended tactile feedback from its vibrations. Though this was not a deliberate effort to combine the senses, it sparked curiosity about how touch and sound could enhance each other. Today, research has shown that integrating these senses can significantly enrich our perception and create more immersive sensory experiences.[17].

Understanding the neural and perceptual mechanisms of these sensory systems is key to advancing multisensory integration technologies, which have applications in assistive devices and human-computer interaction. In this thesis, the focus concentrated more towards the relationship between audition and touch, examining how theoretical insights can be used to the development of a system for texture representation through haptic feedback based on audio signals produced by interactions with textured objects.

2.1.4 Somatosensory System

2.1.4.1 Brief Skin Anatomy

As mentioned before, skin is the largest organ of the human body, consisting of three layers. The upper layer, the one than can be seen, is a waterproof protective wrap of dead cells called epidermis. It contains touch receptors, which are sensitive cells, giving the brain information about the surrounding environment. The second layer is called dermis and contains, among other things such as hair follicles and sweat glands, nerve endings and a variety of touch receptors. The bottom layer, called subcutaneous tissue, is composed of fat, acting as a regulator of the body temperature as well as a protection layer to injuries and connective tissue which connects the skin to muscles and tendons.[18], [19]



Source: https://www.thedermspecs.com/blog/skin-anatomy-101/

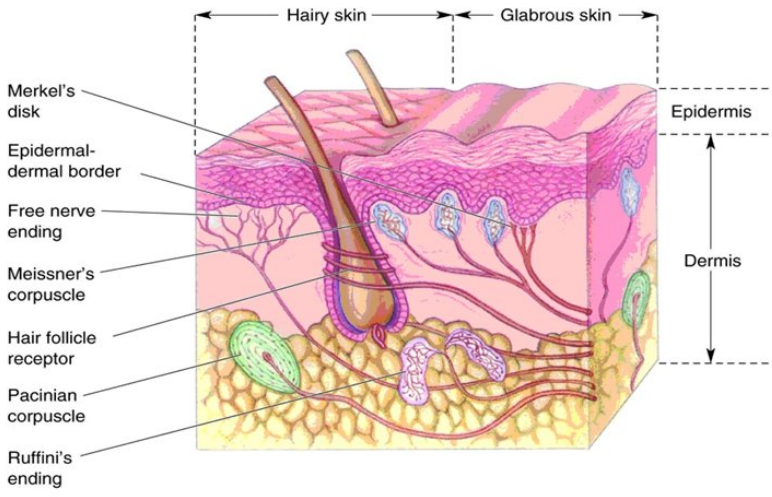
Figure 2.3: Skin Anatomy

2.1.4.2 Types of receptors

The somatosensory system is an intricate network of nerve endings and touch receptors embedded in the skin. These receptors deliver essential tactile information about the position, shape, texture, pressure, and movement of objects humans come into contact with. They are classified into two primary types: rapidly-adapting and slowly-adapting receptors. Rapidly-adapting receptors react immediately to the onset of a stimulus but quickly diminish their response if the stimulus persists, making them crucial for detecting movement. In contrast, slowly-adapting receptors continue to respond to a continuous stimulus, which is vital for recognizing shapes and sizes.[20]

Touch receptors also differ in the size of their receptive fields—the areas they can detect signals from. Generally, numerous small receptive fields allow for superior tactile discrimination compared to fewer, larger receptive fields. The somatosensory system is comprised of three main types of sensory receptors that detect various external stimuli: mechanoreceptors, which sense light touch, vibration, pressure, and texture; nociceptors, which sense pain; and thermoreceptors, which sense temperature. Additionally, vibratory receptors are instrumental in helping us perceive different frequencies of external stimuli.[19]

There are four types of mechanoreceptors found in glabrous skin, like that of the hand.[21]



Source: https://www.researchgate.net/figure/Mechanoreceptors-in-the-glabrous-and-the-hairy-skin-2_fig1_334762306/

Figure 2.4: Types of mechanoreceptors.

- Merkel's Discs (SA1) are slowly-adapting receptors, located in the epidermis, that have very small receptive fields and high spatial resolution. They are especially dense in the fingertips, and are best-suited for processing information about shape

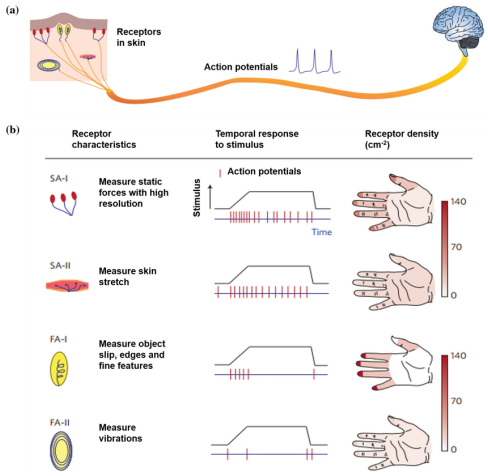
and texture. Selective stimulation of these receptors in humans produces a sensation of light pressure. These characteristics have led to the belief that Merkel's disks play a significant role in the static discrimination of shapes, edges, and rough textures.

- Meissner's corpuscles (RA) are rapidly-adapting receptors located beneath the epidermis. They have relatively small receptive fields, but their spatial resolution is inferior to that of Merkel's discs. They are particularly effective at detecting movement between the skin and another surface, which is essential for sensing texture and identifying when an object is sliding across the skin, thereby aiding in grip maintenance. These corpuscles respond to light touch and are especially efficient in transducing information about low-frequency vibrations (30–50 Hz) that occur when textured objects move across the skin.

- Pacinian corpuscles (PC) are rapidly-adapting receptors with very large receptive fields located in the subcutaneous tissue. They are highly sensitive to high-frequency vibrations and transient pressure changes when objects are contacted or grasped by the hand, making them potentially important for tool use. These receptors are activated by transient disturbances at high frequencies (100–400 Hz) and have a lower response threshold than Meissner's corpuscles. The rapid adaptation and sensitivity to high-frequency vibrations suggest that Pacinian corpuscles play a key role in the discrimination of fine surface textures and other moving stimuli.

- Ruffini's endings (SA2) are slowly-adapting receptors with large receptive fields, located deep in the skin. Although not fully understood, they seem to respond primarily to skin stretching, such as that occurring with finger movements. This sensitivity likely plays a crucial role in making us aware of finger and hand positions. By responding to skin stretching caused by movements of our digits or limbs, Ruffini's corpuscles help us perceive the shape of objects and the position of our hands. Consequently, they contribute to our ability to tactically discriminate larger and more complex objects.

The fingertips are particularly well-suited for tactile exploration and texture discrimination due to their high density of mechanoreceptors such as SA1 and RA along with their small receptive fields. This combination enables high spatial resolution and the ability to detect fine surface features, allowing the fingertips to distinguish very fine textural differences and perceive intricate details of objects. They are very sensitive to light touch, low-frequency vibrations, sustained pressure, and fine spatial details. This high concentration of mechanoreceptors and high sensory resolution allows for the precise detection of small details in textures, making the fingertips especially effective for discriminating between different textures. [22]



Source:
https://www.researchgate.net/figure/Skin-receptors-and-transduction-process-a-Biological-skin-transduction-process-b-Types_fig5_353345353/

Figure 2.5: Skin receptors and transduction process. Rapidly-Adapting receptors are also called Fast-Adapting (FA).

Based on the above, given the fingertip's high sensitivity and spatial resolution, which allows for the precise detection of fine textures, the fingertip of the index finger was selected as the primary point of contact for recording interactions and delivering haptic feedback in this thesis. Its ability to detect subtle differences in texture and respond to low-frequency vibrations makes it ideal for the feedback necessary in this system.

2.2 Material attributes

In order to develop meaningful haptic feedback for virtual textures it is important to study and understand the characteristics of the surfaces that create unique textures. Texture refers to the surface characteristics of an object that can be felt through touch. Roughness, smoothness, surface pattern, and other physical attributes that comprise the overall feel of an object are some of these characteristics. Texture can be described through the spatial frequency of the surface, which refers to the distance between the surface features, and its amplitude, which refers to the height and depth of the features. For instance, a fine sandpaper texture has high spatial frequency since the features are closely spaced, while a coarse wood texture has low spatial frequency because its features are widely spaced. [23]

Surface roughness is a measure of the variations in the surface texture. It is determined by the vertical deviations of the surface from its ideal form. Rougher surfaces have larger deviations, while smoother surfaces have smaller ones. The roughness is quantified using parameters that are essential for defining how a surface interacts with other surfaces or by humans, such as the arithmetic average roughness (Ra) and the root mean square

roughness (Rq). The Ra represents the average height deviations of the surface from the mean line, and Rq provides a root mean square calculation of these deviations, giving information about peaks and valleys on the surface. [24]

The frequency and pattern of a surface's texture is one of the most important factors in the human perception of touch, with higher-frequency textures giving richer sensory feedback. Research that investigated the sensory properties of materials, highlighted that finer textures generated higher vibrations that are perceived by our sensory receptors more intensely. Each kind of material has a specific textural pattern, that makes it its unique tactile signature, producing vibrational feedback that allows the distinction of different materials such as wood, metal and fabric by touch. [25], [26]

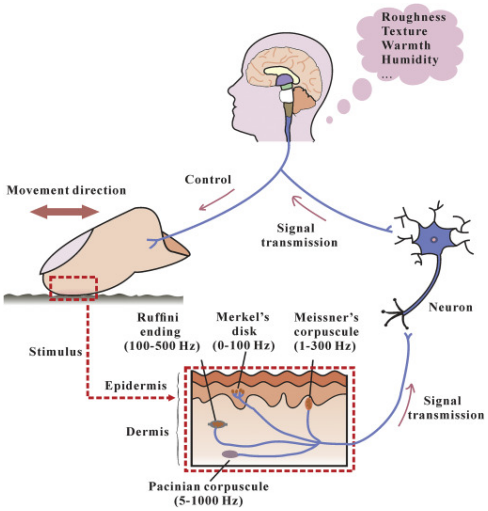
Thermal properties, such as the rate in which a material conducts heat, significantly affect tactile experiences. Materials with higher thermal conductivity, like metals, quickly conduct heat away from the skin and feel cooler, while materials with low thermal conductivity, like wood, feel warmer. When an object is touched, the heat transfer is influenced by thermal conductivity, heat capacity and the contact coefficient of the material. Higher thermal conductivity and heat capacity result in faster and more intense drop in skin temperature, which is perceived as a cool. These variations in thermal properties among different materials creates distinct tactile sensations. The differences arise from the materials' inherent properties, like atomic structure, bonding, and density. Metals for example, have free electrons that enhance heat transfer, while materials like wood and plastic have insulating structures that limit it. These differences affect how materials interact with heat and how they feel to the touch.[27] [28]

Another factor that contributes to the material of an object is its hardness and elasticity. Hardness is a measure of a material's resistance to deformation, determining how the material responds to pressure, while elasticity refers to its ability to return to its original shape after deformation. Harder materials like metal or stone resist deformation more compared to softer materials like rubber or foam that deform more easily under pressure. [29] Materials that have high elasticity on the other hand, like rubber, can become significantly deformed and still return to their original shape in contrast with materials with low elasticity than may remain deformed. [30] These properties affect how a material feels when pressed or touched, contributing to their categorization based on touch alone.

Finally, the chemical composition and surface treatment of materials can significantly influence their tactile properties. Treatments like coating, polishing, or texturing can reform the surface characteristics, affecting how the material feels. For instance, a metal surface can be treated to be smoother and less abrasive, resulting to a different texture than its original. [29], [31]

2.3 Tactile Perception

Tactile perception, the ability to interpret and respond to mechanical stimuli applied to the skin, is influenced by a variety of factors including receptor types, neural processing, and individual differences in sensory experience. Understanding these interactions is essential for developing technologies that replicate or enhance our sense of touch.



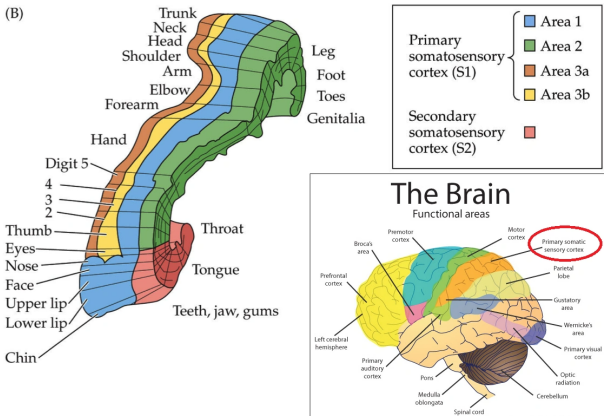
Source: https://www.sciencedirect.com/science/article/pii/S0301679X18301579

Figure 2.6: Tactile Perception Mechanism

2.3.1 Neural Processing of Tactile Stimulus

The neural processing and integration of tactile information are critical for understanding how humans perceive textures and objects through touch. When a tactile stimulus is encountered, mechanoreceptors in the skin transduce this mechanical stimulus into neural signals, which are then transmitted to the brain through the somatosensory pathways. The primary somatosensory cortex (S1) in the brain is a key area where these signals are initially processed. Studies have shown that different regions within the S1 are responsible for processing various aspects of touch, such as texture, pressure, and vibration. [6] [8] [22] Neurons are organized in a somatotopic manner, meaning that specific areas of the cortex correspond to specific parts of the body, allowing precise localization of tactile information. The secondary somatosensory cortex (S2) also plays a significant role in integrating tactile information from both hemispheres of the brain, allowing us to perceive complex tactile stimuli and coordinate movements bilaterally. [32]

Furthermore, neural integration of tactile information involves not only the somatosensory cortex but also other brain regions such as the parietal and prefrontal cortices. These areas contribute to higher-order processing, such as the interpretation of touch sensations based on past experiences and the coordination of sensory input with motor



Source: https://brain-for-ai.fandom.com/wiki/Somatosensory_cortex

Figure 2.7: Brain Anatomy and Somatosensory Cortices.

actions. Neuroimaging studies, including functional MRI, have shown that tactile perception involves a network of brain areas, including regions involved in memory, attention and decision-makings, that collaborate to create a unified perceptual experience. [33]–[36] Additionally, as mentioned in previous sections, the interaction between touch and other senses, such as vision and hearing, enhances the accuracy and richness of tactile perception. This multisensory integration occurs through shared neural pathways and overlapping brain regions that process different types of sensory information at the same time. [37] For instance, visual input can improve the perception of touch by providing context, which helps the brain better identify and distinguish textures and objects. [34]

Tactile perception is influenced by the context in which stimuli are presented as well as by the individual's sensitivity and previous experiences. Differences in the density and distribution of mechanoreceptors, skin properties, and neural processing capabilities can lead to variations in tactile sensitivity among individuals. Also, factors such as age, gender, and even specific experiences can affect how tactile stimuli are perceived and processed. [38] The brain uses prior knowledge to interpret sensory input, which can enhance the accuracy and efficiency of tactile perception, especially in tasks requiring fine discrimination or the identification of objects based solely on touch.

Other factors that affect tactile perception include material properties and skin conditions. Attributes such as texture, hardness, elasticity and thermal conductivity play a crucial role in how materials are perceived through touch, while factors like moisture, temperature and the presence of calluses can alter tactile sensitivity and accuracy of texture discrimination.[27], [28]

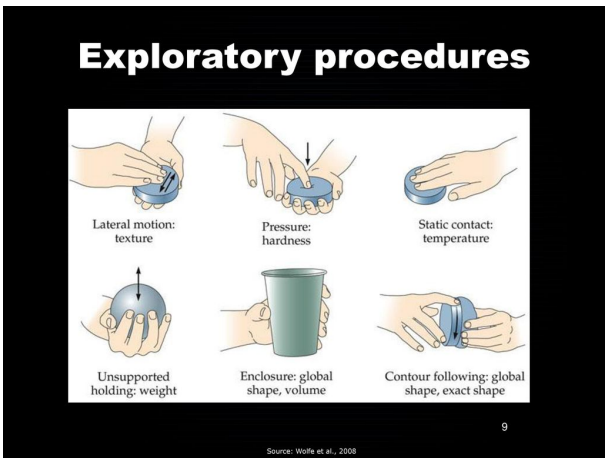
2.3.2 Exploratory Movements for Object Identification

Humans gather tactile information about objects and textures through various exploratory movements. The most common of these is lateral motion, where fingers move back and

forth across a surface to detect texture and patterns. This action generates vibrations that mechanoreceptors in the skin interpret, enabling the detection of surface features like roughness and smoothness [25].

Another important method is applying pressure, which reveals information about an object's hardness and elasticity. This technique activates different mechanoreceptors, such as Merkel cells and Ruffini endings, which respond to sustained pressure and skin stretch, respectively. By varying the pressure applied, individuals can determine whether a material is soft, firm, or elastic [22], [25].

Static contact involves placing the hand on an object's surface to maximize skin contact. This method primarily activates thermoreceptors and slowly adapting mechanoreceptors, providing insights into the object's thermal conductivity and overall shape [27], [39].



Source: https://slideplayer.com/slide/13188787/

Figure 2.8: Exploratory Movements

Enclosure, which involves wrapping the fingers around an object, allows for an assessment of its size and shape. This movement stimulates a wide range of mechanoreceptors, offering comprehensive feedback on the object's dimensions and contours, aiding in the understanding of its three-dimensional structure [25].

Finally, contour following involves tracing an object's edges to understand its outline and finer details. This method helps in precise shape recognition by activating mechanoreceptors that respond to tracing movements [40].

Experimental paradigms often include forced-choice procedures and magnitude estimation tasks to study texture perception. In forced-choice tasks, participants select between different textures based on tactile exploration, while in magnitude estimation tasks, they rate the roughness or fineness of textures on a scale. For example, studies have used comparisons of different materials under varying conditions to understand how factors like texture complexity and exploratory movements influence tactile perception.

These paradigms have shown that fine texture discrimination relies heavily on vibrational cues produced by lateral finger movements, which generate high-frequency vibrations detected by Pacinian corpuscles [26].

2.3.3 Effects of Movement Speed

Texture perception is significantly influenced by the speed at which an object moves across the skin, affecting both the vibrational and spatial cues that the skin detects. Several studies have investigated how movement speed impacts the perception of roughness and texture.

One finding is that the perceived roughness of textures varies with movement speed. The roughness magnitude estimation function shifts along the spacing axis as speed increases, indicating that higher speeds reduce interaction time with texture elements, thus altering roughness perception. The study also noted that this effect was more pronounced under passive touch conditions, where speed is externally controlled, compared to active touch conditions, where participants control the movement themselves. Specifically, they found that at higher speeds, the perceived roughness tended to peak at larger spacing values, suggesting that the temporal characteristics of the interaction play a critical role in texture perception [41].

Further research explored how scanning velocity affects roughness perception using a belt apparatus to vary the speed at which textures are scanned across the skin. Roughness perception for coarse surfaces increased with scanning speed, likely due to the stronger low-frequency vibrations they produce, while for fine surfaces, it decreased. This suggests a complex interaction between surface characteristics and movement speed, supporting a theory of roughness perception where roughness increases with the intensity of texture-induced vibrations. Coarse textures, which produce lower frequency vibrations, became more pronounced at higher speeds, whereas fine textures, which produce higher frequency vibrations, were better perceived at slower speeds [42].

Another study examining the effect of speed on texture perception with a probe found that speed influences both vibratory amplitude and frequency. Higher speeds increase vibratory frequencies, while the effect on amplitude depends on the texture. Coarser textures often generate higher amplitudes at increased speeds, enhancing roughness perception, while for finer textures, higher speeds may reduce vibratory amplitude but increase frequency, altering the perception of these textures. This dual effect highlights the importance of considering both amplitude and frequency when studying texture perception [41].

In a more recent study, haptic texture rendering using adaptive fractional differential methods was investigated. They found that different scanning speeds affected the participants' ability to classify textures accurately. Their experiment demonstrated that the method could effectively convey texture information, with classification accuracies ranging from 72% to 91% depending on the image and speed settings. The study highlighted that higher scanning speeds could enhance the clarity of vibrational cues, aiding in more

accurate texture discrimination [43].

Furthermore, experiments were conducted to understand how speed affects the tactile perception of various textures. They observed that the perceived roughness of fine textures was significantly influenced by scanning speed, with slower speeds enhancing the perception of fine details. Their study confirmed that Pacinian corpuscles play a crucial role in detecting high-frequency vibrations, that are crucial in distinguishing fine texture details and are more pronounced at higher scanning speeds, thus affecting the overall texture perception[26].

Overall, the speed of movement plays a crucial role in texture perception by modulating vibratory and spatial cues. Faster movements generally increase vibratory frequency while the effect on amplitude depends on the texture characteristics, leading to complex interactions that influence perceived roughness. For coarse textures, higher speeds can increase both amplitude and frequency, enhancing the perception of roughness contrasting to finer textures, where higher speeds tend to increase frequency but may reduce amplitude, leading to a smoother perceived texture. These findings highlight the importance of considering movement speed in tactile texture perception studies and the design of haptic devices, where precise control over these variables can enhance user experience and the overall realism. In this thesis, the final feedback is modulated based on the user's hand speed when interacting with virtual objects.

2.3.4 The Duplex Theory

The duplex theory of tactile texture perception, proposed by David Katz, provides a complete model for understanding how humans perceive textures through touch. According to this theory, texture perception is based on two distinct types of cues: spatial and vibrational. Coarse textures are primarily perceived through spatial cues, involving the size, shape, and arrangement of surface elements directly sensed by the skin. Fine textures, on the other hand, are detected through vibrational cues generated during the lateral movement of fingers across a surface. These fine textures generate high-frequency vibrations that are sensed by Pacinian corpuscles, sensitive to high-frequency vibrations (40–500 Hz), while SA1 mechanoreceptors respond to static and low-frequency signals that capture spatial details. This dual mechanism enables humans to distinguish both coarse and fine textures [22], [39], [40].

Dynamic touch, or the lateral movement of fingers, significantly enhances the perception of fine textures by generating vibrational cues. Without movement, fine textures become indistinguishable because static touch lacks the vibratory information needed for detection. Studies have shown that fine textures with element sizes below 100 μm are easily identified when the skin moves across them but become indistinguishable without movement. In contrast, coarse textures, with element sizes above 100 μm, remain perceivable due to their spatial properties [39].

Experiments on roughness perception for linear gratings (surfaces that consist of patterns of grooves and ridges), revealed that the width of the grooves and the force ap-

plied by the fingers significantly affected how rough the textures felt, with grooves wider than 200 μm being mainly detected by SA1 mechanoreceptors.[40] Another study examined how the way that the skin is deformed by different textures, affects our sense of touch. It found that both the shape of these deformations and the vibrations they create play important roles in how humans perceive texture. Fine textures, which create high-frequency vibrations above 100 Hz, were particularly dependent on these vibrational cues. The study showed that Pacinian corpuscles, which are most sensitive around 250 Hz, are crucial for detecting these fine textures [44].

The cerebral cortex processes haptic information from various mechanoreceptors, merging spatial and vibratory cues to create a cohesive perception of texture. Neuroimaging data reveal that distinct cortical areas are activated by spatial and vibrational stimuli, indicating specialized processing pathways for each type of cue, supporting the duplex theory's distinction between spatial and vibrational texture perception [33].

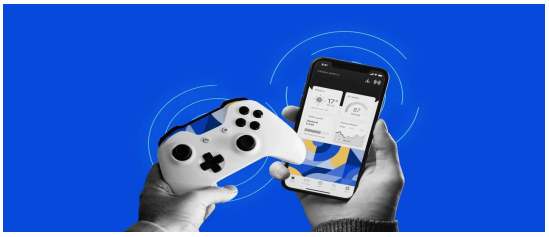
Numerous experimental approaches have been developed to study the duplex theory. These include forced-choice procedures, where participants select between different textures based on tactile exploration, and magnitude estimation, where subjects rate the roughness or fineness of textures on a scale. Comparisons between passive touch (stationary finger, moving surface) and active touch (moving finger, stationary surface) reveal that fine texture discrimination is significantly impaired under passive conditions, underscoring the necessity of movement for generating vibrational cues [40] [39].

In this thesis, the duplex theory provides a fundamental framework for understanding texture perception, explaining how spatial and vibrational cues enable the recognition and discrimination of both coarse and fine textures which are both studied in this work.

2.4 Haptic Technology

Haptic technology identifies all the technologies that provide the sensation of touch feedback, by applying forces, vibrations, or motions to the user, to simulate the sensations that would be felt when they interact directly with physical objects, in remote operations or computer simulations. This technology is often referred to as "haptics", a term that comes from the greek word "haptikos" meaning "concerning the sense of touch" [45]. Haptics field has been studied for many decades now, and haptic feedback technology has evolved significantly over the past few years, transforming the way humans interact with digital and virtual environments. One of the earliest applications of haptic technology was in large aircrafts that used servomechanism systems to operate control surfaces. In more recent years, haptics have become a fast evolving area. With virtual reality technologies maturing, the demand for consumer-grade haptics will also intensify. Given the importance of the hand for interactions and its great sensitivity, many haptic devices have focused on the hand or forearm. The demand for more realistic virtual textures is increasing in areas such as social interactions via computer systems, clinical medicine for guiding refined rehabilitation protocols, in the context of stroke survivors and amputees

and also in gaming and entertainment.



URL Source

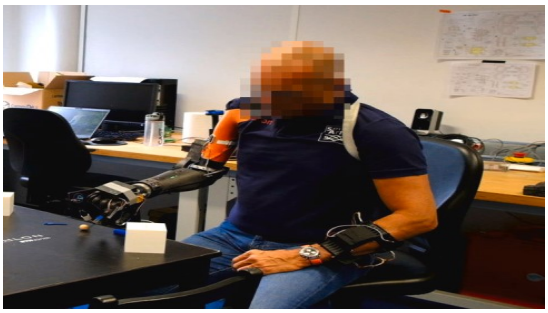
(a) Haptic Feedback through rumble in everyday objects.

URL Source

(b) Haptic Feedback in Virtual Reality Surgeries.

URL Source

(c) HaptX Haptic Glove.



URL Source

(d) MuViSS haptic device in conjunction with the Taska Hand for tactile feedback to amputee users.

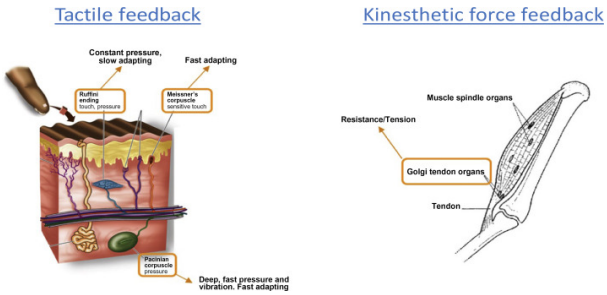
Figure 2.9: Practical application of haptic feedback.

The most well-known examples of haptics are probably the vibration in a mobile phone or the rumble in a game controller. These forms of haptic feedback alert users to notifications or enhance the gaming experience by simulating physical impact. Furthermore, in the premium automotive segment, touch operated haptic devices are integrated into control systems to provide feedback, improving the user's interaction with the vehicle's interface. One example of this application is Mercedes-Benz who introduced haptic touch screens that provide tactile feedback to the driver, enhancing control and reducing distractions. Another area of use of haptic feedback is in medical training simulators, where it replicates the tactile sensation of interacting with human tissues, allowing medical professionals to practice procedures with a high degree of realism. This application improves the training process by providing a safe environment to develop and refine surgical skills [46]. Haptic technologies also become increasingly popular in the entertainment area as well. Haptic devices like gloves and exoskeletons provide immersive experiences by simulating the physical sensations of virtual objects. This technology enables users to feel textures, resistance, and weight, enhancing the realism of virtual reality applications [45], [47]. Designing haptic devices that mimic human touch perception is very important

in teleoperation as well, where operators manipulate distant environments using robotic interfaces that provide feedback, making them feel as though they are directly interacting with objects and also in prosthetics, where integrating haptic feedback can restore a sense of touch to amputees, significantly improving their quality of life. However, current limitations in hardware reduce the expressiveness of haptic feedback, and high costs and complexity hinder widespread adoption in consumer devices. Additionally, many devices fail to stimulate the full range of human mechanoreceptors, limiting the realism of the feedback provided [48].

2.4.1 Types of Haptic Feedback

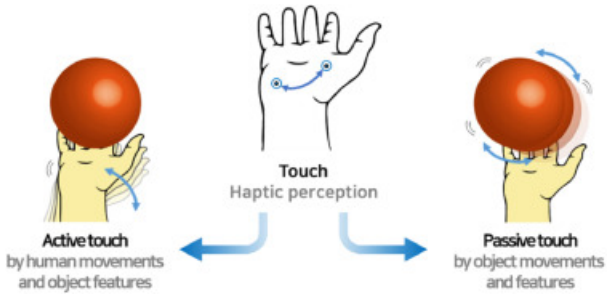
Touch feedback is created by applying forces, vibrations, or motions to the user. Haptic interfaces are divided into two groups based on the type of stimuli: tactile (cutaneous), which applies forces to the skin surface to recreate sensations like texture, pressure, and vibration, and kinesthetic, which involves sensing body movement and muscle strength, allowing users to feel the force and torques exerted upon contact through receptors. Force feedback, typically associated with kinesthetic feedback, involves applying forces that simulate resistance or pressure, allowing the user to sense physical interactions with virtual or real objects.



URL Source

Figure 2.10: Tactile and Kinesthetic Feedback.

There are two main ways to deliver force feedback; active and passive. In active haptics, the user actively controls the interaction, exploring surfaces and shapes by moving their hands. This mode is typically used for detailed texture and shape recognition, providing a higher quality of information. For instance, users can explore virtual textures or the contours of objects in a virtual reality environment to gain a detailed understanding of their properties [45], [49]. On the other hand, in passive haptics, the device controls the interaction, guiding the user's movements. This mode is beneficial for applications like navigation assistance, where the user follows predefined pathways. An example is a navigation system for visually impaired individuals, where the haptic device guides the user through a space, providing directional cues through force feedback [45], [49].



URL Source

Figure 2.11: Active and Passive Haptics.

2.4.1.1 Kinesthetic Feedback

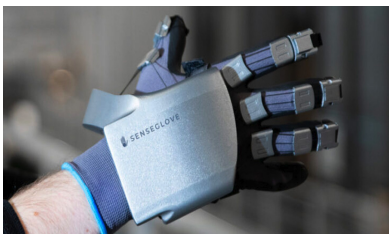
Kinesthetic feedback involves the perception of forces, motions, and resistance through muscles, tendons, and joints, providing information about the position and movement of the body parts interacting with a haptic device. This type of feedback is essential in applications where understanding the force and motion dynamics is critical, such as in surgical training, physical therapy, and virtual prototyping. Furthermore, it finds great use in applications such as virtual reality, gaming, and industrial robotics, where it enhances the user's sense of presence and interaction. Kinesthetic feedback is typically delivered through devices that apply forces or restrict movements to simulate physical interactions. These mechanisms include grounded force feedback systems, exoskeleton-based feedback systems, and cable-driven actuators [45], [50].

URL Source

(a) CyberGrasp.

URL Source

(b) Dexmo



URL Source

(c) SenseGlove

Figure 2.12: Kinesthetic devices.

Exoskeletons are structures that the user wears over their hand, transmitting forces providing kinesthetic feedback. Actuators are used to exert forces and displacements on the order of a few Newtons and centimeters, respectively. Early kinesthetic feedback systems made use of hydraulic systems placed on exoskeleton devices attached to the arms, leading to bulky and difficult to use devices, restraining the use outside of laboratories.

Next, pneumatic actuators became widely used in this field solving the problems of bulkiness, weight and comfort, although the need for a source for pressurized fluid is still a major issue. More recent approaches for kinesthetic feedback make use of cable-based actuators or strings attached to motors that pull them accordingly [45]. CyberGrasp is the most famous commercial exoskeleton that conveys force to the fingertips without constraining other joints. It's maximum force is 12N which completely stops the movement of a finger. However, the users' fingertips are constantly pulled backwards even when nothing is happening due to the fixed design, making the experience less natural. Another commercial exoskeleton is Dexmo by Dexta Robotics which resembles a large claw and uses servo motors for force feedback limited to one degree-of-freedom per finger. The exoskeleton measures finger flexion and abduction, plus one rotation for the thumb. HaptX Glove from Haptx Inc. looks like an armored glove and uses a biomimetic exoskeleton for resistive force feedback delivering up to 22N. HaptX Gloves are focused on VR training and simulation applications for industrial users. Another example is the VRgluv which is actuated by DC motors pulling cables providing 12-DOF with 20N force feedback. SenseGlove uses magnetic brakes instead of traditional motors to transmit force between the wires and the fingertips [51]. More examples include the Ekso Bionics exoskeleton for assisting individuals with spinal cord injuries and the ReWalk exoskeleton designed for paraplegics, enabling them to stand, walk, and even climb stairs. The partial-body exoskeletons focus on specific areas of the body and are used for more targeted support. They are used in various applications, such as MyoPro for assisting individuals with upper limb mobility issues and Keeogo for enhancing lower limb movement and gait improvement [45], [50].

2.4.1.2 Cutaneous Feedback

Cutaneous feedback involves the stimulation of the skin's surface to convey information such as texture, temperature, and vibration. This type of feedback is crucial for simulating surface properties and fine details, enhancing the realism and immersion of the interaction. It relies on the activation of mechanoreceptors located in the skin, primarily at the fingertips, which are responsible for detecting pressure and vibration stimuli. Mechanisms delivering cutaneous feedback typically include vibrotactile stimulation, skin indentation, skin stretch and shear, thermal feedback, and electrotactile stimulation. Most devices provide vibrotactile stimulation, with advantages such as cost, power need and size. All designers are looking for smaller components for these devices and seeking to maximize battery life with solutions that offer maximum vibration while consuming less power [45]. This type of feedback is crucial for tasks that require detailed texture recognition and fine tactile discrimination, enhancing the user's ability to perceive and interact with virtual objects realistically, making it essential for applications such as virtual reality, gaming, and training simulations. The integration of multiple tactile sensations allows for more immersive virtual environments, aiding in tasks like object discrimination and texture identification. Devices implementing cutaneous feedback often employ various actuators

to simulate these sensations accurately, improving user interaction and engagement[46].

To recreate the sensations that are experienced through touch, different techniques have been developed to deliver cutaneous feedback. Each method helps simulate sensations like texture, pressure, or temperature, making virtual interactions feel more realistic, leading to more immersive experiences. Below are some of the ways haptic devices generate these tactile sensations.

- Vibrotactile Stimulation

  Vibrotactile stimulation is the most commonly used form of cutaneous feedback. It uses small actuators to create vibrations that simulate texture and surface variations. This method is widely implemented in various consumer devices like smartphones, gaming controllers, and virtual reality systems such as the Oculus Quest and HTC Vive that use vibrotactile actuators to simulate the feeling of interacting with surfaces in virtural reality environments. Vibrotactile feedback provides the sensation of texture and slippage by creating microvibrations within the range of 1 Hz to 500 Hz, which are perceived by the human touch receptors.

- Skin Indentaion

  Indentation feedback involves actuators that apply a normal force to the skin, creating a sense of weight and shape. Devices using this method often attach directly to the skin or through haptic gloves. For example, TacTiles use pins attached to electromagnetic actuators to simulate the sensation of grasping objects by locking in place. This method is effective, however it can add bulk to the device, potentially restricting movement and usability [52].

- Skin Stretch

  Skin stretch feedback activates mechanoreceptors by stretching the skin, simulating sensations as friction and directional forces. This method often uses platforms attached to the fingers, driven by motors to create normal and shear forces, providing a sense of motion and grip. Some advanced systems use fabric or elastic instead of solid platforms, allowing for a smaller form factor and more flexible application. For instance, hRing uses servomotors to deform the skin on the user's finger.

- Electrotactile Stimulation:

  Electrotactile feedback uses electrical impulses to directly stimulate the nerves in the skin, creating sensations of touch and texture. This method can produce a wide range of sensations and is particularly useful in prosthetics. However, it requires precise control and can be invasive if microelectrodes are used. Non-invasive methods, such as transcutaneous electrical nerve stimulation (TENS), are also employed to provide sensory feedback without the need for surgery. One application is FinGAR, a device that combines electrical land mechanical actuation to simulate different tactile dimensions [45], [46].

- Thermal Feedback

  Thermal feedback involves changing the temperature of the device's surface to simulate the thermal properties of virtual objects. Devices with Peltier elements can heat up or cool down to provide thermal sensations, enhancing the realism of virtual interactions. This type of feedback is particularly effective in simulating different environmental conditions and material properties, such as metal being cold to the touch or fabric being warm [53].

In this thesis, the haptic feedback is delivered by a vibrotactile actuator alone, in order to represent different recorded textures through vibrations, due to their compact size and low power needs.

2.4.2 Classification of Haptic Devices

Haptic devices can be classified based on their form factor and the method of interaction they provide. Each type offers unique advantages and is suited to different applications in fields such as virtual reality, medical simulations, and assistive technology.

- Grounded Devices

  These systems are stationary and provide strong, precise force feedback through an end effector that the user interacts with. They are mounted on a fixed platform and deliver high-fidelity feedback suitable for tasks requiring precision and stability, such as surgical simulations and complex mechanical design. Grounded devices are crucial in environments where exact force replication is necessary, such as virtual prototyping and training simulations [50]. Devices that fall under this category are for example the Phantom haptic device, which uses a stylus to provide detailed force feedback for virtual object manipulation, [49] or joysticks and steering wheels such as the Logitech G29, that simulates the physical sensations of driving such as road texture and collisions. These devices use force feedback to simulate interactions such as pushing, pulling, and feeling the weight of virtual objects [45], [49].

URL Source

Figure 2.13: PHANTOM Omni Haptic Device.

- Handheld Devices

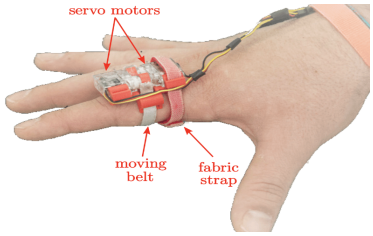
  Handheld haptic devices are portable and often used in applications requiring manual manipulation and feedback. They typically incorporate force feedback to simulate the sensation of interacting with virtual objects, with devices like VR controllers (e.g. Oculus Touch), which use force feedback to enhance the realism of virtual interactions by simulating the sensation of touching and manipulating objects and handheld styluses being common examples. These devices are widely used in gaming and virtual reality applications, where they simulate interactions such as pushing, pulling, and feeling the weight of virtual objects. Handheld devices are known for their mobility and ease of use, making them accessible for a wide range of users [45], [47].

URL Source

Figure 2.14: Oculus touch controllers.

- Wearable Devices

  Wearable haptic devices provide continuous tactile feedback and are mounted on the body. This category includes haptic gloves, fingertip devices, and other wearables that offer detailed feedback for different body parts. Haptic gloves and fingertip devices can provide both kinesthetic and cutaneous feedback to the hands and fingers and are equipped with various actuators to simulate a wide range of sensations, including texture, temperature, and force. Examples include the SenseGlove and Manus VR gloves, which use a combination of force feedback and vibrotactile actuators to enhance the immersive experience in VR environments. Fingertip devices like the HapTip and FinGAR focus on delivering detailed tactile information through mechanisms such as skin stretch, vibration, and electrotactile stimulation. Other wearables, such as the Teslasuit, provide full-body haptic feedback by integrating electrostimulation to simulate various physical sensations. Smaller wearables, such as haptic bracelets and armbands, use vibrations to provide feedback for notifications or guidance, particularly useful in applications such as fitness training, rehabilitation, and assistive technologies [45], [46].



URL Source

(a) hRing fingertip device.

URL Source

(b) TeslaSuit.

URL Source

(c) BalanceBelt.
Used for patients suffering from Bilateral Vestibular Loss.

Figure 2.15: Wearable haptic devices.

- Mid-Air Haptics

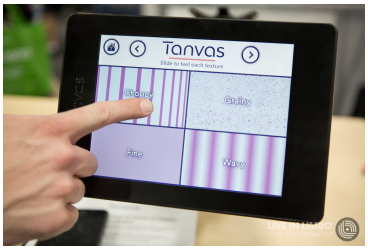
  Mid-air haptic devices create tactile sensations in the air, without direct contact with the user's skin. They use focused ultrasound or air jets to generate touchless haptic feedback. Ultrahaptics technology uses ultrasound waves to create tactile sensations in mid-air. This technology is often used in public interfaces and interactive displays, allowing users to feel virtual objects without physical contact. Mid-air haptics is particularly appealing for hygienic interactions and enhancing user experience in touchless environments [47].

URL Source

Figure 2.16: STRATOS Inspire, by Ultraleap.

- Surface Haptics

  Surface haptic devices provide tactile feedback directly on touch surfaces such as screens and touchpads. These devices use various methods to create the sensation of texture and friction on a flat surface. Technologies like TanvasTouch use electrostatic forces to create the sensation of texture on touchscreens. Users can feel different textures as they swipe across the screen, which enhances the interactive experience of touch interfaces. Surface haptics is useful in applications ranging from automotive controls to advanced touchscreen devices, providing a more tactile interaction with digital content [46].



URL Source

Figure 2.17: TanvasTouch.

- Teleoperation Devices

  Teleoperation haptic devices are used to control remote robots or machinery, providing the operator with tactile and force feedback from the remote environment. This enhances precision and control in remote operations. Master-slave systems in surgical robots like the da Vinci Surgical System provide haptic feedback to the surgeon, allowing for precise manipulation of instruments during surgery. These systems improve the accuracy and effectiveness of remote manipulations, making them invaluable in fields requiring high precision and safety. [47]

URL Source

Figure 2.18: The da Vinci teleoperated surgical system.

In this thesis, the developed interface falls under the wearable category, since all the hardware components are mounted on the user's hand, with haptic feedback delivered on the fingertip of the index finger. This means that size and comfort needed to be considered in the developing stages.

2.5 Haptic Texture Technologies

Texture perception involves a complex interaction between spatial and vibrational cues, both of which are essential for recognizing and distinguishing between different surfaces. Numerous studies have employed advanced methods and conducted experiments to explore these perceptual processes and recreate realistic haptic feedback. In this section, some of these techniques will be explored, as they are enlightening for achieving the primary goal of this thesis which is texture representation.

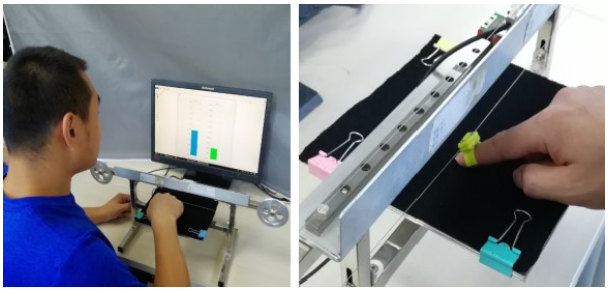
2.5.1 Databases and Toolkits for Haptic Textures

The creation and use of databases and toolkits are crucial for advancing haptic texture recognition and rendering. This section highlights several key resources that provide standardized datasets and tools for modeling and evaluating haptic textures. By offering consistent and reliable data, these databases support researchers in developing new algorithms and techniques as well as promoting collaboration and innovation in the field of haptics.

One study introduces a comprehensive haptic texture database that captures high-frequency acceleration signals during tool-mediated interactions with different surfaces. Using an accelerometer, they record these vibrations during both controlled and free-hand explorations of 43 distinct textures. The main goal was to analyze and classify textures by extracting relevant features from the recorded signals, with Mel-Frequency Cepstral Coefficients (MFCCs) proving particularly effective. The database provides a resource for the evaluation of feature robustness against variations in force and velocity. However, ensuring consistent performance remains challenging due to variability in user interactions and the complexity of capturing high-frequency signals. The study shows that detailed data capture and evaluation remains essential for further advancements in the field [54].

Another resource is the HapTex database, which is designed to support the development of haptic devices, by providing a wide range of fabric textures with corresponding tactile data. The database captured high-quality tactile information using specialized sensors, ensuring accuracy and reliability. When these textures are integrated into haptic devices, users can experience realistic fabric simulations in virtual environments. However, its focus on fabric textures may limit broader applicability and the accuracy of simulations depends on the capabilities of the haptic devices used. This variability leads to the need for continued research and development to ensure consistent and high-fidelity tactile feedback across different haptic platforms [55].

The Penn Haptic Texture Toolkit (HaTT) offers 100 texture and friction models, complete with data, images, and rendering code. Designed to assist researchers in comparing and validating texture modeling methods, the toolkit supports the use of devices like the SensAble Phantom Omni. Nonetheless, the toolkit requires specific hardware for optimal use, which may limit accessibility for some users and the toolkit's complexity may present



URL Source

Figure 2.19: The recording procedure of HapTex database.

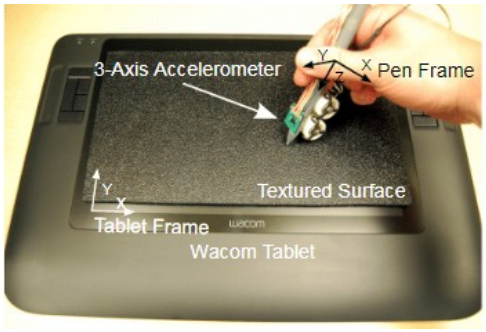
a learning curve for new users [56].

2.5.2 Methods for Virtual Texture Creation

Creating realistic virtual textures is a main challenge in the field of haptics, as it requires sophisticated modeling and rendering techniques. This section focuses on methods for generating and rendering virtual textures that mirror real-world sensations.

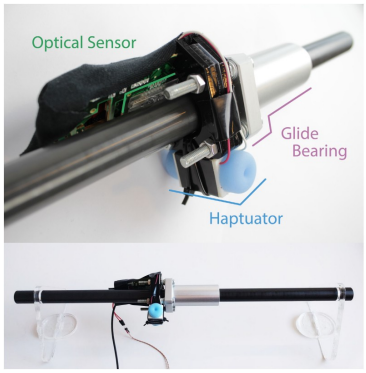
The creation of realistic virtual textures using contact acceleration data collected during interactions with physical surfaces was the scope of one of the studies, with very promising results. Researchers measured vibrations of interactions with isotropic textured surfaces using an accelerometer fixed to a Cintiq 12WX interactive pen, as well as normal force and scanning speed measured by the Wacom tablet. By processing and combining these data with custom procedures and techniques into meaningful signals and creating new hardware for rendering them using voice coil actuators, they succeeded in recreating realistic textures based on user evaluation. However, the complexity of capturing and accurately reproducing high-frequency data poses significant challenges, as advanced and sensitive equipment is required, making the process complex and costly, in addition to hardware limitations that may impact the fidelity of the rendered textures. Despite these challenges, the method's ability to provide detailed tactile feedback marks a significant improvement in the quality of virtual interactions [57].

Another approach simulates various textures by adjusting actuation parameters to movement. A physical slider equipped with a vibrotactile actuator was used, as well as an optical sensor to synchronize vibrations with user actions, enhancing the realism of the simulated textures. An experiment where users interacted with the actuators to experience different vibration patterns replicating sensations like roughness, bumpiness, adhesiveness and sharpness, revealed that higher amplitudes significantly enhanced roughness and bumpiness, while granularity and timbre adjustments played crucial roles in distinguishing textures. While effective for many textures, this method is limited to those that can be conveyed through vibration, lacking the capacity to replicate more complex textures involving additional sensory elements [58].



URL Source

Figure 2.20: The data collection system using the Wacom tablet.



URL Source

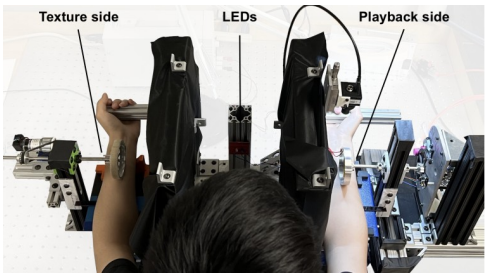
Figure 2.21: The slider used in the above study.

Haptography is analogous to photography, but instead of capturing visual images, it captures the tactile feel of surfaces to recreate them in virtual environments. This method involves recording high-frequency acceleration signals during tool-mediated interactions, using a sensorized tool to capture detailed haptic data, including vibrations and forces. The captured data is then modeled using techniques like linear prediction to create high-fidelity texture models. These models are used in active stylus rendering, where an active stylus reproduces the recorded surface textures, allowing users to experience the tactile qualities of real surfaces virtually. Despite its potential, haptography faces significant challenges. Capturing and accurately rendering real-world textures is complex, and current haptic interfaces often struggle with the bandwidth required for precise feedback. The focus on tool-mediated interactions may limit the applicability to direct-touch scenarios, where users interact with surfaces using their fingers rather than tools [59].

This study investigates the challenges and solutions in achieving realistic haptic texture rendering for virtual environments. By a combination of experimental studies and psychophysical analysis, it focuses on identifying and mitigating "perceptual instability",

as they define the unrealistic sensations users experience with haptic feedback, such as buzzing. They examine various texture rendering models, including sinusoidal gratings, and evaluate their effectiveness in producing stable and realistic haptic sensations. The findings underscore the importance of fine-tuning both the haptic interface and texture-rendering algorithms to overcome computational complexities and deliver perceptually stable, high-quality textures [60].

Another technique simulates realistic tactile textures on the volar forearm by combining skin stretch and vibrotactile feedback. The researchers recorded the mechanical responses of the skin during texture interaction using the digital image correlation (DIC) method, which captured longitudinal displacements of the skin as various textures were scanned across the forearm. These recorded skin displacements were then replayed on the opposite forearm using three different modes: stretch-only, vibration-only, and a combination of both. User feedback showed that a combination of both stretch and vibration produced a more realistic perception of textures compared to either modality alone, however it was not enough to replicate the complexity of natural touch. While this method provides a rich tactile experience, further development is needed to fully replicate real-world textures, with the authors suggesting multi-axis or spatially distributed stimuli to improve realism [61].



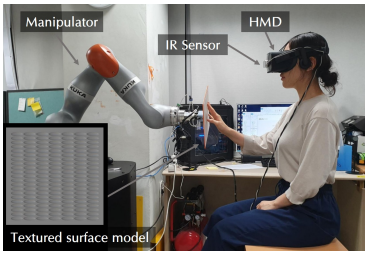
URL Source

Figure 2.22: Top view of the apparatus with the DIC setup on the playback side.

One advanced technique developed in texture perception research is an adaptive fractional differential method for haptic texture rendering, based on 2D images. This method uses a fractional differential operator to enhance texture features by preserving low-frequency image contours and augmenting high-frequency details. Its ability to adaptively select the fractional differential order based on the texture characteristics of the image, enables more accurate extraction of texture information. By analyzing image gradients, they create tactile maps that represent texture depth and roughness, that participants interact with through a Phantom Omni haptic device that renders these maps into tactile sensations, allowing them to feel the textures illustrated in images. The results showed that participants, even when blindfolded, achieved high classification accuracy ranging from 72% to 91% for certain image types, when identifying the rendered textures, demonstrating the method's effectiveness. However, its complexity requires high computational

power, and it may not capture the full range of tactile nuances present in real textures [43] .

A novel method for synthesizing the roughness of textured surfaces using spatiotemporal encoding is introduced by another study. The approach involves a grid of hemiellipsoidal bumps on a surface, where roughness is encoded based on the spatial orientation and velocity of the user's hand relative to the surface. The method allows users to feel different levels of texture roughness without the need for complex actuators, supporting both active and passive tactile experiences. During user psychophysical tests, participants reported perceiving different levels of roughness, confirming the effectiveness of the encoding method. The results showed that increasing surface orientation and hand motion velocity significantly increased perceived roughness. However, the study's limitations include restricted user movement during experiments, which may have affected the realism of the results, and a focus primarily on macro-scale textures without considering fine-scale roughness. Additionally, the limited diversity of participants in the study may impact the generalizability of the findings [62].



URL Source

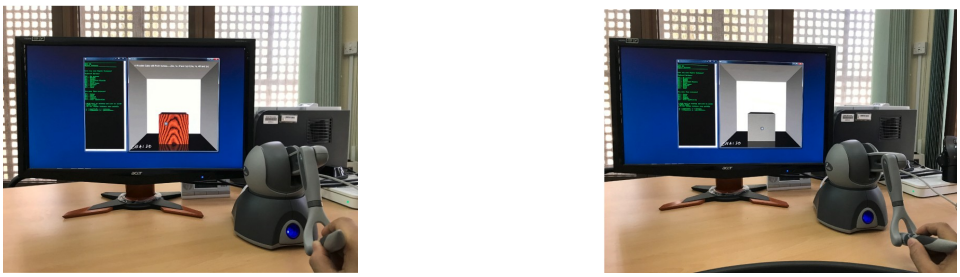
Figure 2.23: The experimental setup of the above study.

Generative adversarial networks (GANs) have also been employed for preference-driven texture modeling, combining user preferences with interactive evolution strategies. This approach generates virtual textures by interpolating between real texture models from the Penn Haptic Texture Toolkit (HaTT), based on user feedback, without requiring direct data collection typically done in data-driven approaches. Users interact with both real and virtual textures through an iterative process, ranking the generated textures based on thei preferences with the algorithm adjusting the models to match the users's expectation. This technique is comparable to data-driven models particularly in terms of roughness and slipperiness perception. While effective, this method has limitations regarding the capture of fine details in some rough textures leading to incosistencies. [63]

Transcutaneous electrical nerve stimulation (TENS) offers an innovative method for delivering tactile feedback for recognizing object shapes and surface topologies without physical contact. Using a 2x8 electrode grid placed on the upper arm to stimulate the median and ulnar nerves, this approach translated fingertip forces detected by a prosthetic hand's fingertip into stimulation intensity and intervals between electrical pulses, allowing users to identify and differentiate textures and shapes. In the conducted experiments, the

electrodes produced sensations at different fingers, enabling participants to recognzie object shapes and surface topologies with accuracies above 84%. Participants achieved high accuracy as well when recognizing both shape and surface features together. Although this non-invasive method showed great accuracy levels, it requires precise callibration and may cause discomfort with prolonged use. The technique is promising for applications in prosthetics and virtual environments, enhancing object manipulation and user experience, but the challenges of careful setup and user adaptation need to be addressed [64].

Lastly an interesting work, introduced a method for generating haptic textures using Perlin noise to simulate texture vibrations in a virtual environment. This method is designed for generating textures at runtime using a PHANToM Omni haptic device, with minimal impact on the haptic rendering performance. The solid noise approach enables texture generation that is independent of the object's geometry, meaning it can be applied to various shapes without requiring additional computation. The texture force was calculated based on the tool's position and involves parameters like frequency and persistence, which control the behavior of the noise function and can recreate a wide range of textures from smooth to rough. Participants interacted with virtual cubes of different textures, in haptic-visual (with graphical textures) and haptic-only (without graphical textures) modes, rating the realism and graphical correspondence of textures and identify them using only haptic feedback. The results showed an average texture recognition accuracy of 84.8%, with textures like gravel and pebbles achieving the highest recognition rates, while textures like wood and granite were more difficult to distinguish. Participants rated the realism and correspondence between visual and haptic textures highly. However, some textures were harder to distinguish, indicating a need for further optimization to improve texture realism and recognition accuracy [65].



(a) With visual graphical textures. (b) Without visual graphical textures.

URL Source

Figure 2.24: The experimental setup of the Perlin Noise study evaluation.

Inspired by studies that use accelerometers and custom hardware to render virtual textures, this thesis captures audio and vibrational data during real-world interactions to create realistic texture feedback. Similar to the work using Wacom tablets and voice coil actuators, this system emphasizes high-frequency vibrational data but achieves it with a

simplified and cost-effective setup. The method focuses on direct mapping to vibrotactile actuators, avoiding the complexities of custom rendering devices while maintaining perceptual realism.

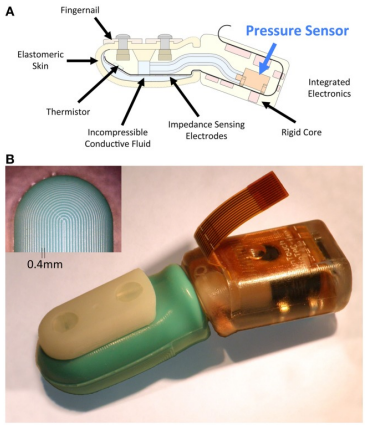
2.5.3 Techniques for Texture Recognition and Classification

Accurate texture recognition and classification are essential for applications ranging from virtual reality to robotic manipulation. This section explores different methodologies and technologies used to classify textures based on haptic feedback. The research presented in this section underscores the importance of precise and reliable texture recognition systems in advancing haptic technology.

One study evaluates various haptic feedback modalities, force feedback, vibrotactile feedback, and direct tactile stimulation, in shape and texture recognition tasks. The experiment involved two sets of patters, unidimensional textures and geometric shapes, that the users experienced through each of the tree modalities, with force feedback being delivered by a Phantom Premium 1.0A device, that generated mechanical responses to users interactions with virtual textured surfaces. Vibrotactile feedback wa delivered through a custom designed glove with actuators on the fingertip and direct tactile simulation involved participants physically touching and identifying patterns made from transparency paper with tangible depressions. The results showed that direct tactile stimulation was the most effective for shape recognition, while vibrotactile feedback excelled at identifying textured patterns, particularly those involving fine-grained surface variations. The study highlights the potential of vibrotactile feedback but notes that challenges remain in improving its precision for complex surfaces [66].

This research investigates texture recognition using data collected from a bionic tactile sensor, BioTac SP, designed to mimic human touch, by analyzing vibration data generated as the sensor slides against different materials under controlled conditions of speed and pressure. The sensor captures detailed information about surface textures, providing a rich dataset for texture classification through processing by three machine learning models. The study demonstrates high accuracy in recognizing different textures by the CNN model, which outperformed traditional machine learning algorithms. The results demonstrate the potential of advanced tactile sensors in achieving high-accuracy texture recognition with relatively low computational time. However, the complexity and cost of developing bionic tactile sensors present challenges [67].

This thesis addresses texture recognition challenges similar to those highlighted in studies using BioTac SP sensors or vibrotactile gloves. Rather than relying on complex sensors or machine learning models, this work extracts features directly from audio and accelerometer signals to classify textures like sandpaper and combs. The approach draws on insights from vibrotactile feedback research, simplifying implementation while maintaining accuracy and responsiveness to variations in interaction speed and force.



URL Source

Figure 2.25: The BioTac bionic tactile sensor.

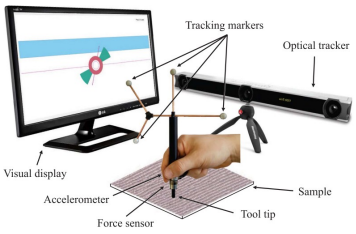
2.5.4 Integration of Multisensory Data and Machine Learning

As mentioned in previous sections, the integration of multisensory data is critical for creating comprehensive and immersive virtual experiences. In addition, machine learning and data-driven approaches have changed the field of haptic texture modeling, enabling the development of algorithms that can predict and render textures in real-time. This section discusses how multisensory cues can enhance texture perception and interaction in haptic feedback as well as the advancements that have occured from learning-based approaches.

One approach, is the investigation of the relationship between visual and haptic features of textures by using a one-dimensional convolutional neural network (1D-CNN) to predict haptic texture attributes from images without the need for direct physical interaction. This method focuses on establishing a haptic texture attribute space, based on psychophysical experiments where participants rated 100 real textures on attributes such as rough-smooth, flat-bumpy, sticky-slippery, and hard-soft. To predict these attributes from images, the study analyzed image features, using extraction techniques, which together capture spatial patterns and high-level features from texture images and were used to train to model. The model's predictions highly correlated with the participant's ratings, with its error ratings once again outperforming traditional machine learning algorithms, showing the potential for accurate prediction of haptic attributes form visual data. However, the method's effectiveness depends on high-quality visual data, and the complexity of real-world textures may limit the accuracy of predictions, plus the implementation and training require high level of expertise in deep learning and image processing [68].

Another study, integrated visual guidance into the acquisition of contact dynamics for haptic texture modeling. The study presents an interactive framework where a user interface provides visual feedback to guide data collection, allowing users to more effec-

tively gather contact dynamics data. They used algorithms for real-time segmentation of input signals, for the collection of data related to force, vibration and friction during tactile interactions. The results of the study demonstrate the enhancement of capturing complex contact dynamics through visual guidance during the data collection process. Again though, the variability of real world textures raises challenges in obtaining a broad and varied dataset for accurate representation [69].



URL Source

Figure 2.26: The data collection setup with visual guidance.

A deep learning-based model for real-time haptic texture rendering has been developed, which uses data from a vision-based tactile sensor and user interaction data to generate realistic haptic textures. The model is designed to be unified across multiple textures, eliminating the need to create a separate model for each texture, which is a limitation in many existing data-driven approaches. The system predicts the vibration acceleration signals that simulate the feeling of a surface when a probe interacts with it and the signals are rendered in real-time using a vibrotactile transducer. This work provides the ability to generalize previously unseen textures allowing for haptic texture rendering without extensive additional data collection, marking an advancement in texture classification and rendering [70].

In another study, the dominance of tactile over visual cues in the speeded discrimination of roughness was examined. Participants involved, had to make rapid judgments about the roughness of "pilled" textile samples in various levels, while they were instructed to focus on either visual or tactile stimuli, with both present simultaneously. The results showed that tactile feedback often prevails over visual feedback, especially for smooth textures, emphasizing the critical role of touch in texture perception. The study employed contrasting combinations of the two modalities to investigate how the two senses interact during roughness discrimination. The findings showed that tactile stimuli significantly influenced visual judgments, even when the discrimination was easier through vision, suggesting that touch is often the dominant modality in texture perception tasks, particularly for smooth surfaces where tactile feedback provided more accurate information than vision. While the study provided valuable insights into multisensory integration and the dominance of tactile feedback, the findings are context-specific and may not fully explore complex sensory integration scenarios. [9]

Hasti project is an advanced method for real-time generation of haptic and audio feed-

back for interactions with textured surfaces in virtual environments. The core technique involved a micro-contact dynamics simulation that operates at a microscopic level. This simulation transforms discrete finger-object contact positions, obtained from a macroscopic physics simulation, into texture image coordinates that are used to sample a one-dimensional signal of surface height at an audio rate of 44.1 kHz, capturing the fine details of the surface texture. The surface height profile was combined with fractal noise, representing microscopic variations in surface geometry and the resulting profile was then smoothed using a second order Butterworth low-pass filter, which helps in reducing any steep transitions and ensures a more realistic tactile sensation. The vertical displacement of the finger pad, modeled as a mass-spring-damper system, was streamed to actuators for vibrotactile feedback, allowing users to feel the detailed texture of virtual surfaces through precise vibrations. Simultaneously, micro-collisions between the finger and the surface generate impulses that drive the modal sound synthesizer.

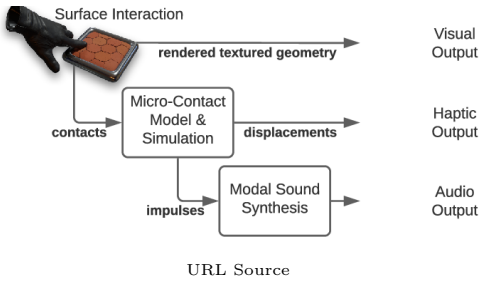


Figure 2.27: Overview of Hasti haptics and audio synthesis method

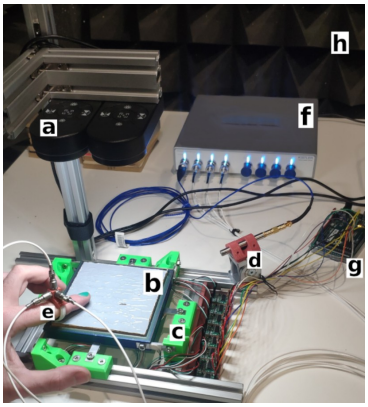
This synthesizer runs the excitation signal through a bank of Infinite Impulse Response (IIR) resonator filters. The modal parameters for these filters are estimated from the recorded impulse responses of real objects, ensuring that the audio feedback closely mimics real-world sounds. The system was evaluated through an absolute identification user study, where participants were asked to identify textures using the combined haptic and audio feedback. The study revealed high accuracy in texture identification, demonstrating the system's effectiveness in creating realistic and distinguishable textures. On the other hand, the complexity of the system requires significant computational resources, which might limit its application in less powerful devices, making further optimization necessary. This study showed how detailed geometric and material representations can be used to generate immersive haptic and audio feedback. By transforming microscopic contact events into vibrotactile and auditory stimuli, the system effectively bridges the gap between visual, auditory, and tactile perceptions, offering a robust framework for enhancing virtual interaction [71].

The integration of visual roughness perception into mid-air haptic texture design in a data-driven approach was explored into another study. Two spaces were created, a visual roughness space, from psychophysical experiments where users rate the roughness of texture images from the HaTT (Penn Haptic Texture Toolkit) database, and a haptic model

space, based on physical interaction data from mid-air haptic stimuli. These two spaces were merged into an authoring space, where new haptic textures were synthesized by matching visual roughness ratings to mid-air haptic stimuli through linear interpolation. The model used a machine learning algorithm to predict visual roughness and map it to tactile sensations felt in mid-air. A user study was conducted, where participants, wearing over-ear headphones to block auditory cues, interacted with mid-air haptic feedback and adjusted a slider to match the perceived roughness of visual textures displayed on a screen. The results of this study showed a strong correlation between the model's predictions and participants' ratings, with an accuracy of 92%, with successful integration of visual and haptic feedback, enhancing realism in virtual environments. While effective, this approach faces difficulties in aligning visual and haptic roughness perceptions across different sensory modalities [72].

An earlier work, presented a system designed to synchronize auditory and haptic feedback in virtual environments, in an effort to resolve the issue of lacking simultaneous sound and tactile sensations. The system generates both sound and reaction forces using a physical model of virtual objects, with the procedure involving collision detection to initiate feedback, vibration calculation using the Finite Element Method (FEM) to simulate how objects vibrate upon impact, and sound emission calculated from sound pressure values derived from partial differential equations. This method ensures realistic sound generation based on the object's properties and the point of impact. A speaker embedded in the grip of the haptic interface was used for spatial sound localization and the physical model was rendered to compute reaction forces according to the object's shape, elasticity, and viscosity, and provide realistic tactile feedback. A 2GHz Pentium 4 PC was used for real-time processing, ensuring immediate response to user interactions. The system's applications include the creation of virtual musical instruments where users can alter material attributes to produce different sounds as well as inspection simulators that let users evaluate the physical properties of virtual objects by tapping them. [73]

In this study, researchers focused on creating a dataset for texture classification using multimodal sensory data, including visual, auditory, and haptic signals during the interaction of a bare finger with textured surfaces. They collected signals like stereoscopic images, vibration data, audio signals from surface interactions, and force data during the sliding motion of a finger across the surfaces. Various machine learning classifiers were trained to categorize surfaces based on features extracted from each signal. For the audio signals, features like Mel Frequency Cepstral Coefficients (MFCC), Spectral roll-off, and Pitch were used, while the vibration data were processed through Euclidean norm. The study found that audio data alone achieved 71% accuracy, but faced challenges due to the variability in sound levels produced by different surfaces with some textures generating minimal sound when touched by the fingertip. Haptic data alone performed better at 76.47% accuracy, while visual classification reached 83%. However, integrating multiple sensory modalities achieved the highest accuracy, 97.3%, by combining audio, haptic, and force data. This emphasizes that while each modality contributes useful information, combining them results in the most accurate texture classification [74].



URL Source

Figure 2.28: The setup as explained in the paper: (a) 4K cameras above a textured surface. (b) load sensing system (c). (d) Directional microphone with adjustable support. (e) Accelerometers affixed to the finger touching the surface. (f) Microphone and accelerometers connected to signal conditioning unit. (g) load sensing system connected to an Arduino. Setup installed in a sound-proofed room

While machine learning has been effectively used for texture recognition and rendering in studies like those using 1D-CNNs, this thesis takes a different approach by focusing on the direct integration of audio and accelerometer data. The emphasis on multisensory fusion, as demonstrated in studies such as Hasti and the concurrent multimodal dataset, aligns with this thesis's use of synchronized audio and vibrational data to recreate realistic haptic textures. By combining audio-derived frequency features and accelerometer-derived amplitude features, this work achieves a fusion that enhances texture realism without the need for computationally intensive algorithms, aligning with findings on the importance of multimodal integration for haptic perception.

2.6 Transforming Audio Signals into Haptic Feedback

The process of transforming audio signals into haptic feedback is a significant advancement in multisensory technology, allowing people to experience sound through touch. This method can be used in broad applications, from enhancing accessibility tools for the hearing impaired to creating immersive experiences in virtual and augmented reality. This section explores the transduction of auditory data into haptic feedback, examining both the theoretical background and practical applications. The procedure of processing and mapping audio data to generate meaningful haptic feedback will be discussed, as well as the technologies and techniques used for texture feedback.

2.6.1 Simplifying Audio Data for Haptic Feedback

To effectively convert audio signals into tactile sensations, it is crucial to simplify complex audio data into components that the human sensory system can interpret through touch. Ensuring that the resulting tactile feedback provides a clear and meaningful representation of the original sound without overwhelming the user with excessive or confusing stimuli is a challenge in this field. This subsection delves into the techniques used to extract essential features from audio data and processing them, to deliver haptic feedback that represent the temporal and spectral qualities of sound.

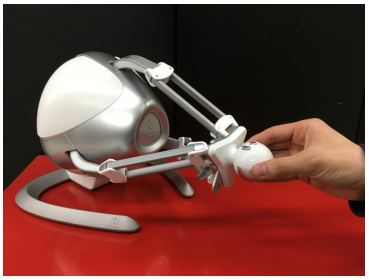
The process of simplifying audio data for haptic feedback, by translating complex auditory information into tactile sensations involves several technical steps to ensure the haptic feedback is accurate and meaningful. The initial step is identifying and extracting audio features such as volume, frequency content, and temporal envelopes, all being critical for generating relevant haptic feedback that accurately reflects the audio signal's characteristics. These features should then be filtered to remove low-level noise and less significant components, to avoid confusion in the tactile feedback. This can be done using threshold gates that filter out audio signals below a certain amplitude, ensuring that only significant audio features are translated into haptic feedback. The simplified audio features are then mapped to haptic signals. For instance, volume can be directly correlated with the intensity of vibrations, meaning louder sounds producing stronger vibrations. Similarly, frequency content can be mapped to the frequency of the vibrations, allowing users to feel different pitches. An important factor of achieving effective haptic feedback is the evaluation and calibration steps, where the algorithms are fine-tuned according to users' feedback, balancing the complexity of the audio signal with the simplicity needed for clear haptic perception.

In the parchment-skin illusion experiment, the sound of hands rubbing together was captured, manipulated by weakening the amplitude and amplifying high frequencies, and then played back to participants, altering their perception of skin dryness. Similarly, the marble hand illusion involved replacing the sound of a hammer hitting a hand with the sound of a hammer hitting marble, gradually inducing the sensation that the hand had material properties of marble. Also, in musical practice, a common approach is to simplify audio data for haptic feedback by rendering auditory and tactile feedback with sinusoidal signals, with amplitudes varying proportionally to the applied force, ensuring that the tactile feedback accurately mirrors the audio signal's characteristics. The process of simplifying audio data for haptic feedback is crucial in providing a clear and intuitive tactile representation of complex auditory information [75].

One approach in haptic technology focuses on converting audio signals into tactile stimuli to enhance music perception through touch. These methods typically involve using actuators and signal processing techniques to translate auditory features, such as rhythm, pitch, timbre, and loudness, into vibrations that can be perceived through the skin. Rhythm is often extracted from audio signals using low-pass filters to enhance bass tones, and then translated into vibrotactile patterns that correspond to rhythmic

elements. For pitch, devices like voice coil actuators (VCAs) are commonly used due to their ability to directly convert audio frequencies into tactile vibrations. Melody, which builds from pitch changes over time, is rendered through a combination of spatial and temporal variations in the tactile feedback, while timbre, the quality that differentiates instrument sounds, is represented by analyzing the spectral content of audio signals and rendering it through techniques like interpolating between sine tones and white noise. Loudness is mapped directly to the intensity of the vibrations, controlled by amplifiers to modulate the displacement of the skin. These techniques improve music perception for the hearing-impaired and provide a more immersive sensory experience for all users. However, there are still some challenges, such as the technical complexity of translating high-frequency audio content into tactile stimuli and the sophisticated signal processing required for high-fidelity tactile rendering [76].

This study, presents a system that translates audio parameters into haptic feedback using the Novint Falcon force feedback device, allowing users to experience sound through touch.



URL Source

Figure 2.29: The Novint Falcon haptic device.

Audio features such as volume, frequency content, and envelopes are mapped to physical forces on a user's hand as they move the device horizontally across a virtual surface. The process of simplifying audio data for haptic representation involved focusing on a single audio parameter at a time, such as volume or frequency and using a threshold gate to filter out low-level noise, making it more interpretable through touch. There were two approaches used for generating haptic feedback: discrete and continuous force mapping. Discrete force mapping involves creating simulated detents at points where the audio parameter exceeds a certain threshold, giving the feeling of small dips in a surface and allowing discrete points of interest in the audio to be identified through physical sensations. Continuous force mapping adjusts the friction felt by the user's hand based on the audio intensity, allowing tactile experiences that reflect audio variations, meaning for example that silent areas offer no resistance. This approach, however, requires careful calibration to ensure that the variations in resistance are noticeable and accurate. Challenges include accurately translating detailed audio data into clear physical sensations and the dependence of continuous force mapping on hand speed, which affects the perception of faint changes. Although users can feel bumps and drags corresponding to audio

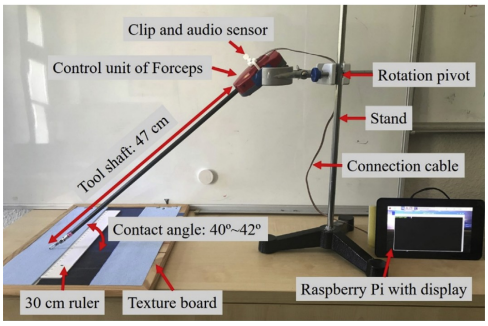
peaks and sustained sounds, conveying subtle changes remains difficult. Additionally, vertical forces proportional to audio amplitude were found to be disruptive and were not fully implemented [77].

This thesis builds on methods such as those using voice coil actuators or electrotactile feedback to convert audio signals into meaningful tactile sensations. By focusing on mapping of audio features like frequency and amplitude to vibrotactile feedback, the approach captures the spectral richness of texture interaction sounds, similar to studies that translate audio data into tactile cues for texture recognition and accessibility applications.

2.6.2 Texture and Tactile Feedback

Texture through haptic feedback has been extensively studied in various fields, particularly in robotics and virtual reality. When texture information is transformed from one sensory modality, into another, the system must maintain enough detail to accurately convey the differences between materials or surface textures. This section presents research into how audio signals can be used to identify textures by translating sound waves into tactile patterns. Several techniques are examined, revealing how audio profiles corresponding to different textures can be converted into tactile sensations. These methods are particularly relevant in fields like minimally invasive surgery, where haptic feedback could enhance a surgeon's ability to distinguish between tissues based on audio generated vibrations.

One study explored a method to enhance tactile feedback in robotic minimally invasive surgery (RMIS) using audio signal analysis, claiming that different textures produce distinct audio signal patterns that are detectable through spectral analysis. They used a da Vinci Prograsp Forceps mounted on a stable platform to ensure consistent audio signal quality, with the forceps performing linear movements over various textures, including synthetic materials like soft cloth and denim and biological tissues of pig liver and pork fillet.



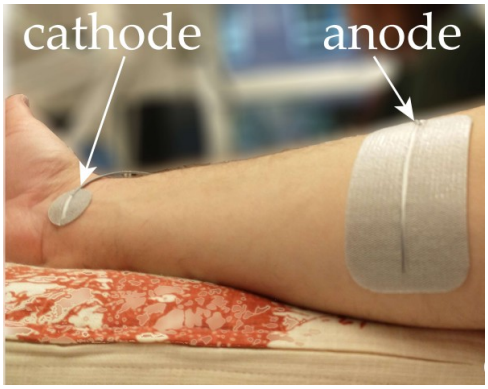
URL Source

Figure 2.30: The experimental setup of the recording.

An Adafruit I2S MEMS microphone, was attached to the forceps to capture the audio signals produced during these interactions. These signals were logged by a Raspberry Pi in WAV format at a 16 kHz sampling frequency. Researchers employed Fast Fourier Transform (FFT) and Auto-Regressive (AR) modeling techniques to estimate the Power Spectral Density (PSD) of the audio signals, that provides a measure of the signal's power across various frequency components. The process involved identifying sweep segments with a CUSUM algorithm, extracting random segments, and calculating the PSD for each. These PSDs were averaged to create spectral models for each texture, which were then compared to test signals using Pearson correlation coefficients, where high correlation values indicate a match between the test signal and the texture model, allowing for differentiation. The results showed that different textures actually produce unique time and frequency domain characteristics, with softer textures generating lower frequency components and rougher textures showing higher spectral energy. The correlation analysis confirmed that both synthetic and biological textures can be distinguished based on their audio spectral characteristics, particularly in the higher frequency ranges of 500-4000 Hz and 4000-8000 Hz. This method is non-invasive, low cost, and shows sensitivity to subtle texture changes. However, the method's sensitivity to noise and the controlled environment of the experiments highlight challenges in applying this technique in real-world surgical settings. [78]

A novel system that enables users to distinguish between various surface textures through electrotactile feedback derived by audio signals was developed in a more recent study. The system used an omnidirectional microphone (Adafruit MAX9814) with a frequency range of 20 to 20,000 Hz and a gain of 60 dBA to capture the friction sounds generated when the microphone interacted with four different textures. The chosen textures were silicone rubber, felt, sponge and string mesh. These signals were processed by a PJRC Teensy 4.0 microcontroller, at a sampling rate of 6 kHz with 16-bit resolution. The signal processing they did involved Fast Fourier Transform (FFT) to compute the frequency content of the audio signal from 2048 samples, with a 50Hz update rate. The system extracted the median frequency from the audio spectrum as the key feature for texture discrimination and then mapped it to the stimulation frequency using a linear transfer function. The total signal energy was used to enable or disable the stimulation based on a threshold, ensuring that only significant tactile events generated feedback. Electrotactile feedback was provided through a custom-made electrical stimulator capable of producing biphasic, charge-balanced, cathodic-first, current-controlled pulses with amplitudes ranging from 0.1 mA to 10 mA and frequencies from 1 to 100 Hz, applied by electrodes placed on the forearm. The cathode was placed over the median nerve to ensure the stimulation's correspondance to sensations in the fingers and palm.

User experiments were conducted in where the participants experienced three phases; training, with-feedback, and without-feedback. During the training phase, participants received both visual and electrotactile feedback while stroking each texture 20 times. In the with-feedback phase, participants, blindfolded and wearing headphones for sound insulation, identified textures using only electrotactile feedback and received verbal feed-



URL Source

Figure 2.31: The placing of the electrodes for the electrotactile feedback.

back on their responses. In the without-feedback phase, participants identified textures relying solely on electrotactile sensations without verbal feedback. Results indicated a median accuracy of 85% in texture discrimination, with felt being the easiest to distinguish due to its higher median frequency and silicone rubber being the most challenging due to its inconsistent friction characteristics. The positive aspects of this work include real-time processing capabilities and high discrimination accuracy, while challenges involve minimizing residual vibrations and improving consistency in manual stroking. This method could be practically applied to users of prosthetic devices and other tactile applications[79].

A prototyping tool for design and education of haptic media was developed as a user-friendly and affordable haptic device aimed at making haptic technology accessible to non-professionals, including designers, educators, and students. The TECHTILE toolkit comprises a haptic recorder that functions like a microphone to capture tactile sensations, small voice-coil vibrators that reproduce these sensations, and a signal amplifier optimized for both audible frequencies (20-20000Hz) and low-frequency vibrotactile sensations (1-20Hz). Users attach the haptic recorder to an object where tactile events occur, such as the bottom of a paper cup and the recorded signal is then amplified and transmitted in real-time to the haptic reactors, allowing users to feel the same tactile feedback. The system also supports recording haptic signals as audio tracks in video files through a USB port, enabling synchronized playback of tactile, visual, and auditory information, which can be shared online.

The toolkit's design, inspired by conventional audio recording setups, ensures ease of use while delivering realistic haptic sensations. Workshops with participants ranging from children to university students demonstrated the toolkit's effectiveness in educational settings, with people quickly learning to create original haptic content using everyday objects. This innovative approach to capturing and sharing tactile sensations, offers a practical framework for converting and communicating tactile experiences digitally. The toolkit's ability to translate complex tactile information into a shareable format

URL Source

Figure 2.32: TECHTILE toolkit.

underscores its potential to enhance multisensory communication and education. [80]

Another approach was a wearable haptic interface that allows users to touch and feel the textures of virtual 3D objects using a haptic copy and paste technique, using the TECHTILE toolkit. The tactile textures of real objects were captured as audio signals by the haptic recorder of the toolkit and then reproduced by the haptic reactors, providing realistic vibrotactile feedback. The system setup included an optical motion capture system, a 3D projector, and a wearable haptic glove equipped with motors and voice-coil vibrators on each finger. The force feedback provided by the glove is calculated based on the user's movements, which are tracked and processed using the PhysX engine by Nvidia. Users, wearing 3D glasses and the haptic glove, can interact with stereoscopically displayed 3D virtual objects, experiencing realistic haptic sensations through a combination of force and vibrotactile feedback. The system was tested with users who could actively touch and feel various textures on virtual objects, achieving a high level of realism and immersion. This system has the ability to provide accurate and realistic tactile feedback, ease of integration into existing multimedia systems, and suitability for real-time interactions. However, challenges remain in optimizing motion capture accuracy and minimizing latency. [81]

A sophisticated method for transforming audio signals into vibrotactile feedback to enhance multimedia experiences was proposed in this work, consisting of an automatic transformation algorithm and a custom made vibrotactile actuator. First, the audio signal was processed through a digital bandpass filter with cutoff frequencies between 50Hz and 12kHz to isolate relevant frequencies and then, Short-Time Fourier Transform (STFT) analysis was used on the filtered signal to break it down into its time-frequency components, using the hamming function as its window. Principal frequencies were extracted by identifying peaks in the power spectrum that were above a certain threshold, ensuring that only significant audio features were converted into tactile feedback, and then they logarithmically scaled them to fit the human body's range of perception, from approximately 1 to 400 Hz. The scaled frequencies generated sinusoidal waves, which

were overlaid to create the final vibrotactile signal. The signal was delivered through a custom vibrotactile actuator based on the structure of voice coil linear motor, as it has the ability to produce precise low-frequency vibrations, embedded in a haptic cushion. For compactness and convenience, an Optimus 2X mobile phone was used as the controller, with the algorithm integrated into the Android framework's "audio flinger" to generate vibrotactile signals. The actuator was controlled by these signals via a power amplifier. The measurement system, including a 1-axis accelerometer and a conditioning amplifier, monitored and amplified the actuator's movement. Data collected through a NI USB data acquisition board was used to evaluate the system's performance. The system's real-time processing capabilities allow immediate response to audio events, verified through tests with audiovisual contents like movies and games. Despite challenges such as unwanted residual vibrations and the need for comprehensive user studies to optimize perception, the method effectively bridges audio and tactile textures, improving interactive multimedia experiences through synchronized auditory and tactile sensations. [82]

URL Source

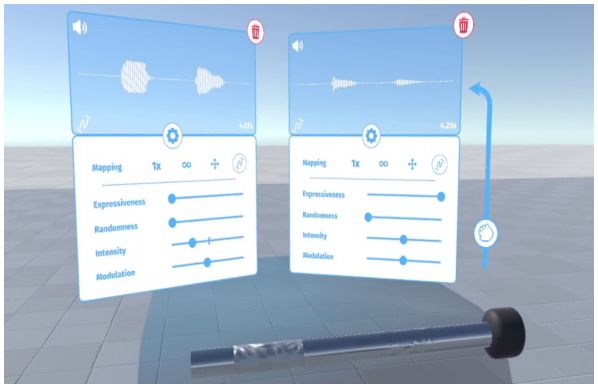
Figure 2.33: The Haptic Cushion.

Lastly, an innovative system for creating vibrotactile feedback in virtual reality by transforming vocalizations into tactile experiences was presented. The initial step is calibrating users' vocal ranges to set bounds for amplitude and frequency, ensuring the inputs are normalized for further processing. Users interact with virtual objects in the virtual environment while vocalizing the intended tactile feedback, which is recorded in real-time through a microphone. The recorded vocalizations are then processed to extract key audio features, specifically frequency and amplitude. Amplitude extraction involves

calculating the signal's envelope and sampling it at 50 ms intervals using the Shockley diode algorithm, while frequency extraction uses the Yin pitch detection algorithm at 100 ms intervals. To ensure temporal alignment, a non-linear time-scaling algorithm from the Rubber Band Library aligns the vocalizations with spatial interactions in real time. The extracted audio features are mapped to vibrotactile feedback using four types of spatio-temporal mappings; instantaneous (for brief events like pressing a button, continuous (for ongoing feedback such as the hum of an electric toothbrush), motion (refering to feedback varying with the speed of the user's movement), and positional (specific spatial feedback for interactions like opening a creaking drawer). The system uses empirically determined thresholds for movement magnitude and recording duration to select the appropriate mapping automatically. Users can dynamically fine-tune the feedback in real-time using modifiers for intensity (adjusting amplitude), modulation (altering frequency), expressiveness (enhancing peaks with a sigmoid function), and randomness (introducing controlled noise to mimic natural variations).

URL Source

(a) The recording of the signal through vocalization.



URL Source

(b) Vocalization layers with settings.

Figure 2.34: The Weirding Haptics design tool.

Each vocalization layer can be independently modified and multiple layers can be combined, enabling detailed and complex haptic feedback. The system supports both VR controllers and custom high-fidelity actuators and also, they created an alternative setup with an actuator controlled by an ESP32-DevKitC V4 microcontroller ensuring high-resolution tactile feedback by normalizing amplitude responses within the frequency range of 150 to 300 Hz. This setup also uses a digital potentiometer and a Class D amplifier to maintain precise control over the vibrations. User validation studies showed that users could effectively design and refine vibrotactile feedback after a short training period. The study found that while most tactile experiences were achieved with a single vocalization layer, the ability to combine multiple layers allowed for enhanced feedback detail. This system provides a powerful and flexible approach with rapid, in-situ prototyping of haptic

feedback, highly personalized through intuitive vocalizations and user friendly. Some of the system's limitations though, are related to vocalization limits, as the users may struggle with vocalizing certain tactile sensations, particularly if their vocal range is limited and layer management complexity, meaning that while layering provides detailed feedback, it can be complex for users to manage multiple layers effectively [83].

In this thesis, the process of texture representation builds upon several approaches discussed in this section, leveraging multimodal data to create realistic vibrotactile feedback. Similar to the RMIS method, where audio signals are analyzed for texture differentiation, this work uses high-resolution audio and accelerometer signals to capture and reproduce tactile nuances. The electrotactile feedback systems, which map audio spectral features to stimulation intensity, inform the mapping strategies employed in this thesis for creating haptic feedback from spectral and temporal characteristics of recorded signals. Additionally, techniques like those used in the TECHTILE toolkit and wearable haptic systems for capturing and reproducing tactile signals in real time, resonate with this thesis's focus on transforming interaction data into perceptually meaningful feedback.

Chapter 3

Technological Background and Definitions

This chapter provides an overview of the key technologies and components that were utilized in this thesis. The system relies on a combination of hardware components, software frameworks, and communication protocols to enable synchronized data capture, processing, and haptic feedback. First the hardware components will be introduced, followed by the software frameworks and development environments that enable the data capture, processing, haptic mapping and delivery. By exploring each of these technologies, this chapter builds the context for understanding the implementation details covered in Chapter 5.

3.1 Hardware Components

This section is dedicated to all the hardware components that were employed, providing information about their technical characteristics that make them suitable choices for this thesis.

ESP32-S3 Microcontroller

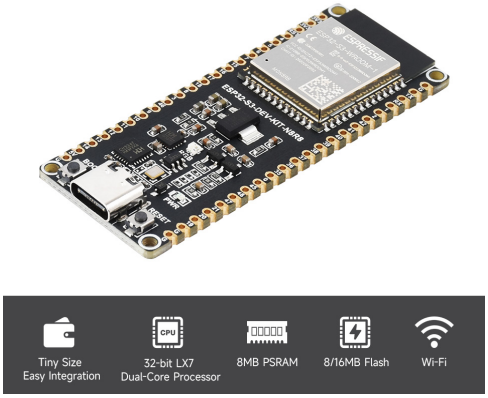
A microcontroller (MCU) is a compact integrated circuit designed to control a specific operation in an embedded system. It contains essential components like one or more CPUs, memory, and I/O peripherals, on a single chip, making it capable of handling specific tasks in real-time with high efficiency. Microcontrollers are widely used in applications like consumer electronics, automotive systems, medical devices, and IoT systems, where they process inputs from sensors and control actuators or other hardware components. Unlike general-purpose microprocessors, MCUs are self-contained, combining the processor with memory and I/O capabilities, which reduces costs and simplifies the design for embedded systems. In terms of architecture, microcontrollers vary in their bit

configurations, with 8-bit, 16-bit, and 32-bit versions being common. Each offers different performance capabilities, from basic control tasks in simpler systems to more complex real-time applications requiring faster processing speeds. They also typically operate at lower power, making them ideal for battery-powered applications [84], [85]. The ESP family of microcontrollers, developed by Espressif Systems, is known for its integration of Wi-Fi and Bluetooth capabilities, allowing for Internet of Things (IoT) applications. The ESP family has evolved from the initial ESP8266, which provided a cost-effective, Wi-Fi-enabled option, to the more advanced ESP32 series. The ESP32 series offers enhanced processing power, dual-core processing, both Wi-Fi (802.11b/g/n) and Bluetooth 4.2 and 5.0 support, and a range of peripherals, all while maintaining a focus on power efficiency [86].

The ESP32-S3-DEV-KIT-N8R8, used in this thesis, is a development board equipped with the ESP32-S3-WROOM-1 module, which includes a dual-core Xtensa 32-bit LX7 processor capable of running at up to 240 MHz. This configuration provides significant processing power for real-time applications and complex multitasking. To optimize power efficiency, it features low-power operation modes and a ULP (Ultra Low Power) co-processor, which can handle simple tasks and control peripherals while the main cores are in deep sleep or low-power modes. This feature allows it to handle sensor readings, GPIO monitoring, or timer operations without fully activating the primary processors, extending battery life in energy-sensitive applications. The board includes 512 KB SRAM, 384KB ROM, 8 MB of flash memory and 8 MB of PSRAM, making it effective for high-speed data acquisition and storage demands like audio and accelerometer data capture in this project. The board's single USB Type-C port enables both power supply and programming functionality, simplifying connectivity, with onboard CH343 and CH334 chips that support USB and UART connections. The ESP32-S3 offers Wi-Fi 802.11 b/g/n and Bluetooth 5.0 LE, which are crucial for wireless communication in IoT systems. The board supports a range of peripheral interfaces, including I2S and I2C, allowing direct connection to audio and sensor modules, as well as an integrated LED for status indication. Its high-speed GPIOs can handle up to 40 MHz and the board also integrates hardware-based security features like secure boot, flash encryption, and AES encryption to ensure data integrity and device security. Finally, the supported development environments are ESP-IDF, Arduino, and MicroPython, among others, making it suitable for rapid prototyping and product applications across various software platforms. [87].

ADXL345 Triple-Axis Accelerometer

An accelerometer is a device that measures the rate of change in velocity (acceleration) which can be both static, such as the force of gravity, and dynamic, such as vibrations or movements. They are based on the principle of detecting changes in capacitance, piezoelectric effect, or micro-electromechanical systems (MEMS), which convert physical movement into electrical signals. Their detection range varies from high-frequency shocks to low-frequency motions, allowing them to detect fine variations in vibrational energy,



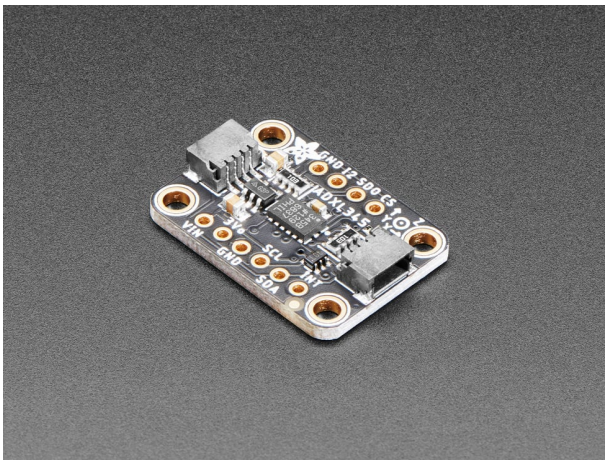
URL Source

Figure 3.1: The ESP32-S3 Microcontroller.

often represented as a voltage proportional to acceleration. This capability makes them valuable in applications such as motion sensing, vibration analysis, and texture recognition, as they capture high-frequency, low-amplitude variations in force and motion. Modern accelerometers are typically compact, highly sensitive, and equipped with digital output capabilities, making them well-suited for integration into portable devices where accurate, low-power vibration measurement is essential. Most commonly, accelerometers measurements are on a single axis, often used in measurement of mechanical vibration levels. However there are also the triaxial accelerometers that create a 3D vector of acceleration among usually x, y and z axis, in the form o orhtogonal components. This type is more valuable in determining the type of vibrations, meaning lateral, transverse or rotational. only a single axis.[88], [89].

In this project, the ADXL345 Triple-Axis Accelerometer from Adafruit was chosen, which is a compact digital accelerometer designed to measure both static and dynamic accelerations. It also is well suited for capturing and analyzing vibrations due to its high sensitivity and overall design. With a range of ±2g to ±16g, it outputs high-resolution data (up to 13-bit) in 16-bit two's complement format, it achieves fine-grained sensitivity (3.9 mg/LSB at ±2g), enabling detection of subtle vibrations. It supports both I2C and SPI communication protocols, offering flexibility in integration. In I2C mode, the ADXL345 operates with a simple two-wire setup (SDA and SCL lines), where it can share the bus with other devices. The SPI mode is another choice, for applications that require faster data transfer, operating in both 3-wire and 4-wire configurations, achieving data rates up to 5 MHz. By providing both protocols, the ADXL345 supports integration into various digital systems, balancing speed, reliability, and flexibility. The ADXL345 integrates two configurable interrupt pins (INT1 and INT2), which can be mapped to various functions, including data readiness, activity/inactivity detection, single and double tap recognition, and free-fall detection. These interrupts enable immediate responses

from the host processor to specific events, reducing the need for constant data polling and conserving system resources. To capture a broad range of vibration frequencies, the ADXL345 provides adjustable bandwidth up to 1600 Hz, with output data rates reaching 3200 Hz in SPI mode. Additionally, its built in 32-level FIFO buffer stores vibration data, minimizing processor load and allowing the system to handle large volumes of data efficiently. The device's ultra-low power consumption, scaling with bandwidth to as low as 23 µA in measurement mode and 0.1 µA in standby, makes it ideal for mobile applications. With features such as activity/inactivity monitoring, single and double tap detection, and free-fall detection—all configurable via user-defined thresholds—the ADXL345 is versatile for a range of applications [90].



URL Source

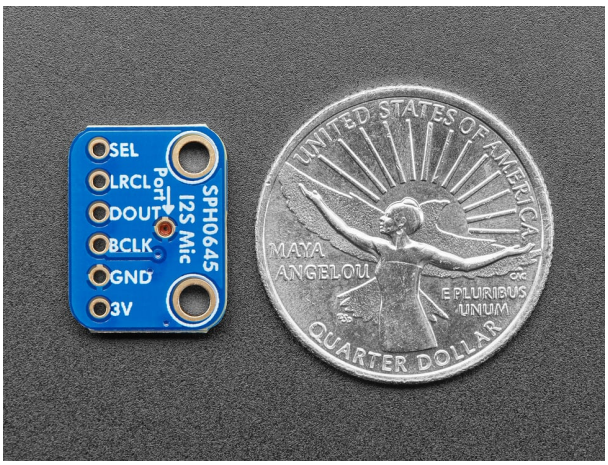
Figure 3.2: The ADXL345 Triple-Axis Accelerometer.

SPH0645LM4H I2S MEMS Microphone

MEMS (Micro-Electro-Mechanical Systems) microphones, or silicon microphones, are compact devices that use a small pressure-sensitive diaphragm (which vibrates in response to sound pressure) and backplate to convert sound waves into electrical signals. These components are mounted on a silicon plate using semiconductor processes, making MEMS microphones compact, energy-efficient, and highly suitable for integration in small, portable electronic devices. Compared to traditional microphones, MEMS microphones are known for their stability, resilience to environmental changes, due to their silicon-based construction as well as ease of digital integration, making them ideal for applications in mobile devices, wearables, and IoT applications. [91]

The SPH0645LM4H-B microphone by Knowles is a digital I2S-output microphone, designed for applications requiring high-quality audio capture with low power consumption and minimal space. This omnidirectional microphone utilizes a MEMS transducer

and outputs a digital signal directly through I2S, allowing integration with microcontrollers and digital processors without needing an external audio codec. It integrates a Sigma-Delta converter that transforms the analog input from the MEMS transducer into a Pulse Density Modulated (PDM) signal, which is then converted into a 24-bit Pulse Code Modulated (PCM) I2S output. This process, which includes decimation and low-pass filtering, enhances audio quality by minimizing high-frequency noise. It supports both active and sleep modes to optimize power usage, operating at clock frequencies above 900 kHz in active mode. In sleep mode, when the frequency drops below 900 kHz or the clock input is removed the power consumption is reduced to 10 µA, making it a good choice for energy sensitive applications. Additionally, the microphone's I2S interface includes tri-state control, allowing two microphones to operate on a single I2S bus. This feature enables stereo recording or spatial audio setups in compact systems, broadening its applications. With a high signal-to-noise ratio (SNR) of 65 dB, low typical operating current of 600 µA, and design elements such as RF shielding and a compact footprint, the SPH0645LM4H-B keeps a good balance in delivering high-quality audio capture within compact electronic designs [92].

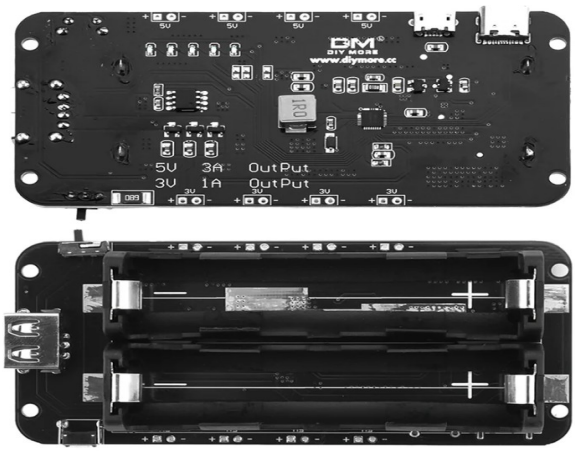


URL Source

Figure 3.3: The SPH0645LM4H - I2S MEMS Microphone Breakout.

18650 Battery Shield Mobile Power Bank

The 18650 Battery Shield V8 is a mobile power bank module designed for use with 18650 lithium batteries, providing reliable power options for Arduino, ESP32, and ESP8266 projects. This shield offers both 3V and 5V outputs, making it compatible with a variety of microcontrollers and IoT devices. Equipped with overcharge, over-discharge, and short-circuit protection, it ensures battery longevity and safe operation. The module includes a USB output and micro-USB input for convenient charging and power delivery, along with an onboard indicator LED to display charging status [93].



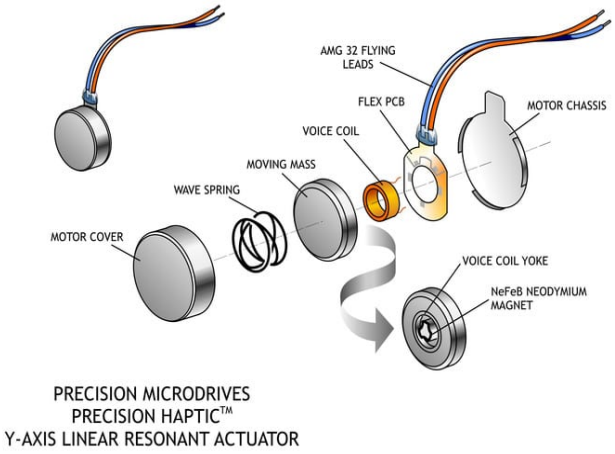
URL Source

Figure 3.4: The 18650 Battery Shield Mobile Power Bank.

Linear Resonant Actuator (LRA)

A Linear Resonant Actuator (LRA) is a a compact and efficient vibration motor that generates oscillating force along a single axis. Unlike traditional vibration motors, such as Eccentric Rotating Mass (ERM) motors, which produce vibrations through the rotation of an unbalanced weight, LRAs operate by using a moving mass suspended by springs within a casing, driven by an electromagnetic coil. AWhen an alternating current (AC) signal is applied, the electromagnetic coil generates a magnetic field that interacts with a permanent magnet attached to the moving mass, causing it to oscillate linearly along a fixed axis, resulting in vibrations. The LRA operates at its natural resonant frequency—typically between 150 Hz and 200 Hz—where the system achieves maximum efficiency, requiring minimal energy to generate strong vibrations. This resonance-based operation offers several advantages. LRAs are highly efficient, consuming less power than ERMs, making them ideal for portable and battery-powered applications. Their design enables faster response times, with short rise and fall times—critical for sharp and precise haptic feedback. Additionally, the linear motion of the oscillating mass results in more uniform and consistent vibrations, providing a superior tactile experience. The robust and compact structure of LRAs, combined with minimal mechanical wear due to their brushless design, ensures longevity and reliability in demanding applications [94].

Despite their advantages, LRAs have some drawbacks that need to be considered for practical implementations. One major limitation is their reliance on an alternating current (AC) signal for operation, which necessitates the use of an external driver circuit. Without a driver, LRAs cannot maintain their resonant frequency or modulate vibration characteristics effectively, making them dependent on precise control mechanisms. Additionally, their performance is highly sensitive to operating conditions, such as mounting position and input signal quality, requiring careful calibration during setup.



URL Source

Figure 3.5: Y-Axis Linear Resonant Actuator.

These dependencies, while manageable, increase the system's complexity compared to simpler actuation methods.

In this thesis, the LRA was chosen for its ability to control the vibration frequency and amplitude independently resulting in haptic vibrations that represent subtle texture variations from the extracted texture features from recorded interactions. Also, its low power consumption, compact size and rapid response times were factors that contributed in the selection of this type of actuation, since the feedback system needs to be lightweight as it is mounted on the user's hand and support real-time interactions. To address the challenges associated with driving the LRA, the DA7280 driver was used, offering precise control to ensure optimal performance.

DA7280 Driver

To effectively drive the Linear Resonant Actuator (LRA) in this project, SparkFun Qwiic Haptic Driver DA7280 was selected due to its advanced features and suitability for precise haptic feedback applications. The DA7280 is a driver designed to support both Linear Resonant Actuators (LRAs) and Eccentric Rotating Mass (ERM) motors, providing robust control mechanisms for high-quality vibration generation. The DA7280 features a closed-loop resonance tracking, which continuously monitors the back electromotive force (BEMF) generated by the actuator to dynamically adjust the drive frequency. This ensures that the LRA consistently operates at its optimal resonant frequency, even as external factors like temperature, mechanical coupling, or aging affect the system. This resonance tracking maximizes vibration strength while minimizing power consumption, addressing one of the key challenges of using LRAs. The DA7280 offers multiple

URL Source

Figure 3.6: The DA7280 Driver with an LRA actuator attached.

control modes, such as I^2C communication, pulse-width modulation (PWM), and general-purpose input triggers making it easy and suitable for integration with microcontrollers. Additionally, it has an integrated waveform memory that allows predefined haptic patterns to be stored and executed with low latency (as short as 0.75 ms). The driver also includes advanced output control, such as active acceleration and braking, which enables sharper vibration onset and decay, critical for delivering responsive and dynamic haptic effects. In terms of efficiency, the DA7280 is designed with a differential output drive architecture, which minimizes energy losses and ensures balanced power delivery to the LRA. It operates on a low supply voltage (2.5 V to 5.5 V) and consumes as little as 0.36 µA in standby mode, making it ideal for battery-powered, portable devices. Its compact design and low external component requirements simplify integration into space-constrained systems. Furthermore, the driver incorporates safety features such as over-temperature protection, short-circuit protection, and fault diagnostics, ensuring reliable operation under various conditions [95].

The DA7280 was chosen for this project because it addresses the specific challenges associated with driving LRAs, such as the need for precise AC signal generation and efficient vibration control. Its ability to independently control vibration frequency and amplitude aligns seamlessly with the project's requirements for precise and dynamic haptic feedback.

Leap Motion

The Leap Motion Controller is an optical hand-tracking device designed to capture the movements of a user's hands and fingers with high precision, enabling natural and

intuitive interaction with digital environments. The earlier version of the Leap Motion device, used in this thesis, uses two infrared (IR) cameras and three IR LEDs to create a 3D interaction space extending up to approximately 60 cm above the device. By capturing data at a rate of over 200 frames per second, it accurately tracks the positions and movements of hands and fingers, reconstructing 27 distinct hand elements, including bones and joints, in real time. With a field of view of 150 degrees wide and 120 degrees deep, it is capable of capturing subtle motions and complex gestures, such as pinching, swiping, and tapping. [96] In this thesis, the Leap Motion Controller is utilized

URL Source

Figure 3.7: Leap Motion Controller.

to monitor the position and velocity of fingertip movements during texture interactions. By providing real-time data on hand and finger positions, it enables the dynamic modulation of haptic feedback, ensuring that the tactile responses correspond appropriately to the user's interaction speed and direction. The device's compact size, low latency, and integration with various development platforms such as Unity make it particularly suitable for developing immersive and responsive haptic systems. Although it requires the user's hands to be within a specific range and may have limitations under certain lighting conditions or occlusions, its capabilities are sufficient for the purposes of this project.

3.2 Software and Frameworks

This section presents all the software tools, environments and frameworks that were employed in this project, giving definitions and insights to key elements that each of them provides.

Visual Studio Code (VS Code)

Visual Studio Code (VS Code)is an open-source code editor developed by Microsoft, that is widely used by developers across various programming languages and environments. Known for its lightweight and user friendly design, VS Code provides features like syntax highlighting, intelligent code completion, and debugging tools, enhancing productivity for developers. Its functionality can be expanded through an ecosystem of extensions, allowing users customization with specialized tools, including PlatformIO and Arduino frameworks, making it particularly useful for embedded and IoT applications. Also, VS Code includes a built-in terminal, integrated Git version control, and remote development environments. These features and adaptability make the VS Code an all in one solution and contribute to its popularity, especially in embedded development [97].

PlatformIO and Arduino Framework

PlatformIO is an open-source development environment for embedded systems, supporting over 800 embedded boards and frameworks, such as Arduino, ESP-IDF, and mbedOS. As an extension for Visual Studio Code, it offers developers a cross-platform solution, integrating advanced tools for coding, building, debugging. It provides features like library management, project-specific dependency handling, unified debugging and a decentralized build system. This allows developers to manage project dependencies per project, preventing conflicts and ensuring consistency across different projects and environments. In addition PlatformIO is compatible with Windows, macOS, and Linux allowing developers to share and manage projects making it suitable for individuals, as well as teams working on scalable and commercial solutions [98], [99].

The Arduino framework, is a widely used open-source electronics platform for embedded systems, due to its simplicity and accessibility. It is designed around user-friendly microcontroller boards, such as the popular Arduino Uno, and an Integrated Development Environment (IDE) with a C/C++-based language, promoting rapid prototyping and experimentation, even for beginners. Its rich library ecosystem, offering pre-written modules for common functionalities like sensor and motor interfacing, supports applications from DIY projects to IoT and education, and its cross-platform compatibility on Windows, macOS, and Linux ensures broad usability. The open-source nature of Arduino creates a community-driven development, further enriching its repository of resources, examples, and support. When combined with PlatformIO, the Arduino framework gains additional scalability and structure, enhancing its use in larger embedded and IoT projects. Together, Arduino and PlatformIO create a versatile, unified development platform for testing and deploying firmware, covering the full development process in embedded applications [100], [101].

FreeRTOS

FreeRTOS is a lightweight real-time operating system (RTOS) designed for embedded systems and microcontrollers, enabling efficient task scheduling and resource management. It supports preemptive, cooperative, and round-robin scheduling which allows the CPU to prioritize tasks based on their priority, maximizing resource use. FreeRTOS provides inter-task communication and synchronization tools like semaphores, queues, and mutexes, ensuring tasks can safely share resources and exchange data. With a minimal memory footprint requiring as little as 9 KB of Flash memory in some configurations, FreeRTOS is ideal for low-power and memory-limited devices. It is compatible with over 40 hardware architectures, including ARM Cortex, ESP32, and RISC-V, and includes extensive libraries for connectivity, security, and over-the-air (OTA) updates. Additionally, FreeRTOS provides long-term support (LTS) versions that provide security patches and bug fixes, ensuring consistent performance over time. Distributed under the permissive MIT license, FreeRTOS is open for modification, making it suitable for both commercial and personal use [102].

The ESP32 uses a modified version of FreeRTOS, integrated within the Espressif IoT Development Framework (ESP-IDF), to support its dual-core architecture and specific hardware features. This version, called ESP-IDF FreeRTOS, is based on Vanilla FreeRTOS but is adapted for symmetric multiprocessing (SMP), optimized to run tasks across two cores independently. Each core can handle its own task scheduling, allowing developers to set task affinity by either pinning tasks to a specific core or leaving them unpinned to run on either core, optimizing CPU usage. Key features of ESP-IDF FreeRTOS include fixed-priority preemptive and time-slicing, which allocate CPU time among tasks based on priority, with modifications to handle dual-core behavior. The framework also supports tickless idle mode, reducing power consumption by entering low-power states during idle times. Additionally, ESP-IDF FreeRTOS offers enhanced inter-core communication with tools like spinlocks and specialized memory management, ensuring efficient multitasking without conflicts between cores. Other adaptations include customizable configuration through ESP-IDF's menuconfig system rather than directly modifying FreeRTOSConfig.h. This allows for streamlined system setup without manual adjustments, supporting connectivity, security, and over-the-air (OTA) updates. The integrated FreeRTOS also initiates app_main as the application's main entry point rather than the traditional vTaskStartScheduler approach used in standalone FreeRTOS, simplifying startup processes for ESP32 applications [103].

Unity

Unity is a powerful and flexible real-time development platform widely used for creating interactive 2D and 3D applications, including mobile devices, desktops, consoles, and virtual reality systems. It provides an integrated development environment (IDE) and supports scripting primarily through C#, allowing developers to design, test, and deploy

applications across various platforms. Unity's physics engine and real-time rendering capabilities make it ideal for projects requiring dynamic simulations and responsive interactions, while its modular architecture enables customization to meet specific project needs. Developers can also take advantage of Unity's Universal Render Pipeline (URP) for optimized graphics performance or the High Definition Render Pipeline (HDRP) for high-fidelity visuals, depending on the application requirements. Additionally, the Unity Asset Store provides a vast repository of user-generated assets, including 3D models, textures, and scripts, which can significantly reduce development time.

Unity's physics engine provides tools for simulating realistic object interactions. Key components such as rigidbodies and colliders define the physical properties and interaction zones of objects within a scene. A rigidbody enables objects to respond to forces such as gravity, collisions, and user-applied inputs, making it essential for simulating dynamic interactions. Colliders define the physical boundaries of objects, ensuring accurate collision detection and interaction between objects in the virtual environment [104].

In this thesis, Unity is used as the main platform for integrating hardware, managing interactions, and delivering haptic feedback. Through the integration of the Leap Motion Controller, Unity facilitates real-time tracking of hand and finger movements, enabling precise mapping of physical actions to virtual interactions. This is achieved using the Ultraleap Unity Plugin, which provides the necessary tools to incorporate hand-tracking data into Unity's environment. The engine also enables wireless communication with the ESP32 system, allowing the synchronization of texture-based haptic feedback with user movements.

Ultraleap SDK

The Ultraleap Unity Plugin is a software development kit (SDK) that enables integration of Ultraleap's hand-tracking technology into Unity applications. The plugin captures hand-tracking data from Ultraleap hardware, which includes the Leap Motion Controller, and processes this data to create 3D representations of hand movements. This functionality allows developers to implement intuitive gesture-based interactions, replacing traditional controllers and creating a more immersive user experience. The Ultraleap Unity Plugin provides a variety of features and tools for development. These include pre-built prefabs for visualizing hands in 3D, scripts for implementing interactions, and utilities for customizing hand physics and gestures. The plugin also includes advanced features like pose detection, which identifies specific hand shapes, and interaction engines for precise manipulation of virtual objects. These features are fully integrated into Unity's physics and rendering systems, ensuring smooth and responsive interactions that feel natural to users. Integrating the Ultraleap Unity Plugin into a Unity project involves a straightforward process. First, the Ultraleap Hand Tracking Software (version 5.2 or newer) must be installed to enable communication between the hardware and Unity. The plugin can the be imported into a Unity project using the Unity Package Manager or a provided .unitypackage file. Once set up, the plugin allows developers to use its

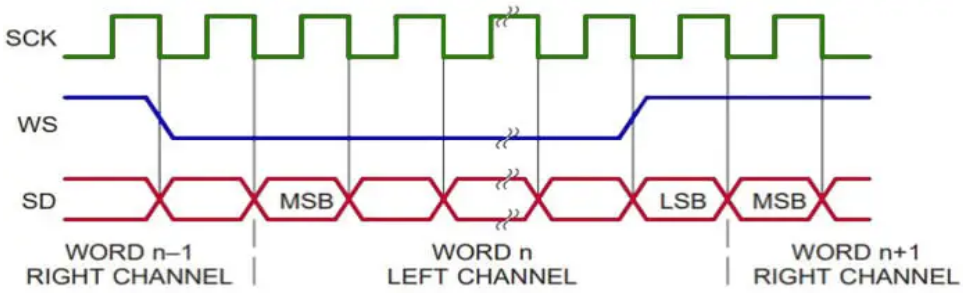
capabilities [105].

In this thesis, the Ultraleap Unity Plugin is used to integrate Leap Motion into the project and connecting real-world hand movements to virtual simulations. It enables real-time tracking and mapping of fingertip velocities and gestures, which are critical for synchronizing haptic feedback with user interactions. By utilizing the plugin's advanced features, this project achieves seamless integration of the Leap Motion Controller into Unity

3.3 Communication Protocols

I2S Protocol

The Inter-IC Sound (I2S) protocol is a specialized serial interface to handle high-quality digital audio transmission between devices like audio codecs, digital-to-analog converters (DACs), analog-to-digital converters (ADCs), and digital signal processors (DSPs). Designed for high-fidelity audio applications, I2S operates with three essential lines: the Serial Data (SD) line, which carries the audio data, the Serial Clock (SCK) or bit clock, which times each data bit and the Word Select (WS) line, which alternates to distinguish between the left and right stereo audio channels.



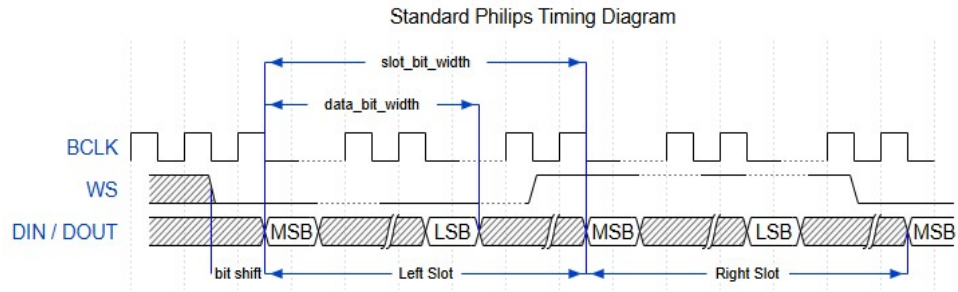
URL Source

Figure 3.8: I2S Protocol Frame Format.

This configuration enables I2S to handle full-duplex communication, meaning that it can send and receive data simultaneously, making it ideal for stereo systems and real-time audio streaming. I2S is versatile, supporting a range of audio data formats, including 8, 16, and 24-bit PCM audio at sample rates from 8 kHz to 192 kHz, allowing it to adapt to various high-fidelity audio standards. To further enhance compatibility and flexibility, I2S supports multiple data alignment modes (left-justified, right-justified, and standard modes), which adjust how data is aligned relative to the WS signal, for different device requirements. Each audio transmission starts with the most significant bit (MSB), allowing I2S to be used by devices with different word lengths by padding or clipping extra bits. Additionally, I2S can be configured in master-slave mode, where the master device

generates the clock signals to maintain synchronized communication across all connected devices. This setup reduces data jitter which a common issue in audio transmission and ensures stable and consistent audio quality. With a low pin count, high precision, and efficient data handling, I2S is widely implemented in high-resolution audio applications such as digital audio workstations (DAWs), automotive sound systems, consumer electronics like smartphones and Bluetooth devices, and professional audio equipment [106].

The ESP32 microcontroller includes unique I2S features that extend beyond the standard I2S protocol. It contains two independent I2S peripherals, which can act as either transmitters or receivers, making it flexible for different audio applications. These peripherals support a range of configurations, including standard I2S and TDM (Time-Division Multiplexing) modes, which enable it to handle complex multi-channel audio streams. Additionally, ESP32's I2S0 peripheral has built-in support for Pulse Density Modulation (PDM) and direct connections to the ESP32's internal ADC (Analog-to-Digital Converter) and DAC (Digital-to-Analog Converter), allowing for direct audio sampling and playback without external converters. Both I2S peripherals utilize ESP32's DMA (Direct Memory Access), which minimizes CPU load by allowing data transfer directly to and from memory, which is important for real-time audio applications. Furthermore, ESP32's I2S drivers include power management capabilities to prevent signal disruptions during light-sleep mode, ensuring stable audio processing even in low-power states [107].



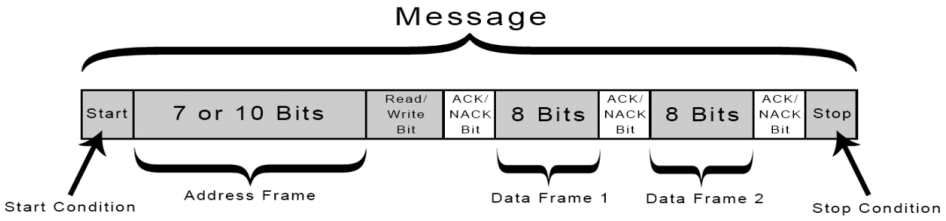
URL Source

Figure 3.9: ESP32 I2S Protocol Standard Philips Format

I2C Protocol

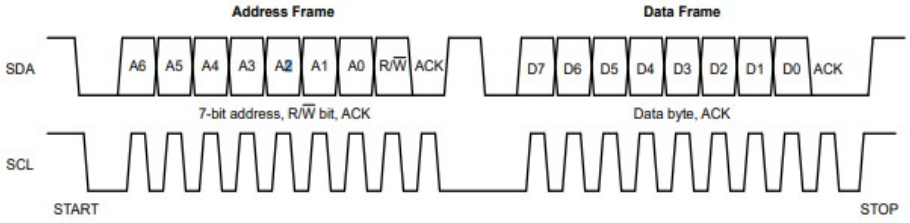
The I2C (Inter-Integrated Circuit) protocol, is a two-wire communication standard, designed for connecting multiple devices over short distances, typically on the same PCB. Using only two lines, Serial Data (SDA) and Serial Clock (SCL), I2C provides bidirectional communication between a controller (master) device and multiple target (slave) devices, each with a unique address. This allows I2C to support multiple devices on a single bus with minimal wire connections, making it widely used in embedded systems and microcontroller projects. The protocol operates synchronously, meaning data

is transmitted in sync with the SCL line, with data transfer rates typically reaching up to 400 kbps in standard mode and up to 3.4 Mbps in high-speed mode. Each transmission begins with a start condition and concludes with a stop condition, helping manage the bus by signaling active communication. Additionally, after each byte of data, the receiver sends an acknowledgment (ACK) bit, ensuring data integrity and error checking.



URL Source

(a) I2C message structure with address and data frames.



URL Source

(b) I2C address and data frames with SDA and SCL waveforms.

Figure 3.10: I2C protocol frame structure

I2C has an open-drain architecture that requires pull-up resistors on both lines, maintaining a high signal when idle and allowing any device to pull the line low to send data. It also supports clock stretching, enabling slower devices to hold the SCL line low while they process data, which is crucial for synchronized communication. In systems with multiple controllers, I2C includes a feature to prevent data collisions, prioritizing transmissions based on their start times. However, it is sensitive to bus capacitance and can its data rates can be reduced with increased wire lengths or device counts, which may result in performance limitations.

The ESP32's I2C implementation supports two I2C controllers (I2C0 and I2C1), enabling multiple independent I2C buses, useful for projects with devices sharing addresses. Any GPIO can serve as SDA or SCL, providing pin flexibility beyond the fixed I2C pins on many microcontrollers. ESP32 also supports frequencies up to 400 kHz in Fast mode, with adjustable clock stretching and timing settings for handling slower devices. Additionally, the ESP-IDF I2C API allows dynamic resource management, letting users easily add or remove devices through handles, which is useful for systems requiring configuration changes [108].

HTTP Protocol

The Hypertext Transfer Protocol (HTTP) is an application-layer protocol that enables data transfer across the web, used for communication between clients (e.g. browsers) and servers over the TCP/IP stack. HTTP uses a request-response model where a client sends a request, often specifying a method like GET, POST, PUT, or DELETE, and the server returns a response, typically with content such as HTML or JSON data. HTTP is stateless, meaning each interaction is independent and session management is handled by headers and cookies. HTTP versions have evolved to enhance efficiency, with HTTP/1.1 introducing persistent connections, while HTTP/2 adding multiplexing, header compression, and server push, to improve loading speed and reduce latency by optimizing how data is transmitted. HTTP's request structure includes a method, headers, and an optional body, while responses are status codes like 200 (OK) and 404 (Not Found), informing clients on the request outcome [109], [110].

The ESP32's HTTP functionality is optimized for embedded applications, with both HTTP client and server APIs available through the ESP-IDF (Espressif IoT Development Framework). The ESP32's HTTP client supports standard HTTP methods like GET and POST and can maintain persistent connections to reuse connections for multiple requests, conserving CPU resources and reducing latency. The ESP32's built-in HTTP server allows the creation of simple, interactive web services on IoT devices. With this capability, the ESP32 can host web pages or APIs, allowing users or other devices to connect directly to it for data viewing and control. For example, it can serve as a dashboard for real-time sensor data or a control panel for device settings. The server supports common HTTP methods like GET and POST, letting developers to set up multiple routes (URI handlers) to handle different actions, such as data requests or commands. Additionally, the ESP32's HTTP server can handle multiple connections simultaneously and support asynchronous processing, which keeps it responsive to other requests while handling longer tasks. Developers can also access socket-level functions if they need custom handling, making it versatile for specific IoT needs. For live updates, the ESP32 can use websockets or server-sent events, allowing clients to receive real-time data, suiting for applications like live monitoring and remote control [111].

UDP Protocol

The User Datagram Protocol (UDP) is a lightweight, connectionless communication protocol within the Internet Protocol (IP) suite, that operates at the transport layer of the TCP/IP stack. Unlike the Transmission Control Protocol (TCP), UDP prioritizes speed and efficiency over reliability by eliminating error-checking and connection-handling overhead, making UDP ideal for applications that require real-time data exchange, such as online gaming, video streaming, and IoT communications. UDP transmits data in discrete packets called datagrams, each of which contains a header (with source and destination port numbers) and a payload, without establishing a prior connection between

the sender and receiver. This connectionless nature provides faster data transfer, as it eliminates the overhead associated with connection setup and management. Although UDP does not provide reliability features, and if needed they need to be managed by application layer, its simplicity allows for lower latency and faster data transmission which is particularly useful in applications where speed is critical and occasional data loss is acceptable[112].

In this thesis, UDP was utilized for communication between Unity and the ESP32, which was programmed as an access point (AP). Unity acted as the client, sending data packets over UDP to the ESP32, which hosted a lightweight UDP server to process incoming commands. This setup allowed for real-time interaction, where Unity could dynamically control the haptic feedback system by sending specific commands to the ESP32. UDP's low overhead ensured minimal latency, a critical factor in achieving responsive feedback for haptic applications. Although UDP does not inherently handle retransmissions or acknowledgments, occasional packet loss does not significantly impact the overall haptic experience, since the somatosensory system has certain threshold (approximately 5 to 10 milliseconds) below which temporal variations in stimuli are not perceivable.

Chapter 4

Implementation

This chapter details the implementation of the proposed wireless haptic system for texture feedback. The implementation begins with a description of the hardware and software setup which enables the capture of audio and accelerometer data from real world textures. This raw data undergoes preprocessing and feature extraction, resulting in the creation of datasets that capture the frequency and amplitude characteristics of the textures. These datasets are the used for driving a Linear Resonant Actuator (LRA) in response to real-time interactions. The Unity integration enables the simulation of textured virtual objects and the interaction of the user's fingertip with these objects using the Leap Motion hand tracking device.

4.1 Data Capture

In this section the Data Capture system will be analyzed, by first introducing the hardware parts that were employed and then the software development that was implemented to achieve simultaneous data capture from a microphone and an accelerometer sensor.

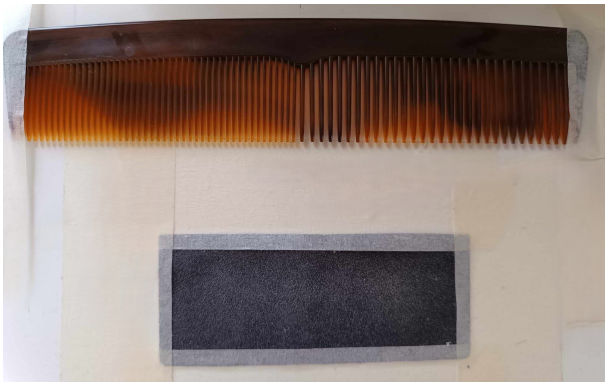
4.1.1 System Overview

The data capture system consists of several hardware parts as well as software development in order to ensure reliability and versatility of the texture recording process. First of all, the ESP32-S3-DEV-KIT-N8R8 was used as the main processor of the system, with the Adafruit I2S MEMS Microphone Breakout SPH0645LM4H and the ADXL345 Triple-Axis Accelerometer soldered directly to its pins for simultaneous data capture. The whole system is powered by a 18650 Battery Shield Mobile Power Bank in order to ensure proper handling of the power needs of each component and also to give the system wireless capabilities. The ESP32 was programmed as a WiFi server and a client was developed for the wireless control of each recording with 4 second duration each. After the recording process finishes, the ESP32 sends the raw data of both sensors to

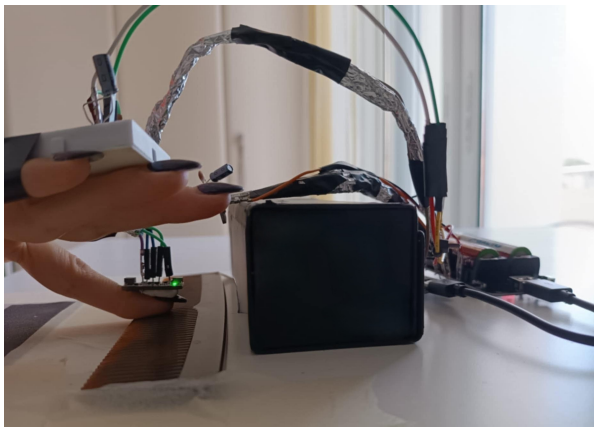
the client, through HTTP requests, and the client saves the audio data in WAV format and the accelerometer data in CSV format in a folder on the PC desktop for further processing.

For this thesis three textures were chosen for recording and processing to deliver haptic feedback: a piece of sandpaper with 180 grit, to capture finer details, and a hair comb featuring two sections with 1-millimeter and 2-millimeter teeth spacing. All textures are made of hard materials, ensuring clear and consistent feedback during haptic interactions. The comb's varied spacing offers mid-level textural detail, complementing the finer granularity of the sandpaper, allowing for the exploration of how different spacing influences tactile perception.

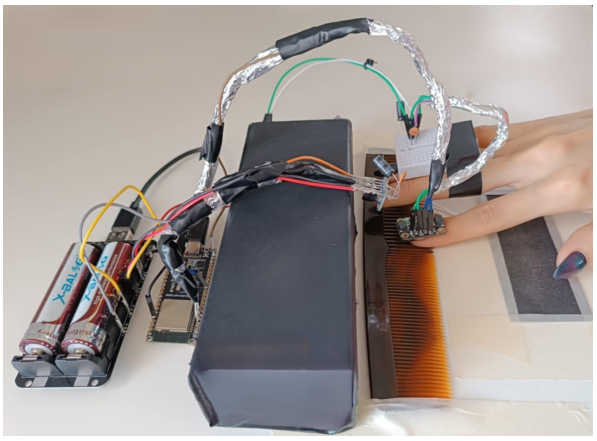
The recording setup involved positioning the microphone directly facing the interaction point to capture sound, while the accelerometer was mounted on the fingernail of the index finger using an adhesive putty for stable and precise data capture during the interaction. All interactions were performed with a bare finger, in a single linear motion from left to right, while trying to maintain a consistent speed and pressure.



(a) Hair Comb and Sandpaper Textures.



(b) Side View of the Data Capture System



(c) Top View of the Data Capture System

Figure 4.1: Overview of the Data Capture System.

4.1.2 Hardware Setup

The hardware setup plays a crucial role in ensuring that the system captures accurate and reliable data during texture interactions. In this section, we'll explore the key hardware components used to gather both audio and accelerometer data, as well as how they were integrated together to form the recording system. As mentioned in the previous subsection, the ESP32-S3 was the main processor managing data acquisition and wireless communication.

4.1.2.1 Battery Power Supply

To ensure stable power supply, the whole system was powered using a 18650 Battery Shield Power Bank. This setup was chosen over powering the sensors directly from the ESP32, as the ESP32's onboard voltage regulator, although capable, can introduce high-frequency noise, especially when Wi-Fi or other peripherals are active. This noise can interfere with sensitive components like the ones used in this system, which rely on clean and stable power to accurately capture data and are highly vulnerable to power fluctuations. The sensors were powered by the 3.3V output pins of the battery shield, while the ESP32 was powered through the USB output of the battery. Additionally, a jumper wire was connected from one of the ESP32 ground pins to one of the battery ground pins, ensuring that all components shared a common ground, preventing issues that could arise from different ground references between the sensors and the microcontroller such as ground loops. The battery pack, which features its own onboard voltage regulator, provides a more stable power source, helping to reduce potential interference. To further improve stability, a 0.1μF and a 10μF capacitor were placed between the power and ground pins of the sensors in parallel. The 0.1μF capacitor helps filter out high-frequency noise, which can be introduced through electromagnetic interference (EMI) or fast-switching components within the system, like the voltage regulator circuit. This helps to reduce spikes and ripples that could affect sensor readings. The 10μF capacitor on the other hand, smooths out lower-frequency voltage fluctuations, helping to buffer any sudden power demands or load changes. By combining both capacitors, a filtering system was created to prevent a broad range of electrical noise and ensure both sensors operate with minimal interference from power supply stability.

4.1.2.2 Accelerometer Setup

The ADXL345 accelerometer was chosen for its compact size and adjustable sensitivity with configurable range (±2g, ±4g, ±8g, or ±16g) enabling the capture of subtle changes in motion. Also, its low power consumption and I2C interface, make it efficient and easy to integrate with the ESP32, ensuring reliable and accurate data transfer. A small breadboard was used for the power connections, utilizing the 3.3V output of the battery pack for power. The ground (GND), 3.3V (VIN), and chip select (CS) pins of the ADXL345 were linked to the battery, including the capacitor filtering system ensuring

stable power delivery, as described in the power supply configuration. To improve signal integrity, the Serial Data (SDA) and Serial Clock (SCL) lines were soldered directly to the corresponding ESP32-S3 GPIO pins (pin 3 for SDA and pin 2 for SCL), establishing an I2C communication. To reduce the risk of electrostatic interference, which could distort the sensor readings, the jumper wires used for the signal lines were wrapped in aluminum foil. Electrostatic noise in this context can arise from several sources, such as environmental static buildup, nearby electronic devices, or human interaction during data capture. Since the accelerometer is particularly sensitive, any electrostatic charge around the unshielded signal lines could introduce distortions. The foil acted as a shield to block external electrostatic noise from affecting the signal integrity, and a grounding wire was connected from the foil to the battery ground pin ensuring that any captured interference was safely dissipated, preventing it from reaching the signal lines. The accelerometer was mounted securely on the user's nail of the index fingertip using an adhesive putty to ensure stable contact with the texture during interactions. This placement allowed for direct motion capture, minimizing unwanted vibrations or motion artifacts.

4.1.2.3 Microphone Setup

The SPH0645LM4H MEMS I2S breakout microphone was selected for several features it provides making it suitable for this project. First, it includes an onboard analog-to-digital converter (ADC), which directly converts the captured analog sound into a digital signal through the I2S interface. This eliminates the need for an external ADC, simplifying the system's design and reducing noise vulnerability that analog systems typically face. Furthermore, the microphone has a high signal-to-noise ratio (SNR), which means it can capture sound while minimizing background noise, making it suitable for recordings in a non soundproof environment. The flat frequency response across its operating range, ensures accurate and consistent sound capture without emphasizing any particular frequency band, making it suitable for capturing realistic fine details from texture interactions. This is critical for capturing the fine details of texture interactions. Finally, its compact size, low power consumption and high fidelity output were some other reasons for choosing this specific microphone for this project. The microphone was powered from the 3.3V output of the battery, with filtering capacitors (0.1µF and 10µF) stabilizing the voltage supply, as discussed earlier. To ensure reliable I2S communication, the microphone's Bit Clock (BCLK), Word Select (LRCL), and Data Output (DOUT) lines were connected to the chosen ESP32-S3 GPIO pins and the jumper wires used for these connections were as short as possible for the setup to minimize noise in the data lines. Signal integrity was further preserved by shielding these connections with aluminum foil to prevent electrostatic interference, which could affect the digital signal, similar to the accelerometer. Also the microphone's channel select pin (Sel) pin was grounded, meaning the left channel mono is selected for transmission. The microphone was strategically placed near the interaction point, with its audio input hole facing directly to the interaction region as shown in the Figure 4.17, to ensure it captured the most accurate sound

during texture recordings.

4.1.3 Software Setup

The software implementation was designed to manage the simultaneous capture of audio and accelerometer data, ensure synchronization, and establish a wireless control system for the recording process. The setup includes several steps such as embedded programming on the ESP32-S3, the configuration of communication protocols to ensure smooth and reliable data flow development and of client software for data acquisition and saving. This section details each of these components and their connections in order to achieve the desired recording setup. In the figure 4.2 there is a flowchart of the ESP32-S3 code. The state "A" refers to the "Wait for Client Command" that due to limited space could not be used for all the required transitions.

4.1.3.1 ESP32-S3 Embedded Programming

The ESP32-S3 was programmed using PlatformIO, an extension for Visual Studio Code (VS Code). PlatformIO was selected over the Arduino IDE because it offers greater flexibility, better library management and debugging tools, which are essential for a project involving multiple sensors and real-time data synchronization. PlatformIO also supports FreeRTOS, the real-time operating system embedded within the ESP32-S3, which allows multiple tasks to run concurrently across the microcontroller's dual cores. The Arduino framework was chosen for its ease of development with its wide range of pre-built libraries,that simplify setting up both I2C communication for the accelerometer and I2S audio streaming for the microphone. While lower-level frameworks such as ESP-IDF would offer finer control over hardware, Arduino's simplicity makes it faster to develop with, without sacrificing functionality. Also its strong community support offers valuable resources for troubleshooting.

The ESP32-S3-DEV-KIT-N8R8 development board was not directly supported in PlatformIO, but its pin configuration is compatible with the ESP32-S3-DevKitC-1 development board, which was selected. However, this configuration did not support PSRAM by default, which was critical in this project to store large data buffers during real-time data capture, ensuring enough memory capacity. To resolve this, specific build flags were added to the platformio.ini file:

```
build_flags =
-DBOARD_HAS_PSRAM
-mfix-esp32-psram-cache-issue
board_build.arduino.memory_type = dio_opi
```

The first flag enables the use of the external 8MB PSRAM on the ESP32-S3 and the second flag resolves know cache handling issues with PSRAM ensuring system stability

during memory operations. The dio_opi setting configures the memory interface to Dual I/O Octal Peripheral Interface, which optimizes PSRAM access speed.

To manage the simultaneous data capture from both the accelerometer and microphone, FreeRTOS was used for task scheduling on the ESP32-S3 dual-core processor. The ESP32-S3's architecture offers two separate cores, Core 0 and Core 1, allowing for custom task distribution and parallel execution. This flexibility was crucial for this project since each of the sensors that were used has different requirements in terms of communication protocols and data rates, leading to different CPU attention needs. Assigning the accelerometer and audio capture to the same core, would likely lead to resource contention, meaning that the CPU would have to constantly switch between the two tasks leading to potential delays, since the I2C communication of the accelerometer requires immediate CPU attention. Based on that, to make sure the system handles the accelerometer and audio data effectively, the tasks were assigned to different cores. The accelerometer task was pinned to Core 0, that typically handles system-critical tasks like Wi-Fi and time-sensitive protocols, making it a fitting choice for the accelerometer task for data acquisition with minimal delays. The audio task was pinned to Core 1 because it uses the I2S peripheral, which is mostly handled by harware reducing the demand on the CPU. Core 1, is more suited for less time-sensitive tasks that do not require constant attention from the processor. This separation allows the two tasks to run smoothly without competing for CPU resources. The stack sizes and priorities of each task were chosen based on the computational requirements and communication protocols used in each, as well as testing. The accelerometer task has a higher priority, as like previously stated, it uses I2C communication which is more sensitive to timing and requires immediate attention from the CPU for handling I2C interrupts. Its stack size is smaller, as the accelerometer is relatively lightweight at 800 Hz rate and does not require as much memory. On the other hand audio has lower priority since I2S data is handled more autonomously without the need for as frequent CPU interventions, but larger stack size since larger buffers are required to handle the 48 kHz rate of data income.

```
BaseType_t result;
result = xTaskCreatePinnedToCore(captureAudio, "CaptureAudio", 8192 ,
                                 NULL, 10 , &audioTaskHandle, 1);
if (result != pdPASS) {
    Serial.println("Failed to create CaptureAudio task");
    while (1);
}

result = xTaskCreatePinnedToCore(captureAccel, "CaptureAccel", 4096,
                                 NULL, configMAX_PRIORITIES - 1, &accelTaskHandle, 0);
if (result != pdPASS) {
    Serial.println("Failed to create CaptureAccel task");
    while (1);
}
```

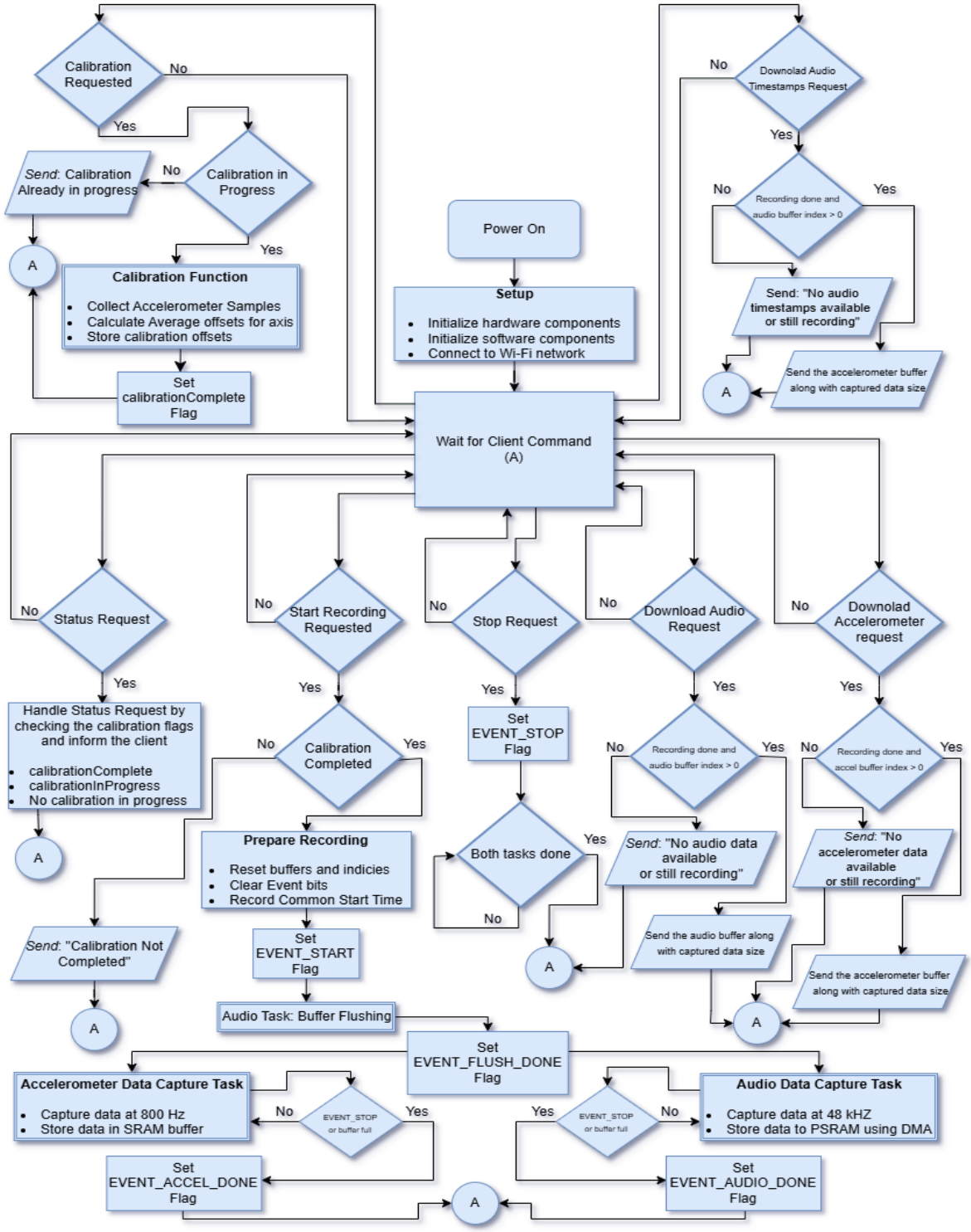


Figure 4.2: Data Capture Flowchart.

To synchronize the tasks, FreeRTOS Event Groups are used which allow tasks to communicate and synchronize their execution by setting and waiting for specific event bits. In this project, the event bits, when set, ensure that both the audio and accelerometer tasks initiate recording at the same moment, aligning the data for accurate post-processing. Event groups are non-blocking, meaning tasks enter a waiting state until an event occurs, freeing the CPU to perform other operations. This is very helpful in handling tasks across different cores as it ensures parallel execution without timing issues or contention. There is also the flexibility of utilizing different event bits, in a single event group so the system can manage different conditions, such as in this project, starting, stopping and confirming task completion. Below is an example of the usage of the event system in this project. First, the event group flags are defined to represent different system states, such as starting and stopping tasks, handling the audio and accelerometer data, and signaling when tasks are complete. The event group is then initialized in the setup function, where it's checked to ensure successful creation. Finally, the system uses the EVENT_START bit to signal the start of both tasks.

```
#define EVENT_START (1 << 0)
#define EVENT_STOP (1 << 1)
#define EVENT_TIMER (1 << 2)
#define EVENT_AUDIO_DONE (1 << 3)
#define EVENT_ACCEL_DONE (1 << 4)
#define EVENT_FLUSH_DONE (1 << 5)

// Create the event group in setup
xEventGroup = xEventGroupCreate();
if (xEventGroup == NULL) {
    Serial.println("Failed to create event group!");
    while(1);
}

// Synchronization for starting tasks
xEventGroupSetBits(xEventGroup, EVENT_START);


xEventGroupWaitBits(xEventGroup, EVENT_START, pdTRUE, pdFALSE, portMAX_DELAY);
// Start audio task...
```

The ESP32-S3 was configured to operate in station mode (STA mode), connecting to an existing Wi-Fi network. This setup was chosen to allow communication with a client device, such as a PC, for remotely controlling the recording process and retrieving data. To ensure that the Wi-Fi connection remained stable during data capture, the Wi-Fi power-saving mode was disabled preventing the ESP32 from automatically entering a lower-power state, which could interrupt the real-time data acquisition from the accelerometer and microphone. Without this setting, network disconnections would

sometimes occur, causing delays or data loss during the recording sessions. An HTTP server was implemented using the ESPAsyncWebServer library, which offers non-blocking communication. This maintains stable data acquisition, as the system can handle network requests asynchronously. The server is configured to listen for HTTP requests from the client, allowing remote control over the recording process. The server handled several HTTP endpoints:

- /prep: Begins the accelerometer calibration process. Once calibration starts, it sets flags for the calibration status and prevents further requests until it completes.

```
server.on("/prep", HTTP_GET, [](AsyncWebServerRequest *request){
    if (!calibrationInProgress) {
        prep = true;
        calibrationInProgress = true;
        calibrationComplete = false;
        request->send(200, "text/plain", "Calibration started.");
    } else {
        request->send(200, "text/plain", "Calibration in progress."); }
    });
```

- /status: Returns the calibration status, that can either be complete, in progress, or hasn't started yet.

- /start: Begins the recording of both sensor data. It clears buffers, resets indices, starts timers and sets flags that trigger the data capture process, while ensuring that recording only begins after the calibration is completed.

- /stop: Stops the recording by setting the relevant flags, waiting for both sensor tasks to finish, and recording the final timestamp.

- /download_audio: Sends the recorded audio data to the client.

- /download_accel: Sends the accelerometer data to the client.

- download_audio_timestamps: Sends the buffer containing each audio sample timestamp.

This setup allows remote devices to trigger recordings, check calibration status, and download the recorded data via simple HTTP requests, providing flexibility during the recording sessions. To further ensure data integrity, custom headers such as X-File-Size were added to the HTTP responses.

4.1.3.2 Sensor Data Capture Logic

The data capture process involves recording synchronized sensor data from both the ADXL345 accelerometer and the SPH0645 microphone. As mentioned before two different protocols are used for communication (I2S and I2C) and each sensor is initialized with specific settings to meet the project's requirements for real-time data capture, ensuring that both the accelerometer and microphone data are gathered and processed efficiently. The system also uses FreeRTOS task management and Event Groups to synchronize the start and stop of data capture for both sensors. The recording process for both sensors has a fixed 4 second duration. This decision was made during the system design, for several practical reasons such as consistency in capturing texture interactions and ensuring synchronization between the two sensors. The 4 second window proved to be sufficient to capture interaction with texture, while maintaining a manageable data size for further processing and without the need to handle varying data stream lengths. Also another important reason behind this decision was that knowing the exact duration of the recording enables pre-calculation of buffer sizes for both sensors, ensuring optimized memory allocation and preventing issues such as memory overflow or corruption and memory fragmentation.

Another design choice that applies to the whole recording process is to store the data of both sensors locally on the ESP32-S3 during the recording rather than streming them in real-time to the client. This approach avoids potential network latency and instability issues that could lead to missing samples or corruption. By locally storing the data, the system ensures that the recording is uninterrupted and free from the risk of network problems. After each recording is completed, the data is safely transferred to the client in a single operation, ensuring full data integrity and reducing the risk of corruption during transmission. With these design choices, the project aims for accurate, high-resolution data capture and efficient post-recording transmission.

Accelerometer

The ADXL345 accelerometer is configured to capture texture interactions at a sampling rate of 800 Hz, chosen to meet both the bandwidth requirements for capturing motion and the limitations of the I2C bus. According to the Nyquist theorem, the 800 Hz sampling rate allows the system to capture frequencies up to 400 Hz, which is sufficient to represent most tactile frequencies that humans can perceive during texture interaction. Tactile feedback typically falls within the range of 0 to 300 Hz, with most of the tactile frequencies that humans can feel through tactile receptors being below 400 Hz, as discussed in Chapter 2. This means that this bandwidth was sufficient for capturing the spectrum of tactile feedback frequencies for meaningful and realistic texture reproduction through haptic feedback. Also, the I2C communiation speed for the ADXL345 accelerometer is limited to maximum clock speed of 400kHz in fast mode. At this speed the accelerometer can achieve data rate of up to 800 Hz, as faster data rates would require

higher communication speeds than the I2C can offer. While 800 Hz rate is the maximum sampling when using I2C, the ADXL345 has capabilities for higher rates up to 3200 Hz using SPI communication. However, the use of I2C is preferred in this project due to its simpler wiring requirements and lower pin usage, lower power consumption and the fact that the 800 Hz were sufficient for capturing the tactile perceptible vibrational range.

The range setting of an accelerometer determines its sensitivity in motion and in this project the ±2g range was selected to capture the fine and small-scale motions and vibrations that occur during texture interaction. Choosing a small range improves sensor's resolution enabling it to detect subtle changes, however there is a noise trade-off since the sensor becomes more sensitive to small fluctuations as well. In this project, this trade-off is handled in post processing steps, to ensure that the noise is minimized. The forces applied when sliding a finger across a surface are relative small and usually fall within the ±2g threshold, making it ideal for accurately capturing tactile features of different textures in high resolution. Below is the code snippet of the initialization of the sensor in the setup function using the Adafruit_ADXL345_U.h library.

```
#include <Adafruit_ADXL345_U.h>
    accel.setDataRate(ADXL345_DATARATE_800_HZ);
    accel.setRange(ADXL345_RANGE_2_G);
    Wire.setClock(400000);
```

For storing the accelerometer data during recording, Static Random Access Memory (SRAM) is used. This choice is based on access speed and real time performance, as SRAM is faster and more efficient in these scenarios than Pseudo-Static Ram (PSRAM) that offers larger (8 MB) but slower memory. For real-time data capture, the speed of read/write cycles is an important factor to consider and SRAM is optimized for minimal latency. However, SRAM is limited in size on the ESP32-S3 (512 KB), and it is shared across various tasks and processes running on the microcontroller like FreeRTOS tasks, Wi-Fi communication, and general program variables. To calculate the buffer size that is needed for storing the whole duration of each recording the following calculations were made:

The accelerometer samples data at 800 Hz, capturing three axes (x, y, z) per sample, each stored as a 32-bit floating-point number (4 bytes), along with a 64-bit timestamp (8 bytes). Each sample, therefore, requires:
$$\boldsymbol{Bytes\ per\ sample = 8\ bytes\ (timestamp) + (3axis * 4\ bytes/axis) = 20\ bytes}$$

The total number of samples captured in 4 seconds is:
$$\boldsymbol{Total\ Samples = 800\ (sample\ rate)\ *\ 4\ (seconds) = 3,200\ samples}$$

The total buffer size required is:
$$\boldsymbol{Total\ Buffer\ Size = 3200\ (samples)\ *\ 20\ (bytes/sample) = 64,000\ bytes}$$

The calculated buffer size at 64 KB, fits within the available 512 KB of SRAM. Before finalizing this decision, it was verified that even after allocating 64 KB for the accelerometer buffer, there would be enough SRAM left to handle other essential tasks and processes.
In the following code we can see the implementation of these calculations, the allocation of the buffer in SRAM and the struct to store the sample format. The AccelData structure is defined to hold the sample values, consisting of a 64-bit timestamp and three 32-bit floating-point values representing the x, y, and z axis accelerations. The timestamp is used to keep track of the precise timing of when each data point was captured, ensuring proper synchronization with the microphone, during post-processing, a crucial addition for aligning data streams with different sampling rates. The __attribute__((packed)) forces the compiler to store the structure in memory without adding any extra padding between its fields. This ensures that the data is storing is compact, exactly as calculated, which is important for minimizing memory usage and having more control over the sytem.

```
#define TOTAL_ACCEL_SAMPLES (ACCEL_SAMPLES_PER_SECOND * TOTAL_RECORDING_SECONDS)
#define ACCEL_BUFFER_SIZE (TOTAL_ACCEL_SAMPLES * sizeof(AccelData))

// Struct to store timestamp and 3-axis accelerometer data
struct __attribute__((packed)) AccelData {
    uint64_t timestamp;  // 64-bit timestamp in microseconds
    float accel_x;       // X-axis
    float accel_y;       // Y-axis
    float accel_z;       // Z-axis
};

// Allocating buffer in SRAM
accelBuffer = (uint8_t*) malloc(ACCEL_BUFFER_SIZE);
if (accelBuffer == nullptr) {
    Serial.println("Failed to allocate memory for accelBuffer in SRAM");
    while (true);  // Halt if allocation fails
}
```

The ADXL345 accelerometer was configured for a 800 Hz sampling rate, but during development, it was observed that despite setting this rate, the output was not consistent in accurate 800 Hz. One issue is that the ADXL345, when read via I2C, leaves the previous sample on its output lines until the next new sample is available, leading to repeated data when the system polls the sensor at irregular intervals for data capture. The ADXL345 has a data ready interrupt feature, that was initially tested, so that the system gets notified when new data becomes available and read the output of the ADXL345 only then. However, this method proved unreliable as inconsistencies were observed during testing.
To address this issue, a precise timing mechanism was implemented to ensure that the microcontroller reads the sensor exactly when new data is available. This is achieved by us-

ing a software timer that triggers the accelerometer reading task every 1250 microseconds, corresponding to the 800 Hz sampling rate since : $\frac{1\ second}{800\ samples} = 1250\ \mu s/sample$

In the following code there there is the implementation of the timer, its creation and initialization to trigger periodically at the calculated interval as well as the handler function that notifies the accelerometer task that its time to read the ADXL345 output.

```
esp_timer_create_args_t timer_args = {
    .callback = &onAccelTimer,
    .arg = NULL,
    .name = "accel_timer"
};

// Create the esp_timer
esp_timer_create(&timer_args, &accel_timer);

// Set the timer to trigger periodically at 1250 microseconds (800 Hz)
esp_timer_start_periodic(accel_timer, 1250);

void onAccelTimer(void* arg) {
    BaseType_t xHigherPriorityTaskWoken = pdFALSE;
    xTaskNotifyGive(accelTaskHandle); // Notify the task
}
```

Using the ESP32's high-resolution software timer (ESP Timer) offers precise microsecond-level control, triggering data reads from the ADXL345 at consistent intervals. The software timer was chosen over hardware timers of the ESP32-S3 because of its flexibility and better integration with FreeRTOS. By running independently it ensures multitasking and also it avoids competition for limited harware resources. Additionally, by using task notifications with FreeRTOS, the system ensures efficient communication between the timer and the accelerometer data capture task. The notification system allows the task to be triggered as soon as the timer reaches the desired interval, providing real-time feedback with minimal overhead.

Before each recording, a calibration step is performed, as there are varying initial positions of the accelerometer throughout recordings. Since the accelerometer is mounted on the fingertip, the zero-point shifts depending on the orientation of the finger during each recording session. To ensure the system captures accurate motion data, calibration is done while the finger is kept as still as possible, allowing the sensor to determine its resting offsets for each axis. During calibration, the accelerometer captures a large set of stationary samples in small fixed intervals and averages the x, y, and z axis values. This averaged data is used to calculate offsets for each axis, which are later subtracted from the accelerometer's real-time measurements. The z-axis is adjusted by subtracting the gravitational constant (9.8 m/s^2) to prevent the influence of gravity. After calibration, the user starts the interaction from the same position, sliding their finger across the textured

surface, ensuring accurate data capture. The following code snippet demonstrates the calibration process, where 1000 samples are collected, averaged, and used to set the calibration offsets:

```
for (int i = 0; i < num_samples; i++) {
    sensors_event_t event;
    accel.getEvent(&event);
    x_samples[i] = event.acceleration.x;
    y_samples[i] = event.acceleration.y;
    z_samples[i] = event.acceleration.z;
    delay(10);
}
//Calculate average values
float x_sum = std::accumulate(x_samples.begin(), x_samples.end(), 0.0);
float y_sum = std::accumulate(y_samples.begin(), y_samples.end(), 0.0);
float z_sum = std::accumulate(z_samples.begin(), z_samples.end(), 0.0);
float x_avg = x_sum / num_samples;
float y_avg = y_sum / num_samples;
float z_avg = z_sum / num_samples;
//Set calibration offsets
x_offset = x_avg;
y_offset = y_avg;
z_offset = z_avg - 9.8; //Subtract gravity
```

The captureAccel task is a permanent while loop that handles the accelerometer data capture process. The task remains stalled until the EVENT_FLUSH_DONE flag is set, signaling that the audio task has completed its initial flushing, ensuring that the accelerometer and audio data start recording simultaneously. Once the flag is received, the task enters the recording loop and starts reading data from the accelerometer at a rate of 800 Hz, maintaining this rate through notifications from the software timer that fires every 1250 microseconds. Within the recording loop each accelerometer sample is read using the Adafruit_ADXL345 library's getEvent() function, which captures the three-axis accelerometer data, along with a timestamp, ensuring the timing of each sample is saved. The accelerometer readings are adjusted by subtracting the calibration offsets that were calculated earlier, ensuring only the relative motion is captured. Each sample, is stored in the pre-allocated SRAM buffer using the memcpy function. This function allows efficient copying of the AccelData structure into the buffer through its low-level memory handling, ensuring that all data points are packed without adding unnecessary overhead or delays. The use of memcpy ensures that data is sequentially stored in memory, avoiding potential fragmentation issues. The loop continues reading and storing samples until either the EVENT_STOP flag is received (indicating that the recording has ended), or the buffer becomes full after collecting the exact number of samples expected for a 4-second duration (3200 samples in total), preventing buffer overflows. Once the recording is complete, the task sets the EVENT_ACCEL_DONE flag, signaling that the accelerometer data capture

is finished and ready for further processing.

Audio

The SPH0645 operates via I2S (Inter-IC Sound), which is a digital audio protocol for transferring audio data between devices and in this case it creates a direct connection of the microphone with the ESP32-S3. The microphone as mentioned before, sends out digital audio samples, meaning there is no need for an additional ADC (Analog-to-Digital Converter), simplifying the system's design.

The microphone was configured to capture audio signals at a sampling rate of 48 kHz. This decision is based on several factors, despite the fact that the perceptible tactile frequencies are below 1kHz. Higher sampling rates can capture micro-details in the audio signal, that can give valuable information when recreating texture through haptics. The higher temporal resolution of this rate results in higher precision when representing the subtle variations and vibrations that occur during the interaction between the fingertip and a textured surface. Also a 48 kHz rate is safer in terms of aliasing when processing the audio for feature extraction and also it is widely used in fields like audio and video processing making the system compatible with standard technologies. Finally, the 48 kHz rate is a compatible match with the ESP32's internal clock system, as it aligns with the ESP's clock divisions, allowing for better synchronization and minimizing timing errors and drifts during recording.

To ensure compatibility between the ESP32-S3's I2S peripheral and the SPH0645 microphone, specific timing adjustments were necessary due to slight differences in data alignment expectations. The SPH0645 microphone uses the Philips I2S protocol, aligning its data to the most significant bit (MSB) on each word select (WS) transition. However, a timing mismatch can occur because the ESP32-S3's I2S peripheral does not inherently account for the precise timing of the SPH0645's data output, potentially leading to misalignment in the data capture. To address this, a delay was introduced by adjusting the falling edge of the clock signal on the ESP32-S3, synchronizing the receiver to the precise moment when the microphone outputs valid data. This adjustment helps the ESP32 capture data correctly as it becomes available, resolving potential timing discrepancies. Additionally, a configuration was applied to force the I2S peripheral into Philips mode (using I2S_RX_MSB_SHIFT), aligning the ESP32's data interpretation with the SPH0645's output format.

```
// Delay by falling edge
REG_SET_BIT(I2S_RX_TIMING_REG(I2S_PORT), BIT(9));
// Force Philips mode
 REG_SET_BIT(I2S_RX_CONF1_REG(I2S_PORT), I2S_RX_MSB_SHIFT);
```

The ESP32 was set to operate in Master mode, generating the necessary clock signals (bit clock, word select, and master clock) for I2S communication. The microphone

outputs 18 bits of audio in a 24 bit lower padded I2S format. However the selected configuration was 32 bits per sample, for smoother data handling and alignment with the microcontroller's architecture, as the I2S peripheral of the ESP works best with 32 bit chunks and does not have a native support for 24 bits. The Philips I2S protocol was selected, which is a widely used standard for digital audio data transfer and is compatible with most audio components, ensuring smooth data transfer.The Audio Phase-Locked Loop (APLL) was used for precise clock generation, ensuring stable and accurate audio capture. The Master Clock (MCLK) is fixed at 12.288 MHz to generate the necessary bit clock (BCLK) for 48 kHz audio at 32 bits per sample, helping to prevent timing errors or drift during recording. Finally, the microphone produces mono audio and is wired to transmit to the left channel, which is also set in the I2S configuration. However, there is an issue with the ESP32 I2S implementation and a workaround to make this work is to set the right channel as the transmission channel, which internally is translated as the left channel.

```
% Initialize I2S
  i2s_config_t i2s_config = {
      .mode = (i2s_mode_t)(I2S_MODE_MASTER | I2S_MODE_RX),
      .sample_rate = SAMPLE_RATE,
      .bits_per_sample = I2S_BITS_PER_SAMPLE_32BIT,
      .channel_format = I2S_CHANNEL_FMT_ONLY_RIGHT,  // Use LEFT channel
      ↪  configuration
      .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_STAND_I2S),
      .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
      .dma_buf_count = 3,
      .dma_buf_len = BUFFER_SIZE_AUDIO,
      .use_apll = true,
      .tx_desc_auto_clear = false,
      .fixed_mclk = 12288000, // BCLK of 3.072 MHz required for 48kHz with 32 bit
      ↪  stereo
  };
```

Direct Memory Access (DMA) is used, to efficiently handle the large amounts of data being transferred from the microphone to memory, by allowing the I2S peripheral to directly access the system memory without involving the CPU. When using DMA, the CPU initiates a transfer between memory and the peripheral and the transfer is taken care by the DMA controller. When the transfer is completed, it sends an interrupt to the CPU that data have been received so the CPU can process them or set up more data to be transmitted, giving time to the CPU to handle other tasks in the meantime. When configuring DMA, two important settings must be chosen: buffer count, meaning how many DMA buffers will be allocated and buffer the length of each of these buffers. A smaller buffer results in more frequent interrupts, increasing CPU load, however with

larger buffers there is a latency trade-off as the DMA transfer must complete before the data can be processed. Also, DMA buffers are sotred in intrernal SRAM and cannot be allocated to the PSRAM, and each buffer is limited to the range 8 to 1024 samples. This limits the amount of data that can be buffered in memory, requiring careful consideration of buffer size and buffer length based on available memory resources. For this project, three DMA buffers are used, with a buffer length of 128, resulting after a lot of testing with different lengths and counts.

The following calculations were made to determine the total data volume of the full duration of the audio recording, just like the accelerometer, to determine the buffer size that was needed to store them before sending them to the client. Each audio sample contains a 32-bit integer (4 bytes).
The total number of samples captured in 4 seconds is:
$$\boldsymbol{Total\ Samples = 48,000\ (sample\ rate)\ *\ 4\ (seconds) = 192,000\ samples}$$

The total buffer size required is:
$$\boldsymbol{Total\ Buffer\ Size = 192,000\ (samples) * 4\ (bytes/sample) = 768,000\ bytes}$$

Given that the ESP32-S3's SRAM capacity of 512 KB was insufficient to store the total 768 KB of audio data, PSRAM was utilized. PSRAM has 8MB of memory, so the space to store the whole recording without risking memory overflows or impacting other tasks depentent on SRAM is more than enough. However, access to PSRAM is much slower that SRAM, so the time it takes for each of the 128 sample DMA buffers to transfer to PSRAM was calculated to avoid performance bottlenecks.
A new batch of 128 samples arrives every: $\boldsymbol{\frac{128\ (samples)}{48000\ (samples/second)} = 2.67\ miliseconds}$

Access speeds to PSRAM vary, due to protocol overheads, system demands and shared resources, and are generally lower especially when multiple tasks are running simultaneously. However, even in the worst case scenario of PSRAM speed being as low as 7MBps, the time it takes to write 128 samples of 32 bits (calculated as 512 bytes), is 73.14 microseconds, which is comfortably within the 2.67 interval it takes for each DMA buffer to be ready for processing. After making sure that the use of PSRAM will not affect the audio data capture, a buffer was allocated for sotring the data of each full recording by the code below.

```
// Allocate the recording buffer in PSRAM
recordingBuffer = (int32_t*)heap_caps_malloc(TOTAL_BYTES, MALLOC_CAP_SPIRAM);
if (recordingBuffer == NULL) {
    Serial.println("Failed to allocate audio buffer in PSRAM");
    while (1); // Halt execution
} else {
    Serial.printf("Allocated audio buffer at address: %p\n", recordingBuffer);
```

```
}
```

The captureAudio task is responsible for capturing audio samples from the I2S microphone and storing them in PSRAM for the recording duration. This task operates in a continuous loop, waiting for the EVENT_START flag to begin recording. When the flag arrives, first it executes a flushing process, reading six batches of 64 samples into tempBufferFlush using the i2s_read command. This clears any residual data in the I2S peripheral buffer, preventing drifts or transients that might occur in the recording without this process. The i2s_read command, reads each audio sample from the I2S input, blocking the task until the full buffer is filled each time. When flush is complete, EVENT_FLUSH_DONE is set, signaling the accelerometer task that the audio is ready to begin the recording process so the accelerometer can start as well. Then, the captureAudio task enters a recording loop where it repeatedly reads data from the microphone into a temporary buffer with 128 samples size, like each of the DMA buffers. Each read operation uses the i2s_read function with a timeout (portMAX_DELAY) to handle any data that arrives asynchronously. The i2s_read works with DMA to manage data transfers directly between the I2S peripheral and memory without loading the CPU. DMA independently moves data in blockes, which then become available for i2s_read to process. The temporary buffer is used as an intermediate step to manage PSRAM's slower access speed, ensuring smooth data transfer without gaps or timing interruptions. The audio samples are then copied from the temporary buffer to the main recordingBuffer in PSRAM using memcpy, chosen for its speed and efficiency in handling large memory transfers. Each read operation is paired with a starting timestamp, and individual timestamps are interpolated for every sample in the batch, then stored in a dedicated array to ensure precise temporal reference for each sample. The task continues this loop until either the entire buffer is filled, or an EVENT_STOP flag is received, and then the EVENT_AUDIO_DONE flag is set, indicating that the audio capture has completed.

Timestamp Synchronization

As mentioned before, the freeRTOS and the Event System are used for ensuring synchronization between tasks. To enhance timestamp management, the ESP32 maintains a common timestamp across both audio and accelerometer data to ensure synchronized time references, which is captured when the recording start command is received from the client. Each sensor's timestamp is saved relative to this common start time by subtracting it from the current reading, allowing all readings to be accurately aligned during post-processing. Additionally, the system uses esp_timer_get_time for timestamping, that provides microsecond-level precision.

Client Software and Data Retrieval

The client software, implemented in Python, manages remote control of the data capture process and ensures reliable retrieval of both audio and accelerometer data from the ESP32. To start, the client is simply run as a Python script from the command line, with the ESP32 powered on and sensors connected. Once launched, the client fully automates the data capture process, from calibration to final data download and formatting, storing both audio and accelerometer data in a specified folder on the client's computer.

At startup, the client first attempts to connect to the ESP32 via an HTTP GET request to the /prep endpoint, triggering the calibration of the accelerometer. The client has retry mechanisms allowing multiple connection attempts if network issues arise. During calibration, the client repeatedly polls the /status endpoint to monitor the server's progress, ensuring the recording starts only after calibration is complete. After calibration, the client sends a start command to the /start endpoint, signaling the ESP32 to begin the recording. To ensure uninterrupted recording, a wait time is set, after which the client sends a stop command to the /stop endpoint.

After the recording is done, the client retrieves data from the ESP32's /download and /download_accel endpoints. Each download operation includes retry logic and checks for file integrity by comparing the received and expected data sizes, confirming data are as expected before processing. The client then converts the raw audio data, to a 24-bit WAV format, preserving the recorded audio quality and aligning with the system's chosen settings for sample rates and bit depth. The accelerometer data, stored as binary data on the ESP32, is unpacked and converted into CSV format, allowing for straightforward analysis of x, y, and z-axis values and timestamps. The raw files, alongside the converted audio, accelerometer and timestamp files, are saved in a specified local directory with unique names generated for each session, allowing organized data collection for multiple recordings. Lastly, reliability measures, like timeout settings and retry attempts, are woven into each data request and download operation to protect against partial transfers or lost connections.

4.2 Preprocessing

After setting up the sensors and the data capture system, this section covers the preprocessing steps needed to prepare the audio and accelerometer signals for feature extraction and texture haptic feedback delivery. In preprocessing issues like noise, DC offsets, and signal alignment are handled, ensuring that the data accurately reflects the actual recorded textures. The preprocessing code was developed in VS Code platform, using Python.

4.2.1 Loading and Initial Preprocessing

In the loading process, raw audio and accelerometer data from pre-recorded files in the directory that they were saved by the client are first retrieved. Each recording has as output a raw audio data file, a CSV accelerometer data file including accelerometer timestamps for each frame and and a text file containing the timestamps of each audio frame. Each data source is handled according to its specific format and structure. Also, during the recording session an audio file capturing the background noise of the environment was recorded separately, so this file can be used to extract a noise profile representing the noise characteristics specific to the recording conditions. By loading this noise profile and handling it just like the audio loading process, the system can later use it for applying noise reduction to the main audio recordings. This process helps in ensuring that the subtle features of the audio signal are preserved while minimizing background noise interference.

The audio data, stored as raw binary in a .raw format, is first loaded and unpacked as 32-bit samples. Since the actual audio information lies in the 18 most significant bits and the signal was lower padded for compatibility reasons in the previous step, shifting to 24-bit removes unnecessary bits, saving space without compromising quality and data integrity. Additionally, this format aligns with standard libraries, leading to more efficient data handling. This 24-bit signal is then normalized to floating-point values in the range of -1 to 1, since in this format is more flexible for further analysis. The loading process of audio files is presented in the code below.

```
def load_audio(audio_file_path):
    with open(audio_file_path, "rb") as raw_file: # Read raw binary audio data
    raw_data = raw_file.read()
    audio_num_samples = len(raw_data) // 4  # 4 bytes per 32-bit sample
    samples_32bit = struct.unpack('<' + 'i' * audio_num_samples, raw_data) # Unpack
    ↪  32-bit samples
    samples_24bit = np.array([(sample >> 8) for sample in samples_32bit],
    ↪  dtype=np.int32) # Convert to 24-bit
    max_24bit_value = 2 ** 23
    audio_data = samples_24bit.astype(np.float32) / max_24bit_value # Normalize to
    ↪  floating point
    return audio_data
```

Additionally, timestamps corresponding to each audio sample are loaded from a text file into a NumPy array to enable synchronization, as these timestamps are essential for aligning the audio with the accelerometer data accurately in microsecond-level precision.

The accelerometer data are loaded by reading from a CSV file containing the three-axis acceleration data along with their timestamps, using pandas library for ensuring that that data remain structured and accessible. This data structure allows each axis (Accel_X, Accel_Y, Accel_Z) to be accessed either independently or all together, depending on the requirements of each stage of the analysis. The three axis of the accelerometer are saved

in a three dimensional array, while the timestamps are extracted and saved as separate integer array to align with the data format of the audio timestamps and facilitate the processing. The following code demonstrates the implementation of the steps described above.

```
def load_accel(accel_file_path):
    accel_data = pd.read_csv(accel_file_path)
    accel_timestamps = accel_data['Timestamp'].values.astype(np.int64)
    accel_data_array = accel_data[['Accel_X', 'Accel_Y', 'Accel_Z']].values
    return accel_data_array, accel_timestamps
```

4.2.2 Initial Offset Correction and Uniform Timestamp Alignment

Since audio and accelerometer data are collected simultaneously on a single ESP32-S3, the start time for each signal can vary slightly, in the scale of a few microseconds, creating a small initial offset between the two signals. After loading all the necessary files, the initial timestamp offset between the two signals is calculated based on their starting timestamps. If one signal begins later than the other, its timestamps are shifted to match the earlier signal's start time. This ensures both signals have a common starting reference point, which is important for comparisons between the data in each signal. The signals also experience non linear microsecond level drifts, which are not consistent across different sessions. These drifts are expected due to hardware limitations and the challenge of synchronizing data capture across different sample rates (48 kHz for audio and 800 Hz for accelerometer), but their scale is below the human perceptual threshold for detecting time differences in haptic feedback, which generally falls in the millisecond range. As a result, while these microsecond level drifts exist, they do not impact the intended user experience in the context of haptic texture representation. However, given these factors the intervals between the samples of each signal show some irregularities beyond the expectations based on their rates, which result in in 20.833 μs for audio and 1250 μs for accelerometer. For that reason, timestamps are reconstructed at regular intervals based on each signal expected rate, creating a uniformly spaced timeline while allowing each signal to remain at its native resolution without altering its original rate or integrity. Linear interpolation is then applied to both signals to map their data onto these regular intervals. Since the accelerometer data have 3 axis, the linear interpolation is applied to each axis separately, ensuring that the three-dimensional data is consistently structured. This process doesn't correct drift directly or alter the raw data, but it does help maintain a consistent frame of reference by reducing irregularities in time intervals. By establishing uniform timestamps, each signal has a consistent and predictable time axis, with each sample point having an exact position on the timeline making data handling smoother and more predictable. Also, since later steps in signal processing involve windowing each

signal in short synchronized time intervals for feature extraction, the uniform timestamps provide a stable temporal framework for comparison between the features of each signal, accounting for small inconsistencies that could make the windows less consistent across the signals. Overall, this approach recognizes that small, microsecond scale drifts are within perceptual thresholds for haptic feedback, meaning these drifts won't be detected by users. Instead, the uniform timestamps help simplify the data processing steps, making sure that time-based windows remain consistent and enhancing overall data integrity without directly modifying the signal content.

4.2.3 Signal Preprocessing

After the above steps, both signals undergo the same process of removing their first 100 milliseconds respectively accounting for any initial transients due to recording process, as shown in the code below. The function takes as arguments the signal, its sample rate and the duration of the part to be removed in milliseconds, and then calculates the equivalent samples based on these inputs and trims the signal respectively. The transient removal is applied to all three axis of the accelerometer signal and to the noise recording as well.

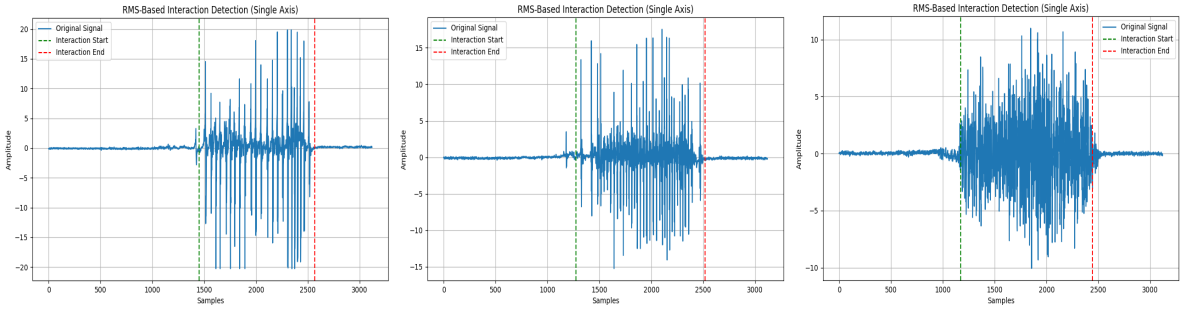
```
def remove_transient_click(signal, sample_rate, duration_ms=100):
    samples_to_remove = int(sample_rate * (duration_ms / 1000))
    trimmed_signal = signal[samples_to_remove:]
    return trimmed_signal
```

The same procedure is applied to both the timestamp arrays, for maintaining consistency. After these steps of preprocessing that were applied to both signals in the same way, the signals are manipulated separately since they are very different in nature, one representing sound waves and the other mechanical vibrations. To achieve that, separate functions have been created called process_accel and process_audio respectively each handling the signals along with their timestamps and returning a trimmed, processed and denoised version of them to the main function.

4.2.3.1 Accelerometer Signal Preprocessing

In the preprocessing of the accelerometer data, the first step is the detection and isolation of the interaction region within the accelerometer data, since in each recording not all 4 seconds represent the interaction with the texture. However, the interaction happens in a single linear motion from left to right, so there is only a single portion in each signal containing the texture information without interruptions. For that reason, the function process_accel_with_interaction_detection is called to identify the relevant interaction region and trim the data to focus only on the segment of the signal containing meaningful texture interaction. The start and stop index of the interaction is detected for the Y

Axis of the signal, since this is the primary interaction axis, through a dedicated function by calculating the root mean square energy (RMS), indicating the signal's activity over time. Other methods were tested, however the RMS showed the best and most consistent results throughout different recordings with varying energy and amplitude characteristics. The function calculates the RMS energy using a moving window across the entire signal, ensuring that energy peaks are localized accurately, helping identify clear start and end points for interaction regions. Smaller window sizes capture finer fluctuation and larger window sizes provide a smoother energy profile. For this specific project the window size that was selected after testing with different recordings was 100. High RMS energy indicate areas with high activity levels, corresponding to interactions, while low RMS values likely reflect noise or idle states. After calculating RMS values, the energy array is padded with edge values to retain continuity of the RMS signal's edges, matching the original signal length. This way the RMS energy aligns with the entire accelerometer signal, preserving temporal alignment and making the mapping of the start and stop interaction indices straightforward. Then a dynamically adjusted threshold is set to filter out background noise and isolate meaningful interaction regions in the accelerometer signal. It is calculated based on the median RMS energy of the signal, which represents the central level of background activity, ensuring the threshold adapts to each recording's unique characteristics. By using a threshold_multiplier of 6, the function scales this median energy value, filtering out low energy fluctuations or subtle noise that do not indicate active interaction. Again, the threshold value was determined after testing different values throughout different accelerometer recordings and evaluating the detection results. Then, samples exceeding this adaptive threshold are saved in an array representing the interaction region and the first and last index of this array represent the starting and ending point of the interaction based on which the signal across all axis and its timestamps are then trimmed to isolate the interaction portion.



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.3: RMS-Based Interaction Detection plots for all three signals on the Y-Axis.

The function returns the trimmed accelerometer data, the adjusted timestamps, and the indices marking the start and stop of the interaction, mapped to corresponding audio indices using the sample rate ratio between the audio and the accelerometer signals. This

mapping ensures that both signals can be trimmed to the exact same region, facilitating synchronized processing in next steps.

The rest processing is applied only on the trimmed portion of the signal, to reduce computational load and also to minimize the chance of distortions from noise in the filtering and denoising stages. Before trimming, a 1 second segment is extracted from the start of the signal creating a noise profile. This segment is chosen since the interaction always start after the first second, so it captures only background noise, for later use as a base for denoising. Every step of processing that is performed on the trimmed accelerometer signal is also performed on the noise profile as well, so that the noise profile can accurately reflect any transformation the main signal goes through, enhancing the accuracy of noise reduction.

Following interaction detection, the DC offset in the accelerometer data is removed. This offset represents a constant shift that causes the signal to be displaced vertically from the baseline. In the ADXL345 accelerometer, it is known that a slight DC offset might arise due to the power supply, mounting position, or sensor calibration, especially when collecting data in dynamic conditions. Removing the DC offset centers the data around zero, allowing for the processing to focus on the true variations in the signal, since the amplitude related features are particularly sensitive to baseline shifts. The offset is removed in each axis separately, by calculating the median value of the signal and subtracting it from each data point. The median was preferred compared to the mean value that is also often used, due to its lower sensitivity to outlier and noise, especially in a signal where occasional irregularities are present.

The next processing step is notch filtering, applied to remove specific frequencies that are associated with persistent noise. Notch filtering is eliminates narrowband interference, such as electrical noise from power sources, often at 50 or 60 Hz. Accelerometers can pick up various mechanical and electronic noise frequencies and the notch filter is designed to specifically target and suppress these without affecting the rest of the signal. In this project, the dominant frequencies in the noise profile were calculated and the filter is fine tuned based on that frequency for each axis, preserving the essential parts of the signal while suppressing power line interference and other consistent, narrow-band noise sources. The quality factor (Q) of the filter was set to 30, a high value so the filter focuses on eliminating narrow-band noise while preserving the surrounding frequencies. Aslo the filter was implemented using the filtfilt function, since it achieves zero phase filtering, by running the filter forward and then backward over the signal, which is important in this project since phase distortion could change the timing of the waveform components affecting the integrity of the signal.

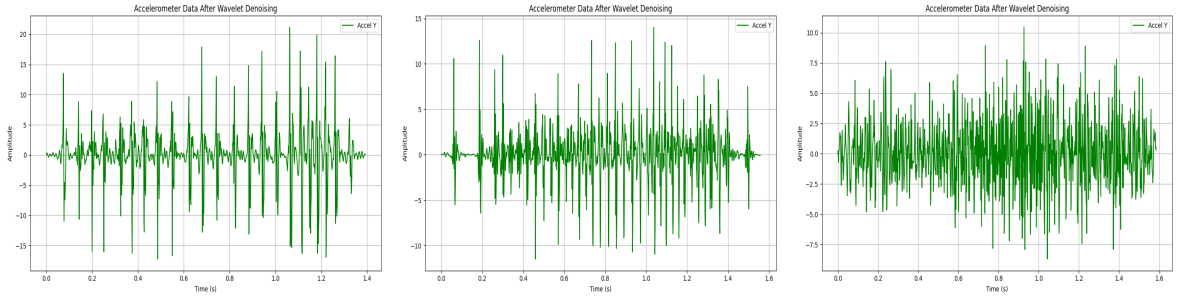
Moving further on the processing pipeline, a second order Butterworth bandpass filter is applied to each one of the signal's axis as well as the noise profile, to isolate the frequency range relevant to the accelerometer signal while removing low frequency noise and high frequency artifacts. The Butterworth filter was chosen for its flat frequency response within the passband, preserving signal characteristics without introducing ripples. The filter range selected is 30 to 350 Hz, chosen based on studies indicating that

mechanoreceptors are highly sensitive to vibration frequencies in this range as discussed in Chapter 2 and common practices in tactile perception and accelerometer analysis. By focusing on this frequency band, the later analysis and feature extraction process will be focused on only the perceptually significant components of the signal. The use of a second order filter ensures a moderate roll-off rate of 40dB per decade, which is sufficient for attenuating frequencies outside the desired band, while avoiding phase distortions that could affect the temporal alignment of the signal's features. Again, the filtfilt function is used for ensuring zero phase shift.

The final step in the accelerometer preprocessing is denoising the signal using wavelet transform techniques and utilizing the noise profile to guide the thresholds. This method separates the signal from noise by decomposing it into different frequency components. First, the signal length is calculated and an array is initialized to store the denoised data. Then the optimal decomposition level for wavelet analysis is determined, with a maximum of six levels to balance detail preservation and computational efficiency avoiding decomposing the signal into components that may not contribute meaningfully. Again, each axis of the accelerometer is processed separately, and for each axis the noise level is estimated by decomposing the noise profile and finding a noise estimate using median absolute deviation (MAD), setting a baseline to what should be considered as noise. The signal is then decomposed to wavelet coefficients and a calculated threshold is applied to each set of the coefficients to remove noise. The threshold is computed dynamically for each level of wavelet coefficients and is based on the estimated noise level in the signal. The noise estimate is used as a baseline and is scaled according to the amount of data in each wavelet decomposition level by a scaling factor, ensuring that it adapts to variations across the signal. Also a predefined threshold multiplier is applied to fine tune the threshold and further balance noise reduction and signal preservation. A multiplier less than 1 reduces the risk of over smoothing and risking loosing valuable nuances of the signal. The multiplier was selected after testing and evaluating the results with different values, by computing signal to noise ratio before and after the denoising, resulting in 0.8. The Daubechies 4 (db4) wavelet was chosen based on its ability to capture both smooth and transient signal features allowing to clean the signal without blurring or distorting important features that represent texture information.

For each filtering step, signal to noise ratio (SNR) was calculated to confirm that noise reduction was achieved without over smoothing or losing valuable details. In the final step, a series of checks is conducted to evaluate the effectiveness of the preprocessing on the signals. First, the mean squared error (MSE) is calculated between the filtered and denoised signals to quantify the difference between the signals. A low MSE value suggests that the denoising has preserved the structure of the signal and important features remain intact. Following this, cross-correlation between the filtered and denoised signals along the Y-axis is performed to check for any phase shifts. The resulting lag array represents the possible time shifts between the two signals and by finding the lag corresponding to the maximum correlation, the sample-level phase shift is determined. Finally, a power spectral density (PSD) plot of the denoised Y-axis data is created to confirm that noise

has been reduced while important frequency details of the signal are kept. Altogether, these checks confirm that the denoising steps preserved the signal's timing, structure, and content, preparing it accurately for further analysis.

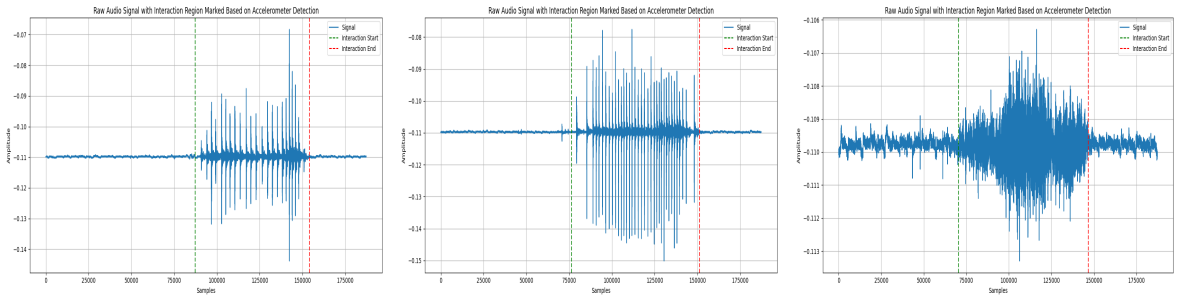


(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.4: Denoised Accelerometer Signals on Y axis.

4.2.3.2 Audio Signal Preprocessing

After the preprocessing of the accelerometer signal is done, the process_accel function returns to the main function the cropped and processed accelerometer signal, the corresponding timestamps and the start and stop indices mapped to the audio sample rate. Then, the audio preprocessing begins with the process_audio function taking as arguments the audio signal and its timestamps, the audio sample rate and the loaded noise profile as well as the start and stop indices. The first thing in the preprocessing function is to trim the audio and its timestamps based on the interaction detected in the accelerometer data, to continue with the processing of the region of interest alone.



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.5: Interaction Detection in Audio Signals Based on Accelerometer RMS Detection.

Again, all the processing steps that are applied to the audio signal are also applied

to the noise profile, and after every step of the process the SNR is computed to keep track of the progress and make sure the processing works as intended. Cross corelation is also performed after filtering steps to ensure that there are no phase shifts introduced. A similar pipeline to the accelerometer processing is followed here as well, with DC offset removal, Notch filtering for dominant frequency suppression, bandpass filtering and two different denoising techniques.

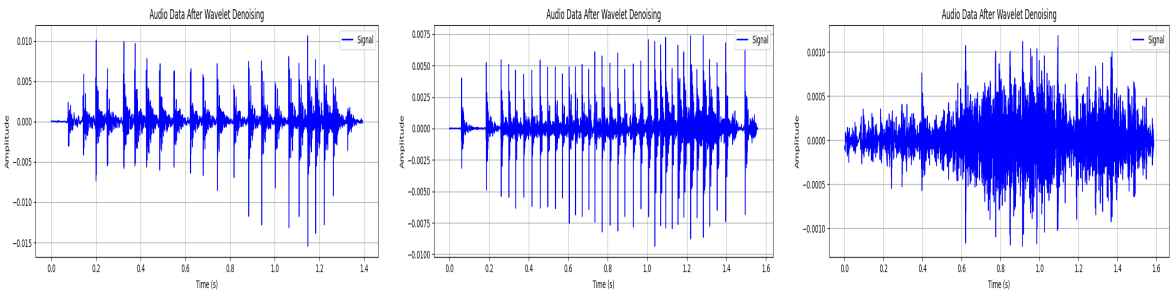
As with the accelerometer data, DC offset removal centers the audio signal around zero, preparing it for further filtering and analysis. This offset is calculated and removed using the median, minimizing the risk of distortion from any outliers. Since audio signals are particularly sensitive to amplitude shifts, centering the baseline is essential for preserving the accuracy of frequency-based features.

Then, Notch filtering is applied to remove specific narrow-band noise frequencies from environmental or electronic interference. First, power spectral density (PSD) analysis is done to the noise profile to reveal the distribution of power across frequencies, identifying the dominant frequency. This dominant frequency often correlates with consistent noise sources, such as electrical interference from power at 50 or 60 Hz. Once this dominant frequency is identified, the notch filter centered on this frequency is applied to attenuate it in both the audio signal and the noise profile. The quality factor of the filter is the same as explained in the accelerometer implementation, ensuring that only the targeted noise frequency is removed while preserving the surrounding frequency content and the filtfilt function used ensures zero-phase distortion, which is important for maintaining the original timing of audio elements. By applying notch filtering based on the noise profile's dominant frequency, the main components of the audio signal remain intact while electrical interference is minimized.

Next, a fifth-order Butterworth bandpass filter is applied to the audio signal to keep only the frequencies relevant to the recorded texture interactions, in the range between 30 and 1000 Hz while filtering out low-frequency noise and high-frequency artifacts. The Butterworth choice aligns with the reasoning in the accelerometer section, chosen for its flat frequency response within the passband. The broader frequency range is selected because audio signals can contain higher frequency components, in comparison to the accelerometer data, that are significant for capturing the nuances of texture related sounds. By including frequencies up to 1000 Hz, the filter ensures that subtle auditory details which may contribute to texture perception are preserved. Also this range falls within the broader tactile perceptible range which is up to 1000 Hz based on research discussed in Chapter 2. The filter order is set to 5, higher than the second-order filter used for the accelerometer data. A higher-order filter provides a steeper roll-off, effectively attenuating frequencies outside the desired band while maintaining the integrity of the passband. This choice ensures that unwanted low and high-frequency noise is minimized, allowing for a clearer representation of the texture related audio components. Using a higher order filter however can introduce phase distortion, which is managed by employing the filtfilt function to ensure zero-phase shift, preserving the temporal alignment and clarity of the audio signal without distortion.

The next step is denoising the signal using a Wiener filter, which is particularly effective for reducing noise in signals with variable noise levels, as it adapts its filtering strength according to the local variance of the signal. For each sample, the Wiener filter considers both the signal's local characteristics and an estimate of the noise power from the noise profile captured at the beginning of the recording session. By setting the noise_power parameter to 1.5 times the mean power of the noise profile, the filter is calibrated to the specific noise conditions of the recording environment. This approach balances noise reduction with preservation of audio detail, as the Wiener filter selectively attenuates the noise while maintaining the important variations in the signal that contribute to texture representation. Unlike simpler smoothing filters, which may blur or lose fine details, the Wiener filter's adaptive nature allows it to respond dynamically to different sections of the audio. This makes it a suitable choice for this project, where it is essential to retain texture-specific audio features while minimizing background noise interference.

In the final denoising step for the audio signal, wavelet-based denoising is implemented using SURE Shrink thresholding. This method applies the wavelet transform to decompose the signal and the noise profile into multiple levels of detail, applying adaptive noise suppression across various frequency bands. To optimize for both noise reduction and feature preservation, the thresholding uses SURE Shrink, which dynamically adjusts the noise threshold at each decomposition level based on the characteristics of the noise profile. This adaptive thresholding ensures that both quieter and signals with more intense features are denoised effectively. A threshold multiplier is also applied to the threshold to avoid over smoothing, which could lead to loss of texture related details, that was chosen based on testing results for different values accross different recordings and evaluating the effectiveness of increasing SNR while avoiding over smoothing. Additionally, soft thresholding is chosen to maintain smooth transitions in the signal. This approach, together with the noise profile information, enhance the SNR without compromising the integrity of the signal, preparing the audio signal for meaningful feature extraction. Again as in the accelerometer implementation, Daubechies 4 (db4) wavelet was selected to capture both smooth and transient features of the audio.



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.6: Denoised Audio Signals.

In this project, both denoising methods are applied to the audio signal because audio data often has more complex noise characteristics than accelerometer data. The recording environment can introduce subtle but variable background noise, which the Wiener filter addresses effectively by adapting to these variations across the signal. This layer of adaptive noise reduction is less critical for the accelerometer data, which generally contains more consistent, narrow-band noise, often caused by sensor-specific factors rather than environmental variability. However, for the audio signal using both Wiener and wavelet denoising helps achieve a cleaner, more refined signal, necessary for accurately capturing the nuances needed for texture representation.

4.2.3.3 Final Processing and Saving

After trimming and processing both audio and accelerometer signals separately, the dedicated functions return the processed signals along with their timestamps back to the main function to apply the final processing steps and to save the signals as WAV and CSV files respectively for feature extraction and haptic mapping.

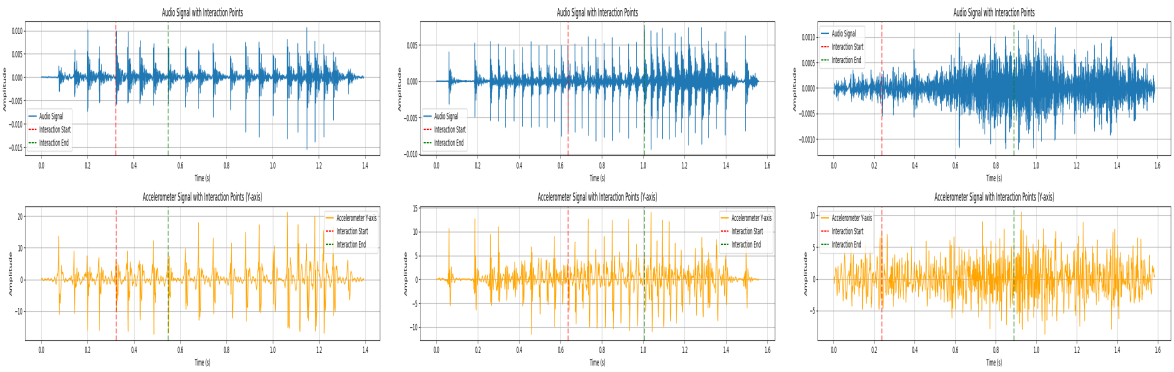
Although the signals were initially trimmed to the interaction region before processing, they undergo a final detection and trimming step after preprocessing to focus on the most relevant and consistent portion of the interaction region. Since the interactions happened with no constraints, while trying to maintain consistent speed and pressure through each texture recording, some inconsistencies naturally occurred. To ensure smoother feedback, a custom adaptive peak detection algorithm was developed to identify regions within the signal that show consistent peaks, accounting for speed variations.

The algorithm begins by dynamically adjusting detection parameters based on the signal type, whether its accelerometer or audio, to account for differences in their characteristics. For each signal, an adaptive peak detection process is applied to identify distinct peaks. This is achieved by calculating a dynamic height threshold based on a specified prominence percentile of the signal's amplitude distribution, as well as an adaptive distance threshold relative to the signal's sample rate. Then the median amplitude of the original signal is used to differentiate and keep only the detected peaks that are most relevant. For accelerometer data, stricter detection thresholds are applied, since accelerometer signals typically exhibit sharp high amplitude changes due to direct contact with the surface. This leads to more frequent peaks in the signal, so a lower prominence percentile and closer peak spacing ensure that high frequency peaks are included. For audio signals, the criteria are more flexible, since the audio signals generally are less sharp and their amplitudes vary more gradually.

After detecting peaks, the algorithm calculates distances between consecutive peaks to determine their regularity, as regular spacing between peaks can be translated to a consistent interaction patttern. The median distance between peaks and the interquartile range (IQR) of these distances are used to compute a density ratio, which gives information about the distribution of the peaks. For more dense peak distributions, the algorithm

increases tolerance and minimum cluster size, allowing more flexibility in grouping peaks that are in close range and capturing a broader interaction region, including minor irregularities in spacing whereas a stricter tolerance with smaller clusters is used for sparse peaks, helping avoid noise or outliers. Using the calculated tolerance bounds, peaks are grouped into clusters based on consistent spacing. Only clusters with sufficient size are kept, as they indicate regions that interaction patterns are consistent and reliable and the longest cluster is chosen as the final region of interest.

The start and stop indices of this region are determined separately for the accelerometer and the audio signal, then they are both converted to time values based on each signal's sample rate and a common region is finally determined by taking the maximum of the start times and the minimum of the end times. Both signals and their timestamps are then trimmed to these common times and the signals.



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.7: Final Interaction Region in both Signals using Adaptive Peak Detection.

Before saving the processed signals, the base velocity during the interaction is calculated from the acceleration data on the Y axis, which is the primary interaction axis, to be used later in the representation stage of the project. Starting with the accelerometer's Y-axis data in units of m/s^2 , the trapezoidal rule is applied to estimate the cumulative acceleration over time, resulting in velocity in m/s. The timestamps of the signal are used to calculate accurate time differences between consecutive samples. By summing these values over time, a velocity profile is created across the interaction region, which is then averaged to give the base velocity. This average reflects the overall interaction speed, providing a foundation for representing the recorded signals in the haptic feedback texture representation.

Finally, the signals are saved to a specified folder in the PC desktop for the next step of processing. The audio signal is saved as a WAV file using the soundfile library and the accelerometer data is saved as a CSV file with three columns, one for each axis, using pandas library.

4.3 Feature Extraction and Mapping

Signal preprocessing, analyzed in the previous section, is critical in ensuring the success of the feature extraction and mapping process. This ensures that the segmented signals represent the most consistent and relevant portions of the interaction, enabling the extraction of features that reflect the physical properties of the textures.

The purpose of feature extraction and mapping in this project is to translate the characteristics of recorded textures into tactile feedback that can be perceived through vibrations generated by a Linear Resonant Actuator (LRA). By analyzing the processed audio and accelerometer signals, meaningful features are identified that capture the unique qualities of texture interactions. These features are then normalized and mapped to the LRA's frequency and amplitude parameters, creating a representation of the recorded textures.

4.3.1 Signal Loading and Concatenation Process

The first step in the feature extraction algorithm is the loading of the preprocessed signals. All the recordings are loaded at once so that they can be interpreted in parallel in some steps of the algorithm to maintain consistency. Audio signals are loaded from WAV files using the librosa library and are stored as numerical arrays. For accelerometer data, CSV files are read using pandas library and only the Y-axis is extracted. This was chosen for simplicity as this is the primary interaction axis. The loading process is shown in the code below.

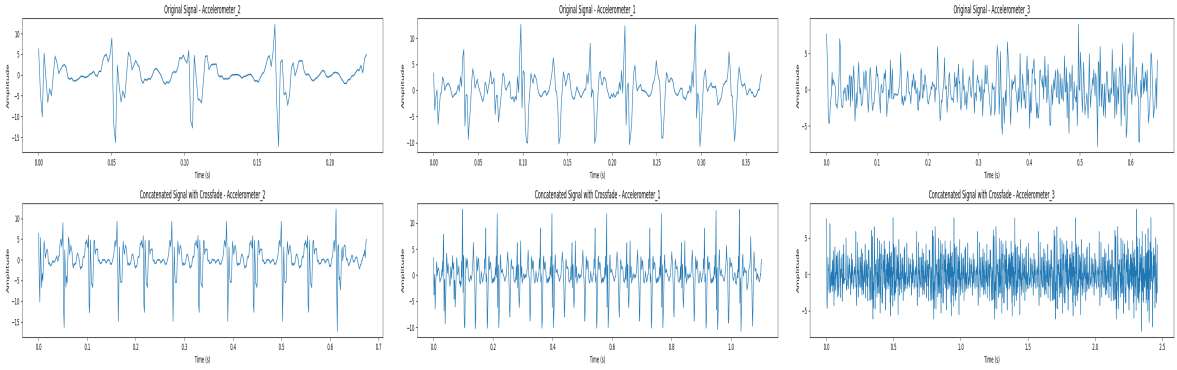
```
audio_signals = [librosa.load(audio_path, sr=audio_sample_rate)[0] for audio_path in
↪  all_audio_paths]
accel_signals = [pd.read_csv(accel_path)['Accel_Y'].values for accel_path in
↪  all_accel_paths]
```

Since the signals were cropped in the preprocessing stage to keep only the most consistent parts within the signals, the next step involves concatenation of each audio and accelerometer signal pair of each recording, to create loopable versions for continuous tactile feedback. Each signal is repeated multiple times determined by a predefined loop count variable, to extend the signal duration and crossfade is applied at the boundaries to ensure seamless transitions between loops. A cosine ramp is used for the crossfade which was selected after testing different methods and evaluating the results visually by plotting the signals and perceptually by driving the Linear Resonant Actuator.

```
loop_count = 5 #Number of times to repeat the signals
crossfade_duration_audio_ms = 50  #Crossfade duration for audio in ms
crossfade_duration_accel_ms = 200 #Crossfade duration for accelerometer in ms
concatenated_audio_signals = []
concatenated_accel_signals = []
```

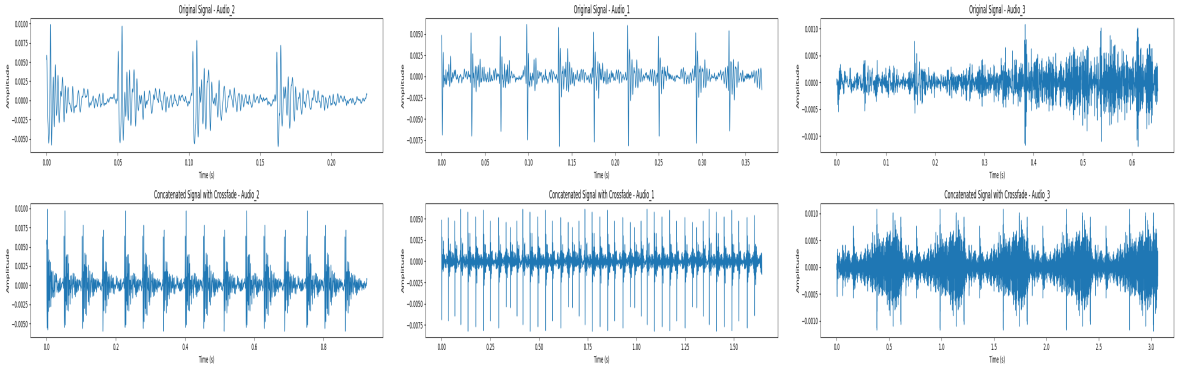
```
for idx, (audio_signal, accel_signal) in enumerate(zip(audio_signals,
↪  accel_signals)):
    concatenated_audio, concatenated_accel, crossfade_length_audio,
    ↪  crossfade_length_accel = process_audio_and_accel(
        audio_signal,
        accel_signal,
        loop_count,
        crossfade_duration_audio_ms,
        crossfade_duration_accel_ms)
```

The crossfade duration is dynamically calculated based on the signal's sample rate and the desired overlap period. The end of one loop is merged with the beginning of the next ensuring that the transitions are imperceptible, while focusing in avoiding the alter of the characteristics of the raw signals. The crossfade and looping was developed through a lot of testing to achieve a balance between smooth transitions and preservation of signal characteristics since the project focuses on representing the recorded texture interactions authentically. To achieve this the crossfade lengths and blending method were fine tuned. To validate the effectiveness of this process both visual and numerical checks were performed, including comparative visualizations of the original and concatenated signals as well as discontinuity checks by calculating the maximum value of the absolute difference between the overlapping regions of the loops indicating how smooth or abrupt the transitions are. In the following figures the original and concatenated signals are shown for all three textures.



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.8: The original and concatenated accelerometer signals for all three textures



(a) Comb with 2mm spacing. (b) Comb with 1mm spacing. (c) Sandpaper.

Figure 4.9: The original and concatenated audio signals for all three textures

4.3.2 Segmentation Logic

The signals were segmented into smaller overlapping windows during the feature extraction process. Each segment captures a small portion of the signal over a short time interval, to extract temporal and spectral characteristics as they evolve throughout the interaction. A 20 millisecond window size was selected with a 50% overlap between consecutive segments, aligning with standard practices in signal processing for feature extraction, balancing temporal and frequency resolution. The 50% overlap doubles the number of segments, enhancing temporal resolution and helping to capture fine changes and temporal variations in the audio and accelerometer data without discontinuities. Also, the overlap reduces edge effects that could distort feature extraction ensuring smoother transitions between segments. For the audio signals captured at 48 kHz, the 20 ms correspond to 960 samples per segment. This duration gives a frequency resolution of 50 Hz (1/0.02 seconds) which is short enough for capturing the temporal details in the signal corresponding to texture interactions and long enough for analyzing the high frequency components of the audio signal. For accelerometer data that were sampled at 800 Hz, a 20 ms window has 16 samples, capturing low-frequency content up to the Nyquist frequency which is 400 Hz in this case, which is sufficient for analyzing the vibrations of texture interactions that typically fall below 500 Hz. The temporal resolution of 20 ms aligns with the human tactile thresholds in vibration changes that are 10-20 ms, resulting in perceptually meaningful feature extraction. A Hanning window was applied to each segment during this process to further reduce spectral leakage and smooth out abrupt transitions at the segment boundaries. This step is crucial to ensure that the extracted features are representative of the actual signal and not get affected by edge effects and discontinuities. The choice to use the same window size and overlap for both signal types was made to ensure consistent feature extraction and also to simplify the synchronization of the signals in later processing steps as well as to maintain coherence between the signals' features in the mapping to tactile feedback process. The implementation of the

above logic is shown in the code snippet below.

```
def segment_signal(signal, sample_rate, window_size_ms=20, overlap=0.5):
    window_size = int(window_size_ms * sample_rate / 1000) #Convert ms to samples
    step_size = int(window_size * (1 - overlap)) #Step size between windows
    segments = []
    for start in range(0, len(signal) - window_size + 1, step_size):
        segment = signal[start:start + window_size]
        if len(segment) > 0:
            hanning_window = np.hanning(len(segment))
            segment = segment * hanning_window
            segments.append(segment)
    return segments
```

4.3.3 Global Feature Ranges

Once the audio and accelerometer signals are preprocessed and concatenated, the next step in the algorithm involves determining the global feature ranges for each feature by scanning through all recordings. These ranges serve as a reference for normalizing extracted features across all signals, ensuring consistency in feature mapping to the LRA's frequency and amplitude parameters.

The global feature ranges are calculated for every feature of interest from both audio and accelerometer signals, defining the minimum and maximum values of each feature across all recordings. A dictionary is created to store these values for each feature and the features are categorized int audio features and accelerometer features. Each signal is segmented into overlapping windows small enough to capture the temporal details of the signal but still provide meaningful feature extraction as explained in the previous paragraph.

For each segment, features are extracted using the corresponding feature extraction functions and the values are separately calculated for each type of signal. The minimum and maximum values are updated dynamically as segments are processed and if a new feature falls outside the current range, the range is extended to include it. After processing all segments from all signals, the final global ranges are saved into the dictionary for use in the next steps. These processes are shown in the implementation code below.

```
for audio_data, accel_data in zip(all_audio_data, all_accel_data):
    audio_segments = segment_signal(audio_data, audio_sample_rate, window_size_ms,
    ↪  overlap)
    accel_segments_y = segment_signal(accel_data, accel_sample_rate, window_size_ms,
    ↪  overlap)
    for audio_segment in audio_segments:
        audio_features = extract_audio_features_per_segment(audio_segment,
        ↪  audio_sample_rate)
        for key in audio_features.keys():
```

```
            feature_value = audio_features[key]
            global_feature_ranges[f'{key}_audio'][0] =
            ↪  min(global_feature_ranges[f'{key}_audio'][0], feature_value)
            global_feature_ranges[f'{key}_audio'][1] =
            ↪  max(global_feature_ranges[f'{key}_audio'][1], feature_value)
    for accel_segment_y in accel_segments_y:
        accel_features_y = extract_accel_features_per_segment(accel_segment_y,
        ↪  accel_sample_rate)
        for key in accel_features_y.keys():
            feature_value = accel_features_y[key]
            global_feature_ranges[f'{key}_accel'][0] =
            ↪  min(global_feature_ranges[f'{key}_accel'][0], feature_value)
            global_feature_ranges[f'{key}_accel'][1] =
            ↪  max(global_feature_ranges[f'{key}_accel'][1], feature_value)
return global_feature_ranges
```

The computation of global feature range was chosen in this project to ensure consistency in the representation of textures by establishing a shared reference for normalization across all recordings. This allows the features to be compared and interpreted uniformly, preserving the unique characteristics of each texture. Without global ranges features could be influenced by the individual recording differences and variability, resulting in imbalance i the tactile feedback.

4.3.4 Feature Extraction

In the feature extraction process details of the recorded textures are captured in order to represent them as accurately as possible. By analyzing the processed signals, features that represent the frequency, temporal, and energy characteristics of the textures, are extracted for creating a meaningful tactile representation. The feature selection in this thesis was based on the identification of audio and accelerometer characteristics respectively that best represent textures captured during interaction, for realistic translation into tactile feedback through vibrations. While literature provides some insights into accelerometer features relevant to texture interaction and tactile feedback, there are no explicit references of audio features for direct texture representation through haptic feedback. In this thesis, features from both signal types were chosen based on their theoretical relevance to capturing variations produced by texture interactions and their potential to map either to frequency or the amplitude of vibrations in haptic feedback for accurate representation of the recorded textures. A wide range of temporal, spectral, and perceptual features were extracted to evaluate their effectiveness in capturing and representing the complex properties of the recorded textures. This way features with different characteristics could be tested and analyzed, for selecting the most meaningful mappings to vibration frequency and amplitude and result to the most realistic representation.

4.3.4.1 Audio Feature Selection

When interacting with a textured surface, different textures generate unique sounds based on how the surface features interact with the finger. Rough textures produce low frequency and high amplitude sound waves due to friction and resistance while smooth textures produce high frequency, more subtle sounds often dominated by sliding noises with less variation in amplitude. The microphone captures these interactions as sound waves ranging from low to high frequencies depending on the texture. For example sliding a finger across a coarse wood might result in a series of low frequency, harsh sounds reflecting the rough texture. Sliding a finger across a smooth plastic will probably generate higher pitched and continuous sounds reflecting less physical interaction and more friction induces vibrations. The frequency range of the audio signal is very important in determining which aspects of the texture are captured. Low frequencies (up to 500 Hz) are typically where most of the tactile texture related information lays, representing physical vibrations such as the feel of bumps, ridges and surface variations. Mid to high frequencies (500-2000 Hz) can represent finer surface details such as friction or small surface imperfections. Very high frequencies above 2 kHz are generally less relevant for texture representation with tactile feedback, but may represent friction or squeaky noises. The extraction of audio features focused on capturing different properties of the recorded signals. The goal was to identify features that could reflect the uniqueness of interactions with different textures, and be directly mapped to frequency or amplitude parameters of the vibrations, leading to realistic texture representation.

Temporal Features

Temporal features are derived from the time-domain representation of the audio signal and were selected for their ability to capture energy, intensity, and variation over time. By analyzing these features, we can directly extract information relevant to how the texture interaction influences the signal amplitude, which is critical for creating realistic haptic feedback. Below is an explanation of the selected temporal features, their relevance to texture representation, and why they were chosen for this project.

- **Root Mean Square (RMS)**: RMS measures the energy or power of the signal within a specific time window, giving the intensity or energy of the interaction. It is calculated by squaring the signal values within that window, finding their mean, and then taking the square root of that mean. Coarser or harder textures generate stronger vibrations, leading to higher RMS values. Conversely, smoother or softer textures result in lower RMS values. RMS was chosen as a candidate for mapping to vibration amplitude in haptic feedback because of its direct correlation with signal intensity. In haptic feedback, stronger RMS values correspond to stronger vibrations, enabling the representation of coarser or more forceful interactions.
- **Short-Time Energy (STE)**: STE is a temporal feature that gives information about the energy of a signal over smaller, overlapping frames within a segment. For

this project, STE was calculated using 10 ms frames with a 50% overlap. This closer and more detailed analysis captures rapid changes in the energy of the signal, such as varying pressure or sudden impacts. The STE complements the RMS by focusing on energy variations across time rather than the average power, making this feature effective in identifying micro-patterns in the signal caused by fine texture details and can be mapped to vibration amplitude.

- **Zero-Crossing Rate (ZCR)**: ZCR calculates the number of times a signal crosses the zero amplitude axis, giving information about the signal's roughness or irregularity. Higher ZCR values typically correspond to interactions with coarse or irregular textures, with rapid amplitude changes. This feature was evaluated for mapping to vibration frequency, as textures with higher ZCR values correspond to faster and higher frequency vibrations that represent the texture's coarseness. This means that this feature could potentially help to recreate the sensation of roughness to the user through vibrotactile feedback.

- **Envelope Mean**: The envelope mean of a signal represents its overall amplitude shape, providing an outline of its variations over time. It calculates the average magnitude of these variations in the signal, capturing the overall dynamics of the interaction. This feature describes the structure and temporal evolution of the signal and was evaluated for amplitude mapping. For instance, envelope mean can represent gradual changes in force or contact pressure during texture interaction, which may not be evident in RMS alone.

- **Peak-to-RMS Ratio**: The peak-to-RMS ratio compares the maximum amplitude of the signal to its average energy (RMS), emphasizing transient spikes or sharp features in the interaction. This feature captures textures with irregularities, such as bumps or ridges, with higher ratios describing surfaces that are more sharp or uneven. By mapping this feature to vibration amplitude, the haptic feedback can replicate the sensation of these tactile irregularities.

Spectral Features

Spectral features describe the frequency-domain characteristics of the audio signal, offering a detailed view of its tonal, harmonic, and noise-like properties. These features provide insights about the distribution of energy across different frequencies, capturing the auditory properties of the textures and were tested for mapping to both vibration frequency and amplitude. For example, sharper textures may produce higher frequency components, while smoother ones might produce lower frequencies.

- **Spectral Centroid**: The spectral centroid is described as the "center of mass" of the signal's frequency spectrum, representing the average frequency weighted by amplitude, indicating the frequency region where most of the energy is concentrated.

It indicated the brightness of the sound correlating to texture smoothness. A higher spectral centroid can be produced by interacting with sharper or finer surfaces. This feature was chosen as a candidate for frequency mapping.

- **Spectral Bandwidth**: Spectral bandwidth measures the range of frequencies present in the signal, describing its complexity and is linked to timbre's spread. A wider bandwidth reflects a rich more complex texture, such as a combination of coarse and fine surface. For this project, spectral bandwidth was tested for mapping to both frequency and amplitude, as wider bandwidths can reflect higher frequency vibrations and stronger tactile sensations.

- **Spectral Rolloff**: Spectral rolloff defines the frequency below which a majority (typically 85%) of the signal's spectral energy is concentrated. Lower rolloff values indicate dominance of low frequency components. Texture with high rolloff values tend to be finer, with higher frequency content while lower values suggest a rougher surface where most energy is concentrated in lower frequencies. Spectral rolloff was explored for mapping to vibration frequency to convey the smoothness of the interaction.

- **Spectral Flatness**: Spectral flatness measures how noise-like a signal is by comparing its geometric mean to its arithmetic mean. Higher values indicate a flatter, more uniform spectrum, found in signals with less tonal structure and more noise like qualities. Textures with irregular patterns often produce signals with high spectral flatness. This feature was chosen for mapping to vibration amplitude to replicate the tactile randomness.

- **Spectral Contrast**: Spectral contrast captures the difference between spectral peaks and valleys, highlighting the texture's harmonic and repetitive characteristics. This feature is particularly relevant for textures with clear patterns or grooves, such as comb ridges. Spectral contrast reveals fine details of the signal's structure and was evaluated for mapping to vibration frequency.

- **Spectral Flux**: This feature captures the rate of change in spectral content over time, reflecting the dynamic variability of the interaction. Higher flux values indicate rapid frequency shifts, which are characteristic of rough or rapidly changing surfaces. For this project, spectral flux was tested for frequency mapping as it allows the haptic feedback system to dynamically respond to the signal's spectral changes.

- **Energy Entropy**: Energy entropy quantifies the unpredictability or randomness in the signal's energy distribution, giving information about the complexity of the texture. Higher entropy values indicate more intricate and less predictable texture, as for example a surface with irregular grooves. This feature was tested for amplitude mapping to reflect the complexity and richness of the textures.

Perceptual Features

Perceptual features are inspired by the way humans process sound, making them especially relevant for aligning the tactile feedback with human sensory perception. Perceptual features focus on qualities like pitch, timbre, and the intensity of sudden changes, connecting the physical properties of a signal with how they are experienced through sound and touch. In this project, these features were explored to examine if the qualities that define how people perceive sound could also enhance the sense of touch, making the haptic feedback feel more natural and realistic.

- **Mel-Frequency Cepstral Coefficients (MFCCs)**: Mel-Frequency Cepstral Coefficients (MFCCs) are features derived from the audio signal's frequency spectrum, designed to capture perceptually significant characteristics of sound. MFCCs are based on the mel scale, a frequency scale designed to reflect the way humans naturally perceive sound. This scale prioritizes frequency bands that match hearing sensitivity, especially emphasizing lower frequencies where perception is most precise. For the calculation of MFCCs first the signal is divided into short, overlapping windows, followed by applying a Fourier Transform to convert the signal into its frequency domain. The resulting spectrum is then filtered through a mel-scaled filterbank, compressing the data into perceptually relevant bands. Logarithmic scaling is applied to reflect the human ear's logarithmic sensitivity to sound intensity and a Discrete Cosine Transform (DCT) is performed to produce the MFCCs, emphasizing the most significant components of the spectral shape. In the context of texture interaction, MFCCs provide insights into the tonal characteristics and broad spectral shape of the audio signal. These coefficients capture variations in sound produced by interactions with different textures, such as coarse, fine, or irregular surfaces. For this project, the mean and variance of the MFCCs were extracted. The mean highlights the steady tonal attributes, while the variance reflects dynamic changes during interaction. MFCCs were tested as candidates for both frequency and amplitude mapping in vibrotactile feedback. MFCCs are widely used in texture classification tasks providing very good results.

- **Onset strength**: Onset strength measures the intensity of energy increases in the signal, capturing sudden impacts or transitions during an interaction. This feature captures the details of sharp or transient textures, such as ridged or patterned surface. Mapping onset strength to vibration amplitude can provide the sensation of sudden changes, providing the tactile feedback of sharpness.

- **Chroma Mean**: The chroma mean captures the harmonic and tonal characteristics of the signal by analyzing its tonal components across distinct frequency bins. While typically used for musical analysis, the chroma mean was included to explore its potential for capturing harmonic or tonal qualities in texture interactions. This feature was evaluated for mapping to vibration frequency, where tonal variations in the signal could be represented through pitch changes in the haptic feedback.

Although MFCC and Chroma are established in speech and music analysis, their suitability for texture representation is exploratory in this context. Including these features allowed investigation of whether their perceptual sensitivity to subtle spectral variations can enhance the realism of the haptic feedback.

4.3.4.2 Accelerometer Feature Selection

The accelerometer signals capture the mechanical vibrations that occur during interactions with textured surfaces. Different surface interactions generate vibrations at different frequencies. Lower frequency bands (up to 200 Hz) are dominated by rougher textures with larger surface features. Mid frequency bands (200-500 Hz) are associated with finer textures and smaller surface details due to fine frictional effects. Frequencies above 500 Hz typically represent frictional noise or sensor noise but may also reflect very fine textures if small grains or micro textures are present. The accelerometer feature extraction process focuses on capturing the characteristics of the recorded signals that reflect the dynamic interactions between the fingertip and the textured surfaces. By analyzing the processed signals, a set of features is extracted to represent the temporal, frequency, and statistical properties of the vibrations, providing insights into the intensity, variability, and distribution of the recorded accelerometer data. The selection of features was guided by common practices in the field as well as by their theoretical relevance to the variations produced by texture interactions and their potential to map either to the frequency or amplitude of haptic vibrations.

Temporal Features

Temporal features of the accelerometer signal capture the variations in vibration and motion over time, reflecting the dynamic interactions between the fingertip and the textured surface. These features provide information about the energy, intensity, and range of motion generated during interaction. Time domain features contribute to identifying the tactile dynamics of different textures, making them strong candidates for amplitude mappings to provide realistic texture representations through haptic feedback.

- **Root Mean Square (RMS)**: RMS represents the average energy of the accelerometer signal over a segment, calculated as the square root of the mean of the squared signal values. While the mathematical calculation is the same as for audio signals, in accelerometer data RMS represents the overall intensity of vibrations caused by texture interaction. Coarser textures or stronger interactions generate stronger vibrations, leading to higher RMS values. This feature was chosen for amplitude mapping, as RMS reflects the intensity of tactile interactions, with stronger signals generating stronger vibrations.

- **Peak-to-Peak Distance**: Peak-to-peak distance is the difference between the maximum and minimum values of the signal within a segment. It quantifies the

range of motion or vibration during interaction. Higher values indicate dynamic or irregular textures such as textures with ridges. This feature was tested for vibration amplitude mapping.

- **Envelope Mean**: The envelope mean represents the average magnitude of the accelerometer signal's amplitude variations, reflecting the overall dynamics of the interaction. Smoother textures produce more consistent envelope values, while rougher textures result in more noticeable variations. This feature was chosen for amplitude mappings as it emphasizes changes in the interaction such as varying pressure during exploration.

Frequency Features

Frequency features provide a detailed representation of the accelerometer signal's spectral content, highlighting the distribution of vibrational energy across different frequency bands during interactions with textured surfaces. These features contribute to identifying the tonal and harmonic components of the vibrations, which are important aspects in differentiating textures. By analyzing the frequency domain, these features help identify distinct vibrational patterns, such as coarse versus fine surfaces. Frequency features were evaluated for both amplitude and frequency mappings, ensuring that the haptic feedback reflects the distinct vibrational characteristics of each texture.

- **Spectral Centroid**: The spectral centroid in the accelerometer signal indicates where the bulk of the vibrational energy is concentrated in the frequency spectrum. Higher spectral centroid values correspond to interactions dominated by higher frequencies, often associated with fine or sharp textures. Conversely, lower centroid values are indicative of smoother textures, where low-frequency vibrations dominate. This feature was selected for frequency mapping, as it captures the textural sharpness or coarseness of interactions.

- **Spectral Bandwidth**: Spectral bandwidth as mentioned in the audio features, measures the range of frequencies in the signal providing a representation of its complexity, with wider values indicating more complex textures. This feature was evaluated for both frequency mapping, as it captures the richness of vibrations across different textures.

- **Power Spectral Density (PSD)**: PSD measures the power distribution of the signal across its frequency spectrum, highlighting the dominant vibrational components of the interaction. For texture representation, PSD can prove helpful because different textures exhibit unique power distributions and it shows how energy is distributed across frequency bands. Smooth textures typically concentrate power at lower frequencies, while coarse textures spread energy across higher frequencies. The mean and standard deviation of PSD provide insights into the average energy

and its variability across the spectrum. These features are useful for identifying textures with distinct vibrational patterns and were tested for both frequency and amplitude mappings to reflect the energy and vibrational patterns of the recorded textures.

- **Energy Entropy**: Energy entropy, as mentioned in the audio feature section measures the randomness or unpredictability in the distribution of vibrational energy across frequencies. High entropy values indicate more rough or uneven textures, while low entropy corresponds to more uniform textures. This feature was considered for amplitude mapping to represent the dynamic and varied nature of the textures through haptic feedback.

- **Low, Mid, and High Frequency Bands**: The low, mid, and high frequency bands divide the spectrum into distinct ranges based on the Nyquist frequency, providing an understanding of the signal's vibrational characteristics across these frequency ranges. Frequencies below 10% of Nyquist, represent low frequency vibrations associated with smooth, consistent textures. Frequencies between 10% and 40% of Nyquist, capture intermediate vibrational details typical of medium coarse textures. Frequencies above 40% of Nyquist frequency highlight high frequency components linked to fine textures or frictional noise. For example, the dominance of the low band suggests smoother textures, while activity in the high band indicates finer textures with rapid transitions. All the bands were tested for frequency mapping across the different recorded textures.

Statistical Features

Statistical features offer an overview of the accelerometer signal by analyzing its overall shape, variability, and distribution. They capture aspects such as symmetry, sharpness, and consistency, which are directly linked to the tactile properties of textures. Statistical features complement temporal and frequency analyses by providing a deeper understanding into the structure and dynamics of the signal. They were considered for both amplitude and frequency mappings to recreate the unique characteristics of each texture, such as the sharpness of ridges or the smoothness of flat surfaces.

- **Peak Count and Peak Height Mean**: Peak count is the number of significant peaks in the signal, while peak height mean represents the average magnitude of these peaks. These features capture the frequency and intensity of distinct vibrational events during texture interaction, such as ridges or bumps. Coarser textures produce higher peak counts and more pronounced peak heights, while smoother textures result in fewer, smaller peaks. These features are directly linked to the perception of surface irregularities, making them strong candidates for both frequency and amplitude mappings. Peak count aligns well with vibration frequency as frequent peaks are linked with rapid changes, while peak hight mean can be

mapped to vibration amplitude, as large peaks correspond to stronger tactile sensations.

- **Skewness**: Skewness captures the asymmetry of the signal's amplitude distribution. Positive skewness reflects sharp rises in the signal, while negative skewness reflects more downward dips. Skewness provides insight into directional tendency of vibrations, which can be linked to specific texture characteristics. This feature can be mapped to amplitude, representing the directional characteristics of the texture interaction.

- **Kurtosis**: Kurtosis reveals the presence of sharp peaks or abrupt transitions in the signal's amplitude distribution. High kurtosis values correspond to signals with sharp peaks, indicating interactions with irregular textures. Low kurtosis values indicate smoother textures. This feature was evaluated for amplitude mappings to represent the irregularities in tactile feedback.

- **Mean Value**: The mean value represents the average of the signal's amplitude, calculated by summing all values and dividing by the number of samples. When interacting with a smooth texture the mean value may remain relatively stable, while coarser textures could produce fluctuating values depending on the irregularities of the surface. This feature was tested for amplitude mapping.

- **Standard Deviation (std_dev)**: Standard deviation quantifies the variability of the signal values around their mean, giving information about the consistency of interaction. It is calculated as the square root of the average squared differences from the mean. Higher values of standard deviation indicate greater variability in the vibration signal, often corresponding to coarser textures, while smoother textures result in lower values representing more uniform vibrations. This feature was chosen for its ability to characterize the consistency or irregularity of tactile interactions. It was tested for amplitude mapping, as it can represent the distinction between smooth and dynamically varying textures.

4.3.5 Feature Normalization and Mapping

After feature extraction, the next step was to map these features to the Linear Resonant Actuator's (LRA) vibration frequency and amplitude parameters. This mapping process ensures that the extracted features effectively drive the LRA to recreate the tactile sensations corresponding to the recorded textures. The mapping involves normalization of features based on global ranges and linear scaling to align the feature values with the operational range of the LRA.

4.3.5.1 Feature Normalization

Normalization is first performed to the extracted features per segment, to ensure consistency in feature representation, by standardizing the extracted feature values across all recordings. This process leads to a uniform mapping of diverse features to the LRA, ensuring that no feature disproportionately influences the tactile feedback enabling meaningful comparison. Each feature is normalized to a range of 0 to 1 using its corresponding global minimum and maximum values, calculated during the global feature range computation step. This way the relative differences between textures are maintained preserving the unique characteristics of each texture. The formula used for normalization is:

$$\text{Normalized Feature Value} = \frac{\text{Feature Value} - \text{Feature Min}}{\text{Feature Max} - \text{Feature Min}}$$

Where feature mean and max are the global minimum and maximum values across all recordings and the feature value is the extracted feature per segment. If the feature minimum equals the feature maximum, the normalized value defaults to 0.5 to avoid division by zero.

4.3.5.2 Amplitude and Frequency Mapping

Normalized features are then scaled to the operational range of the LRA for frequency (150 to 200 Hz) and amplitude (0 to 255). Linear mapping was chosen to preserve the relationships between feature values by maintaining the relative distances between their values while transforming them into the LRA's range. This approach ensures that the normalized features are scaled intuitively with the corresponding physical sensations. Furthermore, the use of global feature ranges for normalization before mapping ensures that the entire operational range of the LRA is utilized, while avoiding exceeding the actuator's capabilities preventing distortions.The mapping formulae are:

- Frequency Mapping:

  $$\text{LRA Frequency} = \text{Freq Min} + (\text{Normalized Feature Value} \times (\text{Freq Max} - \text{Freq Min}))$$

- Amplitude Mapping:

  $$\text{LRA Amplitude} = \text{Amp Min} + (\text{Normalized Feature Value} \times (\text{Amp Max} - \text{Amp Min}))$$

The following table summarizes the mapping of each feature to the corresponding LRA parameter:

In general, features related to the signal's energy or amplitude domain (e.g., RMS, envelope mean, STE) are mapped to LRA amplitude because they directly represent intensity and force in the tactile interaction. In contrast, spectral and frequency-related features (e.g., spectral centroid, bandwidth, MFCC mean) naturally map to frequency variations in vibration, aligning with how changes in friction and surface structure translate into perceived pitch differences in tactile feedback.

Feature Name	Signal Type	Feature Type	Mapping
RMS	Audio,Accelerometer	Temporal	Amplitude
Envelope Mean	Audio, Accelerometer	Temporal	Amplitude
Energy Entropy	Audio, Accelerometer	Frequency	Amplitude
Spectral Centroid	Audio, Accelerometer	Frequency	Frequency
Spectral Bandwidth	Audio, Accelerometer	Frequency	Frequency
Short-Time Energy (STE)	Audio	Temporal	Amplitude
Zero-Crossing Rate (ZCR)	Audio	Temporal	Frequency
Peak-to-RMS-Ratio	Audio	Temporal	Amplitude
Spectral Rolloff	Audio	Frequency	Frequency
Spectral Flatness	Audio	Frequency	Amplitude
Spectral Contrast	Audio	Frequency	Frequency
Spectral Flux	Audio	Frequency	Frequency
MFCC Mean	Audio	Perceptual	Frequency
MFCC Variance	Audio	Perceptual	Amplitude
Onset Strength	Audio	Perceptual	Amplitude
Chroma Mean	Audio	Perceptual	Frequency
Peak-to-Peak Distance	Accelerometer	Temporal	Amplitude
Low Frequency Band	Accelerometer	Frequency	Frequency
Mid Frequency Band	Accelerometer	Frequency	Frequency
High Frequency Band	Accelerometer	Frequency	Frequency
Power Spectral Density (PSD)	Accelerometer	Frequency	Amplitude
Skewness	Accelerometer	Statistical	Amplitude
Kurtosis	Accelerometer	Statistical	Amplitude
Mean Value	Accelerometer	Statistical	Amplitude
Standard Deviation	Accelerometer	Statistical	Amplitude
Peak Count	Accelerometer	Statistical	Frequency
Peak Height Mean	Accelerometer	Statistical	Amplitude

Table 4.1: Summary of Features Extracted from Audio and Accelerometer Signals for Mapping to Vibration Parameters.

4.3.6 Feature Tests

The final stage of the algorithm integrates all the extracted features, to test their effectiveness in driving the LRA and reproduce the recorded textures through vibrotactile feedback. This means that for each texture, the signals were concatenated, segmented, features were extracted, normalized, and mapped to the operational range of the LRA to generate vibration patterns. Various feature combinations were defined to map to the LRA's frequency and amplitude parameters and outputs were generated for each combination, for evaluation of their effectiveness in generating meaningful and realistic haptic feedback. Three types of combinations were created:

- **Audio Features Only**: These tests exclusively use audio features for both frequency and amplitude mapping, to explore how well audio features alone can represent recorded textures through tactile feedback.
- **Accelerometer Features Only**: Correspondingly, these tests rely on accelerometer features alone for mapping, to evaluate their performance in representing the recorded mechanical vibrations produced by the texture interactions.
- **Fused Features**: Finally, a combination of audio and accelerometer features was tested, to investigate the potential synergy between the two data types in recreating realistic textures through haptic feedback. Specifically, these tests mapped one signal type to frequency and the other to amplitude.

More specifically, for each test the algorithm extracts the selected features from each segment of the preprocessed signals, normalizes the features using global minimum and maximum values and maps the normalized features to the LRA's frequency (150-200 Hz) and amplitude (0-255) ranges using linear scaling. The output of the algorithm is a CSV file for each feature combination containing the mapped time varying frequency and amplitude pairs for every segment in the recording. This file is then used as input to the LRA driver to generate vibrations. The CSV file contains two columns, one for frequency and one for amplitude, with each row corresponding to a single segment. The code dynamically names the output files based on the test configuration (frequency and amplitude features, test type), ensuring that each run creates unique results making it easier to compare different feature mappings. The code snippet below demonstrates the test implementation structure.

```
for i, (audio_signal, accel_signal) in enumerate(zip(concatenated_audio_signals,
↪  concatenated_accel_signals)):
        base_filename = os.path.splitext(os.path.basename(all_audio_paths[i]))[0]
        for feature_test in feature_tests:
            test_type = feature_test["test_type"]
            freq_from = feature_test["freq_from"]
            amp_from = feature_test["amp_from"]
            for feature_set in feature_test["features_to_test"]:
```

```
            test_name = f"{test_type}_{feature_set['frequency_feature']}_
                        {feature_set['amplitude_feature']}"
            use_audio = (freq_from == "audio" and amp_from == "audio")
            use_accel = (freq_from == "accel" and amp_from == "accel")
            fusion = freq_from != amp_from
            process_test_case(audio_signal,audio_sample_rate,accel_signal,
            accel_sample_rate,global_feature_ranges,feature_set,
            base_filename,test_name,use_audio=use_audio,use_accel=use_accel,
            fusion=fusion,freq_from=freq_from,amp_from=amp_from)
```

Additionally to the CSV output, the algorithm generates two types of visualization outputs through plots for each feature test, to have deeper insights into the mapping process. The mapped frequency and amplitude values were plotted over time for each combination, allowing an intuitive assessment of how well the selected features captured the dynamics of the recorded textures. It also helped to identify inconsistencies or outliers in the mapping process. The second visualization method were synthesized waveform plots. As explained before, each segment of the processed signal derived form audio or accelerometer data is mapped to a specific frequency and amplitude pair. Using these mappings, individual sine waves are generated for each segment, as the LRA by default produces sinusoidal vibrations. For each segment, the sine wave is calculated based on the formula below:

$$S(t) = A \cdot \sin(2\pi f t)$$

Where A is the mapped amplitude and f is the frequency. These sine waves are then combined into a single continuous waveform, which preserves the segment variations in frequency and amplitude, ensuring that the waveform accurately reflects the characteristics of the recorded textures. The plotted waveforms provide a visual representation of how the mapped features translate into tactile vibrations, showing the dynamic changes in frequency and amplitude over time.

These outputs enable detailed analysis and testing of the haptic feedback system by providing a clear representation of how the extracted features translate into vibrotactile patterns over time. The plots helped to make an initial evaluation of the results and rule out some of the combinations, based on the visual output. Additionally, by saving the results in a standard format, the system allows for further validation, comparison across feature combinations, and easy integration with the LRA control logic.

4.4 Unity Integration

After completing the stages of data capture, preprocessing, and feature extraction, the final step of the implementation brings all components together into an interactive environment. A virtual environment was developed using Unity platform with Leap Motion integrated for hand and finger tracking. Users can directly experience the recorded textures through a virtual scene, with their hand movements influencing tactile feedback generated by a Linear Resonant Actuator (LRA) driven by the ESP32-S3 microcontroller. This integration results into a wireless real-time haptic feedback experience.

4.4.1 Unity Environment

The Unity environment was created as an interactive virtual space where users could experience the recorded textures through haptic feedback. The environment includes a table within a room, with three 3D virtual textured objects that represent the real world objects that were used during the data capture.

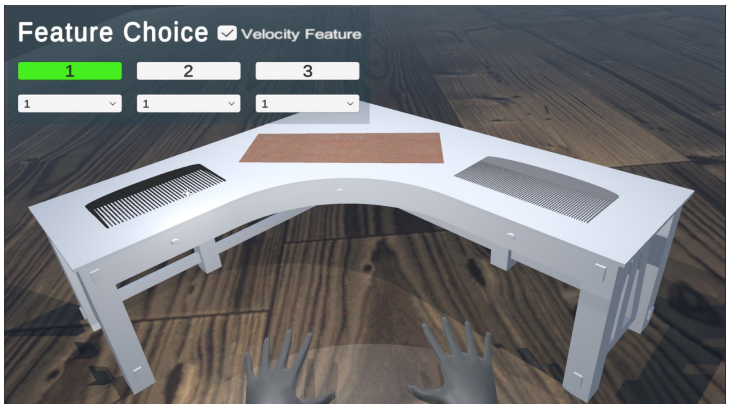
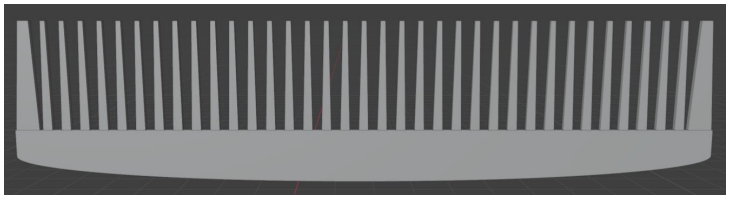
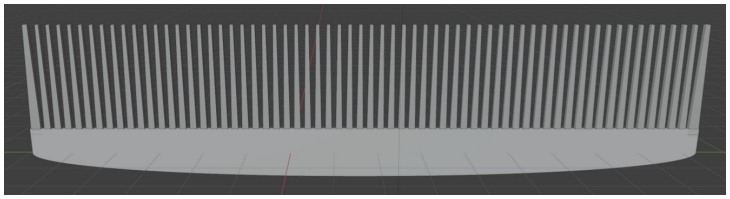


Figure 4.10: Unity Scene Overview.

The sandpaper texture was found in Sketchfab, an open source platform for 2D and 3D assets, while two custom combs with 1mm and 2mm spacing between their teeth were modeled using Blender. Blender is an open source software used for 2D and 3D modeling, texturing and rendering. The figures below demonstrate the models that were designed in Blender.



(a) Comb with 2mm spacing.



(b) Comb with 1mm spacing.

Figure 4.11: The 3D models that were designed in Blender for the two comb textures.

These models enhanced the realism of the texture representation, while making them visually distinct. These gameobjects were imported into the Unity scene and a Rigidbody and a Collider component were attached to each them. The Rigidbody enables Unity's physics system to calculate forces and movements for the objects, ensuring they respond correctly to collisions, while the Collider defines the object's physical boundaries, allowing it to detect contact with the fingertip and trigger interaction events. The Leap Motion device was integrated to provide precise hand and finger tracking, enabling users to interact naturally with the virtual objects. By pressing keys 1, 2 or 3 the camera zooms in to the textures correspondingly for a better perspective for interaction.

(a) Zoom in by pressing 1.

(b) Zoom in by pressing 2.

(c) Zoom in by pressing 3.

Figure 4.12: Close up views of the three textures in Unity.

Pressing key 4 resets the camera to the default overview position allowing users to view the entire scene. The combination of realistic visuals, intuitive interaction and haptic feedback makes this environment the central component for testing and demonstrating the project's outcomes.

For the evaluation phase, an additional feature was implemented to reduce visual bias during user testing. Pressing the A, S, or D keys moves the camera and the hands

to a simplified interaction view where three identical 3D cubes are placed in the center of the screen. Each cube corresponds to one of the textured objects (comb with 1mm spacing, comb with 2mm spacing, and sandpaper). These cubes maintain the same tactile feedback properties as the original objects but lack visual textures, creating a recognition task where users must rely solely on the haptic feedback to identify the textures. This setup allowed for a more objective evaluation of the feedback and user perception of the recorded textures without any visual influence.

4.4.2 UI for Interaction Control

The User Interface (UI) in the Unity environment was designed as a control panel for testing and evaluating the different feature extraction outputs. Each virtual object representing a texture, has an associated tag that corresponds to a specific CSV file generated from the feature extraction tests. As previously explained, the output files are based on three modes: audio only (using amplitude and frequency feature pairs from recorded audio signals), accelerometer only (using features from the recorded accelerometer signals) and fusion mode (combining features from both signal types). Hence, the UI contains three mode selection buttons labeled as 1, 2, and 3 to mask their functionality and prevent user bias. By selecting each button the system dynamically switches the tags of the textured objects to match the corresponding mode and send command to the EPS32 to select the appropriate CSV file to drive the Linear Resonant Actuator (LRA) for producing haptic feedback. This allows for testing and evaluating which signal source between audio, accelerometer or fusion provides the most realistic feedback.

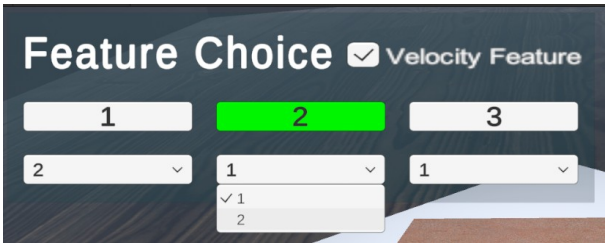


Figure 4.13: UI menu.

Additionally, the UI includes three dropdown menus, one for each of the above modes, that allow the selection of specific feature types, that were determined during the initial evaluation stage, where all feature combinations were systematically tested. These types are categorized into either **common features** that provide a decent representation universal across all textures but still specific to each mode, or **dedicated features** that were optimized individually to provide the most realistic representation for each specific texture within its corresponding mode. This design ensures that the most effective feature combinations can be tested and evaluated, allowing a direct comparison between universal performance and texture specific optimization, to determine whether texture

specific optimizations are necessary for realism. Again, these categories are labeled as 1 and 2 for simplicity, allowing the toggle between common and dedicated features without revealing which is which, ensuring objective evaluation during testing.

A toggle button is also provided in the UI to enable or disable **velocity modulation**. This button operates as a boolean switch that when active, the LRA's driving parameters are modulated based on the user's hand velocity, captured through the Leap Motion device. This modulation enhances the realism of the tactile feedback by dynamically scaling the vibration amplitude and frequency relative to the fingertip's movement speed.

All the functionalities described above are demonstrated in the Figure 4.13 which showcases the layout of the UI.

4.4.3 Leap Motion Integration

The integration of the Leap Motion Controller in the Unity project enabled the users to interact naturally with the virtual textured objects. The device was positioned on the desk in front of the user in a desktop configuration, tracking hand movements within its field of view with sub-millimeter precision, capturing data such as finger positions, velocities and orientations. This tracking capability was integrated into Unity using the Ultraleap SDK, a software that provides prefabs and scripts to handle hand tracking, collision detection, and interaction management.

The Leap Motion Service Provider, which is a core component of the Ultraleap SDK for Unity, processes raw tracking data from the device and deliver it to Unity in a format that can be used for real time hand tracking and interaction with virtual objects. In this project, the service provider was made a child of the scene's main camera, ensuring it dynamically follows the camera's movements when the user switches between perspectives by pressing keys 1, 2, 3 or 4. Also, an offset was applied to the service provider's position relative to the camera. This positioned it slightly below and in front of the camera, ensuring the Leap Motion device's field of view matched the virtual scene. By applying the offset relative to the camera's position, the system maintained proper alignment between the Leap Motion's tracking data and the virtual objects for accurate interactions. This ensures accuracy in the hand interactions, no matter which perspective the user switches to.

The **Interaction Manager** was also included in the Unity scene, which serves as the main controller for all hand interactions. It provides collision detection, interaction events, and communication between the user's hand movements and the virtual environment. It manages the interaction behavior of virtual objects and ensures that the Leap Motion device accurately tracks hand movements, translating them into virtual actions like grabbing, pushing, or touching objects. The **Physical Hands Manager** was also employed to handle the real-time mapping of physical hand movements to their virtual representation, ensuring that the user's gestures and motions were accurately reflected in the scene, by integrating Unity's physics engine. It allows hands to interact with objects equipped with Rigidbody and Collider components, providing responses such as grabbing

or pushing. The Physical Hands Manager offers three contact modes to define how hands interact with objects. Hard Contact which allows the hands to wrap around objects enabling users to pick up and push them naturally, Soft Contact where hands can pass through objects but still allowing for pushing and grabbing and No contact, that lets the hands pass through objects without causing any physical interaction. In this project, the choice of hard contact ensured that the tactile feedback closely matched the sensation of physically engaging with a texture, creating a more immersive and realistic experience. Finally, the **Low Poly Hands with Arms** prefab was used to visually represent the user's hands in the scene, providing a simplified model of the hands and arms for a clear view of interactions with the textured objects.

For the interactions with the virtual objects to function correctly, each object must have an Interaction Behaviour component attached to it. This component is part of the Ultraleap Interaction Engine and handles how virtual objects respond to hand movements, such as grabbing, pushing, or touching. It ensures that interactions are smooth and realistic, with appropriate responses based on Unity's physics engine and the Leap Motion's precise hand tracking data. In this project, the Interaction Behaviour component was added to all textured objects in the Unity scene, allowing the system to interpret user interactions accurately.

To enable the Leap Motion Controller's functionality, the Ultraleap Hand Tracking Software must be installed and running. This software serves as the core driver and service that processes raw input from the device, generating hand tracking data for integration into Unity. It also provides essential tools for configuring the device, testing its functionality, and performing firmware updates. Without this software, the Leap Motion Controller cannot produce any tracking data, making it impossible to use within Unity or any other application. Therefore, ensuring the Ultraleap Hand Tracking Software is active during system use is a critical requirement for this implementation.

4.4.3.1 Hand Structure and Fingertip Detection

The Leap Motion Controller provides detailed skeletal tracking of hands, breaking each hand into individual joints and bones, each equipped with independent colliders. This enables precise interaction with virtual objects but also introduces complexity in managing specific interactions. Since each joint is treated as a separate entity, the system must explicitly identify and manage the fingertip responsible for triggering feedback. The Ultraleap SDK facilitates this detailed hand structure through its hand models, where individual joints and bones are represented as Contact Bones. As shown in Figure 4.14, each joint of the hand, including the palm and fingers, is organized as part of a hierarchy, with individual Capsule Colliders attached to enable Unity's physics engine to detect collisions.

In this project, interactions were designed to be triggered only when the fingertip of the right hand's index finger made contact with the virtual textured objects. For this purpose, a controller script named FingertipColliderManager was created to manage the

collisions of the fingertip with the textured objects and to communicate with the ESP32 tha drives the LRA. However, due to the dynamic nature of Leap Motion, hand objects are generated only when the user's hands enter the device's field of view. Each part of the hand, is dynamically instantiated as individual objects with independent colliders at runtime. Because of this, it is not possible to manually attach scripts to the fingertip within Unity's editor. To address this, a custom FingertipManager script was developed and attached to the Ultraleap Interaction Manager.

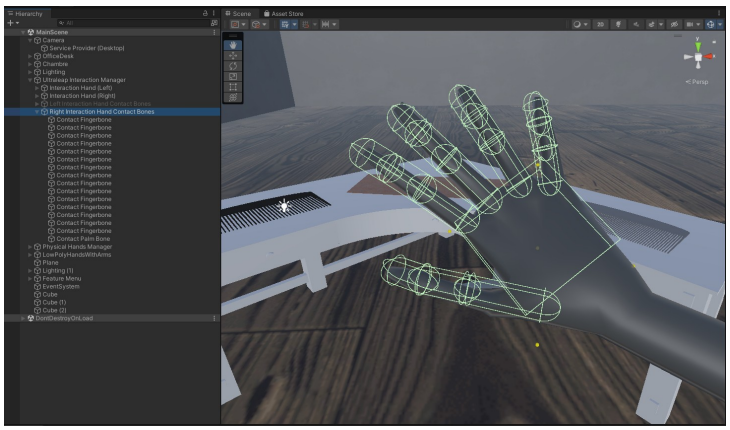


Figure 4.14: Hierarchy and structure of hand colliders provided by the Ultraleap SDK.

The script continuously monitors the Leap Motion's hand-tracking data to detect when the user's right hand enters the field of view. Once the hand is detected, the script locates the dynamically created "Right Interaction Hand Contact Bones" object, which represents the skeletal structure of the right hand. Specifically, it identifies the index fingertip (the fifth child object in the hierarchy) and dynamically attaches the FingertipColliderManager script to it. The Leap Motion dynamically generates hand objects, including their skeletal structure, only when the user's hands are first detected within the device's field of view. The FingertipManager script monitors this initial detection, locates the 'Right Interaction Hand Contact Bones' object, and identifies the index fingertip to dynamically attach the FingertipColliderManager script. This setup ensures that only the fingertip triggers tactile feedback during interactions with the virtual textured objects.

When tracking is temporarily lost, the Ultraleap SDK does not destroy the hand objects, it disables them. This means that when the hand re-enters the field of view, these objects are re-enabled. As a result, the previously attached FingertipColliderManager is retained, eliminating the need for reinitialization or reattachment of the interaction logic.

In Figure 4.15, the FingertipColliderManager is dynamically attached to the index fingertip, ensuring that interactions occur only when the fingertip contacts the textured objects.

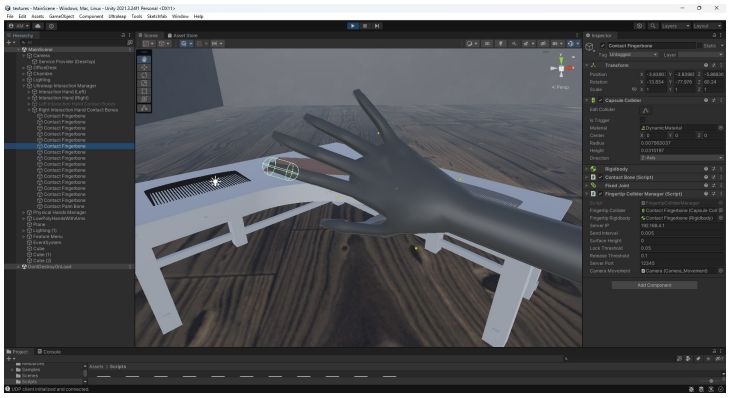


Figure 4.15: Index fingertip collider with dynamic detection enabled.

4.4.4 Fingertip Collider Manager

The FingertipColliderManager script is responsible for managing fingertip interactions within Unity, detecting collisions with virtual textured objects, and establishing wireless communication with the ESP32-S3 microcontroller. This is achieved through a combination of UDP communication, collision detection using Unity's physics engine, handling when the bottom part of the fingertip touches a textured object, velocity-based feedback modulation and surface height locking for smooth sliding interactions. Each component is carefully implemented to ensure efficient and realistic tactile feedback.

4.4.4.1 UDP Communication with ESP32

The script establishes a UDP connection between Unity and the ESP32, which is programmed as an access point (AP), enabling real-time communication. The host device (the laptop running the Unity project) is connected to the the ESP32's Wi-Fi network enabling the data exchange. UDP was chosen for its low latency ensuring real-time tactile feedback. While UDP does not guarantee packet delivery, the system's frequent data transmission ensures that occasional packet loss does not significantly impact the user experience. Tactile perception can tolerate minor inconsistencies without compromising the overall realism of the feedback, due to the adaptability of the human sensory system. It is capable of compensating for slight variations in tactile feedback, especially when supported by other sensory inputs like visual cues. Research indicates that even in cases where packet loss slightly reduces the intensity or synchronization of feedback, the perceptual impact is minimal due to the rapid data updates provided by the system [113].

4.4.4.2 Initialization and Connection

During initialization, the script creates a UDP client using UdpClient class, that connects to the ESP32's fixed IP address (192.168.4.1) and port (12345). A test message, "Hello", is sent immediately after the connection is established to verify the communication channel. If the initial connection attempt fails, a reconnection coroutine is triggered. This coroutine repeatedly attempts to reconnect at regular intervals until a successful connection is established. The reconnection logic includes error handling to ensure the system does not crash in the event of repeated failures, maintaining robust communication.

4.4.4.3 Message Structure and Communication Logic

Data is exchanged between Unity and the ESP32 using a structured UDP message format. Outgoing messages from Unity to the ESP32 consist of four main components, separated by commas in the format `<command>,<textureTag>,<velocity>,<modulationState>` where:

- **Command**: Specifies the type of interaction, when a collision is triggered such as "enter", "stay", or "exit".
- **Texture Tag**: The tag of the collided object that identifies the texture, the mode that is on and the selected type of the features (common or dedicated).
- **Velocity**: The absolute value of the velocity along the X-axis of fingertip movement, derived from the Rigidbody. This value is used for dynamic modulation of the LRA's output.
- **Modulation State**: A boolean indicating whether velocity modulation of the LRA is enabled.

For example, an outgoing UDP message might look like this:

```
stay,2,0.15,true
```

This message communicates that the fingertip has maintains contact with the texture corresponding to the CSV file named 2, is moving at a velocity of 0.15, and that velocity modulation is enabled. If the message format is invalid, the system logs an error and discards the message.

Incoming messages from the ESP32 follow a similar structure, echoing the interaction type, texture tag, velocity, and modulation state. These incoming messages are processed through a callback function asynchronously, which listens continuously for responses from the ESP32. It validates the format of the received data and parses the components. If the connection is temporarily lost and then restored, the callback function ensures communication resumes without manual intervention.

4.4.4.4 Cleanup and Resource Management

To prevent resource leaks or lingering connections, the script ensures cleanup during application shutdown. The CloseUDPClient function closes the UDP client and releases resources when the Unity application is terminated or the FingertipColliderManager is destroyed, preventing interference with future runs of the application.

If the connection is lost, the script initiates the reconnection coroutine that retries at fixed intervals until the connection is re-established. The isConnected flag provides a clear indication of the communication state, allowing other parts of the system to adapt to potential disruptions. Once reconnected, the system resumes sending real-time updates but does not attempt to resend any data missed during the disconnection.

4.4.4.5 Collision Detection and Interaction Logic

The script relies on Unity's collision system to detect and manage interactions between the fingertip and virtual textures, ensuring that tactile feedback is appropriately triggered. Three main collision events handle different phases of the interaction: **OnCollisionEnter**, **OnCollisionStay**, and **OnCollisionExit**. The logic is designed to only detect collisions with the bottom part of the fingertip and avoid unnecessary interactions, maintaining efficient communication with the ESP32.

Bottom Part Detection

The IsBottomPartCollision method ensures that only collisions at the bottom part of the fingertip are valid for triggering interactions. This is implemented by transforming the collision point from world coordinates into the fingertip's local coordinate space. This transformation is needed since Unity's physics system detects collisions in global space, but determining the relative position of the contact point on a specific part of the fingertip requires evaluating it in the fingertip's local coordinate system.

In the local space of the fingertip, the y-axis aligns with the vertical direction relative to the fingertip's orientation. By applying a threshold check on the y-coordinate, the method identifies whether the contact occurred on the bottom surface of the fingertip. This threshold value was empirically chosen to allow for realistic interactions while avoiding false positives from side or top contacts. Using local space ensures that the bottom part detection is consistent regardless of the fingertip's rotation or movement in the scene, since the fingertip's orientation can dynamically change during hand movements. This approach filters out unintended collisions with other parts of the fingertip or hand, such as the sides or top, which might otherwise introduce inaccuracies in the interaction. This logic simulates realistic interactions by ensuring that only intended fingertip contacts trigger feedback.

Velocity Modulation

The velocity modulation feature was implemented to enhance the realism of the haptic feedback by dynamically adjusting the frequency and amplitude of the LRA based on the user's hand movement speed. This approach ensures that tactile feedback mimics the physical sensation of sliding a fingertip over a textured surface, where the speed of movement naturally influences the perceived vibration intensity.

The fingertip's velocity is captured using Unity's Rigidbody component, which provides real-time physics based velocity data. The script monitors the fingertip's horizontal velocity along the interaction axis, which is the the X-axis, corresponding to left or right sliding motions. This axis aligns with the user's natural interaction direction, as captured by the Leap Motion Controller. A threshold value of 0.05 m/s is used to filter out small or unintentional movements and if the fingertip's velocity falls below this threshold, no feedback is triggered. This prevents the LRA from vibrating when the fingertip is stationary or during very slow movements, which might otherwise cause unintended or unnatural sensations.

Collision Functions

The FingertipColliderManager script handles fingertip interactions with virtual surfaces, by employing the Unity's collision system. Each of the functions that are used handle different phases of a collision, with OnCollisionEnter triggered when the collider of one object first makes contact with another, OnCollisionStay called continuously while two objects remain in contact and OnCollisionExit triggered when the colliders stop touching. The logic is further refined to only detect collisions with the bottom part of the fingertip, avoiding unnecessary interactions and maintaining efficient communication with the ESP32. Before executing their respective functionalities, all collision related functions first verify that the collided object has a valid tag using the ValidTag() function. If the tag is valid, they proceed to check whether the collision occurred on the bottom part of the fingertip using the IsBottomPartCollision() method. Only when both conditions are satisfied the script proceeds with the intended interaction logic

OnCollisionEnter

The OnCollisionEnter method is responsible for detecting the initial contact between the fingertip and a virtual textured surface, setting up the necessary parameters for interaction. Upon collision, the Y-coordinate of the contact point is captured and stored as the surfaceHeight, which will be used to lock the fingertip's vertical position during subsequent interactions. Additionally, the isGrounded flag is set to true, indicating that the fingertip is in contact with the surface. To prevent duplicate events, a debouncing mechanism checks whether the fingertip is already colliding and if not, an enter command is sent to the ESP32 via UDP, including the texture tag and an initial velocity of zero. This marks the beginning of the interaction and prepares the system for continuous

feedback as the fingertip moves across the surface.

OnCollisionStay

The OnCollisionStay method handles the ongoing interaction between the fingertip and the virtual textured surface during sliding motions. The method then incorporates the velocity logic and if the velocity exceeds the threshold, the script checks whether enough time has passed since the last data transmission using the sendInterval parameter. This ensures that UDP messages are sent at controlled intervals to avoid overwhelming the ESP32. If the conditions are satisfied, a stay command is sent to the ESP32 via UDP, including the texture tag and the absolute value of the fingertip's velocity. This dynamic feedback allows the system to modulate the vibration amplitude and frequency based on the user's sliding speed, enhancing the realism of the tactile feedback. By continuously monitoring the fingertip's velocity and interaction state, the OnCollisionStay method ensures that tactile feedback reflects real-time fingertip movement while preventing unnecessary data transmission.

OnCollisionExit

The OnCollisionExit method is triggered when when the fingertip lifts off or loses contact with the textured object. When a valid exit event is detected, the method resets the interaction state by marking the fingertip as no longer grounded and sliding, using the isGrounded and isSliding flags. The surfaceHeight value is cleared, ensuring that no height constraints persist for subsequent interactions. Finally, the method sends an exit command to the ESP32 via UDP, including a tag value of 0 and a velocity of 0, signaling the system to stop the vibration feedback. By resetting all relevant states and notifying the ESP32, the OnCollisionExit method ensures termination of the interaction, preparing the system for future fingertip contacts.

Surface Locking for Sliding

The surface height logic ensures smooth and realistic sliding interactions by locking the fingertip's vertical position (Y axis) to the height of the virtual surface during contact, preventing unintended vertical movement and simulating a natural interaction as the fingertip glides over the textured surface. The logic begins in the OnCollisionEnter method, where the surface height is initially set by capturing the Y coordinate of the contact point during the first valid collision. Specifically the script first ensures that the collision originates from the bottom part of the fingertip to avoid false triggers and if the interaction is valid it assigns the surface height using the collision data. Once the surface height is established, the Update() method which runs once per frame, continuously enforces the Y axis lock. If the fingertip's distance to the surface falls within a predefined lockThreshold, the isGrounded and isSliding flags are activated and the fingertip's position is dynamically constrained to the surface height using MovePosition(), allowing free movement only along the horizontal axes (X and Z axis). To maintain realism, the

script checks if the fingertip rises above the surface beyond a specified releaseThreshold, where the flags are reset and the surface height is cleared, restoring free fingertip movement. The OnCollisionExit method complements this logic by resetting the surface height and interaction state when the fingertip leaves the surface, ensuring the system is ready for reinitialization on subsequent collisions. Together, these components eliminate vertical jitter, enable stable and responsive sliding to ensure realistic experience in tactile feedback.

4.5 Driving the Linear Resonant Actuator

This section outlines the complete implementation of the system developed to drive the Linear Resonant Actuator, which is the primary component for delivering tactile sensations corresponding to texture interactions. The implementation is divided into hardware and software components, providing a comprehensive approach to designing a robust and responsive system. The hardware setup focuses on the physical connections needed and parts that were used, while the software setup explains the communication protocols and real-time task management needed to ensure accurate feedback generation. Together, these components enable a connection between user interactions captured in Unity and the physical feedback rendered by the LRA.

4.5.1 Hardware Setup

The hardware configuration for the LRA consists of three main components: the DA7280 Haptic Driver, the ESP32-S3 microcontroller, a coin Linear Resonant Actuator and a 18650 battery shield. These components were integrated into a compact wearable system for real time haptic feedback.

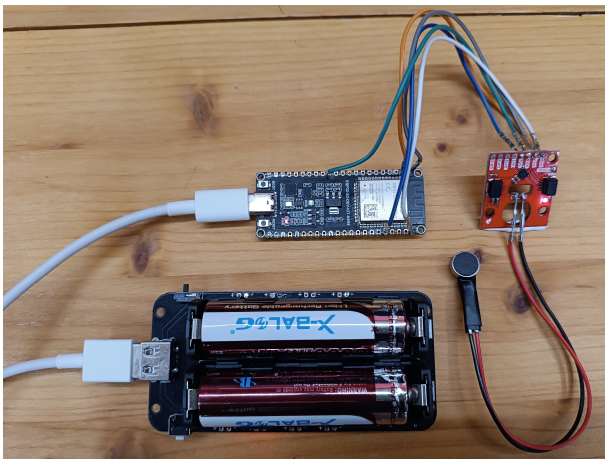
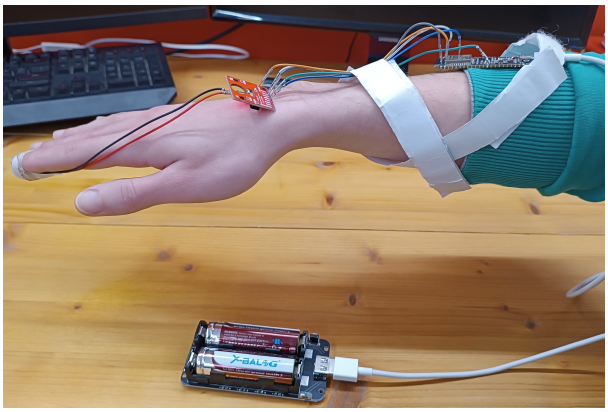


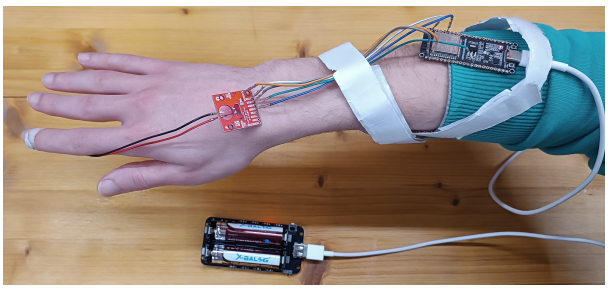
Figure 4.16: Hardware configuration for the Linear Resonant Actuator.

In this configuration, the DA7280's VDDIO and VDD pins are connected to the 3.3V output pins of the ESP32-S3, providing the necessary operating voltage for the driver. The GND pin of the DA7280 is connected to a ground (GND) pin on the ESP32-S3, establishing a common reference point for the circuit. The driver communicates with the ESP32-S3 using the I2C protocol, with its SDA (data line) and SCL (clock line) pins connected to the ESP32's pins 3 and 2, respectively.These GPIO pins are designated for I2C communication, as indicated in the ESP32-S3 datasheet, allowing data exchange between the microcontroller and the haptic driver. The ESP32-S3 is powered via a USB connection to a battery shield, which supplies the necessary voltage and current to both the microcontroller and the connected peripherals.

The ESP32-S3 serves as the primary controller for the system. It manages the software logic for driving the LRA and handles I2C communication with the DA7280 driver. The ESP32 is powered via a USB connection to a 18650 battery shield, which provides consistent and portable power for the entire system. The battery shield supports the power demands of both the ESP32 and the DA7280, ensuring reliable operation. The LRA is directly driven by the DA7280 whichi is powered by the ESP32-S3 and its power requirements are typically between 80-120 mA at 3.3V which fall well within the capabilities of the ESP32's power supply.



(a) Side View of the LRA interface mounted on the right hand of a user.



(b) Top View of the LRA interface mounted on the right hand of a user.

Figure 4.17: Overview of the LRA interface.

The coin LRA is physically attached to the user's fingertip using Velcro straps, ensuring direct contact for effective haptic feedback. Power and Ground wires from the LRA run along the user's arm to the DA7280 driver, which along with the ESP32-S3 and the battery, are secured to the user's arm using additional Velcro straps. This solution allows to test the haptic feedback system in a completely wireless and wearable mode.

4.5.2 Software Setup

On the embedded side, the ESP32-S3 which is programmed again using the PlatformIO, as a Wi-Fi access point, is the controller for the haptic feedback. The frequency-amplitude output CSV files of the feature extraction and mapping algorithm are pre stored on the ESP32's SPIFFS file system, so they can be used to drive the Linear Resonant Actuator. Each CSV file corresponds to a particular texture and chosen feature combination, such as audio-only features, accelerometer-only features or fused features and can represent either a common feature set suitable for multiple textures or a dedicated set fine tuned for each individual from the recorded textures. The feedback is generated based on user interactions tracked in Unity and sent to the ESP32-S3 microcontroller. The key functionalities of this implementation include UDP communication, CSV file processing, dynamic modulation based on fingertip velocity, and FreeRTOS-based task management to ensure non-blocking operation.

4.5.2.1 System Initialization

The system initialization stage implemented in setup function which runs every time the microcontroller is powered on, sets up all the key components required for wireless communication, haptic feedback control, and file management. Additionally, task creation and resource synchronization mechanisms are initialized to handle concurrent processes efficiently.

Wi-Fi Access Point Setup and UDP Communication

The ESP32-S3 microcontroller is configured as a Wi-Fi Access Point (AP) to establish a dedicated communication link with the Unity host device. This ensures a stable and low-latency UDP communication channel for transmitting tactile feedback parameters. The AP is initialized with a predefined SSID (ESP32_AP) and password (12345678) for secure access. After enabling the AP, the ESP32's IP address is printed to the serial monitor for debugging purposes and to allow the Unity system to know where to send UDP packets. Using the WiFiUDP library, the ESP32 sets up a UDP listener on port 12345, where continuously listens for incoming packets from the Unity application, ensuring real-time data exchange. The following code demonstrates the Wi-Fi AP setup and UDP listener.

```
WiFi.softAP(ssid, password);
IPAddress IP = WiFi.softAPIP();
Serial.printf("AP IP address: %s\n", IP.toString().c_str());

if (!udp.begin(localUdpPort)) {
    Serial.println("Failed to start UDP");
    return;
}
Serial.printf("Now listening at IP %s, UDP port %d\n", IP.toString().c_str(),
↪  localUdpPort);
```

I2C Initialization and Haptic Driver Setup

The system initializes the I2C bus to communicate with the DA7280 Haptic Driver, using the Wire library. The I2C pins are configured as SDA (pin 2) and SCL (pin 3) and a clock speed of 400 kHz is set for high-speed communication. Then, the DA7280 is initialized using the SparkFun Qwiic Haptic Driver Library, that provides functions to configure and control the driver. The actuator type is set as LRA and the operation mode to Direct Register Operation (DRO), which provides real time control over the LRA's driving parameters, such as frequency and amplitude. DRO mode allows the system to dynamically adjust the vibration output based on incoming data by directly overriding the driver's registers, unlike other available modes such as waveform memory with pre-stored sequences. The DA7280 can be configured to operate in either open or closed loop modes. In closed-loop, the system actively monitors the back electromotive force (BEMF) and adjusts the output signal to optimize operation, by utilizing features like Frequency Tracking, Active Acceleration and Rapid Stop. Frequency Tracking is used to automatically track and maintain the LRA's resonant frequency during operation. This ensures optimal driving conditions for maximum efficiency and vibration intensity, particularly for narrowband LRAs where the resonant frequency is critical. However, in this implementation where the frequency values are dynamically adjusted, the frequency tracking would override these values and attempt to re-lock to the LRA's natural resonant frequency, conflicting with the purpose of this project. The Active Acceleration mode optimizes the acceleration performance of the LRA, to achieve higher acceleration during activation, while Rapid Stop mode is designed to quickly stop the LRA motion by applying an opposing waveform to cancel the vibration. This means that Active Acceleration and Rapid Stop features enable automated driving, reducing the time to reach the target acceleration level and the time for the actuator to come to a complete stop. While this mode optimizes performance and energy efficiency by continuously adapting to the actuator's natural resonance, it is not suitable for applications requiring user defined frequency control. In this project the open-loop mode was selected, where the driver operates without feedback from the actuator with the signal output being directly controlled by user defined settings. It allows driving the actuator at frequencies different

from its natural resonance, resulting in Wideband Frequency operation.

```
if (!Wire.begin(3, 2, 400000)) {
    Serial.println("Failed to initialize I2C bus.");
}
if (!haptic.begin()) {
    Serial.println("Could not communicate with Haptic Driver.");
    return;
}
haptic.setActuatorType(LRA_TYPE);
haptic.setOperationMode(DRO_MODE);
haptic.enableAcceleration(false);
haptic.enableRapidStop(false);
haptic.enableFreqTrack(false);
```

File System Initialization (SPIFFS)

Next, the SPI Flash File System (SPIFFS) was initialized to enable the ESP32 to store and retrieve files from its internal flash memory. As mentioned before, the selected CSV files which contain the frequency and amplitude data from the feature mapping algorithm are pre-stored in the flash memory of the ESP32-S3. To achieve that, a dedicated data folder within the PlatformIO project directory was created and the selected CSV files were saved in this directory. Using PlatformIO's Upload Filesystem Image feature, the data folder is packaged into a SPIFFS image and flashed onto the ESP32's internal storage, ensuring that the files are correctly formatted and accessible during runtime. Each file corresponds to a specific texture and can be referenced by its path within SPIFFS, however due to the file system's length limitation of 31 characters, the filenames were kept as short as possible for efficient file handling.

The SPIFFS.begin(true) function mounts the file system making the flash memory ready for file operations like reading, writing and deleting, while the true parameter allows the SPIFFS to format the storage in case of corruption.

```
if (!SPIFFS.begin(true)) {
    Serial.println("An error has occurred while mounting SPIFFS");
    return;
}
Serial.println("SPIFFS mounted successfully");
```

FreeRTOS Task Management and Resource Synchronization

The freeRTOS is employed to manage concurrent tasks efficiently while ensuring synchronization between shared resources. Two tasks were created and pinned on separate

cores of the ESP32-S3 for parallel processing. The UDP task manages the UDP communication with Unity and is pinned to core 0, as this core handles system level interactions including WiFi stack. Since the UDP communication uses WiFi, placing this task in the same core ensures minimal context switching overhead. The Haptic Control task is responsible for reading the frequency and amplitude data from the CSV files and driving the LRA, pinned to core 1. This core is available for application specific tasks and is less impacted by system level operations, allowing for a more stable and consistent execution of the LRA control logic, given that the task involves computationally intensive operations. This division of tasks across cores utilizes the full processing power of the microcontroller enabling real-time haptic feedback. The system can process incoming data and drive the LRA concurrently, minimizing latency. Also, since the system is battery dependent, distributing the tasks across the two cores leads to better power management as it reduces the need for task switching.

Additionally, shared resources across tasks require synchronization to prevent race conditions. Mutexes ensure that only one task at a time can modify or read the shared resources so, in this implementation, two mutexes were created. The sharedVarMutex was used to protect variables that are updated by the UDP handling task, which processes incoming data from Unity and modifies the system's interaction state, while being read and used by the haptic control task to dynamically adjust LRA output. This mutex was crucial, as it prevents the simultaneous access to these variables that can result in using stale or partially updated values. The dataRowsMutex was also created for managing access to the dataRows vector, which stores the frequency and amplitude pairs that are loaded from the CSV files. This resource is updated when a new CSV file is read during texture changes and is accessed continuously by the haptic control task for driving the LRA. This mutex ensures that only one task at a time can modify or read this vector, preventing issues such as tasks accessing the vector while it is being cleared or updated with new data.

The initialization of the tasks and mutexes implemented in this project is displayed in the following code snippet.

```
sharedVarMutex = xSemaphoreCreateMutex();
dataRowsMutex = xSemaphoreCreateMutex();
if (sharedVarMutex == NULL || dataRowsMutex == NULL) {
    Serial.println("Failed to create mutex");
    return;
}
xTaskCreatePinnedToCore(handleUdpTask, "Handle UDP Task", 4096, NULL, 10,
↪  &taskUDPHandle, 0);
xTaskCreatePinnedToCore(hapticControlTask, "Haptic Control Task", 4096, NULL, 10,
↪  &taskHapticControlHandle, 1);
```

4.5.2.2 CSV Management

To read the frequency and amplitude pairs from the pre-stored CSV files in SPIFFS to dynamically drive the LRA, two functions that work collaboratively were created to load and parse the CSV data into an std::vector named dataRows. This vector is the central data structure that is accessed by the Haptic Control task at runtime to drive the LRA.

The redaCSVFile function, when called with a specific argument that represents the name of the CSV file that corresponds to the texture that the user interacts with at the moment, loads the CSV file into the vector. It begins by locking a mutex on the dataRows vector using xSemaphoreTake(dataRowsMutex, portMAX_DELAY), to ensure that no other tasks access or modify the vector during the file reading process. After locking, the function clears the vector to remove any residual data from previous loads and opens the specified CSV file from SPIFFS. The file is read line by line, with each line being processed using the processCSVLine function, with the first line always being skipped to avoid including metadata in the data rows. Once all lines are read, the file is closed and the mutex lock is released.

The processCSVLine function processes a single line of the CSV file at a time. Since each line contains frequency and amplitude pairs separated by a comma, the function identidfeis the comma's position and splits the line into two components. These values are stored in a DataRow structure which is then added to the dataRows vector.

4.5.2.3 Controlling the LRA and Velocity Modulation

The LRA is driven using a dynamic control logic implemented through several functions, to ensure that the vibration output reflects the texture propertied and user interaction dynamics in real time.

The controlLRA function is the central function for configuring the LRA's vibration parameters. It operates in two modes, based on whether the velocity modulation feature is on or not. In velocity modulation mode the vibration parameters dynamically adjust to the user;s speed. The velocity is first normalized within a predefined range to ensure smooth scaling and then the amplitude and frequency are adjusted using the mapAmplitude and mapFrequency functions respectively. The mapAmplitude function scales the input amplitude quadratically with velocity simulating a natural increase in vibration intensity as the user's speed increases. Quadratic scaling reflects the fact that higher velocities generate stronger tactile sensation, while maintaining sufficient sensitivity at lower velocities, providing a smooth transition between low and high intensity. The mapFrequency function adjusts the base frequency linearly with velocity, with a factor of 5 as scaling coefficient. This results in a piecewise linear modulation where the slope of the adjustment is consistent across velocities but the starting point varies depending on the texture's base frequency. This scaling ensures that frequency changes are proportional to velocity changes, while preserving the unique vibrational characteristics of each texture, as alternative methods might introduce perceptual inconsistencies where small changes in velocity result in disproportionately large or small changes in feedback. Since

the system is real time the scaling methods selected with simplicity and computational efficiency in mind.

```
int mapAmplitude(int amplitude, float velocity) {
    return (int)(amplitude * velocity * velocity);
}
float mapFrequency(float frequency, float velocity) {
    return frequency + (velocity * 5);
}
```

The delay between updates is calculated using the calculateDelay function, which scales inversely with velocity, creating shorter intervals for faster movements and maintaining smooth feedback. By comparing the current velocity to a predefined base velocity for the texture, it adjusts the delay clamping to a stable range to maintain perceptibility. The clamping limits that were selected were 10 to 50 ms. This ensures that the changes are withing the tactile resolution of human perception, as updates below 10 ms can result in unstable sensations. The maximum delay of 50 ms was chosen to prevent the feedback from being unresponsive during slower movements. Also this range falls within the operational limits of the LRA ensuring high performance.

In static mode, the parameters are directly retrieved from the feature mapping data without modification, and the interval between updates of the values is fixed at 20 ms. This choice ensures that the delay falls well within the human perception threshold and also aligns with the 20 ms feature extraction windows used.

4.5.2.4 Handle UDP Task

The handleUdpTask function is responsible for managing real-time communication between the Unity application and the ESP32 microcontroller. This task continuously listens for incoming UDP packets, which convey commands and interaction parameters from Unity. it begins by checking if a packet has been received and if so, it reads its content into a buffer. The packet is parsed into the key components of the UDP message structure explained in the Unity Integration section, meaning the command, the texture tag, the velocity and the modulation state. These components are extracted using indices for comma delimiters and validated to ensure proper format, with malformed packets being skipped with a corresponding debug message.

If a packet is valid, the function locks the sharedVarMutex to avoid race conditions in the shared variables and updates the system state based on the parsed command. If the command is ”enter” or ”stay”, the shared variables such as currentVelocity, which tracks the fingertip’s velocity received from Unity to dynamically adjust the haptic feedback based on movement speed, currentStateModulation, indicating whether velocity modulation is enabled or disabled, and contact, which is a flag that determines whether the fingertip is in contact with a textured surface used to start or stop the haptic feedback, are updated. If a new texture is selected, determined by the texture tag, the corre-

sponding CSV file is loaded using the readCSVFile function and the base velocity for the specified texture is updated to ensure accurate feedback modulation. The baseVelocity represents the reference velocity of each texture and is derived from the accelerometer signals from the recorded interaction during data capture. It normalizes the feedback modulation process giving a baseline for user's velocity ensuring consistency in scaling of the vibration parameters regardless of texture differences. If the command is exit, the function resets the contact flag and stops the LRA vibration immediately, signaling the end of the interaction.

```
xSemaphoreTake(sharedVarMutex, portMAX_DELAY);
if (command == "enter" || command == "stay") {
    currentVelocity = receivedVelocity;
    currentStateModulation = receivedStateModulation;
    contact = true;
    if (textureTag != currentTextureTag) {
        currentTextureTag = textureTag;
        if (textureTag == "1") {
            baseVelocity=0.0146;
            readCSVFile("/1.csv");
        } else if (textureTag == "2") {
            baseVelocity=0.0106;
            readCSVFile("/2.csv");
        } else if (textureTag == "3") {
            baseVelocity=0.0039;
            readCSVFile("/3.csv");
        }
    } else if (command == "exit") {
        contact = false;
        haptic.setVibrate(0); // Stop vibration immediately
    }
    xSemaphoreGive(sharedVarMutex);
    udp.beginPacket(udp.remoteIP(), udp.remotePort());
    udp.printf("Acknowledged: %s", incomingPacket);
    udp.endPacket();
}
```

After processing the packet, the mutex is released and the function acknowledges the received packet by sending a confirmation response back to Unity via UDP.

4.5.2.5 Haptic Control Task

The hapticControlTask is responsible for driving the LRA to provide haptic feedback based on the data received from Unity. This task continuously monitors the system state and dynamically adjusts the LRA output to ensure feedback that aligns with user interactions.

The task begins by locking the sharedVarMutex to check the values of the shared

variables updated in the UDP handle task. If the contact flag is true, and the dataRows vector is not empty, the system proceeds to get the next frequency and amplitude pair for driving the LRA. The dataRowsMutex is used to lock access to the dataRows vector ensuring that no other task modifies the vector while the Haptic Control task reads from it. The currentRowIndex determines which row in the CSV data to process, and the corresponding row is retrieved safely. The core of the task involves calling the controlLRA function, which adjusts the LRA's vibration frequency and amplitude based on the velocity and modulation state. After processing the current row, the currentRowIndex is incremented, with a modulo operation ensuring the task loops back to the start of the vector when all rows have been processed. This allows continuous looping of the vibration pattern, creating seamless feedback during longer interactions. If the contact flag is false or the dataRows vector is empty, the task stops the LRA vibration by setting the amplitude to zero and introduces a short delay using vTaskDelay(pdMS_TO_TICKS(1)). This prevents the task from consuming unnecessary processing power while waiting for new interaction events. This implementation is shown in the following code snippet.

```
void hapticControlTask(void *parameter) {
while (true) {
    xSemaphoreTake(sharedVarMutex, portMAX_DELAY);
    bool isContact = contact;
    float velocity = currentVelocity;
    bool useModulation = currentStateModulation;
    xSemaphoreGive(sharedVarMutex);
    if (isContact && !dataRows.empty()) {
        xSemaphoreTake(dataRowsMutex, portMAX_DELAY);
        DataRow row = dataRows[currentRowIndex];
        xSemaphoreGive(dataRowsMutex);
        controlLRA(row, velocity, baseVelocity, useModulation);
        xSemaphoreTake(sharedVarMutex, portMAX_DELAY);
        currentRowIndex = (currentRowIndex + 1) % dataRows.size();
        xSemaphoreGive(sharedVarMutex);

    }else {
      haptic.setVibrate(0);
      vTaskDelay(pdMS_TO_TICKS(1));
    }
  }
}
```

4.5.2.6 System Monitoring and Recovery

The loop function in this implementation, serves as a lightweight system monitor to ensure uninterrupted communication with the DA7280 driver. The function continuously checks for any communication issues and attempts to recover without requiring a system

restart. First the function clears any pending interrupt flags from the driver using the haptic.clearIrq() method of the SparkFun library, ensuring a clean state and then the communication with the DA7280 is verified through a dedicated function. If the driver is unresponsive, the system retries communication every second by reinitializing the I2C bus and attempting to reconnect to the DA7280. When the reconnection successfully happens, the haptic driver is reinitialized to restore its configuration.

Chapter 5

Evaluation

In the Evaluation chapter, an analysis of the haptic feedback system's effectiveness in representing recorded textures through direct mapping of extracted features to vibrotactile parameters is presented. First a feature selection process was conducted by employing the think-aloud methodology, where audio and accelerometer features were analyzed to identify the most effective combinations for accurately representing textures. A user evaluation followed, which assessed realism, consistency and responsiveness through participant testing using the features selected during the first phase. The findings in this chapter highlight the strengths and limitations of different feedback modes, feature types and their impact on the overall haptic experience.

5.1 Feature Selection Process

The evaluation phase of this project focuses on identifying the most effective features from audio and accelerometer signals for representing textures through vibrotactile feedback. The objective of the first stage was to evaluate the initially extracted features, that were chosen for their theoretical relevance to texture representation, ensuring that the final selection provides a realistic and perceptually consistent haptic experience. This evaluation was conducted through a combination of visual elimination, perception-based testing, and iterative refinement processes to narrow down and optimize the feature set.

5.1.1 Feature Elimination

The first stage of the evaluation process aimed to narrow down the extensive list of extracted features from both audio and accelerometer signals as described in the Implementation chapter of this thesis in table 4.1, ensuring their suitability for direct mapping to the frequency and amplitude of the vibrotactile output. The features were initially chosen based on their theoretical relevance to texture representation and were tested by pairing frequency-mapped and amplitude-mapped features. For each pair, LRA was

driven to produce vibrations that were directly felt on the fingertip, allowing the vibrational patterns to be perceptually assessed. Given the large number of features, a preliminary elimination was necessary to refine the list. This involved a visual inspection of the frequency and amplitude mappings plotted over time for each recorded texture, where features producing flat lines, unrealistically low or high values, or patterns inconsistent with the recorded textures were removed.

Features producing flat lines in their frequency or amplitude mappings were eliminated because they failed to capture dynamic variations in the signal over time. The purpose of direct mapping is to reflect the temporal and spatial nuances of texture interactions. Flat line behaviors suggest that these features do not capture any meaningful texture specific variations, failing to provide perceptually distinguishable feedback. Additionally, features that generate extremely low values result in vibrations that are too week to be perceptible, while excessively high values might lead to uncomfortable and non realistic tactile sensations. Ensuring a perceptible yet comfortable range is essential for accurate haptic representation. Finally, features whose mapped outputs did not align with the characteristics of the recorded textures were excluded. For instance, if the mapping failed to reflect the coarseness, smoothness or regularity of a texture, it was judged unsuitable. Consistency between recorded data and haptic output is important for creating a realistic texture experience. Also, some features that had a promising background and are known to perform well in texture classification tasks such as Chroma Mean, did not provide adequate feedback in direct vibration parameter mapping and got eliminated.

Furthermore, an initial self-administered perceptual testing phase was conducted, where combinations that felt irrelevant to the recorded textures were eliminated. This process led to a reduced and refined list of candidate feature tests that served as the foundation for the next stage of perceptual testing. In the table 5.1 the results of the first stage of the feature elimination process are shown, with each extracted feature from both signals accompanied with a status column indicating if it is retained for the next stage of the evaluation.

5.1.2 Perception Testing and Think-Aloud Methodology

The updated list of feature combinations was organized into four distinct categories based on the source of the frequency and amplitude mappings. These categories were designed to explore different combinations of feature derived from the audio and accelerometer signals, ensuring a comprehensive evaluation of their contributions to vibrotactile feedback. The categories are the following:

- **Audio Only**: In this category, both frequency and amplitude were derived exclusively from features extracted from the audio signal. This approach allowed for testing the effectiveness of audio-based features in representing texture vibrations without input from the accelerometer.

Feature Name	Signal Type	Status
RMS	Audio, Accelerometer	Retained
Envelope Mean	Audio, Accelerometer	Eliminated (for Audio)
Energy Entropy	Audio, Accelerometer	Eliminated
Spectral Centroid	Audio, Accelerometer	Retained
Spectral Bandwidth	Audio, Accelerometer	Retained
Short-Time Energy (STE)	Audio	Eliminated
Zero-Crossing Rate (ZCR)	Audio	Eliminated
Peak-to-RMS-Ratio	Audio	Retained
Spectral Rolloff	Audio	Retained
Spectral Flatness	Audio	Eliminated
Spectral Contrast	Audio	Retained
Spectral Flux	Audio	Retained
MFCC Mean	Audio	Retained
MFCC Variance	Audio	Retained
Onset Strength	Audio	Eliminated
Chroma Mean	Audio	Eliminated
Peak-to-Peak Distance	Accelerometer	Retained
Low Frequency Band	Accelerometer	Retained
Mid Frequency Band	Accelerometer	Retained
High Frequency Band	Accelerometer	Retained
Power Spectral Density (PSD)	Accelerometer	Eliminated
Skewness	Accelerometer	Eliminated
Kurtosis	Accelerometer	Eliminated
Mean Value	Accelerometer	Eliminated
Standard Deviation	Accelerometer	Eliminated
Peak Count	Accelerometer	Eliminated
Peak Height Mean	Accelerometer	Eliminated

Table 5.1: Summary of Features Extracted from Audio and Accelerometer Signals, Indicating Retained and Eliminated Features.

- **Accelerometer Only**: Here, frequency and amplitude were derived solely from features extracted from the accelerometer signal. This category evaluated the ability of accelerometer-based features to represent textures independently.
- **Audio Frequency with Accelerometer Amplitude**: This category combined frequency features derived from the audio signal with amplitude features derived from the accelerometer signal. The goal was to test whether this hybrid approach could leverage the strengths of each signal type for more realistic texture representation.
- **Accelerometer Frequency with Audio Amplitude**: The inverse of the previous category, this group used frequency features from the accelerometer signal paired with amplitude features from the audio signal. This allowed for further exploration of how combining features from both signals could enhance the vibrational feedback.

Within each category, all possible frequency and amplitude feature combinations were tested to ensure every pairing was evaluated. This included pairing every frequency-mapped feature with every amplitude-mapped feature for each category. For instance, in the *Audio Only* category, frequency features extracted from the audio signal were paired with amplitude features from the same signal. Similarly, hybrid categories like *Audio Frequency with Accelerometer Amplitude* paired audio-derived frequency features with accelerometer-derived amplitude features to explore cross-signal mapping.

5.1.2.1 Think-aloud Methodology

The think-aloud method was conducted separately for each texture, allowing for a focused evaluation of how well the vibrations represented the unique characteristics of each recorded texture. It is a qualitative evaluation approach widely used to gain insights into user experiences by encouraging participants to articulate their thoughts, feelings and perceptions while interacting with a system. In this project, it was employed to evaluate the perceptual realism of vibrational feedback generated by the LRA for representing recorded textures. Each signal type (audio, accelerometer and combinations of the two) was tested independently, ensuring that the contribution of each signal type to the vibrational feedback could be thoroughly evaluated. For every texture and signal type, all possible feature combinations were tested. This included evaluating how features mapped to frequency and amplitude interacted to produce vibrational patterns. Participants provided real-time verbal feedback on both the acceptability and perceptual relevance of each combination. Finally, based on participants' feedback, the most perceptually meaningful feature combinations for each signal type and texture were determined.

The evaluation was conducted within a Unity-based virtual environment designed specifically for this purpose. Participants interacted with three virtual cubes, each representing a different texture, in a controlled setting where all visual and auditory biases

were eliminated. The cubes had no visual indications or labels and participants wore noise-canceling headphones to remove auditory distractions, ensuring that the evaluation focused solely on the vibrational feedback. The Leap Motion sensor tracked hand movement, and the haptic interface, mounted on the participant's hand, provided real-time vibrotactile feedback through the LRA, driven by the selected feature combinations. The velocity modulation feature was not used for this method, meaning the delay between the value changes was fixed at 20 ms. Below, the findings for each category and texture are summarized, along with the features retained and eliminated based on the participants' feedback.

Comb 2 mm spacing

Audio-Only Tests

When participants explored this texture using only audio-derived features, they consistently emphasized the importance of perceiving distinct tooth ridges without extra vibrations between them. The term "dirty" as used in their feedback, referred to the presence of additional noise or artifacts between peaks. While the peaks were noticeable, it was harder to clearly distinguish individual ridges and the overall realism of the haptic feedback was slightly reduced. The pairing of **Spectral Rolloff** and **MFCC variance** stood out the most, with users saying it delivered a smooth loop and maintained a realistic sense of comb ridges, providing the best balance between perceptual realism and smoothness.

On the other hand, feature pairs like Spectral Flux with MFCC Variance and Spectral Rolloff with RMS were eliminated due to their tendency to generate random bursts and inconsistent vibrations, which distracted from the tactile experience and reduced the feedback's overall realism. Participants highlighted combinations like Spectral Centroid with Peak to RMS ratio as being "slightly more clean" and providing a "very good crossfade" creating a more realistic sense of tooth spacing.

Accelerometer-Only Tests

In the accelerometer-only tests for the comb with 2 mm spacing, the feature pair of **Low Band Energy** and **Peak to Peak Distance** was the most highly rated combination for representing the comb's ridges. Participants consistently praised this pairing for its ability to produce clean, distinct peaks that closely matched the physical sensation of sliding a finger across the comb.

Other combinations such as Low Band Energy and RMS, introduced subtle noise between the ridges and degraded the overall experience, making the representation less precise. Also, combinations like Mid Band Energy and Peak to Peak Distance were criticized for their inconsistent feedback, with participants describing the peaks as uneven in intensity and lacking the sharpness needed to clearly distinguish the ridges. Another eliminated feature pair was Spectral Centroid with Envelope Mean, with participants

describing it as "confusing" due to the lack of clear peaks and a generalized blur in the feedback pattern.

Fusion: Audio Frequency and Accelerometer Amplitude

When testing the Comb's with 2mm spacing representation using audio frequency and accelerometer amplitude features, the pair of **Spectral Rolloff** and **Peak to RMS Ratio** provided the most realistic and perceptually meaningful feedback, with participants emphasizing its ability to maintain peak clarity. This pairing was particularly praised for its natural rhythm and smooth transitions, creating a tactile experience that felt both sharp and immersive without becoming overwhelming.

In contrast, the combination of Spectral Rolloff and RMS was eliminated due to feedback describing its output as having sharp and unnatural edges. Other feature pairs, such as Spectral Flux and Envelope Mean, were also excluded for producing inconsistent or "dirty" vibrations. Another combination, Spectral Centroid with Peak to Peak Distance, was described as offering a "nice representation" but was considered less effective due to minor inconsistencies in peak clarity and a slightly worse crossfade compared to the chosen one.

Fusion: Accelerometer Frequency and Audio Amplitude

When frequency came from the accelerometer, users often picked High Band Energy or Spectral Centroid since those features map well to spaced-out ridges. In the final chosen combination, **High Band Energy** and **MFCC variance**, testers described it as "clean," with tooth peaks standing out clearly and the amplitude introducing moderate but perceptible fluctuations.

On the other hand, Low Band Energy and RMS was considered "too noisy" with participants reporting excessive vibrations between the ridges. Similarly, Mid Band Energy and Envelope Mean was marked as "confusing" by participants, as it lacked the sharpness needed to represent the ridge pattern distinctly. Other combinations, such as Spectral Bandwidth and MFCC Variance, received some positive feedback for their clean representation. Combinations like Spectral Centroid and MFCC Variance were also noted as "good" but introduced minor inconsistencies that prevented them from being the top choice.

Comb 1 mm spacing

Audio-Only Tests

For the Comb with 1mm spacing in the audio-only tests, participants found **Spectral Centroid** mapped to frequency and **Peak to RMS Ratio** mapped to amplitude the best. This combination was praised for its ability to deliver a perceptually clean and balanced feedback, capturing the dense ridge spacing of the texture.

In contrast, feature pairs such as Spectral Flux with MFCC Variance were criticized for introducing excessive noise and uneven vibration patterns, leading to feedback de-

scribed as "dirty" or "confusing." Similarly, combinations like Spectral Rolloff and RMS were eliminated for creating overly intense or exaggerated feedback. Some combinations, such as MFCC Mean with RMS, were described as "better than others" but still failed to convey the same level of ridge clarity as the selected pair.

Accelerometer-Only Tests

When testing the texture using accelerometer-only features, the combination of **Mid Band Energy** and **Peak to Peak Distance** stood out with users describing it as "clean" and effective in creating a tactile experience where each ridge could be distinctly felt. This pairing offered smooth transitions between peaks while avoiding excessive noise or distracting vibrations.

On the other hand, combinations like Spectral Bandwidth and Envelope Mean were criticized for being "confusing," as the feedback lacked precision and did not distinctly convey the ridges resulting in less intuitive feedback. Similarly, Low Band Energy and RMS introduced inconsistencies and background noise, which covered the ridge details leading to its elimination. Additionally, combinations like Spectral Centroid with Envelope Mean and Mid Band Energy with RMS were described as "decent" but they either lacked the dynamic variation or introduced minor distractions.

Fusion: Audio Frequency and Accelerometer Amplitude

In the Audio Frequency and Accelerometer Amplitude tests, the pairing of **MFCC Mean** as frequency and **Peak to Peak Distance** as amplitude was the most effective and perceptually meaningful. Participants consistently highlighted this combination's ability to convey the tightly spaced ridges of the comb with clarity and smooth transitions.

Other combinations, like Spectral Flux with Envelope Mean were found to have a lack of clarity, often described as "confusing" and "dirty". Other feature pairs, such as Spectral Rolloff with RMS, were noted to produce overly sharp or unnatural feedback, feeling inconsistent with the comb's physical properties. Participants particularly appreciated the smooth crossfade in the chosen combination, which avoided abrupt transitions. Although Spectral Rolloff with Peak to Peak Distance showed potential and received some positive feedback, it was dismissed due to unnatural transitions and inconsistencies.

Fusion: Accelerometer Frequency and Audio Amplitude

In the Accelerometer Frequency and Audio Amplitude tests, the pairing of **Spectral Centroid** as frequency and **RMS** as amplitude was identified as the most effective representation. The use of Spectral Centroid as the frequency mapping provided a stable and well-balanced rhythm, accurately reflecting the fine and closely spaced ridges of the comb.

In contrast, combinations like High Band Energy with MFCC Variance were eliminated due to their inability to provide a clean representation. Similarly, pairings such as Low Band Energy with Peak RMS Ratio introduced uneven intensities, which participants found confusing and less representative of the texture. Combinations like Mid Band

Energy with RMS were considered acceptable but lacked the nuanced clarity provided by the final choice.

Sandpaper

Audio-Only Tests

For the Sandpaper texture in the Audio Only tests, the feature combination of **MFCC Mean** mapped to frequency and **Peak RMS Ratio** mapped to amplitude concluded as the most effective representation. Participants chose this pairing for its ability to convey the coarse and granular nature of the texture. The MFCC Mean frequency mapping introduced a dynamic, irregular pattern that represented well the roughness of sandpaper, while the Peak RMS Ratio amplitude mapping provided a steady intensity that felt natural and perceptually engaging. Another strong candidate was the pairing of Spectral Rolloff and Peak RMS Ratio, which effectively represented much of the texture's coarse characteristics. However, it was eliminated due to its slightly higher intensity, which some participants found overly harsh during extended interactions. Despite this, the combination was recognized for its realistic transitions and nuanced representation.

Conversely, combinations such as Spectral Rolloff with RMS or Spectral Flux with MFCC Variance were eliminated with participants describing these pairings as "flat" or "too uniform" lacking the dynamic irregularities that define sandpaper. RMS as an amplitude mapping from the audio signal, while effective for textures like the combs, demonstrated a significant limitation for the sandpaper texture. Its reduced dynamic range led to vibrational feedback that was barely perceptible making it unsuitable for capturing the texture's granular irregularities.

Accelerometer-Only Tests

For the Sandpaper texture in the Accelerometer Only tests, the feature combination of **High Band Energy** and **Envelope Mean** was the top choice for capturing the texture's coarse and irregular characteristics. Participants consistently described the feedback as "realistic" and "detailed" aligning well with the tactile perception of sandpaper. The frequency mapping from High Band Energy was particularly well-suited for conveying the fine-grained texture, while the amplitude mapping from Envelope Mean provided a perceptible but not overwhelming variation, making the feedback intuitive. Other combinations, such as High Band Energy and RMS and Low Band Energy and Envelope Mean also received positive feedback, however they noted that the amplitude felt slightly less dynamic compared to Envelope Mean, which reduced the overall tactile realism.

Several combinations, such as Low Band Energy and Peak-to-Peak Distance and Mid Band Energy and Peak-to-Peak Distance, were eliminated due to feedback describing their output as "less realistic" or lacking the granularity necessary for sandpaper. Participants commented that these pairings failed to fully evoke the texture's irregularities, making the vibrations feel less intuitive. Additionally, Spectral Centroid and Envelope

Mean was rated as "good" but was not selected due to minor inconsistencies in capturing the tactile granularity of the sandpaper.

Fusion: Audio Frequency and Accelerometer Amplitude

For the Sandpaper texture in these tests, the feature combination of **Spectral Flux** and **Envelope Mean** was selected as the most effective for the representation of the texture. Participants noted that this pairing provided "smooth and consistent feedback" with the frequency effectively capturing the subtle, irregular shifts of the texture while the amplitude delivered sufficient clarity without overpowering the tactile sensation. Similarly, Spectral Flux paired with Peak to Peak Distance stood out for its smooth and natural tactile feedback, offering a perceptual clarity that participants found acceptable for representing the texture.

In contrast, feature combinations such as Spectral Bandwidth mapped to frequency and RMS mapped to amplitude, as well as Spectral Rolloff with Envelope Mean, were eliminated due to producing repetitive and overly intense loops. Participants described these outputs as artificial and disconnected from the expected tactile sensation of sandpaper. Other combinations, including MFCC Mean with Peak to Peak Distance, were noted as "okay" but lacked the distinctiveness and clarity necessary to capture the texture's granular nature.

Fusion: Accelerometer Frequency and Audio Amplitude

In the Accelerometer Frequency and Audio Amplitude tests, the feature combination of **Spectral Centroid** and **Peak RMS Ratio** was the most effective in representing the texture's coarse characteristics. Participants described this combination as providing a "balanced and clear representation," with the frequency capturing the granular irregularity of sandpaper while the amplitude provided enough intensity to make the feedback perceptible without overwhelming the sensation. High Band Energy with Peak RMS Ratio, Low Band Energy with Peak RMS Ratio, Mid Band Energy with Peak RMS Ratio, and Spectral Centroid with Peak RMS Ratio all received "okay" ratings from participants. These combinations provided perceptible feedback with a balanced intensity, but none stood out as exceptional for representing sandpaper's fine, coarse qualities.

On the other hand, combinations such as Spectral Centroid with RMS, High Band Energy with RMS, and Low Band Energy with RMS were eliminated due to feedback describing the vibrational patterns as barely perceptible, significantly reducing their effectiveness. Other combinations, such as High Band Energy with MFCC Variance and Spectral Centroid with MFCC Variance, were dismissed for producing intense looping sensations.

General Results about Features and Patterns

The results of the think-aloud method revealed consistent patterns and valuable insights into how specific feature combinations perform across different textures. These

findings highlighted the importance of carefully selecting and pairing frequency and amplitude features to achieve perceptually meaningful and realistic haptic feedback.

In comparing the two fusion approaches, using audio features for frequency and accelerometer features for amplitude and using accelerometer features for frequency and audio features for amplitude, participants favored the first option for all three textures. This outcome suggests that audio-based frequency generally captured the essential variation of each texture more effectively, while the accelerometer's amplitudes gave an accurate representation without creating random spikes or silence. Testers also noted that audio-based frequency often felt less "chaotic", likely because direct mechanical vibrations from the accelerometer when used for frequency, could become too pronounced unless carefully paired with a mild audio amplitude. By contrast, using audio derived features for frequency with accelerometer amplitude provided a more consistent balance, leading participants to select that configuration in the fusion mode.

Overall, the results revealed both alignment and divergence between theoretical expectations and practical outcomes. Amplitude features such as RMS, Envelope Mean, and MFCC Variance largely aligned with theory, delivering smooth and consistent feedback, especially for continuous textures like Sandpaper. However, RMS's limited perceptibility in audio-only tests for Sandpaper highlights the need for adaptive dynamic range adjustments in certain contexts. Similarly, frequency features like Spectral Centroid and Spectral Rolloff successfully represented structured textures like comb ridges but underperformed when poor crossfade design or signal irregularities introduced noise or "dirty" artifacts, as seen with Spectral Flux. Fusion mappings, demonstrated their theoretical advantage by producing realistic and nuanced representations, but only when crossfade transitions were smooth. User feedback provided subjective variability, with preferences for intensity and clarity differing across participants, underscoring the need for flexible designs that accommodate individual differences. These findings highlight that while theoretical models provide a starting guidance, iterative user testing and practical refinements are crucial for achieving perceptually convincing and realistic haptic feedback systems.

5.1.3 Final Candidate Selection

As detailed in the previous section, the Think-Aloud Methodology resulted in specific feature pairs for each texture and testing method that provided the most accurate and perceptually meaningful individual representations. However, the evaluation also considered whether common feature pairs could effectively represent all textures within each mode (audio-only, accelerometer-only, or fusion). To explore this, the feature pairs that were not rejected during the Think-Aloud stage were compiled into tables 5.2, 5.3, 5.4 for each mode and texture. These tables naturally emerged as a result of this process, listing the combinations that consistently performed well. Within these tables, common feature pairs that performed well across multiple textures were highlighted in bold. These highlighted pairs were then candidates for a secondary perceptual evaluation to assess

their viability as common features for all textures within each mode.

In this stage of evaluation, the focus shifted from testing each texture individually to assessing all textures simultaneously. Users were presented with three cubes per texture (one for each texture), arranged side by side, for each feature pair. The number of cube triplets corresponded to the total feature pair candidates for each mode. The evaluation was conducted separately for each signal type, and users interacted with all textures represented by each feature pair in sequence. After testing, participants were tasked to select the feature pair they favored most for representing all textures based on realism and quality of representation. This process led to the identification of the final common feature pairs for each signal type, balancing perceptual fidelity with consistency across textures.

This follow-up evaluation aimed to find features to later determine whether universal feature pairs could balance simplicity and perceptual fidelity or if dedicated feature pairs remained necessary for achieving optimal texture representation.

Table 5.2: Accelerometer Signal Features Only

Sandpaper		**Comb 1mm**		**Comb 2mm**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
High Frequency Band	Envelope Mean	High Frequency Band	Envelope Mean	**High Frequency Band**	**Peak-to-Peak Distance**
High Frequency Band	**Peak-to-Peak Distance**	**High Frequency Band**	**Peak-to-Peak Distance**	**High Frequency Band**	**RMS**
High Frequency Band	**RMS**	**High Frequency Band**	**RMS**	Low Frequency Band	Envelope Mean
Low Frequency Band	Envelope Mean	Low Frequency Band	RMS	Low Frequency Band	Peak-to-Peak Distance
Mid Frequency Band	Envelope Mean	Mid Frequency Band	Peak-to-Peak Distance	Low Frequency Band	RMS
Mid Frequency Band	**RMS**	**Mid Frequency Band**	**RMS**	**Mid Frequency Band**	**RMS**
Spectral Bandwidth	Envelope Mean	Spectral Bandwidth	Envelope Mean	Spectral Bandwidth	RMS
Spectral Centroid	Envelope Mean	Spectral Bandwidth	RMS	Spectral Centroid	Peak-to-Peak Distance
Spectral Centroid	RMS	Spectral Centroid	Envelope Mean	Spectral Centroid	RMS

Table 5.3: Audio Signal Features Only

Sandpaper		**Comb 1mm**		**Comb 2mm**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
MFCC Mean	Peak-to-RMS Ratio	**Spectral Centroid**	**Peak-to-RMS Ratio**	MFCC Mean	MFCC Variance
Spectral Bandwidth	Peak-to-RMS Ratio	Spectral Contrast	RMS	Spectral Bandwidth	Peak-to-RMS Ratio
Spectral Centroid	**Peak-to-RMS Ratio**	Spectral Flux	RMS	Spectral Bandwidth	RMS
Spectral Contrast	Peak-to-RMS Ratio	Spectral Rolloff	RMS	**Spectral Centroid**	**Peak-to-RMS Ratio**
Spectral Rolloff	Peak-to-RMS Ratio			Spectral Flux	Peak-to-RMS Ratio
				Spectral Rolloff	MFCC Variance

Table 5.4: Fusion Features-Audio Frequency,Accelerometer Amplitude

Sandpaper		**Comb 1mm**		**Comb 2mm**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
MFCC Mean	Peak-to-Peak Distance	MFCC Mean	Peak-to-Peak Distance	MFCC Mean	Envelope Mean
Spectral Bandwidth	Envelope Mean	**Spectral Bandwidth**	**Peak-to-Peak Distance**	**Spectral Bandwidth**	**Peak-to-Peak Distance**
Spectral Bandwidth	**Peak-to-Peak Distance**	Spectral Bandwidth	RMS	**Spectral Centroid**	**Peak-to-Peak Distance**
Spectral Centroid	Envelope Mean	Spectral Centroid	Envelope Mean	Spectral Centroid	Peak-to-Peak Distance
Spectral Centroid	**Peak-to-Peak Distance**	**Spectral Centroid**	**Peak-to-Peak Distance**	Spectral Contrast	Peak-to-Peak Distance
Spectral Contrast	Envelope Mean	Spectral Contrast	Peak-to-Peak Distance	Spectral Contrast	RMS
Spectral Flux	Peak-to-Peak Distance	Spectral Contrast	RMS	Spectral Flux	Envelope Mean
Spectral Rolloff	**Peak-to-Peak Distance**	**Spectral Rolloff**	**Peak-to-Peak Distance**	Spectral Flux	Peak-to-Peak Distance

5.1.4 Final Choice of Features

The final features that were selected based on the above evaluation processes are gathered in the table 5.5 for the Common feature pairs within each mode and in tables 5.6, 5.7, 5.8 for the Dedicated feature pairs for each mode.

As we can observe in the Accelerometer-only mode in the Dedicated Feature pairs, the frequency features selected for each texture aligned intuitively with their granularity. Participants chose Low Band Energy for Comb 2 mm, Mid Band Energy for Comb 1 mm, and High Band Energy for Sandpaper as shown on table 5.6). This result aligns with theoretical predictions in haptic perception, which suggest that the spatial frequency of a texture correlates to its vibrational signature. The ability of participants to conclude on these mappings demonstrates the perceptual significance of frequency band selection in representing texture properties. Additionally, both Comb 2 mm and Comb 1 mm used Peak-to-Peak Distance for amplitude, highlighting that textures with similar structural characteristics but differing in ridge spacing, do not require distinct amplitude mappings for effective representation from accelerometer signals. This suggests that amplitude features primarily affect the perception of intensity rather than structural differentiation.

In the Audio-only mode as observed in table 5.7, the feature pairs selected for each texture were entirely distinct, unlike the Accelerometer-only. This phenomenon can be attributed to the nature of audio signals, which are highly sensitive to spectral variations and transient changes. Each texture generates a unique spectral profile based on its physical properties such as ridge spacing or surface coarseness, leading to a need for texture-specific frequency features. For instance, Spectral Rolloff was selected for Comb 2 mm, as it captures the upper frequency boundary, emphasizing sharp ridges. Similarly, Spectral Bandwidth was chosen for Comb 1 mm, likely due to its ability to reflect the spread of frequencies around a central value, aligning with the denser ridge pattern. For Sandpaper MFCC Mean was preferred, reflecting its broad and irregular spectral characteristics which align well with the perception of coarse friction.

This outcome highlights a key distinction between audio and accelerometer signals. While accelerometer data maps spatial frequency and vibrational intensity consistently across textures, audio data relies more on capturing the unique spectral signature of each texture. These results align with theoretical expectations in auditory signal processing, which suggest that audio features are inherently texture-specific, requiring tailored frequency mappings to achieve perceptually meaningful haptic feedback. This also underscores the challenge of finding a single common Audio-only feature pair that generalizes well across diverse textures.

In Fusion mode, where the frequency is derived from audio features and the amplitude from accelerometer features, in the Dedicated Feature pairs, the amplitude feature selected for both Comb textures aligns directly with the choices made in the Accelerometer-only tests. Similarly, the Sandpaper texture's amplitude feature in Fusion mode corresponds to its Accelerometer-only amplitude, highlighting the consistency of amplitude mappings that come from accelerometer signals. However, while the Comb 2 mm has

identical frequency in the Fusion mode for Dedicated Feature pairs as in the Audio-only features, the other two textures turned out to have completely different frequency mappings, than the single mode selections. This divergence in frequency mappings for the other two textures can be attributed to the interaction between the audio-derived frequency and accelerometer-derived amplitude features in Fusion mode. The combination of modalities likely introduces new dynamics that affect the effectiveness of certain features, requiring adjustments to the frequency selection to achieve the desired tactile representation. This highlights the complexity of multimodal integration, where the interplay between frequency and amplitude from different sources can influence the overall perceptual outcome.

In the Common Feature pairs of Fusion mode, the final selected feature pair combined the frequency feature from the Audio-only mode with the amplitude feature from the Accelerometer-only mode, as shown in table 5.5. This result reflects that the Fusion mode effectively combines the strengths of both sensors for achieving realistic and perceptually meaningful haptic feedback.

Table 5.5: Common Feature Pairs

Accelerometer-Only		**Audio-Only**		**Fusion (Audio Frequency, Accelerometer Amplitude**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
High Band Energy	Peak-to-Peak Distance	Spectral Centroid	Peak-to-RMS Ratio	Spectral Centroid	Peak-to-Peak Distance

Table 5.6: Dedicated Feature Pairs Accelerometer-Only

Comb 2 mm		**Comb 1 mm**		**Sandpaper**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
Low Band Energy	Peak-to-Peak Distance	Mid Band Energy	Peak-to-Peak Distance	High Band Energy	Envelope Mean

Table 5.7: Dedicated Feature Pairs Audio-Only

Comb 2 mm		**Comb 1 mm**		**Sandpaper**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
Spectral Rolloff	MFCC variance	Spectral Bandwidth	RMS	MFCC Mean	Peak-to-RMS Ratio

Table 5.8: Dedicated Feature Pairs Fusion (Audio Frequency, Accelerometer Amplitude)

Comb 2 mm		**Comb 1 mm**		**Sandpaper**	
Frequency	**Amplitude**	**Frequency**	**Amplitude**	**Frequency**	**Amplitude**
Spectral Rolloff	Peakt-to-Peak Distance	MFCC Mean	Peak-to-Peak Distance	Spectral Flux	Envelope Mean

5.2 User Evaluation

The purpose of the User Evaluation phase was to assess the effectiveness and realism of the haptic feedback system in representing the recorded textures using the feature pairs from the tables 5.5, 5.6, 5.7, 5.8. This evaluation aimed to determine the level of realism in texture representation across different feedback modes (audio, accelerometer and fusion) and to evaluate the performance of common versus dedicated feature pairs in providing realistic feedback. Additionally, the evaluation explored whether the use of dedicated features was essential for enhancing realism and whether velocity modulation contributed to the perceived realism of the feedback. To ensure a comprehensive evaluation, a questionnaire was designed and was completed during the testing process. The questions addressed various aspects of the haptic feedback experience, including its consistency, realism, and responsiveness to interaction speed variations. The feedback modes and feature types were tested to identify the best-performing configurations. Ten participants took part in the evaluation and they were both male and female aged between 22 and 63.

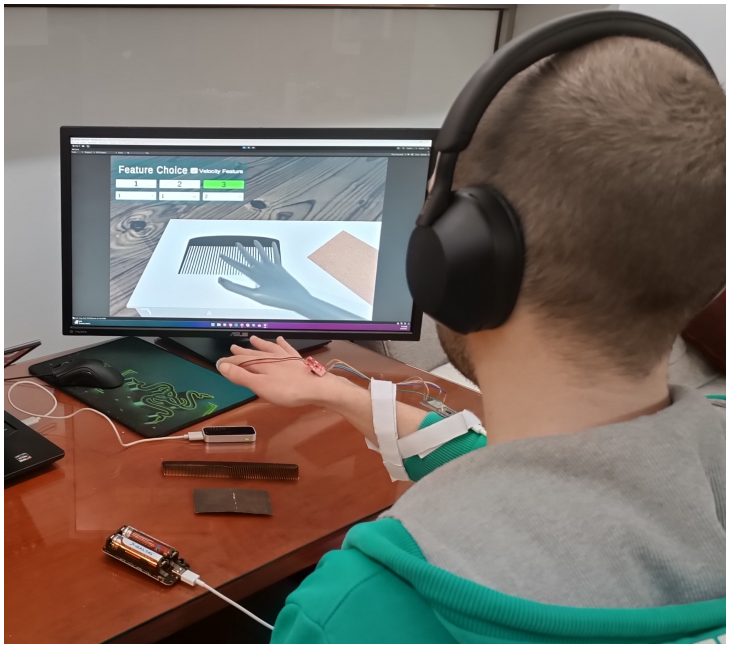


Figure 5.1: User Evaluation process where a user is interacting with a virtual texture wearing noise-canceling headphones.

Participants wore noise-canceling headphones throughout the testing sessions, to minimize auditory bias. The Linear Resonant Actuator was mounted on the fingertip of the participants using Velcro straps, while the ESP32-S3 microcontroller and the DA7280 haptic driver were secured to their arm for a completely wireless experience. The evaluation was conducted in a custom virtual environment developed in Unity as thoroughly

explained in the Implementation Chapter of this thesis, which allowed users to interact with virtual textures while receiving real-time haptic feedback. The Unity environment wirelessly communicated with the ESP32-S3, which controlled the LRA based on feature pairs of frequency and amplitude extracted from the recorded signals, with the selection of the CSV file driving the LRA being controlled through a Unity User Interface (UI). To prevent bias, the UI used numbers instead of descriptive labels for modes and feature types. The evaluation process consisted of three main stages:

- Familiarization with the real and virtual textures.
- Texture identification in a simplified interaction scene.
- Systematic testing of different feedback modes and feature types.

5.2.1 Evaluation Process

Initially, participants were introduced to the three real world textures that were used during the data capture phase and were asked to interact with these textures by sliding their index finger across the surfaces, from left to right, to develop an understanding of the physical sensations of each texture.

Then, participants were introduced to the Unity virtual environment, which included 3D models of the textures placed on a table. Using the Leap Motion controller for hand and finger tracking, participants interacted with each of the virtual textures to familiarize themselves with the vibrotactile feedback and the Leap Motion interface. During this phase, velocity modulation was activated, and the feedback configuration was set to the fusion mode with dedicated feature pairs. This setup was chosen as it was expected to provide the most realistic representation.

After familiarization, participants were moved to a simplified interaction scene within the Unity environment, that featured three identical all white cubes, each corresponding to one of the textures presented in a random order. This setup aimed to eliminate visual bias, requiring participants to rely solely on haptic feedback to identify each texture. Participants slid their fingers across each cube and identified which cube represented each texture. The fusion mode with dedicated feature pairs and velocity modulation was used during this stage to maintain consistency.

In the next stage, participants returned to the main scene to evaluate the different feedback modes and feature types for each texture. For each feedback mode (audio, accelerometer, and fusion), participants interacted with all three textures and rated the realism of each texture's feedback on a scale of 1 to 5. The modes were presented in a randomized order, with participants unaware of which mode they were experiencing at any given time. Velocity modulation was activated, and the dedicated feature pairs were selected for all modes.

Next, participants evaluated the effect of common versus dedicated feature pairs. For this phase, the fusion mode was used exclusively, as it was anticipated to deliver the

most realistic feedback. Velocity modulation was disabled to standardize the feedback across both feature types and prevent bias from user input. Participants interacted with each texture using both common and dedicated feature pairs and rated the realism on a scale of 1 to 5. This phase aimed to determine whether dedicated features significantly enhanced realism and whether common features could provide adequate representation.

During the mode evaluation, each mode was tested sequentially across all three textures before proceeding to the next mode. Conversely, when evaluating the feature type both types were tested on the same texture before moving to the next texture. This approach was chosen because during the think-aloud method, users reported perceiving greater differentiation in the texture features when changes occurred sequentially within the same texture. By structuring the experimental flow this wat, participants were able to compare feedback in real time, without relying on memory of earlier textures, allowing for more accurate and immediate assessment.

5.2.2 Results

Participant responses were collected in real-time and the data gathered provided insights into the performance of different feedback modes and feature types, as well as the overall realism and effectiveness of the haptic feedback system.

In figure 5.2, we can see for each texture how many people identified them correctly. Overall, the identification task was successful as the identification rate was 80 to 90 % for each texture.

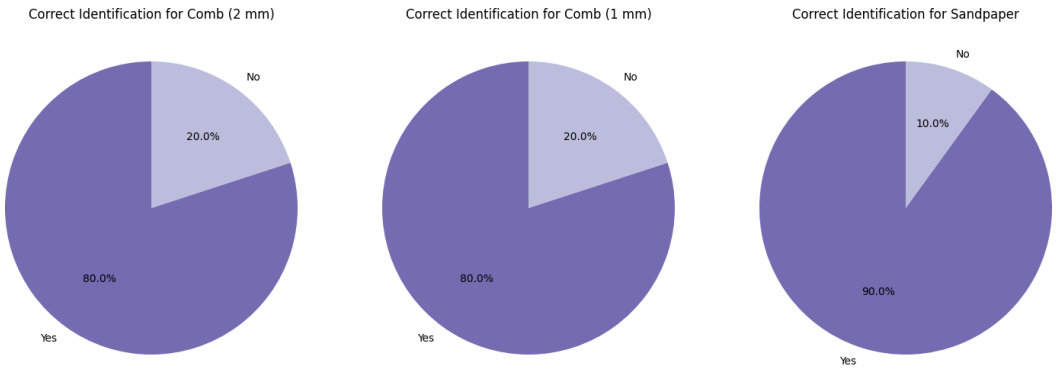


Figure 5.2: Correct Identification Rates for Each Texture Based on Feedback.

Figure 5.3 shows how realistic was the feedback for each mode and each texture. Based on the results, it appears that the Fusion Mode provided the most realistic representation across all textures. This finding is not surprising, as the Fusion Mode has the advantage of combining the strengths of both signals resulting in enhanced texture representation.

Also, it can be observed that the Accelerometer Mode resulted in better representation the two Combs, while the Audio Mode performed best for the Sandpaper representation. This can be explained by the fact that the Sandpaper is a fine texture that exhibits richer spectral characteristics that can be better captured and represented by audio features. The Combs on the other side, are coarser textures that produce more distinct vibrations that can be accurately captured by the accelerometer.

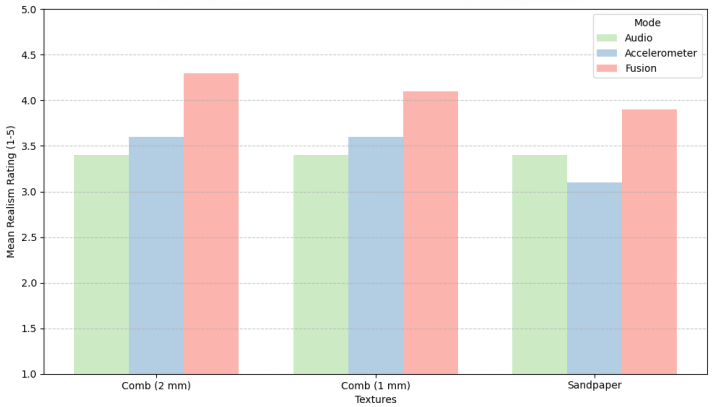


Figure 5.3: Mean Realism Ratings for Feedback Modes Across Textures.

To cross-validate the results about the performance of different modes across the textures, an additional question was included that prompted the participants to indicate the mode that provided the best representation for each texture. The responses to this question were aligned with the results from the previous questions.

Figure 5.4 shows the feedback effectiveness of Common and Dedicated feature pairs for each texture. The difference between the two feedback types is negligible for the two Comb textures. The difference in the Sandpaper texture is more pronounced, as the Dedicated feature pairs scored higher values, with the mean rating difference between the two types ranging from about 3.25 to 4.1. This difference can be attributed to the complexity of the Sandpaper's surface, which features microstructures that produce high frequency vibrations when interacted with. The Dedicated feature pairs are tailored to capture these specific patterns leading to a more accurate and realistic haptic representation, in comparison to the Common feature pairs that are more general. On the other hand, Comb textures have regular coarse patterns that generate more straightforward vibrations, resulting in simpler more uniform textures that can be represented with Common feature pairs more accurately. Interestingly, when asked which approach provided the most realistic feedback overall between the two types, the majority of participants (70%) chose the Dedicated feature pairs.

These results are complemented by those in figure 5.5, where most participants reported that the use of Dedicated feature pairs marginally improved the realism of the feedback. In particular, 6/10 participants noticed an improvement in Sandpaper texture

when using the Dedicated feature pairs, 5/10 when using the Comb 1mm and 4/10 when using the Comb 2mm. However, 9/10 users rated from 1 to 3 out of 5 when asked if the improvement in feedback when using Dedicated feature pairs justifies the additional effort it requires. Taking all the above into consideration, while the Dedicated feature pairs demonstrated some benefits, the Common feature pairs were found to provide sufficient realism and accurate representation across textures. Considering the extra effort and time required to extract dedicated features for each texture and given the small differences between the two types and the user feedback, the Common feature pairs can be chosen as the preferred solution for representing textures based on feature extraction and direct mapping to vibration parameters.

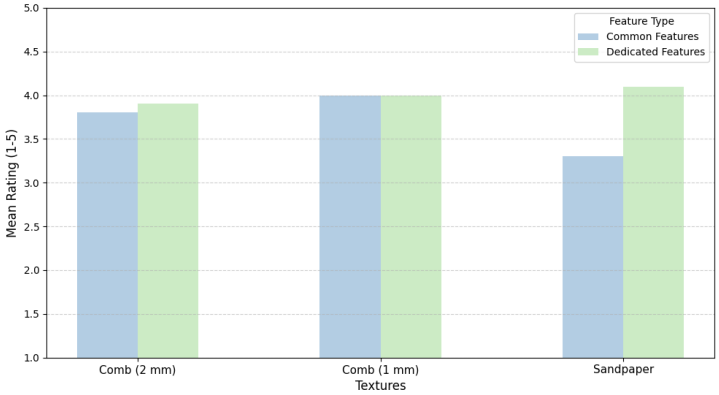


Figure 5.4: Mean Ratings of Feedback Effectiveness of Common and Dedicated Feature Pairs Across Textures.

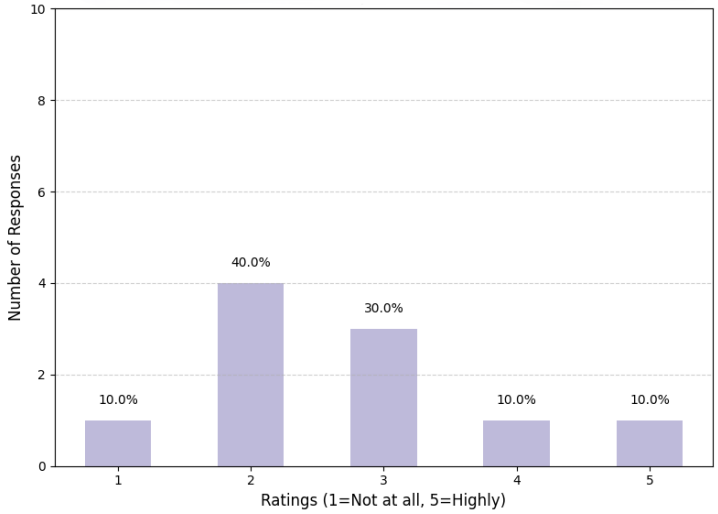


Figure 5.5: Overall Ratings for Realism Improvement with Dedicated Feature Pairs.

When it comes to the impact of interaction speed in the texture representation experience, users reported that the feedback felt natural and consistent as they varied their interaction speed across all textures, as shown in figure 5.6. Additionally in figure 5.7, on a scale of 1 to 5, users gave an average rating of 4 for the system's responsiveness to interaction speed changes across all textures. When comparing the use or absence of speed modulation, it was clear that the use of speed modulation with user's interactions increased the overall realism of the haptic texture feedback.

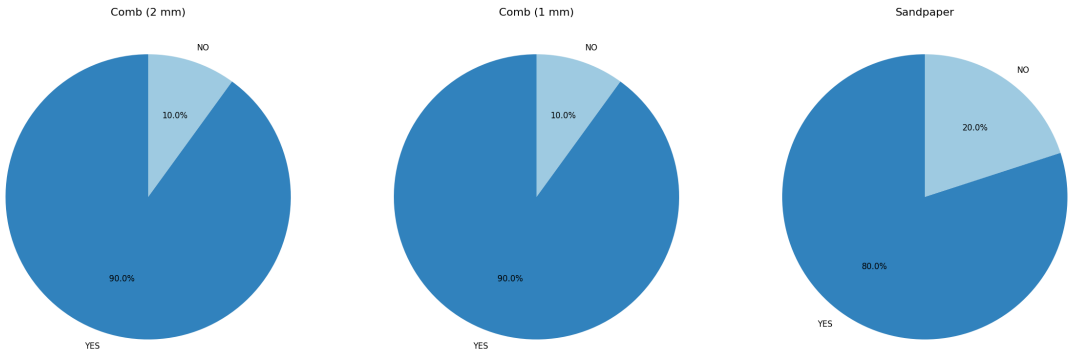


Figure 5.6: Perceived Consistency and Realism of Feedback During Interaction Speed Variations Across Textures.

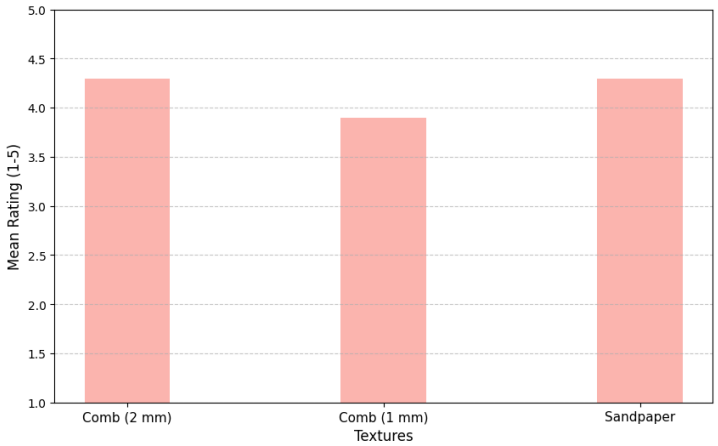


Figure 5.7: Mean Ratings of Effectiveness of Feedback in Adapting to Interaction Speed Changes Across Textures.

As a reminder, the texture representation was created by extracting features from small concatenated signal segments and looping them while the users interacted with the textured surfaces allowing for continuous feedback. Therefore it was important to evaluate the existence of any unnatural repetitions. In figure 5.8, it is shown that users could

sometimes identify looping artifacts, especially in Comb with 1 mm spacing. However, when asked to rate how distracting these artifacts were from 1 to 5 for each texture, most users answered with 1, indicating that these looping artifacts are not distracting at all.

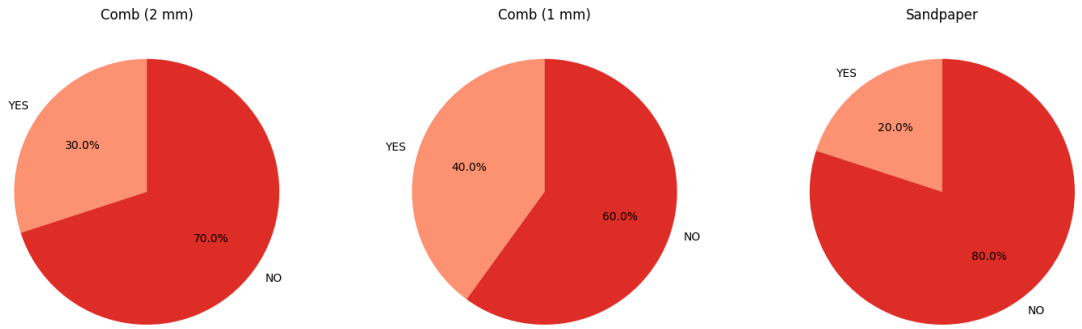


Figure 5.8: Unnatural Repetitions (Looping Artifacts) Detected Across Textures.

Finally, the overall results for the realism of the feedback for each texture are shown in figure 5.9. Users found the sandpaper representation highly realistic with mean ratings close to 4.5 out of 5. The two Combs were also rated around 4, which shows the system's success at representing all the textures accurately. Similarly, figure 5.10 presents user opinions on whether the haptic feedback was seamless and continuous during their interactions, with the results showing a very similar pattern to the realism assessment.

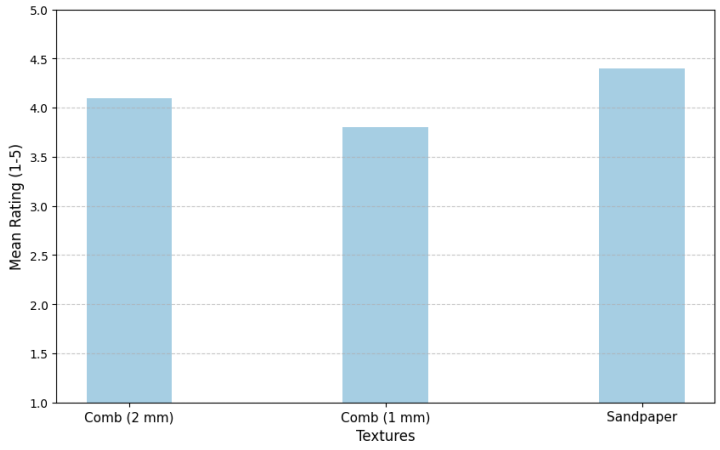


Figure 5.9: Mean Ratings for Perceived Realism of Haptic Feedback for Each Texture.

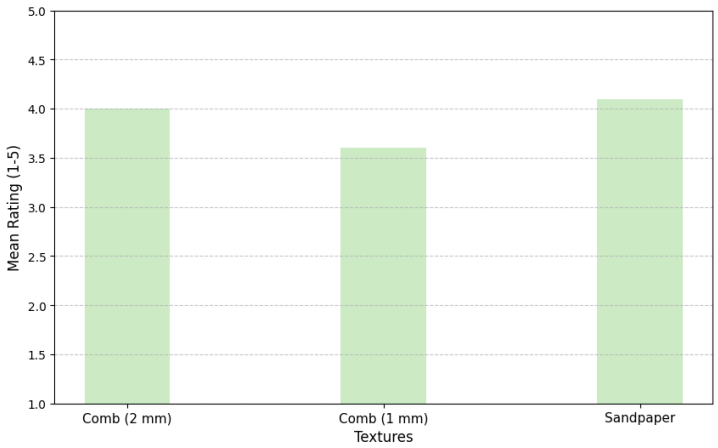


Figure 5.10: Mean Ratings of How Seamless and Continuous the Feedback Felt During Interaction Across Textures.

Chapter 6

Conclusions & Future Work

This chapter concludes the thesis by summarizing its main findings and contributions in the field of haptic feedback and texture representation. The discussion highlights the effectiveness of the proposed texture haptic feedback system, while addressing the challenges encountered during implementation and testing. Finally, directions for future research are proposed to address existing limitations and explore new opportunities in the field.

6.1 Conclusions

In this thesis, a wireless system for capturing, processing, and representing textures through vibrotactile feedback was developed. The system captures texture data through simultaneous recordings of accelerometer and microphone sensors during bare-fingertip interactions with real-world textures. These signals are preprocessed and meaningful features are extracted from them, for direct mapping to vibration frequency and amplitude in order to drive a Linear Resonant Actuator (LRA). This results into a multisensory approach to texture representation, combining the spectral richness of audio data and the vibrational precision of accelerometer readings. The primary goal of this work was to evaluate and compare the effectiveness of audio-only, accelerometer-only and fused features in providing realistic texture haptic feedback. Additionally, this thesis aimed to determine whether Dedicated feature pairs tailored for each texture are necessary for accurate representation, or if Common feature pairs can effectively represent multiple textures. Through this comparison, this thesis aimed to determine which approach delivers the most perceptually meaningful and immersive texture representation.

The system's user evaluation, conducted in a custom virtual environment, demonstrated that the system performed well in terms of realism, consistency and responsiveness. Participants achieved high texture identification rates of 80–90% across all modes, highlighting the system's success in accurately representing the textures. The Fusion mode, which combined features from both audio and accelerometer signals, consistently

provided the most realistic texture representations across the evaluated textures. These results highlighted the importance of multimodal feedback in creating immersive haptic experiences. Additionally, hand velocity modulation was shown to significantly enhance feedback realism, emphasizing its importance in dynamic texture interactions, despite its occasional misbehavior.

The feature selection and mapping processes revealed valuable insights into how specific features contribute to texture representation. For instance, audio features were more effective for fine textures like Sandpaper, while accelerometer features outperformed the audio in capturing the distinct patterns of coarser textures like the Combs. Dedicated feature pairs offered improvements in realism for more complex textures and were favored by most users, but Common feature pairs were sufficient for most applications, balancing simplicity and performance.

Despite its success, the system faced some challenges. Minor looping artifacts were occasionally detected particularly in tightly spaced textures. While these artifacts were not found to be distracting by most participants, they were still perceptible and could affect the overall experience. Additionally, while the mapping process was effective, it required manual selection and refinement of features, which could limit scalability and adaptability in real-time applications. The evaluation was conducted with a small dataset of three textures and a limited participant number of ten users, which may restrict the generalizability of the findings. These constraints provide opportunities for future research and development to enhance the system's performance and adaptability.

6.2 Future Work

This section explores potential enhancements and expansions to the haptic feedback system developed in this thesis, addressing current limitations and identifying opportunities for advancing the field of texture representation.

The outcomes of this project provide a foundation for incorporating machine learning in future iterations of the haptic feedback system. By insights gained from feature mapping and user evaluations, machine learning algorithms could optimize feature selection, mapping processes, and texture representation.

The feature selection stage identified specific frequency and amplitude mappings that were most effective for different textures, revealing patterns that machine learning algorithms could use to automate and optimize feature pairing. For instance, AI models could analyze the relationships between features like Spectral Centroid or Peak-to-Peak Distance across various textures to uncover broader principles for feature selection. Additionally, user evaluation results, such as feedback on realism, consistency and the effectiveness of fusion modes, could inform reinforcement learning models that iteratively refine feedback mappings based on user satisfaction.

These insights could also enable the development of adaptive systems capable of dynamically adjusting feedback based on interaction context. For example, generative ma-

chine learning models trained on the evaluated data could create new, realistic textures for unexplored materials, while predictive models could anticipate user actions and fine-tune haptic feedback in real time. Furthermore, the feedback on the trade-offs between common and dedicated feature pairs could guide the design of hybrid approaches. Systems could use common features as a baseline but dynamically switch to texture-specific features for complex or fine-grained surfaces.

Additionally, the current evaluation was limited to three textures, all of which were hard surfaces and involved ten participants. To enhance the generalizability of the findings, future work should expand the dataset to include a wider variety of both hard and soft textures such as fabrics, foams, and rubber, to capture a broader range of material properties. Additionally, increasing the number of participants and incorporating a more diverse demographic range would provide statistically robust insights and improve the system's adaptability to different user profiles and preferences.

Minor looping artifacts were occasionally detected, particularly in textures with tightly spaced patterns. Investigating advanced crossfade techniques to optimize transition smoothing between looped segments could eliminate abrupt changes that might break immersion.

In the current Fusion mode, a single feature from one sensor is mapped to vibration frequency and a single feature from the other to amplitude. Future work could investigate combining multiple features from one or both sensors to produce these outputs, potentially capturing richer and more complex texture characteristics. For example, features like Spectral Flux and Spectral Centroid from the audio signal could be combined to enhance frequency mapping, while Peak-to-Peak Distance and RMS from the accelerometer could contribute to a more precise amplitude representation. This approach could better reflect textures with both fine and coarse characteristics. Additionally, dynamic fusion strategies where the system adjusts feature combinations based on the texture or interaction context, could improve adaptability and realism. These enhancements would allow the system to handle a broader range of textures with greater perceptual accuracy and create a more immersive haptic experience.

Incorporating additional haptic modalities beyond vibration can significantly enhance the realism and immersion of virtual texture simulations. Real-world texture perception is inherently multimodal, combining various sensory inputs to convey information about an object's properties. For instance, thermal feedback can simulate temperature variations, providing cues about material composition, while force feedback can replicate resistance and compliance, offering insights into an object's stiffness or elasticity. Integrating these modalities into haptic systems allows for a more comprehensive sensory experience, making virtual textures feel more lifelike and broadening the system's applications.

Electromagnetic interference (EMI), as highlighted during the implementation phase, can significantly affect signal integrity, leading to potential inaccuracies in data acquisition. To address this, future improvements could include enhanced shielding techniques such as the use of conductive enclosures to minimize EMI exposure. Employing twisted pair wiring for differential signals can further reduce interference by canceling out noise.

Additionally, incorporating adaptive filtering algorithms could dynamically identify and mitigate EMI artifacts in real time. These advancements would ensure more reliable data acquisition, particularly in wireless setups, and enhance the overall robustness and performance of the system.

Additionally, designing a lightweight, wearable interface would improve user comfort and expand the system's usability in diverse applications. A custom-printed PCB could consolidate components into a compact form factor, reducing wiring complexity and improving signal stability.

Bibliography

[1] R. W. Proctor and J. D. Proctor, "Sensation and perception", in *HANDBOOK OF HUMAN FACTORS AND ERGONOMICS*. John Wiley & Sons, Ltd, 2021, ch. 3, pp. 55–90, ISBN: 9781119636113. DOI: https://doi.org/10.1002/9781119636113.ch3. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/9781119636113.ch3. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/9781119636113.ch3 (cit. on p. 10).

[2] A. Pérez-Bellido, R. D. Pappal, and J. M. Yau, "Touch engages visual spatial contextual processing", *Scientific reports*, vol. 8, no. 1, p. 16 637, 2018 (cit. on p. 11).

[3] C. Lunghi and D. Alais, "Congruent tactile stimulation reduces the strength of visual suppression during binocular rivalry", *Scientific Reports*, vol. 5, no. 1, p. 9413, 2015 (cit. on p. 11).

[4] K. Sathian and S. Lacey, "Cross-modal interactions of the tactile system", en, *Curr. Dir. Psychol. Sci.*, vol. 31, no. 5, pp. 411–418, Oct. 2022 (cit. on p. 11).

[5] H.-C. Sun, A. E. Welchman, D. H. F. Chang, and M. Di Luca, "Look but don't touch: Visual cues to surface structure drive somatosensory cortex", en, *Neuroimage*, vol. 128, pp. 353–361, Mar. 2016 (cit. on p. 11).

[6] H. P. Saal and S. J. Bensmaia, "Touch is a team effort: Interplay of submodalities in cutaneous sensibility", *Trends in Neurosciences*, vol. 37, no. 12, pp. 689–697, 2014, ISSN: 0166-2236. DOI: https://doi.org/10.1016/j.tins.2014.08.012. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0166223614001556 (cit. on pp. 11, 18).

[7] R. L. Klatzky, S. J. Lederman, and C. Reed, "There's more to touch than meets the eye: The salience of object attributes for haptics with and without vision", en, *J. Exp. Psychol. Gen.*, vol. 116, no. 4, pp. 356–369, Dec. 1987 (cit. on p. 11).

[8] C. Spence and N. Di Stefano, "Sensory translation between audition and vision", en, *Psychon. Bull. Rev.*, vol. 31, no. 2, pp. 599–626, Apr. 2024 (cit. on pp. 11, 18).

[9] S. Guest and C. Spence, "Tactile dominance in speeded discrimination of textures", *Experimental Brain Research*, vol. 150, no. 2, pp. 201–207, May 2003, ISSN: 1432-1106. DOI: 10.1007/s00221-003-1404-x. [Online]. Available: https://doi.org/10.1007/s00221-003-1404-x (cit. on pp. 12, 41).

[10] A. Pérez-Bellido, K. Anne Barnes, L. E. Crommett, and J. M. Yau, "Auditory frequency representations in human somatosensory cortex", en, *Cereb. Cortex*, vol. 28, no. 11, pp. 3908–3921, Nov. 2018 (cit. on p. 12).

[11] M. Özcan, U. Baumgärtner, G. Vucurevic, P. Stoeter, and R.-D. Treede, "Spatial resolution of fmri in the human parasylvian cortex: Comparison of somatosensory and auditory activation", *NeuroImage*, vol. 25, no. 3, pp. 877–887, 2005, ISSN: 1053-8119. DOI: https://doi.org/10.1016/j.neuroimage.2004.11.037. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1053811904007220 (cit. on p. 12).

[12] M. D. Fletcher, H. Song, and S. W. Perry, "Electro-haptic stimulation enhances speech recognition in spatially separated noise for cochlear implant users", en, *Sci. Rep.*, vol. 10, no. 1, p. 12 723, Jul. 2020 (cit. on p. 12).

[13] N. Kitagawa, "Link between hearing and bodily sensations", *NTT Technical Review*, vol. 11, no. 12, pp. 30–33, 2013. [Online]. Available: https://doi.org/10.53829/ntr201312fa6 (cit. on p. 12).

[14] S. Merchel and M. E. Altinsoy, "Psychophysical comparison of the auditory and tactile perception: A survey", *Journal on Multimodal User Interfaces*, vol. 14, no. 3, pp. 271–283, 2020. DOI: 10.1007/s12193-020-00333-z. [Online]. Available: https://doi.org/10.1007/s12193-020-00333-z (cit. on p. 12).

[15] J. M. Yau, J. B. Olenczak, J. F. Dammann, and S. J. Bensmaia, "Temporal frequency channels are linked across audition and touch", en, *Curr. Biol.*, vol. 19, no. 7, pp. 561–566, Apr. 2009 (cit. on p. 12).

[16] G. Gescheider, "Some comparisons between touch and hearing", *IEEE Trans. Man Mach. Syst.*, vol. 11, no. 1, pp. 28–35, Mar. 1970 (cit. on p. 13).

[17] S. Connor, "Edison's teeth: Touching hearing", in *Hearing Cultures: Essays on Sound, Listening and Modernity*, ser. Wenner-Gren International Symposium Series, V. Erlmann, Ed., Oxford, New York: Berg, 2004, pp. 153–172, ISBN: 1-85973-823-0. [Online]. Available: https://voidnetwork.gr/wp-content/uploads/2016/09/Hearing-Cultures-Essays-on-Sound-Listening-and-Modernity-edited-by-Veit-Erlmann.pdf (cit. on p. 13).

[18] HST, "Sense of touch", [Online]. Available: https://learning-center.homesciencetools.com/article/skin-touch/ (cit. on p. 13).

[19] D. Purves, G. J. Augustine, D. Fitzpatrick, *et al.*, Eds., *Neuroscience*, 2nd. Sunderland, MA: Sinauer Associates, 2001, ch. Psysiology, Vibratory Sense. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK542288/ (cit. on pp. 13, 14).

[20] D. Purves, G. J. Augustine, D. Fitzpatrick, *et al.*, Eds., *Neuroscience*, 2nd. Sunderland, MA: Sinauer Associates, 2001, ch. Cutaneous and Subcutaneous Somatic Sensory Receptors. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK11162/ (cit. on p. 14).

[21] D. Purves, G. J. Augustine, D. Fitzpatrick, *et al.*, Eds., *Neuroscience*, 2nd. Sunderland, MA: Sinauer Associates, 2001, ch. Mechanoreceptors Specialized to Receive Tactile Information. [Online]. Available: https://www.ncbi.nlm.nih.gov/books/NBK10895/ (cit. on p. 14).

[22] R. S. Johansson and Å. B. Vallbo, "Tactile sensory coding in the glabrous skin of the human hand", *Trends in Neurosciences*, vol. 6, pp. 27–32, 1983, ISSN: 0166-2236. DOI: https://doi.org/10.1016/0166-2236(83)90011-5. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0166223683900115 (cit. on pp. 15, 18, 20, 22).

[23] H. Zuo and M. Jones, "An investigation into the sensory properties of materials", (cit. on p. 16).

[24] F. Blateyron, "The areal field parameters", in Jun. 2013, pp. 15–43, ISBN: 978-3-642-36457-0. DOI: 10.1007/978-3-642-36458-7_2 (cit. on p. 17).

[25] R. L. Klatzky and S. J. Lederman, "Stages of manual exploration in haptic object identification", *Perception & Psychophysics*, vol. 52, no. 6, pp. 661–670, 1992, ISSN: 1532-5962. DOI: 10.3758/BF03211702. [Online]. Available: https://doi.org/10.3758/BF03211702 (cit. on pp. 17, 20).

[26] Y. Yamada, S. Okamoto, and H. Nagano, "Psychophysical dimensions of tactile perception of textures", *IEEE Transactions on Haptics*, vol. 6, pp. 81–93, Jan. 2013. DOI: 10.1109/TOH.2012.32 (cit. on pp. 17, 21, 22).

[27] H.-N. Ho and L. A. Jones, "Contribution of thermal cues to material discrimination and localization", *Perception & Psychophysics*, vol. 68, no. 1, pp. 118–128, 2006, ISSN: 1532-5962. DOI: 10.3758/BF03193662. [Online]. Available: https://doi.org/10.3758/BF03193662 (cit. on pp. 17, 19, 20).

[28] H.-N. Ho, "Material recognition based on thermal cues: Mechanisms and applications", *Temperature*, vol. 5, no. 1, pp. 36–55, 2018, PMID: 29687043. DOI: 10.1080/23328940.2017.1372042. eprint: https://doi.org/10.1080/23328940.2017.1372042. [Online]. Available: https://doi.org/10.1080/23328940.2017.1372042 (cit. on pp. 17, 19).

[29] R. Fagiani, F. Massi, E. Chatelet, Y. Berthier, and A. Akay, "Tactile perception by friction induced vibrations", *Tribology International*, vol. 44, pp. 1100–1110, Sep. 2011. DOI: 10.1016/j.triboint.2011.03.019 (cit. on p. 17).

[30] S. Ding, Y. Pan, M. Tong, and X. Zhao, "Tactile perception of roughness and hardness to discriminate materials by friction-induced vibration", *Sensors*, vol. 17, no. 12, p. 2748, 2017. DOI: 10.3390/s17122748. [Online]. Available: https://doi.org/10.3390/s17122748 (cit. on p. 17).

[31] S. Sharma and H. Grewal, "Tribological behavior of bioinspired surfaces", *Biomimetics*, vol. 8, p. 62, Feb. 2023. DOI: 10.3390/biomimetics8010062 (cit. on p. 17).

[32] J.-H. Kim and S.-P. Kim, "Neuroimaging of tactile information processing", *Investigative Magnetic Resonance Imaging*, vol. 27, no. 1, pp. 1–9, 2023 (cit. on p. 18).

[33] K. Sathian, "Analysis of haptic information in the cerebral cortex", *Journal of Neurophysiology*, vol. 116, no. 4, pp. 1795–1806, 2016. DOI: 10.1152/jn.00546.2015. [Online]. Available: https://doi.org/10.1152/jn.00546.2015 (cit. on pp. 19, 23).

[34] B. Dandu, Y. Shao, and Y. Visell, "Rendering spatiotemporal haptic effects via the physics of waves in the skin", *IEEE Transactions on Haptics*, vol. 14, no. 2, pp. 347–358, 2021. DOI: 10.1109/TOH.2020.3029768 (cit. on p. 19).

[35] F. Rabe, S. Kikkert, and N. Wenderoth, "Finger representations in primary somatosensory cortex are modulated by a vibrotactile working memory task", *bioRxiv*, 2022. DOI: 10.1101/2021.10.29.466459. eprint: https://www.biorxiv.org/content/early/2022/03/10/2021.10.29.466459.full.pdf. [Online]. Available: https://www.biorxiv.org/content/early/2022/03/10/2021.10.29.466459 (cit. on p. 19).

[36] L. R. Manfredi, A. T. Baker, D. O. Elias, *et al.*, "The effect of surface wave propagation on neural responses to vibration in primate glabrous skin", *PLOS ONE*, vol. 7, no. 2, pp. 1–10, Feb. 2012. DOI: 10.1371/journal.pone.0031203. [Online]. Available: https://doi.org/10.1371/journal.pone.0031203 (cit. on p. 19).

[37] S. Brewster and L. M. Brown, "Tactons: Structured tactile messages for non-visual information display", in *Proceedings of the Fifth Conference on Australasian User Interface - Volume 28*, ser. AUIC '04, Dunedin, New Zealand: Australian Computer Society, Inc., 2004, pp. 15–23 (cit. on p. 19).

[38] A. Zimmerman, L. Bai, and D. D. Ginty, "The gentle touch receptors of mammalian skin", *Science*, vol. 346, no. 6212, pp. 950–954, 2014. DOI: 10.1126/science.1254229. [Online]. Available: https://doi.org/10.1126/science.1254229 (cit. on p. 19).

[39] M. Hollins and S. R. Risner, "Evidence for the duplex theory of tactile texture perception", *Perception & Psychophysics*, vol. 62, no. 4, pp. 695–705, 2000, ISSN: 1532-5962. DOI: 10.3758/BF03206916. [Online]. Available: https://doi.org/10.3758/BF03206916 (cit. on pp. 20, 22, 23).

[40] S. J. Lederman, "Tactual roughness perception: Spatial and temporal determinants", *Canadian Journal of Psychology / Revue canadienne de psychologie*, vol. 37, no. 4, pp. 498–511, 1983. DOI: 10.1037/h0080750. [Online]. Available: https://doi.org/10.1037/h0080750 (cit. on pp. 20, 22, 23).

[41] R. L. Klatzky, S. J. Lederman, C. Hamilton, M. Grindley, and R. H. Swendsen, "Feeling textures through a probe: Effects of probe and surface geometry and exploratory factors", en, *Percept. Psychophys.*, vol. 65, no. 4, pp. 613–631, May 2003 (cit. on p. 21).

[42] S. J. BensmaIa and M. Hollins, "The vibrations of texture", en, *Somatosens. Mot. Res.*, vol. 20, no. 1, pp. 33–43, 2003 (cit. on p. 21).

[43] H. Hu and A. Song, "Haptic texture rendering of 2D image based on adaptive fractional differential method", en, *Appl. Sci. (Basel)*, vol. 12, no. 23, p. 12 346, Dec. 2022 (cit. on pp. 22, 37).

[44] R. Grigorii, J. Colgate, and R. Klatzky, "The spatial profile of skin indentation shapes tactile perception across stimulus frequencies", *Scientific Reports*, vol. 12, Aug. 2022. DOI: 10.1038/s41598-022-17324-7 (cit. on p. 23).

[45] A. R. See, J. A. G. Choco, and K. Chandramohan, "Touch, texture and haptic feedback: A review on how we feel the world around us", en, *Appl. Sci. (Basel)*, vol. 12, no. 9, p. 4686, May 2022 (cit. on pp. 23–30).

[46] C. Pacchierotti, S. Sinclair, M. Solazzi, A. Frisoli, V. Hayward, and D. Prattichizzo, "Wearable haptic systems for the fingertip and the hand: Taxonomy, review, and perspectives", *IEEE Transactions on Haptics*, vol. 10, no. 4, pp. 580–600, 2017. DOI: 10.1109/TOH.2017.2689006 (cit. on pp. 24, 28, 30, 31).

[47] F. G. Hamza-Lup, K. Bergeron, and D. Newton, "Haptic systems in user interfaces: State of the art survey", in *Proceedings of the 2019 ACM Southeast Conference*, ser. ACM SE '19, Kennesaw, GA, USA: Association for Computing Machinery, 2019, pp. 141–148, ISBN: 9781450362511. DOI: 10.1145/3299815.3314445. [Online]. Available: https://doi.org/10.1145/3299815.3314445 (cit. on pp. 24, 30–32).

[48] H. Culbertson, S. Schorr, and A. Okamura, "Haptics: The present and future of artificial touch sensation", *Annual Review of Control, Robotics, and Autonomous Systems*, vol. 1, May 2018. DOI: 10.1146/annurev-control-060117-105043 (cit. on p. 25).

[49] J.-L. Rodríguez, R. Velázquez, C. Del-Valle-Soto, S. Gutiérrez, J. Varona, and J. Enríquez-Zarate, "Active and passive haptic perception of shape: Passive haptics can support navigation", en, *Electronics (Basel)*, vol. 8, no. 3, p. 355, Mar. 2019 (cit. on pp. 25, 29).

[50] T. A. Kern, C. Hatzfeld, and A. Abbasimoshaei, *Engineering haptic devices*. Springer Nature, 2023 (cit. on pp. 26, 27, 29).

[51] J. Perret and E. Vander Poorten, "Touching virtual reality: A review of haptic gloves", in *ACTUATOR 2018; 16th International Conference on New Actuators*, VDE, 2018, pp. 1–5 (cit. on p. 27).

[52] O. J. Ariza Nuñez, "Wearable haptic technology for 3d selection and guidance", Ph.D. dissertation, Staats-und Universitätsbibliothek Hamburg Carl von Ossietzky, 2020 (cit. on p. 28).

[53] D. Wang, K. Ohnishi, and W. Xu, "Multimodal haptic display for virtual reality: A survey", *IEEE Transactions on Industrial Electronics*, vol. 67, no. 1, pp. 610–623, 2019 (cit. on p. 29).

[54] M. Strese, J.-Y. Lee, C. Schuwerk, Q. Han, H.-G. Kim, and E. Steinbach, "A haptic texture database for tool-mediated texture recognition and classification", in *2014 IEEE International Symposium on Haptic, Audio and Visual Environments and Games (HAVE) Proceedings*, 2014, pp. 118–123. DOI: 10.1109/HAVE.2014.6954342 (cit. on p. 33).

[55] J. Jiao, Y. Zhang, D. Wang, X. Guo, and X. Sun, "Haptex: A database of fabric textures for surface tactile display", in *2019 IEEE World Haptics Conference (WHC)*, 2019, pp. 331–336. DOI: 10.1109/WHC.2019.8816167 (cit. on p. 33).

[56] H. Culbertson, J. Delgado, and K. Kuchenbecker, "The penn haptic texture toolkit for modeling, rendering, and evaluating haptic virtual textures", Feb. 2014 (cit. on p. 34).

[57] J. M. Romano and K. J. Kuchenbecker, "Creating realistic virtual textures from contact acceleration data", *IEEE Transactions on Haptics*, vol. 5, no. 2, pp. 109–119, 2012. DOI: 10.1109/TOH.2011.38 (cit. on p. 34).

[58] P. Strohmeier and K. Hornbæk, "Generating haptic textures with a vibrotactile actuator", in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, ser. CHI '17, Denver, Colorado, USA: Association for Computing Machinery, 2017, pp. 4994–5005, ISBN: 9781450346559. DOI: 10.1145/3025453.3025812. [Online]. Available: https://doi.org/10.1145/3025453.3025812 (cit. on p. 34).

[59] K. J. Kuchenbecker, J. Romano, and W. McMahan, "Haptography: Capturing and recreating the rich feel of real surfaces", in *Robotics Research*, C. Pradalier, R. Siegwart, and G. Hirzinger, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 245–260, ISBN: 978-3-642-19457-3 (cit. on p. 35).

[60] S. Choi and H. Tan, "Toward realistic haptic rendering of surface textures", *IEEE Computer Graphics and Applications*, vol. 24, no. 2, pp. 40–47, 2004. DOI: 10.1109/MCG.2004.1274060 (cit. on p. 36).

[61] Z. Liu, J.-T. Kim, J. A. Rogers, R. L. Klatzky, and J. E. Colgate, "Realism of tactile texture playback: A combination of stretch and vibration", *IEEE Transactions on Haptics*, pp. 1–10, 2024. DOI: 10.1109/TOH.2024.3355982 (cit. on p. 36).

[62] Y. Kim, S. Kim, U. Oh, and Y. J. Kim, "Synthesizing the roughness of textured surfaces for an encountered-type haptic display using spatiotemporal encoding", *IEEE Transactions on Haptics*, vol. 14, no. 1, pp. 32–43, 2021. DOI: 10.1109/TOH.2020.3004637 (cit. on p. 37).

[63] S. Lu, M. Zheng, M. C. Fontaine, S. Nikolaidis, and H. Culbertson, "Preference-driven texture modeling through interactive generation and search", *IEEE Transactions on Haptics*, vol. 15, no. 3, pp. 508–520, 2022. DOI: 10.1109/TOH.2022.3173935 (cit. on p. 37).

[64] L. Vargas, H. Huang, Y. Zhu, and X. Hu, "Object shape and surface topology recognition using tactile feedback evoked through transcutaneous nerve stimulation", *IEEE Transactions on Haptics*, vol. 13, no. 1, pp. 152–158, 2020. DOI: 10.1109/TOH.2020.2967366 (cit. on p. 38).

[65] O. Halabi and G. Khattak, "Generating haptic texture using solid noise", *Displays*, vol. 69, p. 102 048, 2021, ISSN: 0141-9382. DOI: https://doi.org/10.1016/j.displa.2021.102048. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0141938221000597 (cit. on p. 38).

[66] J. Martínez, A. S. García, J. P. Molina, D. Martínez, and P. González, "An empirical evaluation of different haptic feedback for shape and texture recognition", *The Visual Computer*, vol. 29, no. 2, pp. 111–121, Feb. 2013, ISSN: 1432-2315. DOI: 10.1007/s00371-012-0716-x. [Online]. Available: https://doi.org/10.1007/s00371-012-0716-x (cit. on p. 39).

[67] S. Huang and H. Wu, "Texture recognition based on perception data from a bionic tactile sensor", en, *Sensors (Basel)*, vol. 21, no. 15, p. 5224, Aug. 2021 (cit. on p. 39).

[68] W. Hassan, J. B. Joolee, and S. Jeon, "Establishing haptic texture attribute space and predicting haptic attributes from image features using 1D-CNN", en, *Sci. Rep.*, vol. 13, no. 1, p. 11 684, Jul. 2023 (cit. on p. 40).

[69] A. Abdulali, I. R. Atadjanov, and S. Jeon, "Visually guided acquisition of contact dynamics and case study in data-driven haptic texture modeling", *IEEE Transactions on Haptics*, vol. 13, no. 3, pp. 611–627, 2020. DOI: 10.1109/TOH.2020.2965449 (cit. on p. 41).

[70] N. Heravi, H. Culbertson, A. Okamura, and J. Bohg, "Development and evaluation of a learning-based model for real-time haptic texture rendering", *IEEE transactions on haptics*, vol. PP, Mar. 2024. DOI: 10.1109/TOH.2024.3382258 (cit. on p. 41).

[71] S. Chan, C. Tymms, and N. Colonnese, "Hasti: Haptic and audio synthesis for texture interactions", in *2021 IEEE World Haptics Conference (WHC)*, 2021, pp. 733–738. DOI: 10.1109/WHC49131.2021.9517177 (cit. on p. 42).

[72] D. Beattie, W. Frier, O. Georgiou, B. Long, and D. Ablart, "Incorporating the perception of visual roughness into the design of mid-air haptic textures", in *ACM Symposium on Applied Perception 2020*, ser. SAP '20, Virtual Event, USA: Association for Computing Machinery, 2020, ISBN: 9781450376181. DOI: 10.1145/3385955.3407927. [Online]. Available: https://doi.org/10.1145/3385955.3407927 (cit. on p. 43).

[73] H. Iwata, H. Yano, and H. Igawa, "Audio haptics", in *ACM SIGGRAPH 2002 Conference Abstracts and Applications*, ser. SIGGRAPH '02, San Antonio, Texas: Association for Computing Machinery, 2002, p. 66, ISBN: 1581135254. DOI: 10.1145/1242073.1242101. [Online]. Available: https://doi.org/10.1145/1242073.1242101 (cit. on p. 43).

[74] A. Devillard, A. Ramasamy, D. Faux, V. Hayward, and E. Burdet, "Concurrent haptic, audio, and visual data set during bare finger interaction with textured surfaces", in *2023 IEEE World Haptics Conference (WHC)*, 2023, pp. 101–106. DOI: 10.1109/WHC56415.2023.10224372 (cit. on p. 43).

[75] M. Geronazzo and S. Serafin, *Sonic interactions in virtual environments*. Springer Nature, 2023 (cit. on p. 45).

[76] B. Remache-Vinueza, A. Trujillo-León, M. Zapata, F. Sarmiento-Ortiz, and F. Vidal-Verdú, "Audio-tactile rendering: A review on technology and methods to convey musical information through the sense of touch", en, *Sensors (Basel)*, vol. 21, no. 19, p. 6575, Sep. 2021 (cit. on p. 46).

[77] T. Mudd, "Feeling for sound: Mapping sonic data to haptic perceptions", in *New Interfaces for Musical Expression*, 2013. [Online]. Available: https://api.semanticscholar.org/CorpusID:2863121 (cit. on p. 47).

[78] C. Chen, T. Sühn, M. Kalmar, *et al.*, "Texture differentiation using audio signal analysis with robotic interventional instruments", *Computers in Biology and Medicine*, vol. 112, p. 103 370, 2019, ISSN: 0010-4825. DOI: https://doi.org/10.1016/j.compbiomed.2019.103370. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0010482519302471 (cit. on p. 48).

[79] P. Svensson, C. Antfolk, A. Björkman, and N. Malešević, "Electrotactile feedback for the discrimination of different surface textures using a microphone", en, *Sensors (Basel)*, vol. 21, no. 10, p. 3384, May 2021 (cit. on p. 49).

[80] K. Minamizawa, Y. Kakehi, M. Nakatani, S. Mihara, and S. Tachi, "Techtile toolkit: A prototyping tool for design and education of haptic media", in *Proceedings of the 2012 Virtual Reality International Conference*, ser. VRIC '12, Laval, France: Association for Computing Machinery, 2012, ISBN: 9781450312431. DOI: 10.1145/2331714.2331745. [Online]. Available: https://doi.org/10.1145/2331714.2331745 (cit. on p. 50).

[81] Y. Takeuchi, S. Kamuro, K. Minamizawa, and S. Tachi, "Haptic duplicator", in *Proceedings of the 2012 Virtual Reality International Conference*, ser. VRIC '12, Laval, France: Association for Computing Machinery, 2012, ISBN: 9781450312431. DOI: 10.1145/2331714.2331749. [Online]. Available: https://doi.org/10.1145/2331714.2331749 (cit. on p. 50).

[82] Y. Cho, S. Kim, M. Joung, and J. Lee, "Haptic cushion: Automatic generation of vibro-tactile feedback based on audio signal for immersive interaction with multimedia", Messe Bremen, 2014 (cit. on p. 51).

[83] D. Degraen, B. Fruchard, F. Smolders, *et al.*, "Weirding haptics: In-situ prototyping of vibrotactile feedback in virtual reality through vocalization", in *The 34th Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '21, Virtual Event, USA: Association for Computing Machinery, 2021, pp. 936–953, ISBN: 9781450386357. DOI: 10.1145/3472749.3474797. [Online]. Available: https://doi.org/10.1145/3472749.3474797 (cit. on p. 53).

[84] Wikipedia contributors. "Microcontroller". (), [Online]. Available: https://en.wikipedia.org/wiki/Microcontroller (cit. on p. 56).

[85] IBM. "Microcontroller". (), [Online]. Available: https://www.ibm.com/think/topics/microcontroller (cit. on p. 56).

[86] DigiKey. "A guide for the esp32 microcontroller series". (), [Online]. Available: https://www.digikey.gr/en/maker/blogs/2024/a-guide-for-the-esp32-microcontroller-series (cit. on p. 56).

[87] Waveshare. "Esp32-s3-dev-kit-n8r8". (), [Online]. Available: https://www.waveshare.com/wiki/ESP32-S3-DEV-KIT-N8R8 (cit. on p. 56).

[88] N. Instruments, *White paper: Measuring vibration with accelerometers*. [Online]. Available: https://media.digikey.com/pdf/data%20sheets/National%20Instruments%20Corp%20PDF's/White%20Paper_Measuring%20Vibration%20with%20Accelerometers.pdf (cit. on p. 57).

[89] Endaq, *Vibration measurements: Accelerometer basics*. [Online]. Available: https://blog.endaq.com/vibration-measurements-accelerometer-basics (cit. on p. 57).

[90] Adafruit, *Adxl345 - triple-axis accelerometer*. [Online]. Available: https://www.adafruit.com/product/1231 (cit. on p. 58).

[91] U. Wafer, *Silicon microphone*. [Online]. Available: https://www.universitywafer.com/silicon-microphone.html?srsltid=AfmBOoqGyEK8SBGWoJTl_w2YtArsjj6x-gX4hm_fBGhPDfPHzM7xQN0g (cit. on p. 58).

[92] Knowles, *Sph0645lm4h-1 - mems microphone datasheet*. [Online]. Available: https://www.knowles.com/docs/default-source/default-document-library/sph0645lm4h-1-datasheet.pdf (cit. on p. 59).

[93] DIYMORE. "18650 battery shield v8 mobile power bank 3v 5v for arduino, esp32, esp8266 wifi". (), [Online]. Available: https://www.diymore.cc/collections/hot-sale/products/18650-battery-shield-v8-mobile-power-bank-3v-5v-for-arduino-esp32-esp8266-wifi (cit. on p. 59).

[94] P. MICRODRIVES. "Linear resonant actuators". (), [Online]. Available: https://www.precisionmicrodrives.com/linear-resonant-actuators-lras (cit. on p. 60).

[95] RENESAS. "Da7280 datasheet". (), [Online]. Available: https://www.renesas.com/en/document/dst/da7280-datasheet?srsltid=AfmBOory_A1uEc-JDGbwy9UZNHCCSRcvBl (cit. on p. 62).

[96] ultraleap. "Leap motion controller datasheet". (), [Online]. Available: https://www.ultraleap.com/datasheets/Leap_Motion_Controller_Datasheet.pdf?utm_source (cit. on p. 63).

[97] Microsoft. "Visual studio code documentation". (), [Online]. Available: https://code.visualstudio.com/docs (cit. on p. 64).

[98] PlatformIO. "What is platformio?" (), [Online]. Available: https://docs.platformio.org/en/latest/what-is-platformio.html (cit. on p. 64).

[99] D. Workshop. "Getting started with platformio". (), [Online]. Available: https://dronebotworkshop.com/platformio/ (cit. on p. 64).

[100] Arduino. "What is arduino?" (), [Online]. Available: https://docs.arduino.cc/learn/starting-guide/whats-arduino/ (cit. on p. 64).

[101] C. Staff. "What is arduino, how it works and what you can do with arduino". (), [Online]. Available: https://www.circuitschools.com/what-is-arduino-how-it-works-and-what-you-can-do-with-arduino/ (cit. on p. 64).

[102] freertos. "Rtos fundamentals". (), [Online]. Available: https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals (cit. on p. 65).

[103] Espressif. "Freertos overview". (), [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/system/freertos.html (cit. on p. 65).

[104] Wikipedia. "Unity (game engine)". (), [Online]. Available: https://en.wikipedia.org/wiki/Unity_(game_engine) (cit. on p. 66).

[105] ultraleap. "Ultraleap in unity". (), [Online]. Available: https://docs.ultraleap.com/xr-and-tabletop/xr/unity/index.html?utm_source=chatgpt.com (cit. on p. 67).

[106] PiEmbSysTech. "I2s protocol: Framing, working applications". (), [Online]. Available: https://piembsystech.com/i2s-protocol-framing-working-applications/ (cit. on p. 68).

[107] Espressif. "Inter-ic sound (i2s)". (), [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/peripherals/i2s.html (cit. on p. 68).

[108] Espressif. "Inter-integrated circuit (i2c)". (), [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-reference/peripherals/i2c.html (cit. on p. 69).

[109] M. W. Docs. "An overview of http". (), [Online]. Available: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview (cit. on p. 70).

[110] Postman. "What is http?" (), [Online]. Available: https://blog.postman.com/what-is-http/ (cit. on p. 70).

[111] E. Systems. "Http server". (), [Online]. Available: https://docs.espressif.com/projects/esp-idf/en/v4.2/esp32/api-reference/protocols/esp_http_server.html (cit. on p. 70).

[112] Wikipedia. "User datagram protocol". (), [Online]. Available: https://en.wikipedia.org/wiki/User_Datagram_Protocol?utm_source=chatgpt.com (cit. on p. 71).

[113] A. E. Saddik, M. Orozco, M. Eid, and J. Cha, "Haptics: General principles", in *Haptics Technologies: Bringing Touch to Multimedia*, ser. Springer Series on Touch and Haptic Systems, Berlin, Heidelberg: Springer, 2011, pp. 1–20. DOI: 10.1007/978-3-642-22658-8_6. [Online]. Available: https://doi.org/10.1007/978-3-642-22658-8_6 (cit. on p. 126).