Technical University of Crete
School of Electrical and Computer Engineering

A Lightweight Haptic Glove for Enriched Tactile Feedback, Including Normal Indentation, Lateral Skin Stretch, and Precise Softness and Hardness Kinesthetic Rendering in 3D Interactive Applications

Michail Roumeliotis

A thesis presented for the degree of Master of Science.

Chania, Crete , October 2023

Declaration of Authorship

I, Michael Roumeliotis, declare that this thesis titled, "A Lightweight Haptic Glove for Enriched Tactile Feedback, Including Normal Indentation, Lateral Skin Stretch, and Precise Softness and Hardness Kinesthetic Rendering in 3D Interactive Applications" and the work presented in it are my own. I confirm that:

- This work was done wholly while in candidature for a research degree at this University.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.

Jury

3-member Committe

Professor Aikaterini Mania	ECE, Technical University of Crete
Professor Konstantinos Balas	ECE, Technical University of Crete
Professor Georgios Papagiannakis	UOC, University of Crete

"The value of an idea lies in the using of it."

Thomas Edison

Abstract

Current haptic devices typically rely solely on vibrations to provide tactile feedback, which addresses just one aspect of cutaneous sensation. Moreover, existing devices for kinesthetic feedback often involve bulky and cumbersome exoskeletons that limit users' mobility. This thesis introduces an innovative haptic glove that is lightweight, flexible, and easy to wear. It offers realistic tactile feedback by incorporating normal indentation, lateral skin stretch, and vibrations, along with high-fidelity kinesthetic feedback through strings manipulated by servo motors.

Unlike current systems, this haptic glove is cost-effective, using small vibration motors embedded in the fingertips to deliver tactile feedback. Additionally, it generates normal indentation and shear forces by employing moving platforms to apply pressure to the skin. The kinesthetic feedback is achieved through small strings attached to servo motors placed on the glove, simulating both soft and hard virtual object manipulation. Controlled by a compact microcontroller, the glove receives input from a computer, which sends commands to the motors and actuators.

To evaluate these features, three 3D interactive applications were created using the Unity game engine, where the users performed different tasks and were able to interact and feel various haptic cues. These interactive applications that have been developed can be seamlessly adapted into a 3D or gaming application for Virtual Reality (VR) or Augmented Reality (AR) headsets, as well as for mobile platforms, with minimal software adjustments required.

Study results suggest that users can better perceive directional information and surface geometry when fingertip vibration is incorporated. Furthermore, users excel in distinguishing softness levels when the differences in softness are distinct.

Acknowledgements

I would like to express my sincere gratitude to my thesis supervisor, Professor Katerina Mania, for her invaluable guidance, support, and expertise throughout this research journey. Her mentorship has been important in shaping this work.

I extend my thanks to the committee members, Professor George Papagiannakis and Professor Konstantinos Balas, for their time and effort in reading and evaluating my work.

Furthermore, I would also like to extend my heartfelt appreciation to my friends and family for their constant encouragement and understanding during the ups and downs of my academic pursuit.

Publications

- **M. Roumeliotis, K. Mania. A Lightweight Haptic Feedback Glove Employing Normal Indentation, Lateral Skin Stretch and both Softness and Hardness Rendering. 2023 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct). IEEE, 2023.**

Contents

List of Figures

Listings

Chapter 1

Introduction

1.1 Brief Description

Haptic technology has made remarkable progress in recent years. Its main goal is to recreate the sense of touch, pressure, and other tactile sensations, making it possible for users to interact with digital and remote environments in ways that feel tangible. This technology has found applications across various domains, such as robotics, virtual reality (VR), teleoperation, rehabilitation, and cultural heritage.

The haptic feedback in these domains is significant. In VR, for instance, it has the potential to greatly enhance the user experience by providing a profound sense of immersion and realism [1], [2]. When users can not only see and hear but also feel virtual objects and environments, the boundary between the real and the virtual blurs. Similarly, in teleoperation, haptic feedback gives the ability to users to engage with distant or hazardous environments with a degree of sensitivity and precision that would be otherwise impossible. It allows operators to "feel" and manipulate objects in a remote setting as if they were physically present [3], [4], [5]. In rehabilitation, haptic technology plays an important role in assisting patients to regain their sensory and motor functions. The ability to simulate tactile and kinesthetic sensations aids in the recovery process, making it more engaging and effective [6], [7].

However, despite these promising applications, several challenges persist in the realm of haptics. One of the primary challenges is the design and implementation of wearable haptic systems that have a balance between functionality and user comfort. Many existing devices are cumbersome, heavy, and restrict users' natural range of motion, diminishing the overall user experience [8]. Furthermore, these devices often fail to truly replicate the sense of touch

and force feedback as experienced in the real world, leading to a noticeable gap in simulation fidelity [9]. Affordability is another hurdle because the majority of haptic devices available in the market remain relatively expensive.

Haptic feedback can be categorized into two types: tactile and kinesthetic feedback. Tactile feedback aims to replicate the sensations felt in the fingers, including vibrations, normal indentation, and lateral skin stretch [10], [11]. More specifically, normal indentation refers to the sense of pressure felt on the fingertip by pressing a button, lateral skin stretch is the shear forces applied to the skin to simulate directional information, friction, and curvature display. Kinesthetic feedback, on the other hand, involves replicating sensations related to muscle, joint, and tendon movements through exoskeletons [12], pneumatic actuators [13], etc.

While many devices that provide tactile feedback, focus solely on vibrations, these represent only a small fraction of the sensations that human skin can perceive [14]. Also, existing kinesthetic feedback systems have their own set of limitations, such as discomfort, weight, and restrictions on users' movements [15]. These systems struggle to provide an accurate perception of soft and deformable objects, limiting their utility in various scenarios.

1.2 Purpose of the Thesis

To address these critical challenges and present an innovative solution, our work focuses on the design and implementation of a custom haptic glove that is lightweight, flexible, and easy to wear. Importantly, it offers both tactile and kinesthetic feedback, providing users with a more realistic haptic experience. The tactile feedback includes not only vibrations but also normal indentation and lateral skin stretch, simulating the sensations of interacting with objects of varying textures and shapes. Another feature of our haptic glove is its ability to provide kinesthetic feedback through a system that pulls strings and is attached to the glove. To evaluate the effectiveness of our haptic glove, we have conducted a series of three comprehensive experiments. The first experiment evaluates the capability of the glove to offer accurate feedback on the softness/hardness level of objects. The second experiment was divided into two parts, evaluating the perceived directional information, with and without vibration. Finally, the last experiment focuses on measuring users' perception of the surface geometry, with and without vibrations. Our specific contributions include:

- A custom and flexible haptic device offering tactile feedback in the form of normal indentation by pressing a moving platform against the fingertip. It provides lateral skin stretch

(surface geometry) by applying shear forces to the fingertip and vibration by using an Eccentric Rotating Mass (ERM) motor on each fingertip.

- Unlike previous systems, we present a haptic device providing kinesthetic feedback based on strings attached to the glove which can be pulled to simulate both soft and hard objects.
- We conduct three formal experimental protocols to investigate users' perception of either softness or hardness as well as directional information and surface geometry of virtual objects.
- We determine the accuracy of the perceived haptic feedback and whether vibrations can improve the feedback provided to the users related to directional information and surface geometry.

Figure 1.1: Our Haptic Glove Implementation

1.3 Structure of the Thesis

In the following chapters as well as in this one, the whole thesis is presented in full detail.

Chapter 1 provides a concise overview of the purpose and functionality of the haptics field. It provides an analysis of the current state of haptics used in various fields along with its challenges and limitations. Additionally, it offers a brief description of haptic feedback and its two types: tactile and kinesthetic. The chapter also defines the methodology and the implementation used in this thesis to address the research questions and limitations.

Chapter 2 offers a research overview of various haptic devices. It starts with explaining different dorsal-based haptic devices, then proceeds to analyze devices, where the grounding point

is moved on top of the fingertip(finger-based devices). Then, it analyzes implementations that utilize custom handheld controllers and moves on to present devices used for cultural heritage. Finally, it presents different approaches for haptic feedback, such as dielectric elastomers, chemical haptics, etc. Overall, this chapter provides a comprehensive understanding of the research topics and sets the stage for further exploration and development within the thesis.

Chapter 3 provides an extensive exploration of the technological background and definitions essential for this thesis. It presents different approaches for vibration, such as ERM, LRA, piezo sensors, and shape memory alloys. It also shows how kinesthetic can be achieved. Moreover, it provides an overview of different tracking technologies, such as ultrasonic tracking, magnetic tracking, Inertial Measurement Unit(IMU), Leap Motion, etc. It includes an extensive presentation of the CAD software, the Arduino IDE, and Unity3D that is used in this thesis. This chapter provides a comprehensive understanding of the technical aspects and advancements utilized throughout the thesis.

Chapter 4 offers an exploration of the glove design and implementation. It presents the design procedure followed to design each 3D part placed on the hand. Then, it provides an extensive presentation of the electronic devices used in our implementation. It also covers a detailed analysis of the Unity scenes employed in the experiments. Finally, it provides an insight into the code for each experiment for both Arduino and Unity. Overall, this chapter offers a detailed account of the technical implementation aspects, serving as a foundation for the subsequent chapters of the thesis.

Chapter 5 focuses on the evaluation of the implemented haptic glove, presenting the experiments conducted and the results obtained. Overall, this chapter provides a comprehensive overview of the evaluation process, offering insights into the perception of haptic feedback provided and the level of immersion, and contributing to the overall assessment of the implemented haptic glove.

Chapter 6 provides a comprehensive summary and conclusion of the findings and contributions presented in this thesis. It acknowledges the limitations presented throughout the study. Furthermore, it offers valuable insights and recommendations for future research and potential areas of exploration.

Chapter 2

Research Overview

2.1 Introduction

2.2 Dorsal-Based Haptic Devices

Past research has focused on moving the grounding point of the system on top of the hand closer to the actuation point, developing exoskeletons, i.e., a robotic system blocking the hand's movement when there is contact between the virtual hand and a virtual object.

Figure 2.1: Dexmo Gloves. **Source:** https://www.dextarobotics.com/

Dexmo Gloves [16] are haptic feedback devices designed to provide users with realistic touch and force feedback in virtual reality (VR) and augmented reality (AR) environments. Dexmo gloves are equipped with small vibration motors, placed on the fingertips, that generate vibrations at different frequencies and intensities to replicate the sensation of touching different surfaces and textures. The researchers also integrated pressure sensors, which adjust the vibra-

tion patterns of the actuators accordingly. The gloves can exert resistance against the user's fingers, simulating the sensation of touching and holding objects of varying weight and stiffness. They are wireless with an overall latency of 20-50ms. However, there is no enriched tactile feedback and they are quite heavy and cumbersome. In this thesis, we provide enriched tactile feedback in the form of normal indentation, and lateral skin stretch, and our implementation is lighter to Dexmo since we use strings instead of an exoskeleton.

Figure 2.2: Haptx Gloves DK2. **Source:** [17]

Haptx [17], created a haptic feedback glove that uses microfluidic technology to generate precise tactile feedback on the user's skin. By applying varying pressure and vibrations through microfluidic actuators, the gloves simulate the sensations of touching different textures, surfaces, and objects. For the kinesthetic feedback, they use pneumatic actuators that enable the exotendons to apply up to eight pounds of force per finger. They use magnetic motion tracking with 6 DOF per finger. However, the system requires an air compressor to control the actuators, which makes the interface heavy. Also, the precision of small objects is off. In this thesis, we have implemented a lighter device since we use an actuation system based on strings instead of pneumatic motors that require an external air compressor.

Hosseini et al. [18] introduced a wearable haptic glove called the ExoTen-Glove, which employs a twisted string actuation (TSA) system. This system is designed to provide users with force feedback, as they interact with virtual objects. The glove was designed to be lightweight. The technology behind the ExoTen-Glove is the twisted string actuation system, which consists of two independent actuators. These actuators are equipped with force sensors and small DC motors, allowing them to dynamically adjust the tension in twisted strings. This tension manipulation results in force feedback being transmitted to the user's fingertips during virtual

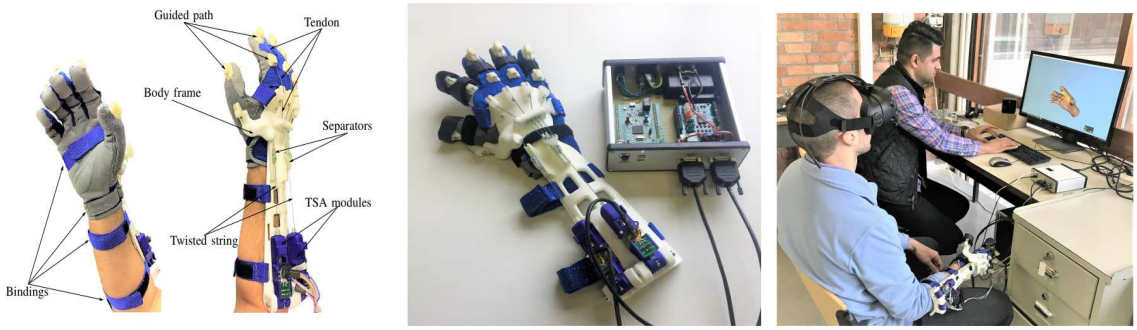

Figure 2.3: Exoten Glove. **Source:** [18]

object-grasping scenarios. To validate the effectiveness of the ExoTen-Glove, the paper presents preliminary results from experimental evaluations. In a virtual reality environment, participants engaged in a specific task where they squeezed a physical spring using their thumb and index finger. This real-world action was then compared to the experience of squeezing a simulated spring with adjustable stiffness. The results of this comparison demonstrated the glove's ability to accurately convey force sensations and simulate varying levels of stiffness. However, the device is quite cumbersome and limits finger motion. It also requires an external control box to operate. In this thesis, we tackle the problem of weight by designing a state-of-the-art actuation system and controlling it without the need for external components.

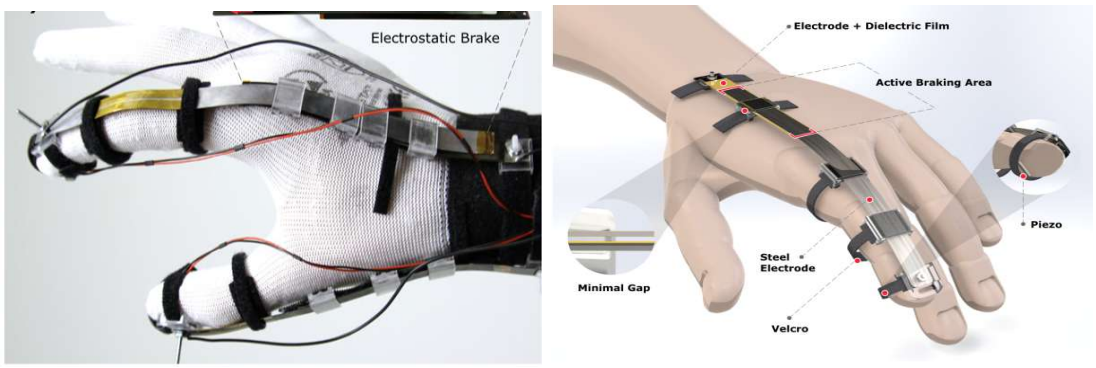

Figure 2.4: a) "DextrES" capabilities overview, b) Mechanical representation of "DextrES" components. **Source:** [19]

Hinchet et al. [19] presented the "DextrES" glove, a lightweight and flexible wearable that integrates kinesthetic and cutaneous feedback in a compact form (weighing less than 8g). The glove employed an electrostatic clutch system that can produce up to 20 N of holding force for

each finger. This force was generated by controlling the electrostatic attraction between flexible elastic metal strips, resulting in an adjustable friction force that provides rapid kinesthetic feedback. The glove placed the electrostatic brake on the index finger and thumb, utilizing 3D-printed guides to ensure smooth movement of the metal strips. Additionally, piezo actuators on the fingertips delivered cutaneous feedback. A controlled experiment showed that "DextrES" enhances the accuracy of grasping across a range of virtual objects. They also reported findings from a psychophysical study that established the thresholds for different levels of holding force discrimination. In this thesis, we provide enriched tactile feedback compared to only vibrations, and kinesthetic feedback for objects with different softness levels.

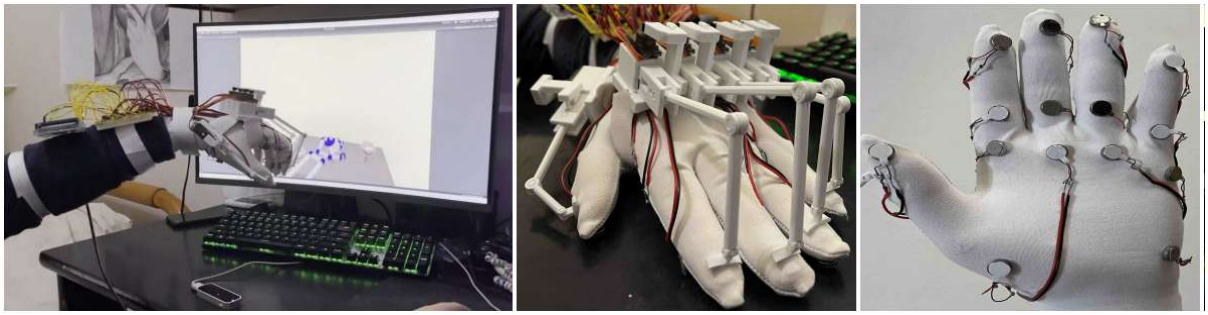

Figure 2.5: a) Implementation during the experiment, b) Implementation front view, c) Implementation side view. **Source:** [20]

Roumeliotis et al. [20] created a wearable embedded system, designed to deliver both tactile and kinesthetic haptic feedback for interactive 3D applications. By combining both types of haptic feedback, the proposed system aims to create a more immersive and convincing interactive experience. The authors developed a printed circuit board (PCB) to house the necessary circuitry, and they opted for cost-effective components to make the system affordable and accessible. For kinesthetic feedback, a 3D-printed exoskeleton is employed, which houses five servo motors on the back of the glove. This setup allows users to experience motion-related feedback. Tactile feedback is facilitated through fifteen coin vibration motors positioned on the inner side of the hand, capable of producing vibrations at three different levels. However, this implementation was cumbersome and fatiguing. The tactile feedback is provided only through vibrations and the kinesthetic feedback does not offer varied levels of stiffness, unlike our implementation.

Efraimidis et al. [21] proposed a wireless embedded system integrated into a glove, offering both hand motion capture and tactile feedback for remote interaction in VR without the need

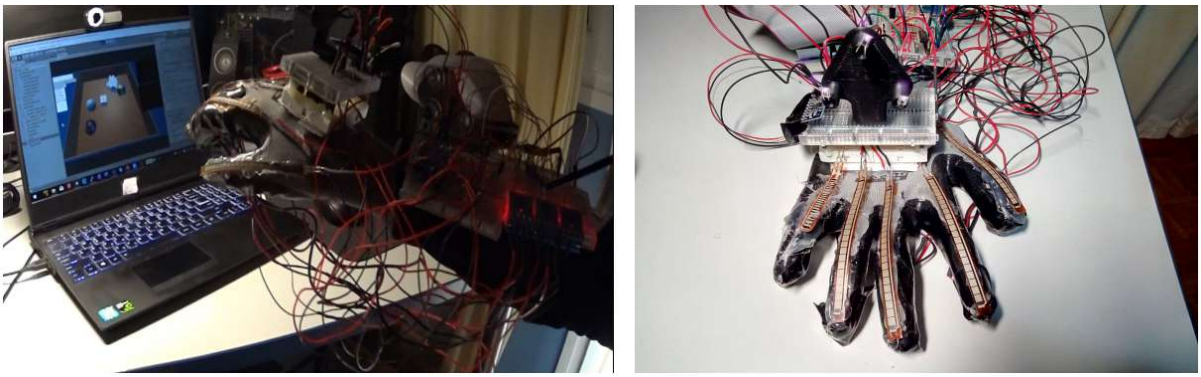

Figure 2.6: a) Implementation during the experiment, b) Implementation front view. **Source:** [21]

for an additional finger-tracking device like the Leap Motion. The key components of the system include an infrared filter pass camera that tracks the user's hand movements. This camera detects infrared LEDs attached to a 3D-printed base on the glove. This configuration eliminates the need for a separate finger-tracking device. The wireless nature of the system removes the restrictions imposed by wired connections, thereby enhancing user mobility and engagement in VR experiences. An accelerometer/gyroscope for the pitch and roll was added and tactile feedback was communicated through five vibration motors attached to the fingertips. However, this implementation lacks kinesthetic feedback. In our work, kinesthetic feedback is provided by strings instead of an exoskeleton, reducing weight. Also, we provide enriched tactile feedback via vibration, normal indentation, and lateral skin stretch, instead of just vibration.

2.3 Finger-Based Haptic Devices

Another approach to haptic systems is to move the grounding point on top of the fingertip to provide enriched tactile feedback, even closer to the actuation point.

Chinello et al. [22] presented a modular wearable finger interface designed to provide users with both cutaneous (tactile) and kinesthetic (force-related) interactions. An embedded system is developed that offers 3-DoF fingertip cutaneous feedback through a mobile platform providing normal indentation and simulation of the curvature on the fingertip. It provides 1-DoF finger kinesthetic feedback on the proximal and distal interphalangeal finger articulations through an exoskeleton that employs a single servo motor anchored to the proximal phalanx. The results from their experiments indicated that integrating cutaneous and kinesthetic feedback through

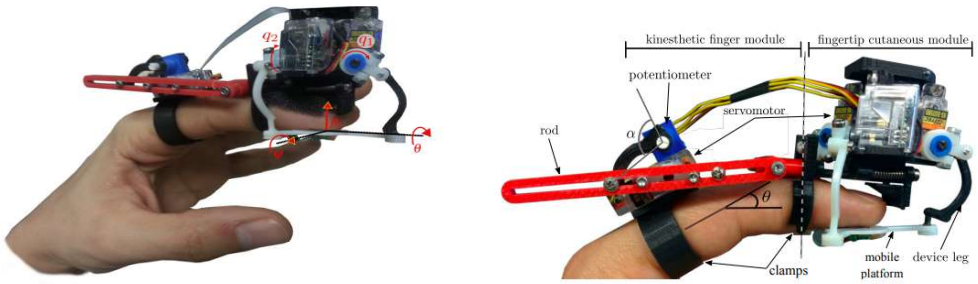

Figure 2.7: a) Kinematics of the moving platform, b) Mechanical components used in this implementation. **Source:** [22]

the wearable interface significantly enhances the performance of all tasks. While cutaneous-only feedback demonstrated promise, the addition of kinesthetic feedback yielded improvements in most performance metrics. However, the interface is cumbersome, difficult to use, and provides haptic feedback to only one finger. In this thesis, we move the actuation point to the dorsal area of the hand to reduce the weight from the fingers, and we provide feedback to all fingers.

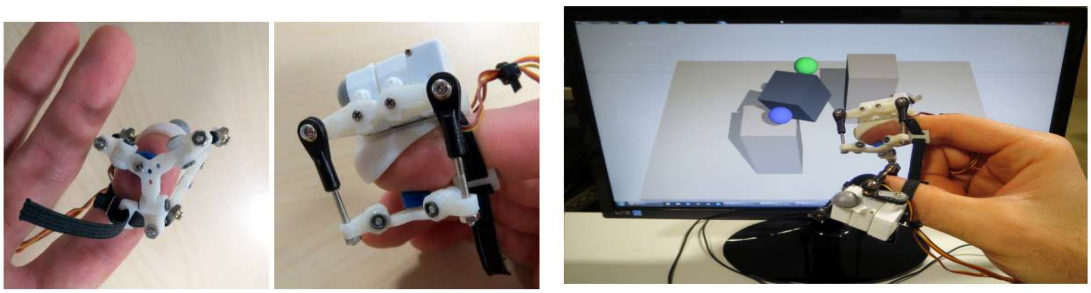

Figure 2.8: a)Device bottom and side view, b)Experimental setup. **Source:** [23]

Leonardis et al. [23] introduced a wearable haptic device designed to manipulate the sensation of contact forces at the fingertips. By utilizing skin deformation across three different directions, the device offered the ability to simulate both contact and non-contact scenarios. This functionality is achieved through a rigid parallel kinematics system. The device stands out due to its distinct three-revolute-spherical-revolute (3-RSR) configuration, which ensures a compact design without compromising the hand's range of motion. To effectively control the device in real time, a unique differential method was introduced to solve the intricate challenge of inverse kinematics. The paper outlined the results of three separate experiments aimed at assessing the device's performance and user interaction. The first experiment involved partici-

pants distinguishing between different directions of fingerpad stretching. The second experiment engaged 19 participants in a virtual manipulation task to evaluate the quality of tactile feedback provided by the device. In the third experiment, 10 participants engaged in a virtual task that simulated lifting and holding objects. The experimental findings indicated that participants exhibited better control over interaction forces when tactile feedback was active, as opposed to relying solely on visual or visuo-haptic cues. In this thesis, we placed the actuation point on the dorsal area of the hand to reduce unnecessary weight from the fingertips. Additionally, our implementation offers kinesthetic feedback for different softness-level objects.

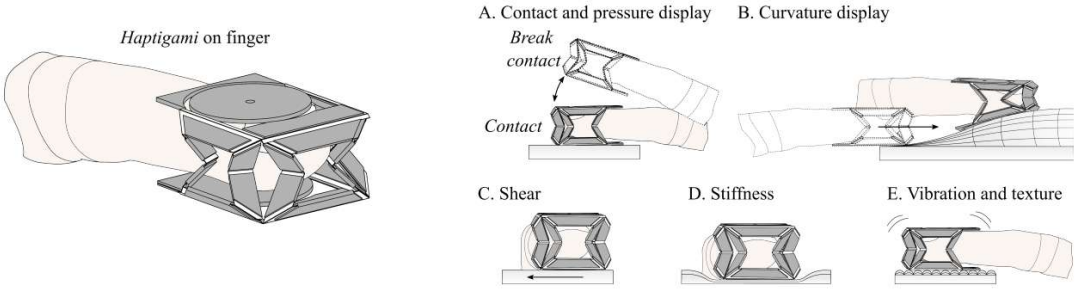

Figure 2.9: a) "Haptigami" overview on finger, b) "Haptigami" types of haptic rendering. **Source:** [24]

Giraud et al. [24] developed a wearable fingertip haptic device named "Haptigami", designed to provide haptic feedback. The 36 x 25 x 26 mm device, which weighs 13g, can provide contact and stiffness display via compression, shear, and curvature display via roll and pitch, as well as texture via vibration. The device consists of an origami-based mechanism, which is powered by two piezo motors each one controlling two embedded slider-crank transmissions. The current prototype of "Haptigami" can generate 678 mN of compression force. It can also produce 400 mN and 150 mN of shear force in the Y and X directions, respectively. However, it cannot achieve compression and pitch simultaneously, since compression requires the use of two motors. Their implementation is also designed for one finger only. In this thesis, we use different motors for each action, and we are able to provide compression and pitch simultaneously.

Salazar et al. [25] explored the enhancement of haptic feedback in Virtual Reality (VR) and Augmented Reality (AR) environments through the combination of passive tangible objects and wearable haptic devices. To achieve this, they created a wearable cutaneous device for

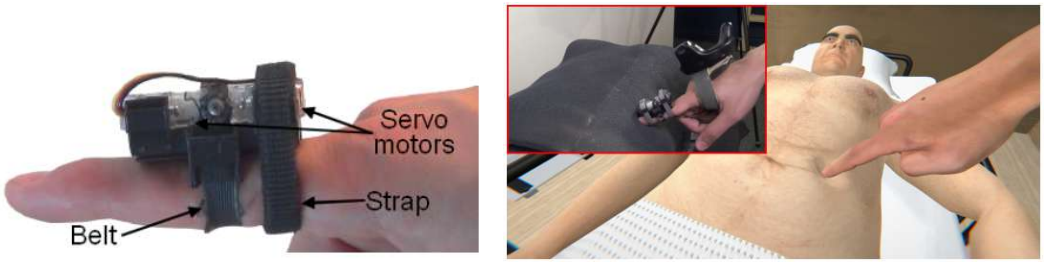

Figure 2.10: a) Wearable tactile interface for the finger, b) Application example. **Source:** [25]

the fingertip providing pressure and skin stretch stimuli by two servo motors and a fabric belt. When the motors rotate in the same direction, shear forces are applied to the finger. When they rotate in a different direction, they provide force to the finger. However, simple passive tangible objects were combined with this device to provide the perception of shape, stiffness, and friction. They conducted three experiments to evaluate the effectiveness of their approach. The results indicate that the compliance of a palpable object can be increased by conforming to the pressure applied through the wearable device. Furthermore, the simulation of bumps and holes is achieved by delivering appropriate pressure and skin stretch sensations. In this thesis, we don't use passive haptics to achieve enriched tactile cues.

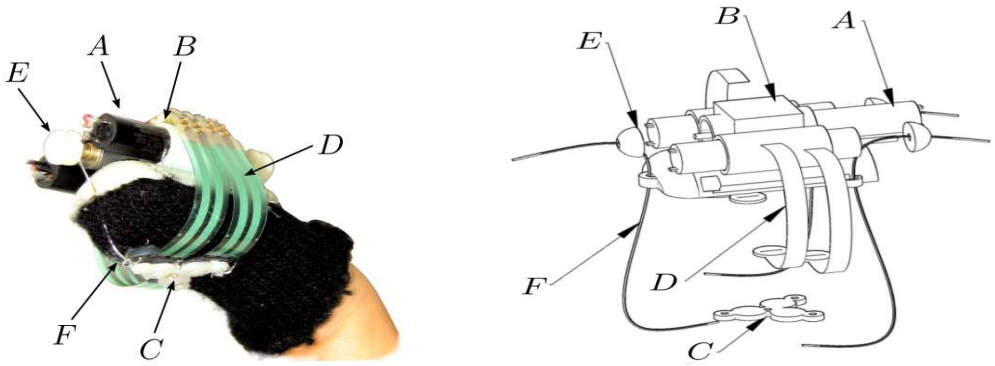

Figure 2.11: a) Prototype device worn on the index, b) CAD sketch of the implementation. **Source:** [26]

Prattichizzo et al. [26] developed a 3-DoF wearable device for tactile feedback that applies normal and tangential shear forces to the fingertip by controlling three cables through three actuators. The device is also composed of two platforms: a static one situated on the back of the finger, and a mobile platform responsible for applying the forces to the finger pad. The wearable haptic device can exert forces of up to 1.5 newtons, with a maximum platform inclination of

30 degrees. The authors conducted an experiment involving curvature discrimination. The wearable device, when used in conjunction with a popular haptic interface, led to improved performance compared to using the haptic interface alone. In this thesis, our implementation provides both kinesthetic feedback and tactile feedback to all fingertips.

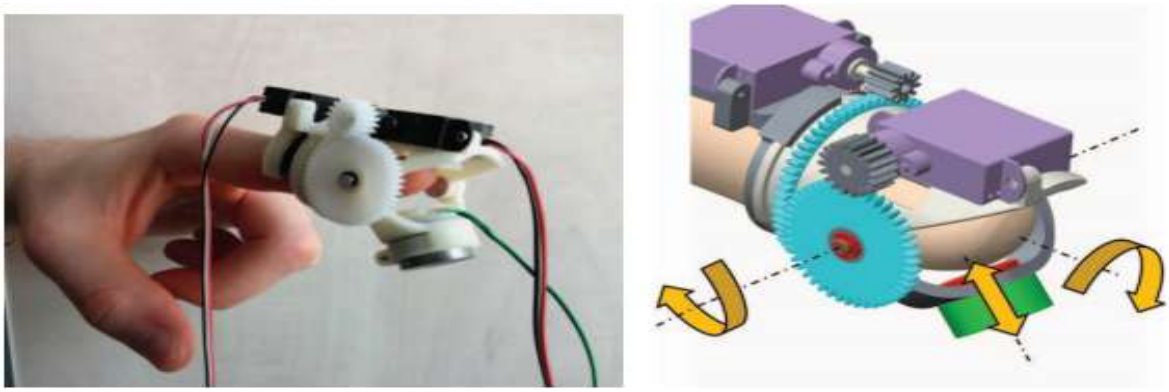

Figure 2.12: a) "Haptic Thimble" mounted on a finger, b) Scheme of the kinematics. **Source:** [27]

Gabardi et al. [27] developed the "Haptic Thimble", a novel wearable haptic device designed for surface exploration. The "Haptic Thimble" is capable of rendering surface orientation and delivering tactile cues with a wide frequency bandwidth, enabling users to interact with surfaces in virtual environments while receiving rich tactile feedback. The device's features allow it to simulate reactive contact, transitions between contact and no contact, collisions, surface textures, and asperities, enhancing the user's perception of the virtual environment. The "Haptic Thimble" achieves these capabilities through a unique serial kinematics design that wraps around the finger. The implementation is actuated by a compact servo motor for the orientation and a custom voice coil for the actuation of the plate in contact with the fingerpad. The performance of the voice coil is evaluated in both static and dynamic conditions, demonstrating its ability to reproduce wide-bandwidth (0-300 Hz) tactile cues. The usability and effectiveness of the "Haptic Thimble" are explored through various experiments. Within a virtual environment, participants use the device to explore virtual surfaces, assessing its overall usability. However, the actuators were placed on the finger, making the system cumbersome and difficult to use for long. In this thesis, we move the placement of the actuators on the dorsal area of the hand and reduce the weight by implementing a string-based system.

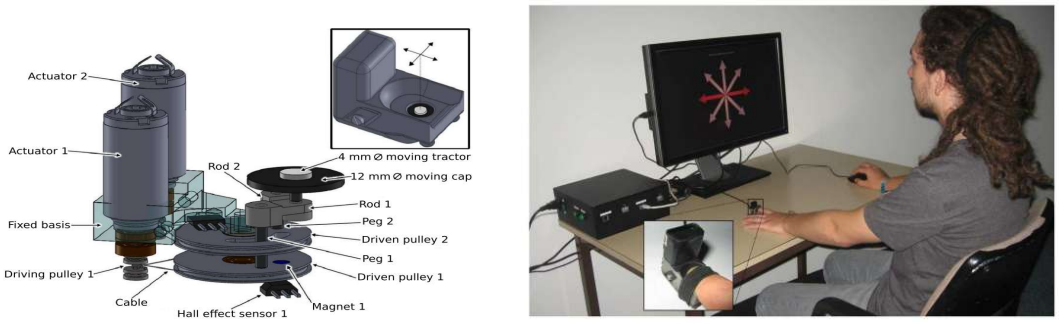

Figure 2.13: a) CAD view of the "HapTip", b) Experimental setup for "HapTip", **Source:** [28]

Girard et al. [28] introduced "HapTip," a compact wearable device capable of generating two Degrees of Freedom (2 DoF) shear forces on the fingertip, allowing for movements within a range of ±2 mm. The modularity of "HapTip" enables multiple devices to collaborate, allowing scenarios where multi-finger or bimanual interactions in virtual environments. The authors present four use cases illustrating how "HapTip" can facilitate interactions involving touching or grasping virtual objects. The results from the experiments demonstrate that participants are able to accurately discriminate the directions of the 2 DoF stimulation provided by the haptic device. Additionally, participants are shown to effectively perceive different weights of virtual objects simulated using two "HapTip" devices. In this thesis, we add kinesthetic feedback and enriched tactile feedback to every finger.

2.4 Handheld Controllers

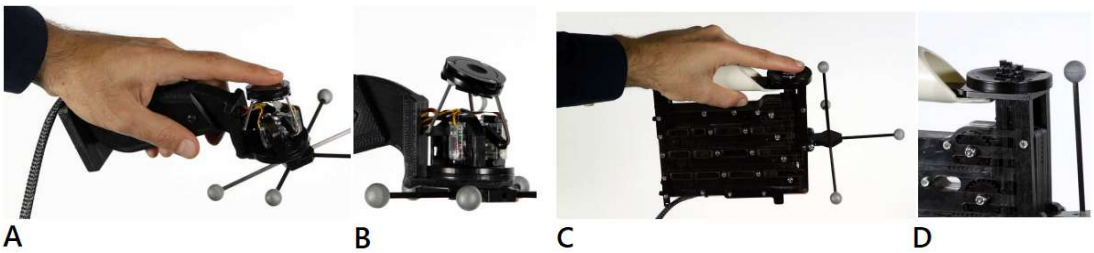

Figure 2.14: a) "NormalTouch" implementation, b) Close-up of the tiltable and extrudable platform, c) "TextureTouch" implementation, d) Close-up of the 4×4 array of pins. **Source:** [29]

Benko et al. [29] designed mechanically-actuated handheld controllers to provide users with

tactile perception of virtual object shapes, textures, and forces that align with their visual representation. The authors introduce two controllers, "NormalTouch" and "TextureTouch," both capable of producing spatially-registered haptic feedback to the user's finger while being tracked in 3D space. "NormalTouch" employs a tiltable and extrudable platform to haptically represent object surfaces and deliver force feedback. On the other hand, "TextureTouch" employs a 4x4 matrix of actuated pins to render the shape of virtual objects, including intricate surface structures. By manipulating these controllers while maintaining finger contact with the actuated platform, users can perceive a more extensive 3D shape perception by integrating the tactile sensations over time. The results from their experiments, demonstrate that haptic feedback significantly enhances the accuracy of VR interaction, particularly when rendering high-fidelity shape output, as demonstrated by the authors' controllers. However, their implementations are cumbersome, limiting users' immersion, since they grab a controller, and the tactile feedback is limited only to the index finger. In this thesis, we designed a wearable and easy-to-wear haptic glove that allows users to grasp virtual objects like in the real world.

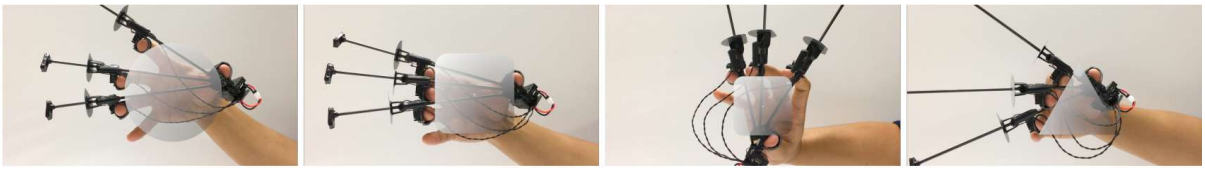

Figure 2.15: Overview of the different shapes simulated with "Wolverine". **Source:** [30]

Choi et al. [30], designed a wearable haptic device to simulate the haptic feedback provided when users grasp rigid objects in virtual reality (VR). The focus was on creating an affordable and lightweight solution. The "Wolverine" creates forces between the thumb and three fingers to simulate "pad opposition" grasps. Using low-power brake-based locking sliders, the system withstands significant forces (up to 100N) between fingers and thumb while consuming minimal energy. The device incorporates sensors for feedback and input: time-of-flight sensors track finger positions, and an IMU monitors orientation. To evaluate the effectiveness of the "Wolverine" interface, the authors conduct user studies comparing the interface's haptic feedback to other hand controllers. The results demonstrate that the "Wolverine" interface significantly improves the users' perception of grasping and manipulating virtual objects, leading to a more engaging and immersive VR experience. However, in this thesis, we provide tactile feedback to all fingertips as well and our device is wearable and easy to use in different case scenarios.

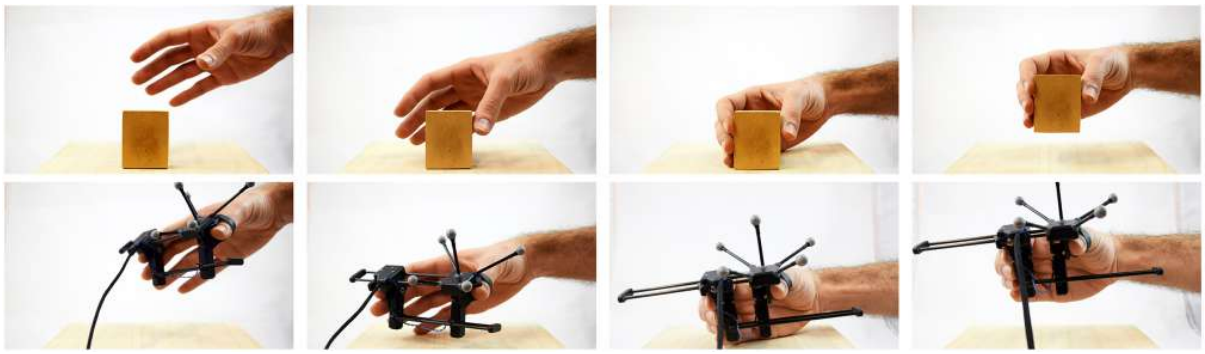

Figure 2.16: Overview of "Grabity" grasping motions. **Source:** [31]

In [31], the authors designed a wearable haptic device to simulate the sensation of grasped virtual objects' rigidity and weight within virtual reality (VR) applications. "Grabity" is created to provide users with a more immersive experience by simulating both the kinesthetic forces associated with a pad opposition grip and the sensation of weight when interacting with virtual objects. The device applies forces to the index finger and the thumb, allowing for a precision grasp. A unidirectional brake mechanism generates the grasping force feedback, enabling tactile sensations. The interface incorporates two voice coil actuators that produce virtual forces tangential to each finger pad by inducing uneven skin deformation. These forces emulate the gravitational and inertial sensations of virtual objects. The paper assesses "Grabity's" performance through two separate user studies. In the first study, the authors explore the device's ability to convey varying levels of weight and force sensations. The second study focuses on users' capacity to differentiate between various weights in a VR environment using "Grabity". However, their implementation hinders user immersion in virtual reality (VR) due to the design not being wearable, unlike our haptic glove. Also, our implementation offers tactile feedback to all fingertips and can be used in different scenarios.

Figure 2.17: Overview of "CLAW" haptic rendering capabilities. **Source:** [32]

In [32], they created a handheld virtual reality (VR) controller that adds features such as force feedback and actuated movement for the index finger. The primary innovation of "CLAW" is its ability to enable three distinct interactions: grasping virtual objects, touching virtual surfaces, and triggering actions. The device incorporates a servo motor connected to a force sensor to deliver controllable forces to the index finger, creating feedback during grasping and touching interactions. Additionally, a voice coil actuator at the fingertip generates synchronized vibrations corresponding to different textures, enhancing the sense of touch. The "CLAW" also provides haptic force feedback during trigger actions, as is common in scenarios involving virtual guns. To assess the performance of the controller, the authors conducted two user studies. The first study gathered qualitative feedback regarding the device's naturalness, effectiveness, and comfort. The second study focused on evaluating the ease of transitioning between grasping and touching modes while using "CLAW". Users reported a heightened level of engagement and satisfaction with their interactions in virtual environments when using the "CLAW". However, this device is heavier than our implementation, not wearable, and limited to one finger.

2.5 Haptic Devices used for Cultural Heritage

Figure 2.18: a) Experimental setup for the implementation, b) The simulated virtual object. **Source:** [33]

Sreeni et al. [33] addressed the issue of virtual representation of cultural heritage objects for haptic interaction. Their main goal is to provide haptic access to art objects at any scale for people with disabilities. This is a low-cost system that, when combined with a stereoscopic visual display, provides a better immersion experience even for people with vision. To achieve this, they propose a simple, multi-layered method of haptic-optical rendering via a proxy for point cloud data, which includes the highly desirable scalability that allows users to adaptively change

the scale of objects during haptic interaction. In the proposed haptic rendering technique, the proxy update loop is executed 100 times faster than the required haptic update frequency of 1KHz. They found that this functionality contributes very well to the realism of the experience.

Figure 2.19: a) Experiment setup, b) The object the user interacts with. **Source:** [34]

Krumpen et al. [34] focused on enriching object inspection in VR with additional haptic feedback to create a tangible heritage experience. To this end, they present an approach for interactive and collaborative object inspection based on VR and annotation. Their system supports high-quality 3D models with accurate reflective properties and additionally provides haptic feedback on the shape of the object based on a 3D printed replica. The digital object is stored in a compact and streamable representation on a central server, which transmits the data to remotely connected users/clients. The latter can collectively perform an interactive inspection of the object in VR with additional haptic feedback provided by the 3D-printed copy. System performance evaluations, the visual quality of the models viewed, and findings from a user study indicate improved interaction, evaluation, and experience of the viewed objects.

Figure 2.20: a) The multitouch table, b) The physical rings that are used for the exhibition. **Source:** [35]

Ma et al. [35] presented a study comparing the behavior of museum visitors to an interactive

exhibition that used physical and virtual objects to explore a large scientific dataset. The exhibit depicts the distribution of phytoplankton in the world's oceans on a multi-touch table. In one version, visitors used physical rings to see the type and proportion of phytoplankton in different regions of the oceans, while in the other version they used virtual rings. The findings suggest that the physical rings allowed for more touching and manipulation and attracted more people, who discussed and exchanged ideas. However, the comparison did not identify measurable differences in the completeness of visitors' interactions, the questions they asked, or their discussions on the topic with others in the exhibition.

2.6 Other Haptic Feedback Approaches

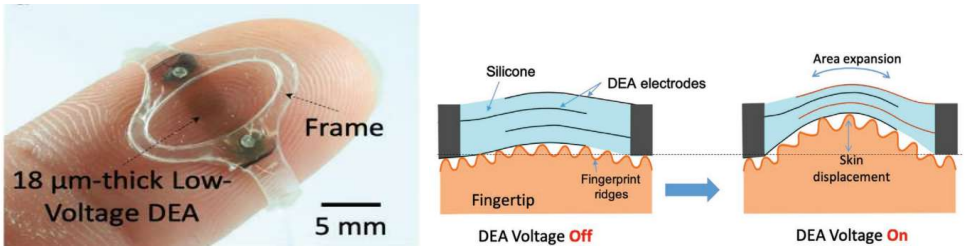

Figure 2.21: a) Close-up to the dielectric elastomer actuator, b) Presentation of the haptic feedback. **Source:** [36]

Ji et al. [36] introduced an innovative approach to wearable haptic interfaces, focusing on "feel-through" haptics using ultra-thin dielectric elastomer actuators (DEA) with an 18 μm thickness. This technology aims to provide tactile feedback for virtual and augmented reality experiences while maintaining a soft and imperceptible feel when not in use. The paper addresses the limitations of current wearable haptic devices, which are often bulky, tethered, and lack high-fidelity feedback. The proposed solution involves ultra-thin DEA applied directly to the skin, enabling users to perceive force and texture changes associated with virtual objects. Experiments were conducted to assess the effectiveness of the approach. The 18 μm thick DEA was found to generate rich vibrotactile feedback across a wide frequency range. Users were able to accurately identify different frequency and sequence patterns when the devices were applied to their fingertips, achieving success rates ranging from 73% to 97%. An untethered version of the technology, weighing just 1.3 grams, allowed blindfolded users to recognize letters by "feeling" them through their fingers. The use of ultra-thin DEA offers a solution to the

challenges of bulky and tethered haptic devices. However, it requires a 500V-1kV power supply to actuate the interface, unlike our proposed haptic glove that requires only 5V to operate.

Figure 2.22: Overview of the implementation that provides haptics through chemicals. **Source:** [37]

Lu et al. [37] explored a novel approach to haptic feedback by using topical stimulants to create tactile sensations. Unlike traditional methods that rely on mechanical actuators or vibrations, this approach leverages chemicals to elicit haptic sensations in a non-invasive manner. While traditional haptic technologies often involve mechanical components that can be bulky or lack precision, the proposed "chemical haptics" approach introduces a different paradigm by using substances applied topically to the skin. By applying these substances to the skin, users can experience haptic feedback that simulates different textures, forces, or even temperature changes. They use Menthol to simulate the sense of cooling, Capsaicin for warming, Sanshool for tingling, and Lidocaine for numbing. However, they do not transition from one sensation to another and sensations do not last for a long time. In this thesis, haptic sensations can last as long as required.

Chapter 3

Technological Background and Definitions

3.1 Haptic Feedback

3.1.1 Types of Haptic Feedback

In order to implement haptic technology, researchers create special devices that provide the appropriate feedback to the user. The type of feedback that can someone find in these devices is **kinesthetic** and **tactile**. Tactile feedback refers to the things you feel in your fingers—vibration, softness, hardness, warmth, while kinesthetic is associated with the things you feel from sensors in your muscles, joints, and tendons—weight, stretch, shape, etc.

3.1.2 Tactile Feedback

Vibration Motors Due to their small size and enclosed vibration mechanism, coin vibration motors are a popular choice for many applications. They are mainly used in applications like smartwatches and other wearable devices to provide tactile feedback. These vibration motors divide into two categories—**ERM**(Eccentric Rotating Mass) and **LRA**(Linear Resonant Actuator).

Eccentric Rotating Mass Motors are small, compact, and lightweight and are mainly found in a wide range of electronic devices, from smartphones and smartwatches to video game controllers and remote controls. They consist of an off-center, non-symmetric mass that is attached to the motor shaft. To activate these motors, an electrical voltage is applied across the motor's terminals. When the motor is powered, it generates a magnetic field inside its

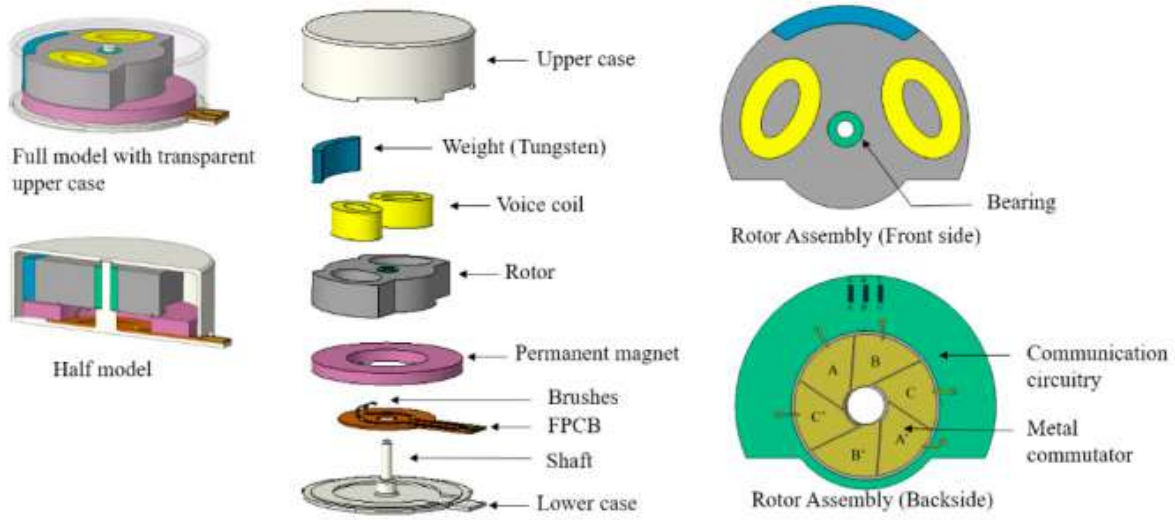

Figure 3.1: ERM Coin Vibration Motor

coil making the armature rotate. As the armature moves, it turns the eccentric mass. The centripetal force is asymmetric, which results in a centrifugal force that displaces the motor. This displacement of the motor causes the vibration. The movement of the vibration could be represented as a sinusoidal wave. The frequency of the wave is the frequency of the vibration which results from the applied voltage. The equation for the centrifugal force is:

$$F = m\omega^2 r$$

From the centrifugal equation we could derive that if the voltage is increased, the vibration frequency (ω) will increase along with the vibration amplitude(F).

Linear Resonant Actuators is a recently developed alternative to ERM. These motors consist of a mass, permanent magnet, voice coil, and spring. These motors are also used in various electronic devices, providing precise and customizable haptic feedback. The core principle of LRA is resonance. Resonance happens when a system vibrates most effectively at a specific frequency. The resonant frequency is determined by the mass, the spring stiffness, etc. Additionally, LRA motors have to work on the resonant frequency because their efficiency and performance drop instead. These motors are driven by alternating current (AC) voltage sources. When voltage is applied to the voice coil, it generates an electromagnetic field. This field interacts with the permanent magnet attached to the mass. This mass is allowed to be moved back and forth by the spring. The movement of the spring is linear along the axis of

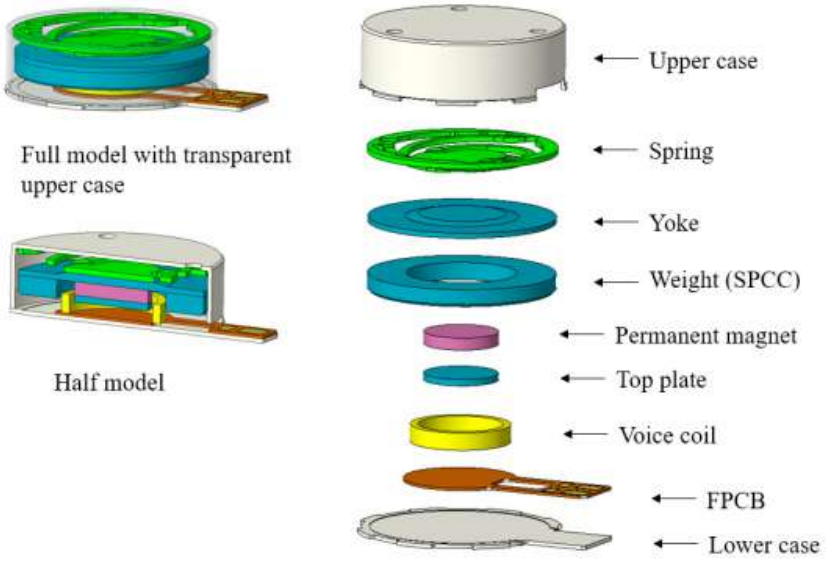

Figure 3.2: LRA Coin Vibration Motor

the LRA. Operating at or near the resonant frequency, these motors consume minimal power. Finally, LRA motors offer precise control over vibration amplitude, frequency, and duration.

Piezoelectric Motors Piezoelectric Actuators are devices that use the piezoelectric effect to convert electrical energy to mechanical motion. Common piezoelectric materials are quartz, lead zirconate titanate (PZT), etc. The piezo motors consist of one or more piezo materials sandwiched between two electrodes. When voltage is applied to these electrodes, it creates an electrical field within the piezoelectric material and it deforms/changes shape. This change is subtle but precise. Moreover, piezoelectric actuators consume very little power when idle, which makes them energy-efficient. Also, these motors don't generate a magnetic field, thus magnetic interference is not a concern. However, these motors require high-voltage electrical signals to operate effectively. Also, piezo motors need complex control circuitry and are quite brittle.

Shape Memory Alloys Shape Memory Alloys(SMAs) are materials that can be used in haptic devices to provide tactile feedback and simulate various sensations. SMAs are able to "remember" and return to a predefined shape when are subjected to stimuli, typically changes in temperature. These materials have two phases: austenite(high-temperature phase) and martensite(low-temperature phase). The amount of force produced by the SMAs depends on

the design of the actuator, the size of the SMA element, and the temperature change applied. In haptics, SMAs are used to simulate sensations like pressing, squeezing, bending, or twisting. These actuators can respond relatively quickly to temperature changes and are compact and lightweight. However, it is challenging to control precisely the temperature in order to transition from one phase to the other. Also, the circuit to control these actuators is complex and will add extra weight to the implementation. Finally, repeating cycles from the austenite to the martensite phase will fatigue the SMAs and will cause degradation to the material.

This thesis used ERM motors to achieve tactile feedback through vibration because they are more available in the market, they are powered with DC voltage, and the circuit to drive them is not as complex as LRA's, the piezo sensors', and the SMAs'.

3.1.3 Kinesthetic Feedback

The haptic applications that use kinesthetic feedback, look to provide force to hands or fingers to recreate the virtual object. The most popular ways that this is accomplished are with electric motors(servos) or pneumatic systems among others.

Figure 3.3: Servo Motor

Servo Motors Servo Motors are small-sized and energy-efficient components that have been around for a long time. The circuit of the servo is built inside the motor unit, which gives motion to the shaft that is mounted on top of the motor. The basic components that the servo consists of are a small DC motor, a potentiometer, and a control circuit. When the DC

motor rotates, the potentiometer's resistance changes. Then, the control circuit can regulate the movement and its direction. Once the shaft reaches the desired position, the power supplied to the motor is stopped. The control of servos is achieved by sending an electrical pulse and the movement is limited to only 180^o.

Pneumatic Systems Pneumatic Systems can provide kinesthetic feedback in haptic applications by using controlled air pressure to create resistance or motion depending on the user's interaction. These systems typically use pneumatic actuators, which are devices that convert compressed air into mechanical motion. These actuators could vary from pneumatic cylinders to inflatable tubes. The key to providing accurate kinesthetic feedback with pneumatic motors is precise pressure control. This requires a compressor. Similar to servo motors, pneumatic actuators operate within a closed-loop control system. However, designing and controlling these systems is complex. Also, due to the usage of a compressor, they consume a significant amount of power. Finally, the design of pneumatic systems often results in heavy and cumbersome systems, which limits the user's immersion.

In this thesis, we used servo motors, since they don't require an external air compressor that adds extra weight to the implementation.

3.2 Tracking Technologies

In haptic interfaces, a very important aspect is to detect the position and rotation of the user's hand in space. This could be accomplished in various ways like optical tracking solutions, commercial solutions, IMU tracking systems, flex sensors, etc.

3.2.1 Ultrasonic Tracking

Ultrasonic tracking is achieved by ultrasonic waves, which are emitted from transmitters and received by receivers, that measure distances and positions in the 3D space. The measurements are done by calculating the time it takes for a signal to travel to an object and back. STRATOS Explore by Ultraleap tracks the hands using the Leap Motion Controller and provides the user with tactile feedback using ultrasonic. This tracking option provides touchless interactions, works in occluded environments, and can enhance immersion by applying haptic sensations to the hand tracking. However, the range of tracking is limited compared to other options, it

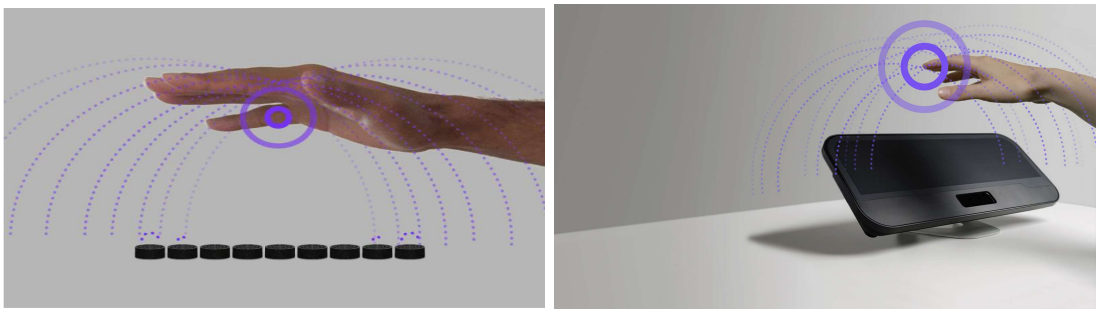

Figure 3.4: Ultrasonic tracking by Ultraleap's STRATOS Explore. **Source:** https://www.ultraleap.com/

requires precise positioning and calibration of ultrasonic sensors, and the accuracy is affected by ambient noise.

3.2.2 Magnetic Tracking

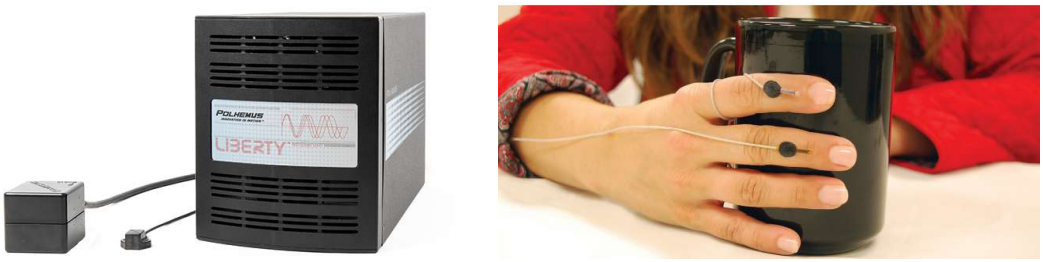

Figure 3.5: a) Magnetic tracking device by Polhemus, b) Sensors that track fingers by Polhemus. **Source:** https://polhemus.com/

Magnetic tracking is accomplished by using electromagnetic fields to track positions and orientations in the 3D space. Polhemus LIBERTY uses a source that emits an electromagnetic field. Then some sensor units are added to the fingers in order to be tracked. When the sensor enters the electromagnetic field, it changes the field's strength and direction at the sensor location. These changes are collected from the sensors as data and then processed by the system's software to determine the positions and orientations of the sensors. This tracking option offers advantages such as high accuracy and low latency tracking, minimal occlusion compared to other optical systems, and sensors that can be hidden from view. However, these sensors are sensitive to metal and magnetic materials in the environment, require calibration for improved accuracy and the setup/configuration of this system is complex.

3.2.3 Inertial Measurement Unit

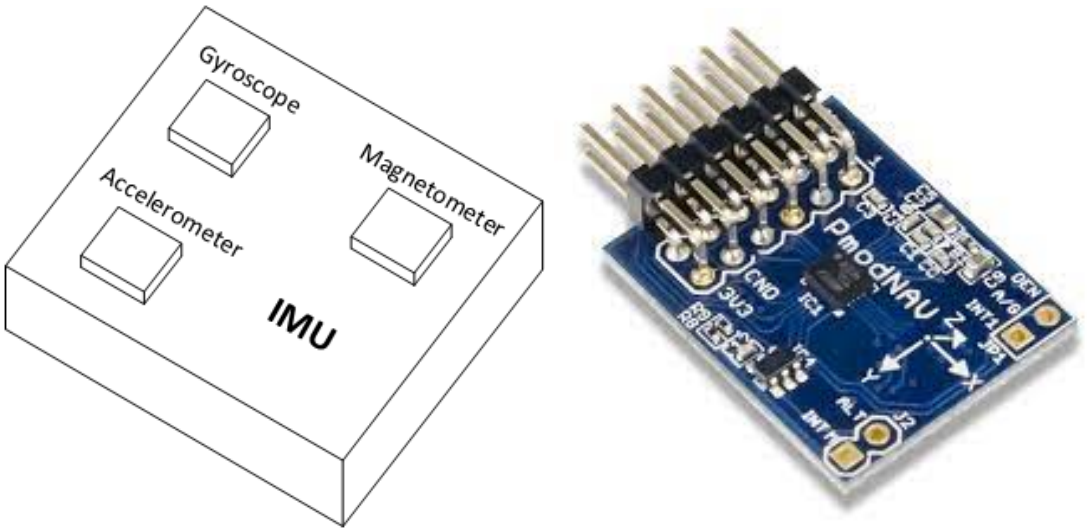

Figure 3.6: Inertial Measurement Units

IMUs are electronic devices, that measure a set of factors with the usage of some tools inside them and offer up to 10 Degrees Of Freedom. These tools could be an accelerometer to measure velocity and acceleration. Accelerometers contain MEMS devices that move in response to acceleration forces. IMUs usually have three accelerometers, one on each axis(X, Y, Z). Another tool used in IMUs is gyroscopes, which measure rotation and rotational rate. Gyroscopes contain MEMS devices that move in response to rotational forces. IMUs usually have three gyroscopes, one on each axis(X, Y, Z) aligned with the accelerometers. Sometimes IMUs contain a magnetometer to establish the direction of the magnetic field. Tracking with IMUs comes with the advantage that is a portable and wearable solution. These sensors also provide real-time tracking with low latency and are suitable for both indoor and outdoor environments. However, IMUs often drift over time, which causes accuracy issues. One additional issue is that the magnetometers in the IMUs are affected by local magnetic interference.

3.2.4 Oculus Quest

Oculus Touch is a commercial solution and provides 3/5 Degrees Of Freedom for every hand. Also, it consists of two handheld units, which have an analog stick, three buttons, and two triggers. Each unit has a ring on the upper side of the device, that contains infrared LEDs. Those LEDs are the ones responsible for the tracking of the controller's position. However, the

Figure 3.7: a) Oculus Quest 2 Controllers, b) Oculus Quest 2 Setup

device focuses on tracking the position and orientation of the controller, thus lacking individual finger-tracking. Also, Oculus Touch relies on cameras placed on the headset for tracking, which could affect the accuracy if the controllers are outside the camera's field of view.

3.2.5 Leap Motion Controller

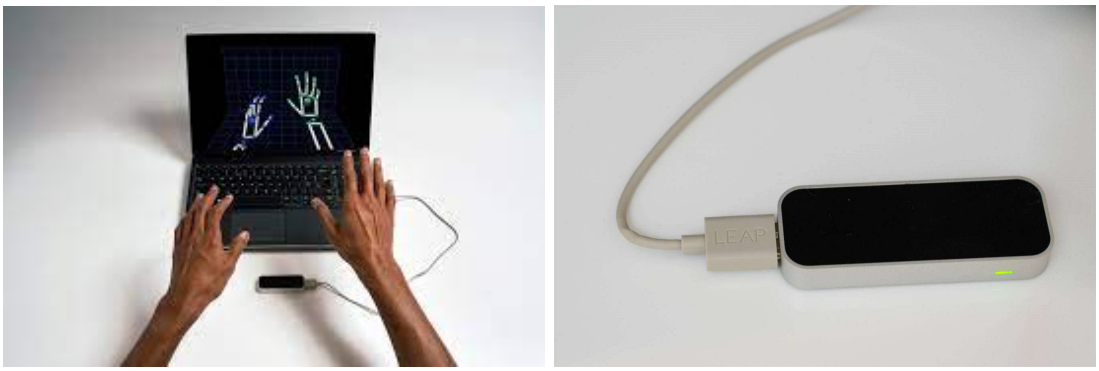

Figure 3.8: a) Setup for the Leap Motion Controller, b) Leap Motion Controller

Leap Motion is an optical tracking solution, that can offer all 27 Degrees Of Freedom of the hands. This device works with two cameras and three infrared LEDs, that track infrared light with a wavelength of 850 nanometers. Also, the controller is capable of tracking hands up to 60cm above the device and within 140^o field of view. Then, the data is sent via USB to the Leap Motion Software, which processes the images and then reconstructs a 3D representation of what the device sees. Although the device provides precise hand/finger tracking and eliminates the need to hold controllers, it has limited tracking area and objects on top of the hand may

hinder the accuracy of the tracking of the hand.

3.3 CAD Software

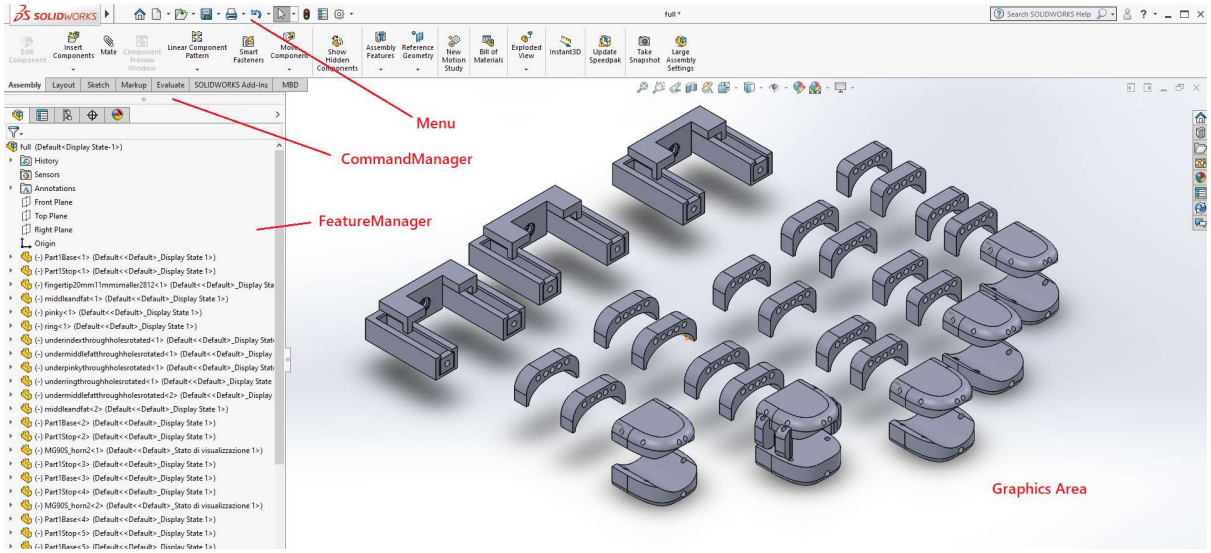

Figure 3.9: SolidWorks Window

In this thesis to create the 3D parts for the kinesthetic feedback, we used SolidWorks. SolidWorks is a Computer-Aided Design(CAD) software, that is used by engineers and designers to create, simulate, and analyze how various types of digital objects fit together and interact. These objects could be from simple 3D objects to a piece of industrial machinery.

3.3.1 Graphics Area

The Graphics Area is central in the SolidWorks window. There the user interacts with the design in a dynamic manner. The Graphics Area shows a real-time visual representation of the 3D model, where the user can clearly view the model's geometry, features, etc. By using the mouse controls, such as rotating, zooming, and panning, the user is able to navigate through the design and view it from different angles and perspectives. Also, the user can interact with his model directly from the Graphics Area. When creating new sketches, the users typically start by sketching in the graphics area.

3.3.2 CommandManager

The CommandManager is located at the top of the SolidWorks window. In this section of the software, the user is able to access a wide range of tools, commands, and functions to create and modify 3D models. It is organized into different tabs, where each one corresponds to a specific task. Some important tabs are the following:

Features Tab In the Features Tab, the user can create or modify features on his 3D models. Some common features used are the following:

- Extrude, which creates a solid or a surface by extending a sketch in a direction.
- Revolve, which created a solid or a surface by revolving a sketch around an axis.
- Sweep, which creates a solid or a surface by sweeping a surface along a path.
- Loft, which creates a solid or a surface by blending two or more sketches along specified curves.

Sketch Tab In the Sketch Tab, the user can create and edit 2D sketches that can be used as a foundation for building 3D models. These 2D sketches are created by tools such as lines, circles, rectangles, arcs, etc. In the Sketch tab, the user can also add dimensions to his sketch in order to define distances and sizes.

Evaluate Tab In the Evaluate Tab, the user can evaluate and analyze different aspects of their 3D models. This tab offers tools to measure distances and angles, along with mass properties tools to calculate mass volume and other physical properties of the 3D model.

3.3.3 FeatureManager Window

The FeatureManager Design Tree is located on the left side of the SolidWorks window. This window provides a hierarchical representation of the different features of the design. This representation allows users to see how each feature contributes to the final design. Through the FeatureManager Window, the users are able to easily access and modify the sketches and features. Also, the features can be reordered, suppressed, or unsuppressed and these changes are applied immediately to the 3D model. The FeatureManager allows the users to control the visibility of each feature or part when they work with complex models.

3.3.4 PropertyManager Window

The PropertyManager Window is located on the left side of the Solidworks window. It offers options and settings specific to the currently selected item. For example, if the user selects a sketch, the PropertyManager window will show settings related to the sketches. If the selection is a feature, the window will show options for possible modifications of that feature. Through the PropertyManager window, the user can adjust parameters and properties directly without the need to search for them manually. These changes happen in real time on the 3D model.

3.3.5 ConfigurationManager Window

The ConfigurationManager Window is also located on the left side of the SolidWorks window. It allows the user to manage different configurations of parts and assemblies. Configurations are variations of a design that can have different dimensions, features, and other properties. Through the ConfigurationManager the user can create multiple configurations and edit each one independently.

3.4 Arduino IDE

The Arduino IDE is a software application designed for programming and developing projects with Arduino microcontrollers. It is available for all operating systems i.e. Windows, Linux, and MacOS. Before Arduino IDE, programming microcontrollers and embedded systems required more specialized tools and a deeper understanding of hardware and low-level programming. However, Arduino IDE introduced a more straightforward programming language based on C++. Also, it integrated all the necessary tools, from coding to compiling and uploading, together. Additionally, Arduino utilizes an open-source community, which encourages users to develop custom libraries and other hardware/software components. The Arduino IDE consists of four main sections. These sections are the following:

Menu Bar The bar that is placed on the top of the IDE window is called Menu Bar. The Menu Bar contains 5 options, which are File, Edit, Sketch, Tools, and Help.

With the File option, the user is able to:

- press **New** to create a new file.
- press **Open** to open a saved file.

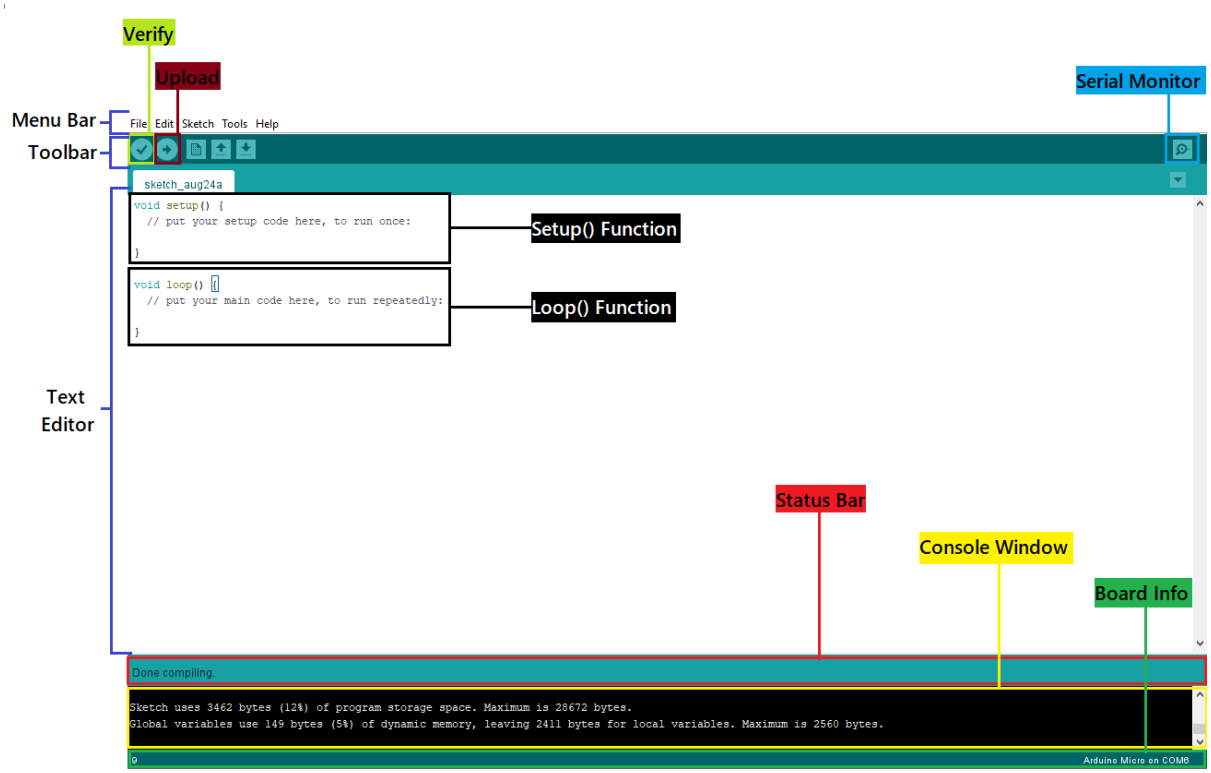

Figure 3.10: Arduino IDE Window overview

- press **Examples** to see default examples already stored in the IDE software.
- press **Close** to close a file.
- press **Save** to save a file.
- press **Quit** to quit the application.

With the Edit option, the user can copy and paste code and modify the font size.

With the Sketch option, the user can compile and upload the software to the board.

With the Tools option, the user can test his project or burn the bootloader to a new microcontroller in a different port.

The Help option is mainly used for troubleshooting.

Toolbar The six buttons under the Menu Tab form the Toolbar. The first button, which is a checkmark, is used to verify the code. The arrow key pointing to the right, is used to upload

and transfer the required code the the Arduino microcontroller. The third button is used to create a new file, while the fourth one is used for opening an existing project. The next button is used to save the current Arduino project and the last button, which is placed on the right side of this section is called Serial Monitor. When the user presses the Serial Monitor, a new window appears, which works as an independent terminal and is important for sending and receiving the serial data. It is mainly used for debugging.

Text Editor The text editor is the interface, where the user writes and edits his Arduino code. It is the central component in the IDE window. The editor uses syntax highlighting so that different parts of the code have different colors to be easily identified. Also, each line of code is numbered, which helps with debugging. The indentation happens automatically. Additionally, the editor allows for multiple tabs, to work on different sketches in the same window.

Output Panel The bottom part of the window is called the Output Panel and it contains the Status Bar, the Console Window, and the Board Information. The Status Bar shows the uploading status. If everything is correct, the user will see a Done uploading message. The Console Window shows the memory used by the code and any possible error that occurred in the program. Finally, the Board Information shows information about the board that is used and the port that is connected.

3.5 Unity

The implementation of the experience within the framework of the research project takes place entirely within the Unity Game Engine. Unity is a cross-platform 3D application development platform developed by Unity Technologies. As of 2018, Unity has expanded to support more than 25 platforms. It can be used to create games and interactive applications with 3D or 2D graphics, Virtual Reality and Augmented Reality games, as well as simulations and other experiences. Unity has been adopted by industries outside of game development, such as the film industry, automotive, architecture, and construction. Unity offers an original scripting API in C, both for Unity itself in the form of plugins and for game programming. Before C# became the main programming language used it previously supported Boo, which was removed in Unity 5, as well as a JavaScript version called Unity Script, which was removed in August 2017 after the release of Unity 2017. Unity has support for the following graphics APIs: Direct3D on Windows and XboxOne. OpenGL on Linux, MacOS, and Windows, OpenGLES on

Android and iOS, WebGL on the web, core APIs on video game consoles as well as applications for Augmented Reality masks such as the LuminOS of the Magic Leap Augmented Reality mask. Additionally, Unity supports the low-level Metal APIs on iOS and macOS and Vulkan on Android, Linux, and Windows, as well as Direct3D 12 on Windows and XboxOne. For 3D games, Unity allows for defining image compression, creating mipmaps, and adjusting resolution settings for each platform the game engine supports. It provides support for elevation mapping, reflection mapping, distortion mapping, Screen Space Ambient Occlusion (SSAO) maps, dynamic shading, and shadow maps as well as rendering-to-texture and post-processing effects.

3.5.1 Basic Structure of Unity

This and the following sub-chapters of our master research thesis will take a close look at the technological terminology regarding the development of the thesis in the Unity game Engine. The following terminologies analyzed here will be greatly used in later Chapters. Unity's workflow builds around the structure of components. Each component has its own specific job, and can generally accomplish its task or purpose without the help of any outside sources. Each game or application created in Unity is called a Project. Each Project consists of the used Assets and one or more Scenes. The Scenes consist of GameObjects and Prefabs, each of which has one or more Components and Scripts attached to it.

Assets Assets are all the building blocks of all Unity 3D projects. It refers to all the files that one will use to create a game such as models, scripts, textures, sounds, etc. A good classification from early on helps for the future of the project.

Scenes In Unity, the Scenes function as individual levels, or areas of the game, although some developers create entire games in a single scene, such as puzzle games, loading dynamic content through code. By building a game in many scenes, the developer is able to allocate loading and testing times separately. At any time there is only one open Scene which is the one we are working on, as it is not possible for two scenes to work at the same time.

GameObjects Each active object in the current open scene is called a GameObject. All GameObjects contain at least one Component (Component) which is the Transform Component. The transformation informs Unity about the position, rotation, and scaling of an object through the Cartesian coordinates X, Y, and Z. Through code, the component can determine

the coordinates of the object. Thus, from this initial component, it is possible to add other Components to the object by adding the required functionality that we desire.

Components Components come in many forms and are attached to GameObjects. They are used to create behavior, determine appearance, and affect other aspects of the operation of an object in the game. By placing Components in an object, one can immediately give them new functions. Common components of the game production come integrated with Unity, such as the Rigidbody component, lights, cameras, particle transmitters, and more. To build further interactive elements of the game, we write Scripts, which are also treated as components that extend or modify the existing available Unity functionality.

Scripts A scripting language is a programming language that allows us to control one or more applications. Scripting is a key ingredient in all games. Even the simplest game will need scripts to meet the participant's input and take care of the gameplay events to take place when needed. Beyond that, scripts can be used to create visual effects, control the physical behavior of objects, or even implement a custom AI system for game characters or environment animals. At Unity, Scripts are Components that extend or modify its existing available functionality. They are created and placed as Components in the GameObjects we want by giving them functionality, without any restrictions on how many Scripts can be used. It also offers a full-fledged scripting code editor, MonoDevelop, allowing developers to program either JavaScript or C .

Prefabs Prefabs are prefabricated and stored versions of an object that can be reused in various parts of our program. With the use of Prefabs, complex objects with different elements and settings can be used at any time, allowing each one to be modified individually whenever and however we wish.

3.5.2 Unity Architecture & Project Structure

Unity's architecture is based on its Components. In order to be able to use the Components of objects in the right way, it is necessary to understand the architecture on which this GameEngine was built and the way it uses these Components. The first and most important Component is the Transform, determining the either Local or World Space Coordinates. Then we have the Mesh Component and Rigidbody Physics which approach the real world collisions and gravity. In order to determine the way an Object is rendered we use Textures, Materials, and Shaders.

In order to simulate sunlight and other lights we use the Lighting Components. For more advanced simulations such as fog, fire, etc. we need the Particle System. For animating 3D Objects we need the Animation Components. For having a Menu, we need the User Interface (UI) Components. Lastly, to be able to render all these on the computer screen (or VR headset in our case) we need the Camera Component and to accompany the image with sound, we need the Audio Components.

Coordinates - Transform At a two-dimensional level, each point is uniquely determined by a pair of numerical coordinates X (horizontal axis), and Y (vertical axis). In three dimensions and consequently in all 3D applications of Unity, there is a third axis Z, which represents the depth with the result that each point is represented by a trinity (X, Y, Z). Thus, a cube at point (5,4,3) in the three-dimensional world means that it is 5 points away from point 0 on the X axis, 4 points away from the Y axis, and 3 points away from the Z axis. This format is known as the Cartesian coordinate system. Every object in a scene has a transform component. It is used to store and treat the position, rotation, and scaling of an object. Each transform component can have a parent, which allows the developers to move the object, rotate it, and scale it hierarchically.

Local Space vs World Space Coordinates In any three-dimensional world, there is a point of origin, often referred to as origin or world zero as it is represented in position (0.0.0). All the positions of the objects in the three-dimensional worlds have zero as their reference point. However, to make things simpler, we use local space to define object positions relative to other objects.

Mesh Component 3D Meshes are the main graphics primitive of Unity. They define the shape of an object as a polygon mesh consisting of vertices, edges, and faces. The vertices are points that represent positions along with other information such as color and texture coordinates. An edge is a connection between two vertices and a face is a closed set of edges.

Rigidbody Physics The Rigidbody component controls an object's position through physics simulation. Unity's physics machine allows developers to simulate real-world responses to objects. Unity uses Nvidia's PhysX engine, which is a popular and accurate physics machine. In GameEngines there is no assumption that an object should be affected by gravity, firstly

because it requires a lot of processing power and secondly because there is no reason to do so. The physics machine uses the Rigidbody dynamic system to create realistic motion. This means that instead of objects being static, they can have properties such as mass, gravity, velocity, and friction. As far as processing power increases, the Rigidbody physics system applies to games, as it allows for more realistic simulations.

Materials, Textures & Shaders Materials define how a surface should be rendered, by including references to the Textures it uses, tiling information, Color tints, and more. The available options for a Material depend on which Shader the Material is using. Textures are bitmap images. A Material can contain references to textures so that the Material's Shader can use the textures while calculating the surface color of a GameObject. In addition to the basic Color (Albedo) of a GameObject's surface, Textures can represent many other aspects of a Material's surface such as its reflectivity or roughness. Shaders are small scripts that contain mathematical calculations and algorithms for calculating the Color of each pixel rendered, based on the lighting input and the Material configuration.

Particle System In a three-dimensional game, most characters, sets, and elements are generally represented as Meshes. Mesh is the ideal way to depict close objects with a well-defined shape. However, there are other entities that are fluid and intangible and therefore difficult to represent with Meshes. For effects such as moving liquids, smoke, clouds, flames, and others, a different approach is used to graphics, known as Particle System. A Particle System simulates and renders many small images or Meshes, called particles, to produce a visual effect. Each particle in a system represents an individual graphical element in the effect. The system simulates every particle collectively to create the impression of the complete effect.

Lighting To calculate the shading of an object, Unity needs to know the intensity, the address, and the color of the light falling on it. These properties are provided by Light Objects. The basic color and intensity are determined in the same way for all lights, but the direction depends on the type of light we use. Also, the light can be reduced by the distance from the source. The four types of light available in Unity are described below:

- Point Light: It is placed at a point in space and sends light in all directions evenly. The direction of light that hits a surface is the line from the point of contact to the center of the light source. The intensity decreases with distance from the light source, reaching

zero after a certain range. This type of light is ideal for simulating lamps and other local light sources.

- Spot Light: Like Point Light, it has a specific position and range, around which the light intensity decreases. However, this light is limited to one corner resulting in the illumination of a cone-shaped area. It is generally used for artificial light sources such as lenses and headlights.

- Directional Light: Represents large, remote sources that come from a location outside the game world. In realistic scenes, they can be used to simulate the sun or the moon. In an abstract world of play, it can be a useful way to convincingly shadow objects without determining exactly where the light is coming from.

- Area Light: Light is emitted in all directions, but only on one side of the rectangle and decreases in a specified range. It can be used to create a realistic street light as well as for indoor home lighting.

Camera The camera is one of the most important elements in a 3D game. It acts as the participant's eyes. Cameras are devices that capture and display the world to the participant. By customizing and manipulating cameras, you can make the presentation of your game truly unique. You can have an unlimited number of cameras in a scene. They can be set to render in any order, at any place on the screen, or only certain parts of the screen letting the participants see the game world from different points of view.

Audio The sound in Unity consists of two basic components which are described below:

- Audio Source: It is the source of the sound and reproduces a sound clip which can be either two-dimensional, three-dimensional, or a mixture (SpatialBlend). This component contains settings for volume, repeatability, tone, priority over other sources, and a variety of other settings and effects.

- Audio Listener: It basically works like a microphone device, receiving input from the audio sources of the scene and playing sound from the computer speakers or headphones. There must always be a Listener activated at a given time during the game. It is usually placed on the main camera as the participant hears and sees from there.

User Interface (UI) Unity provides the User Interface (UI) system that allows the developers to create intuitively and quickly presentable graphical interfaces. The User Interface system consists of a canvas (Canvas), in which all (UI) elements are placed. These elements can be texts, images, etc., as well as more interactive elements such as buttons and sliders.

Animations Unity's Animation enables the developers to create and shape their own animated designs to create creative behaviors. In addition, Unity allows the developers to import animations from other graphics design programs such as Blender3D, which is used in this dissertation. So when an object is inserted, it needs an Animation Controller component, which allows the animation to be managed.

Chapter 4

Glove Design & Implementation

4.1 Hardware Assembly

The haptic feedback device is presented in Figure 4.1 and its components are explained below:

- In Figure 4.1, the purple dots are the moving platforms for each finger, which were 3D printed and connected through strings with the servo motors.
- In Figure 4.1, the blue dots indicate the softness level controllers for each finger. These are 3D-printed and provide variable softness.
- In Figure 4.1, the green dots are the hard object controllers for each finger. These 3D-printed parts control the hardness rendering of virtual objects.
- In Figure 4.1, the yellow dots are the servo motors that are responsible for controlling which tactile cue the user will feel. More specifically, if the object is soft it rotates to the left, otherwise to the right.
- In Figure 4.1, the red dots indicate the 4 servo motors that are controlling the moving platform on the index finger
- In Figure 4.1, the box presents the PCA9685 and the circuit board, which consists of the Arduino Micro(which is powered with 5V from the USB port), five n-type MOSFETs, five 100pF capacitors, five 50KOhm resistors, and five Schottky diodes. This board is connected with the pair of cables of every vibration motor, the PWM pins, the 5V, and GND from Arduino. Furthermore, the PCA9685 is connected with the 7 servo motors. In order to power these motors, an external power of 5V and 2.5A was used.

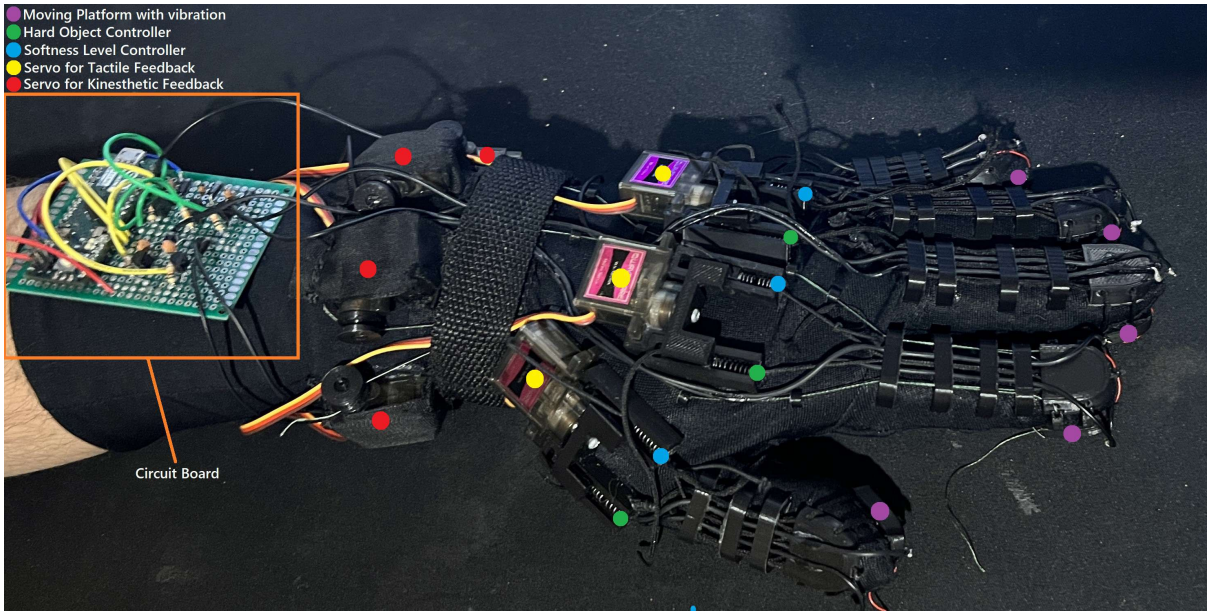

Figure 4.1: Top view of the haptic device

4.2 Design of the 3D parts

4.2.1 Part on the top side of the fingertips

Index Finger To design the part placed at the top side of the index fingertip, which drives the strings for the kinesthetic feedback and the strings connected with the moving platform, we followed these steps:

- **Step 1: Create a 2D Sketch**

  Open a 2D sketch by going to the CommandManager and selecting "Sketch". In this sketch, we drew a 20mm straight line as our initial reference.

  We added a perpendicular line on both edges of the initial 20mm line, each measuring 6.5mm long.

  We set a reference point located 20mm away from the middle of the initial straight line. This point is used for positioning and aligning a curved spline.

  To create a curved shape, we drew a spline starting from one perpendicular line, passing through the reference point, and ending on the other perpendicular line.

- **Step 2: Extrude the 2D Sketch**

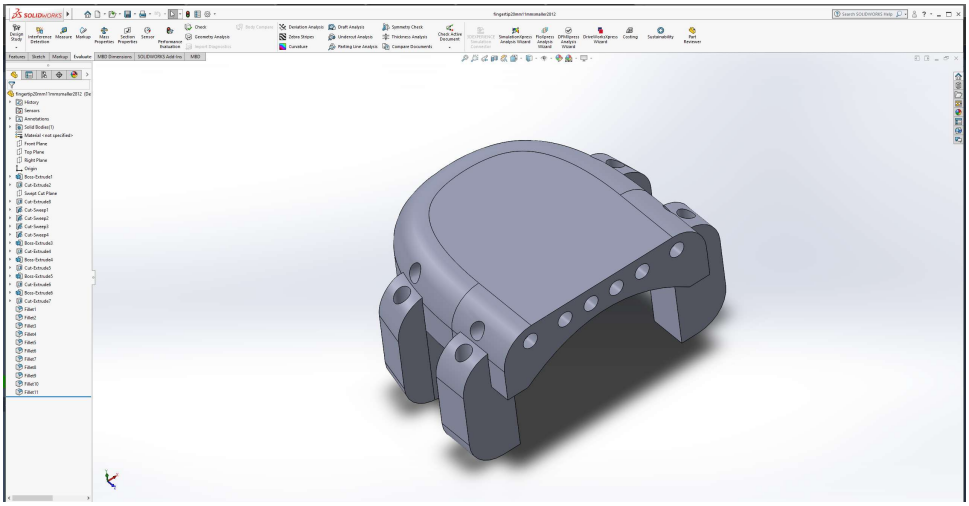

Figure 4.2: Index fingertip top side part

The next step was to extrude the 2D sketch we've created to give it depth and turn it into a 3D object. First, we selected the 2D sketch from the FeatureManager window on the left.

Once the sketch was selected, we went to the "Features" tab in the Command Manager and clicked the "Extrude Boss/Base" command. In the PropertyManager window on the left, we set the depth at 6mm.

- **Step 3: Cut-Extrude the 3D object**

  The following step was to extrude the 3D object so that it matches the user's fingertip shape. Before we started the extrude-cut, it was necessary to design the 2D sketch on the 3D object's side we wanted to extrude-cut.

  We drew a 16mm straight line, as shown in Figure 4.5. Then, we set a reference point located 3mm away from the middle of the initial straight line. This point is used for positioning and aligning a curved spline.

  To create a curved shape, we drew a spline starting from one edge of the initial straight line, passing through the reference point, and ending on the other end of the straight line.

  In the "Features" tab in the Command Manager, we clicked on the "Extruded-Cut" command, and in the PropertyManager window, we set the depth at 20mm.

- **Step 4: Cut-Extrude two holes for the kinesthetic feedback strings**

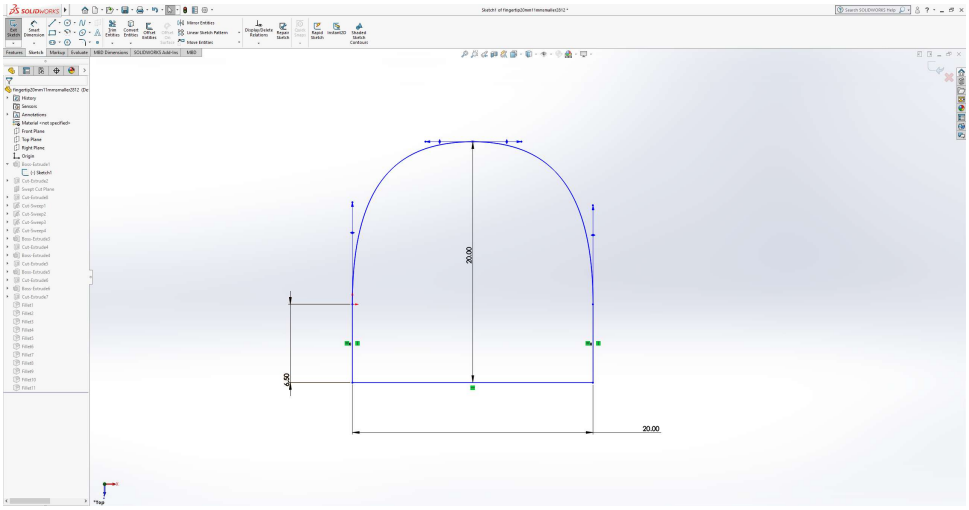

Figure 4.3: Step 1: Create a 2D Sketch

The next step was to create two holes, that are used to drive the kinesthetic feedback strings. We designed two circles with a 0.7mm radius on one side of the 3D object using the "Sketch" command in the Command Manager.

Then we selected the "Extruded-Cut" command in the "Features" tab to create the two holes, each measuring 20mm in length.

- **Step 5: Cut-Sweep four holes for the moving platform strings**

  Then, we needed to create holes that have a curved path to drive the strings to the sides of the 3D model. This is achieved by creating cut-sweeps.

  First, we needed to sketch the profile that we wanted to use for the cut. The profile represents the shape we want to cut. By selecting the "Sketch" command in the Command Manager, we drew a circle with a 0.7mm radius.

  Then, we defined the path for the sweep-cut. We drew a straight line perpendicular to the center of the circle. After the line, we created an arc, that reaches the side of the 3D model. We did this process twice on each side. The straight line measures 1mm for the holes closer to the back side of the 3D model and 6.5mm for the further ones. Similarly, the arc radius for the holes closer to the back side of the 3D model is 2.75mm and 5.59 for the further ones.

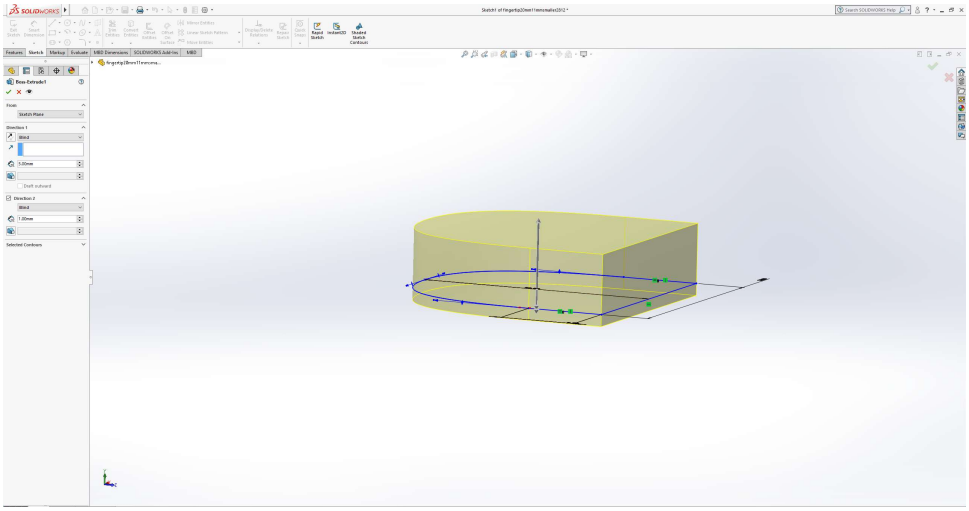

Figure 4.4: Step 2: Extrude the 2D Sketch

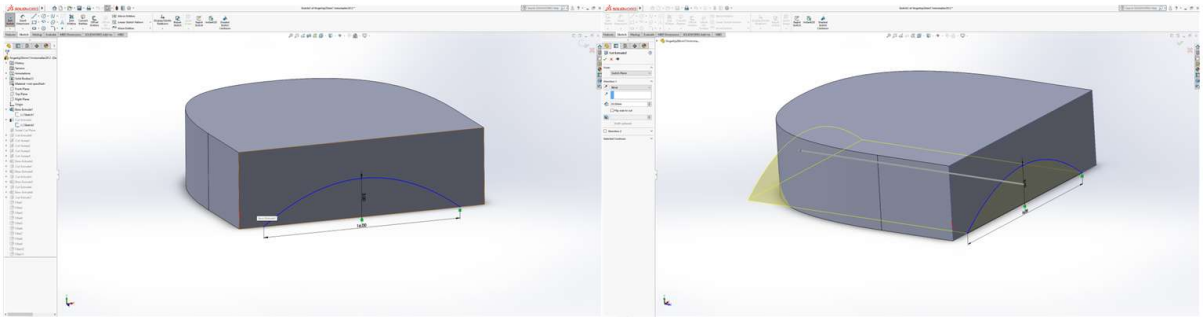

Figure 4.5: Step 3: Cut-Extrude the 3D object

Finally, we went to the "Features" tab in the Command Manager and clicked on the "Sweep-Cut" command. By selecting the profile and path we designed, a curved hole is created.

- **Step 6: Create 2D sketch for the "legs"**

  The next step was to create four "legs", one on each corner, to drive the strings smoother to the moving platform and change the direction of the strings from horizontal to vertical.

  Through the "Sketch" command in the Command Manager, we designed a 2D sketch whose shape is rectangular. This sketch extends from the side of the 3D model by 3mm. The width of the rectangle is 3.41mm.

- **Step 7: Extrude the 2D sketch for the "legs"**

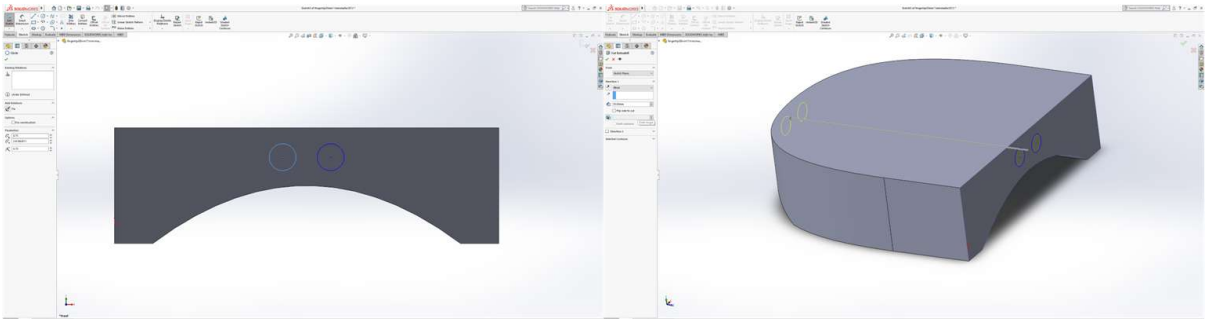

Figure 4.6: Step 4: Cut-Extrude two holes for the kinesthetic feedback strings

After creating the 2D sketch, we needed to extrude it. First, we selected the 2D sketch from the FeatureManager window and clicked on the option "Features" in the Command Manager. Then, we chose the Extruded Boss/Base command and extruded the design by 11mm.

- **Step 8: Cut-Extrude hole on each "leg"**

  After extruding the 2D sketch, we created a hole, so that strings were able to pass through to reach the moving platform. These holes are created by the "Extruded Cut" command in the "Features" tab.

  We repeat the process from step 6 to step 8 for each "leg".

- **Step 9: Fillet edges to be smoother**

  The final step was to fillet the edges through the "Fillet" command in the "Features" tab in the Command Manager window. In the PropertyManager on the left side of the window, we entered the desired fillet radius on each fillet. This radius is the radius of the fillet radius. We could either type in a value or use the drag handles to visually adjust the radius.

Middle Finger & Thumb To design the part placed at the top of the thumb and middle fingertip, which drives the strings for the kinesthetic feedback, we followed the same steps as for the index fingertip. However, there were some minor modifications. We didn't create "legs" for these fingers because we don't move these fingers' platforms in many directions like in the index fingertip part.

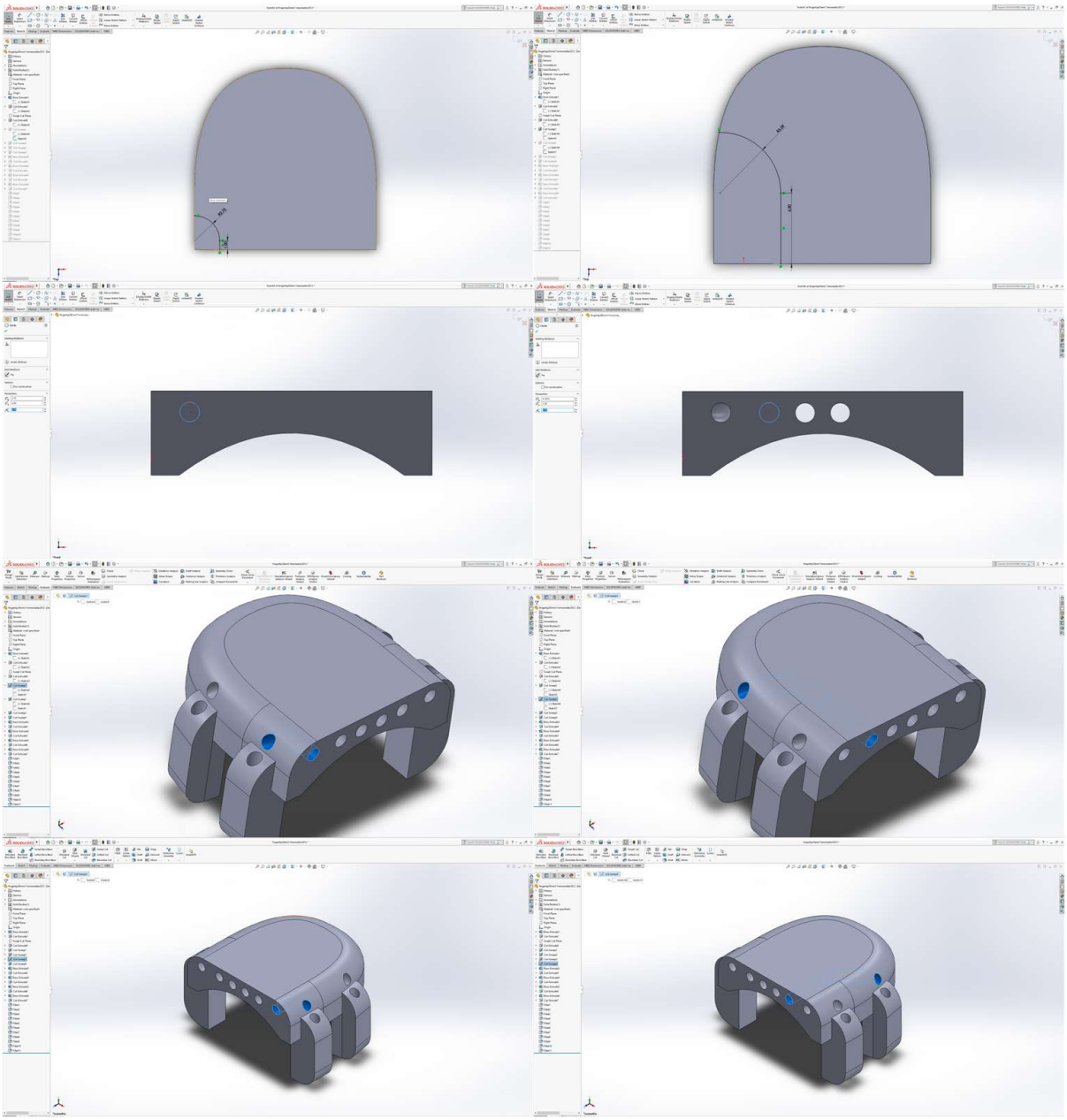

Figure 4.7: Step 5: Cut-Sweep four holes for the moving platform strings

Ring Finger To design the part placed at the top of the ring fingertip, which drives the strings for the kinesthetic feedback, we followed the same steps as for the index fingertip. For this finger, however, we didn't create "legs" because we don't move these fingers' platforms in many directions like in the index fingertip part. Also, since this fingertip is smaller, in Step 1

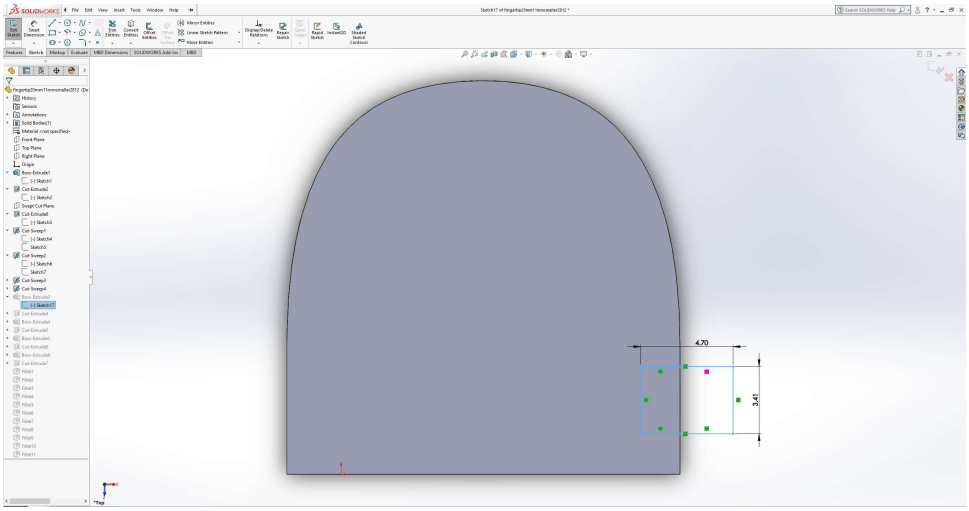

Figure 4.8: Step 6: Create 2D sketch for the "legs"

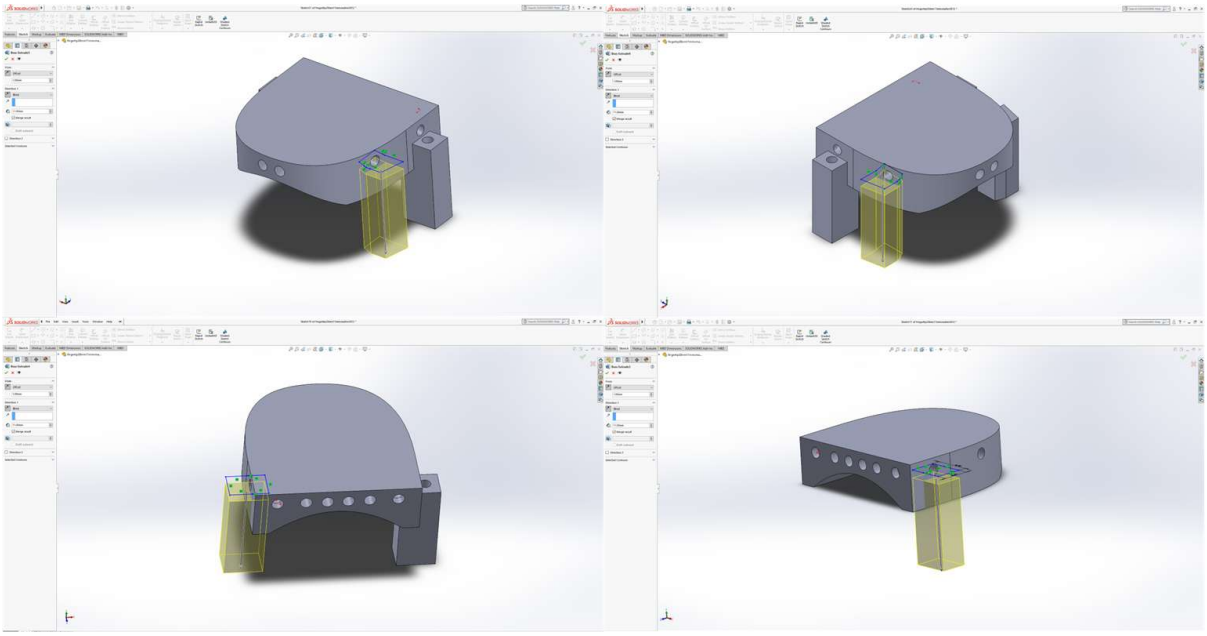

Figure 4.9: Step 7: Extrude the 2D sketch for the "legs"

we reduced the initial straight line from 20mm to 18mm.

Pinky Finger Similarly to the ring finger, we followed the same steps and modifications. More specifically, we also didn't create "legs" and we reduced the initial straight line to 17mm

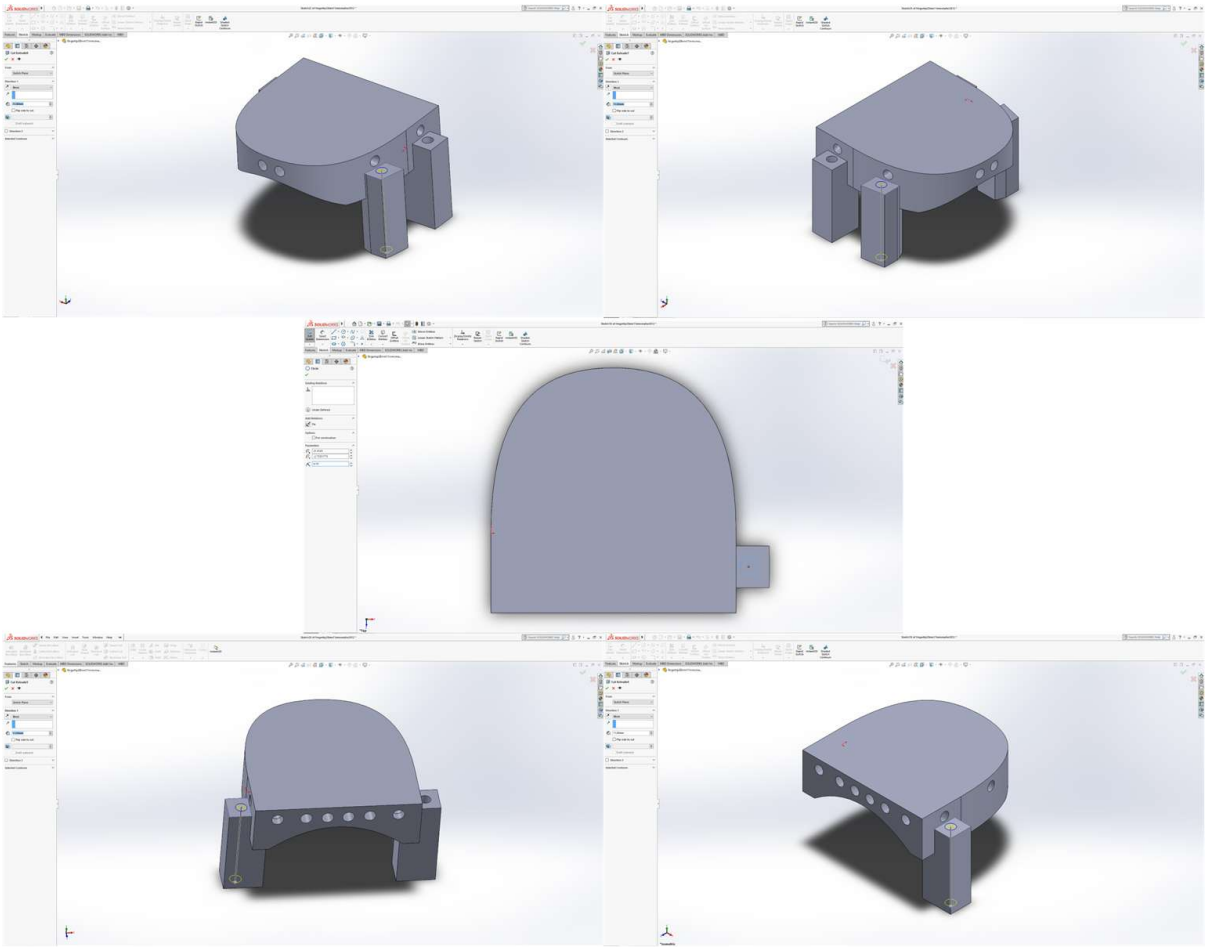

Figure 4.10: Step 8: Cut-Extrude hole on each "leg"

since the pinky finger is even smaller than the ring finger.

4.2.2 Part on the bottom side of the fingertips

Index Finger To design the part placed at the bottom side of the index fingertip, which is the moving platform, we followed these steps:

- **Step 1: Create a 2D Sketch**

  We created a 2D sketch by selecting the "Sketch" command found in CommandManager. This sketch is the same as the one shown in Figure 4.3.

- **Step 2: Extrude the 2D Sketch**

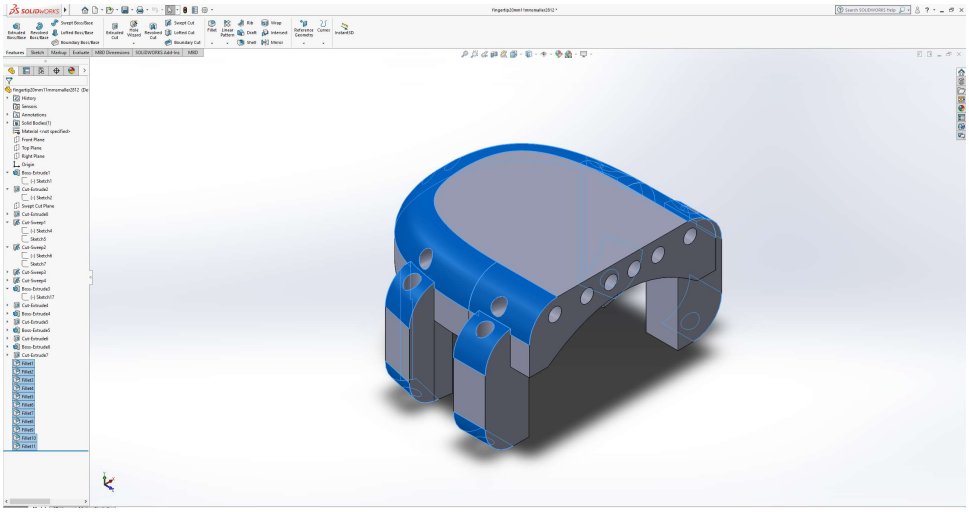

Figure 4.11: Step 9: Fillet edges to be smoother

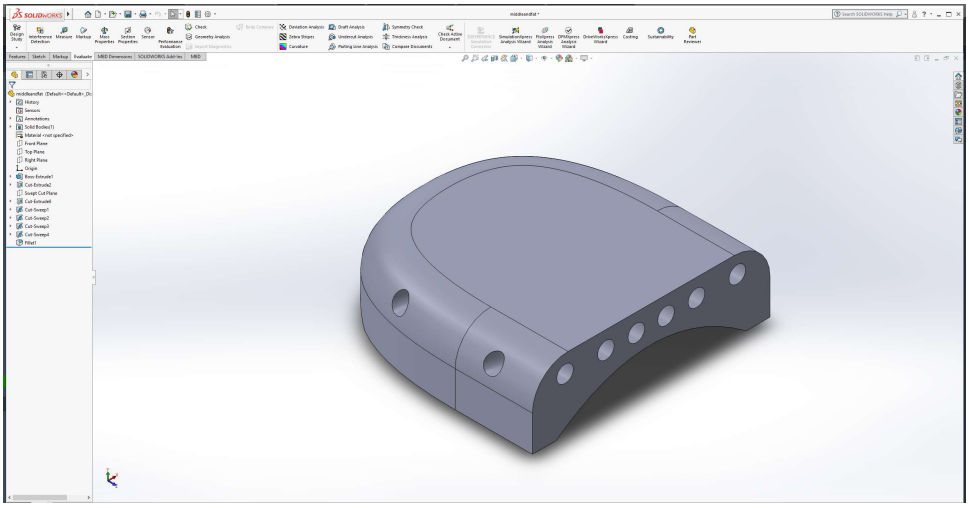

Figure 4.12: Middle and Fat fingertip top side

The next step after creating the 2D sketch was to extrude it. By extruding the sketch by 7.5mm, we created a 3D model. The extrusion of the model was done by selecting the 2D sketch from the FeatureManager window, clicking on the "Features" tab in the CommandManager, and pressing the "Extruded Boss/Base" button.

- **Step 3: Create the vibration motor socket**

  After extruding the 2D sketch and creating a 3D model, the next step was to create a

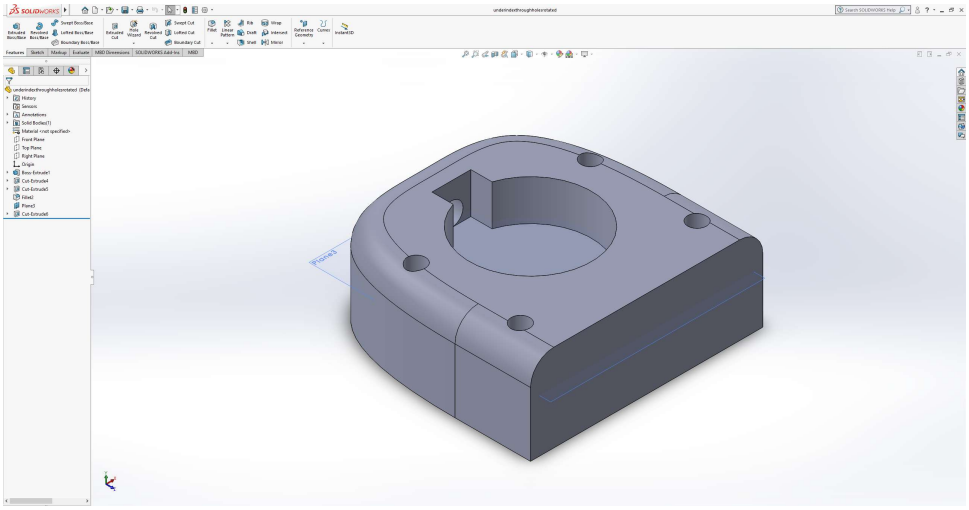

Figure 4.13: Index fingertip bottom side

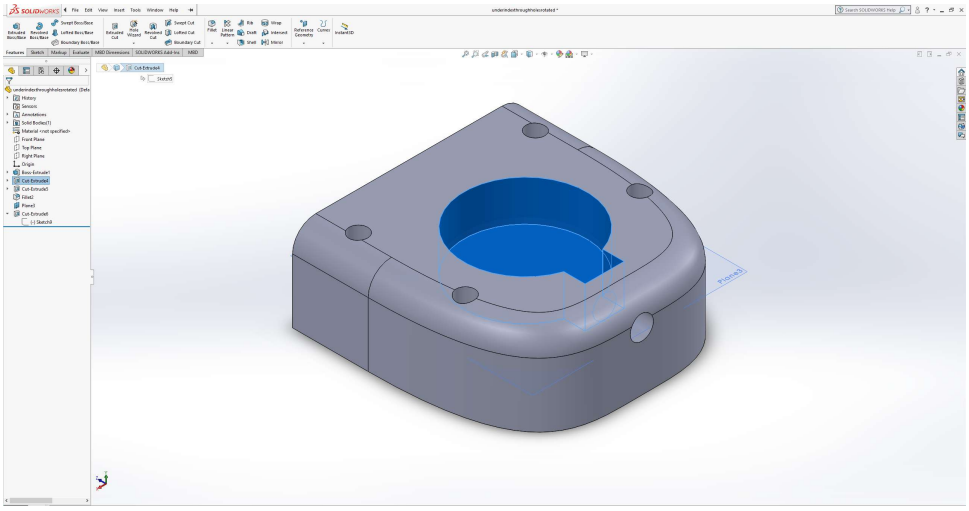

Figure 4.14: Step 3: Create the vibration motor socket

socket for the vibration motor. On the top side of the 3D model, we used the "Sketch" from the CommandManager to create the sketch for the socket. The sketch consists of a 5.25mm radius circle connected with a rectangle(1.84mm x 3.34mm).

After creating the sketch for the vibration motor casing, the next step was to cut-extrude it by 3.5mm. This was accomplished by selecting the "Extruded-Cut" command in the CommandManager window.

- **Step 4: Create a hole for the vibration motor cables**

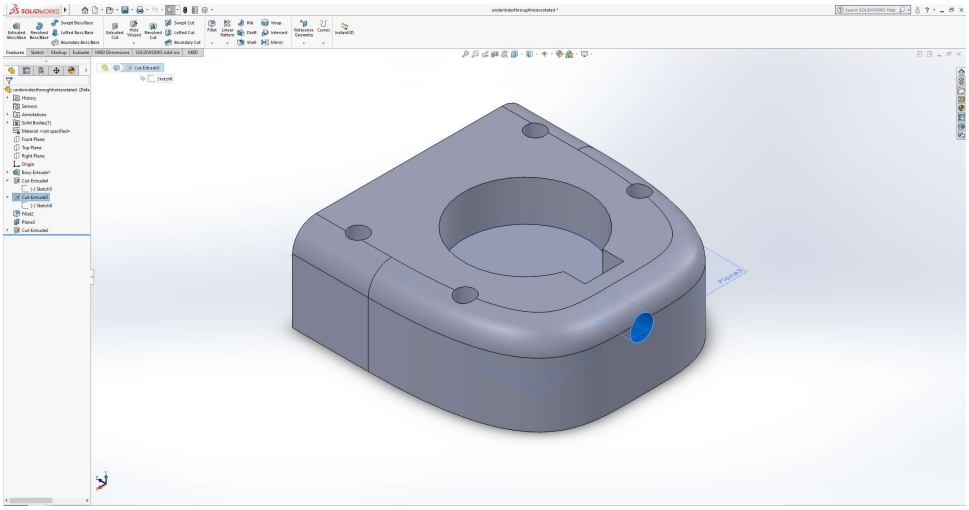

Figure 4.15: Step 4: Create a hole for the vibration motor cables

After creating the motor casing, we created a hole for the vibration motor cables. We made that by sketching a circle on the front side of the 3D model(CommandManager → Sketch) and cut-extruding it(Features → Extruded-Cut).

- **Step 5: Cut-Extrude holes for the haptics feedback strings**

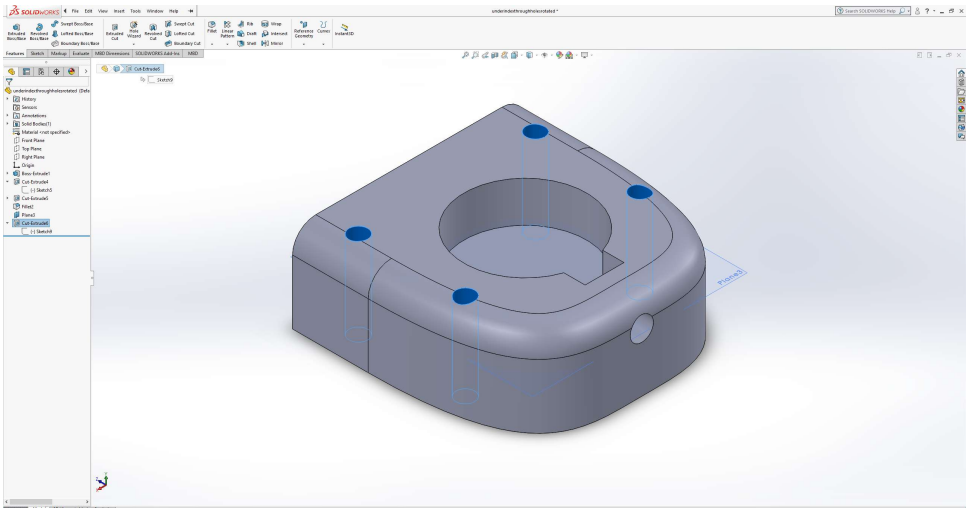

Figure 4.16: Step 5: Cut-Extrude holes for the haptics feedback strings

The next step was to create holes for the haptic feedback strings. By selecting the top

side of the 3D model and clicking on the "Sketch" command in the CommandManager, we create four holes as shown in 4.16 whose radius is 1mm each. Then we selected all of the sketched circles and we used the "Extruded Cut" command from the "Features" tab in CommandManager and created the holes measuring 7.5mm.

- **Step 6: Fillet the edges of the 3D model**

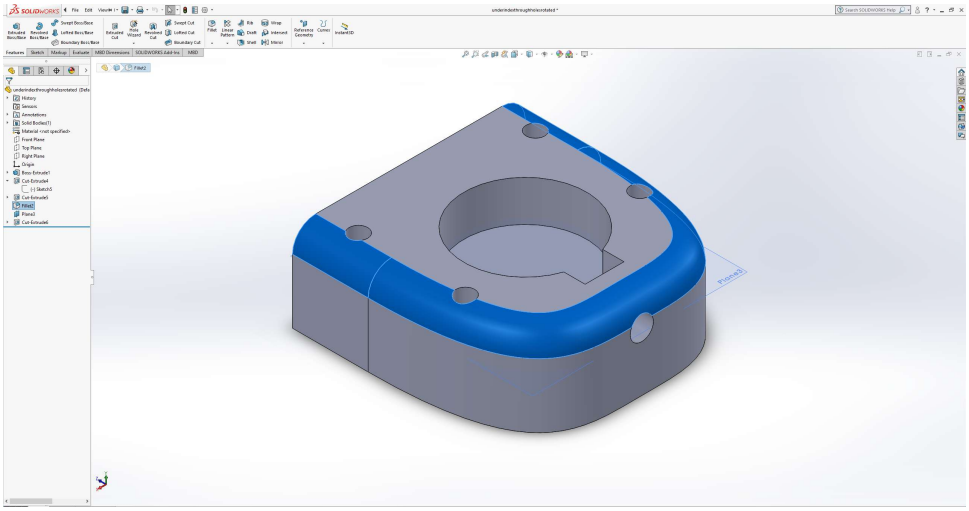

Figure 4.17: Step 6: Fillet the edges of the 3D model

The final step was to fillet the edges through the "Fillet" command in the "Features" tab in the Command Manager window. In the PropertyManager on the left side of the window, we entered the desired fillet radius. By filleting we made the edges smoother.

Middle Finger & Thumb To design the part placed at the bottom of the thumb and middle fingertip, we followed the same steps as for the index fingertip. However, there were some minor modifications. Since on these fingers, we provided tactile feedback through vibration and only normal indentation, there was no need to create a flat surface underneath the middle and thumb fingertip. Thus, after step 3 we added one additional step. In this step, we first designed a 2D sketch of the 3D object's back side. More specifically, we drew a 16mm line, we then set a reference point located 3mm from the middle of the initial line and used this point to design a curved spline, similar to the Figure. Then we extruded-cut the 3D object for 20mm depth, so that it matches the user's fingertip shape, through the "Features" tab in the Command Manager.

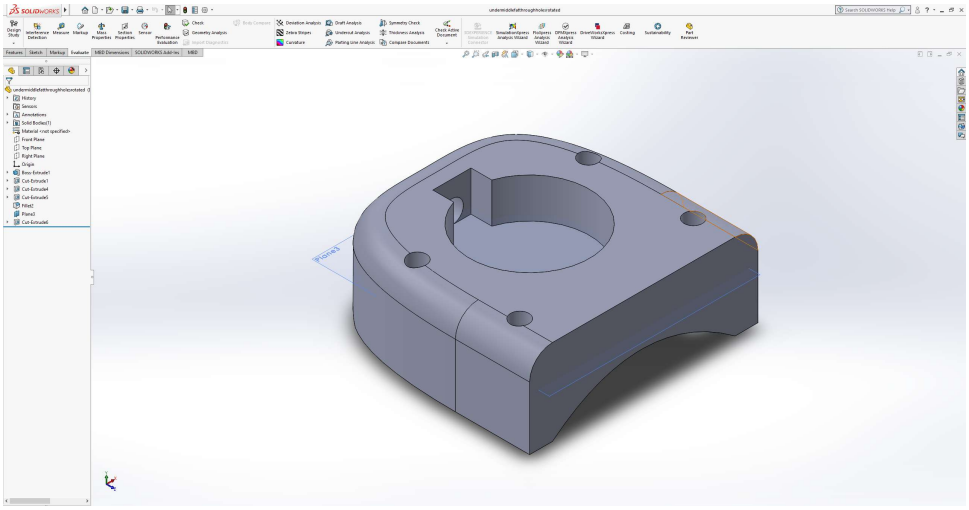

Figure 4.18: Rest of the fingertips bottom side part

Ring Finger Similarly to the thumb and middle finger, we followed the same process with the only difference being in Step 1, where the initial straight-line length was changed to 18mm from 20mm.

Pinky Finger To design the part underneath the pinky fingertip, we also followed the same process with the middle and thumb fingers, but with two modifications. The first modification was in Step 1 to change the initial straight-line length from 20mm to 17mm. The second change was in the additional step, where we changed the line length from 16mm to 15mm.

4.2.3 Parts responsible for driving cables

To design the parts responsible for driving the cables and the strings, we followed these steps:

- **Step 1: Create the 2D sketch and Extrude it**

  The first step in creating these parts was to design a rectangle with dimensions 22mm x 5mm. Then we extruded this 2D shape by 10mm using the "Extruded Boss/Base" in the "Features" tab.

- **Step 2: Create holes for the finger and the cables/strings**

  After creating the basic 3D model, the next step was to create five holes that were used to drive the cables from the circuit all the way to the fingertips and the strings that control

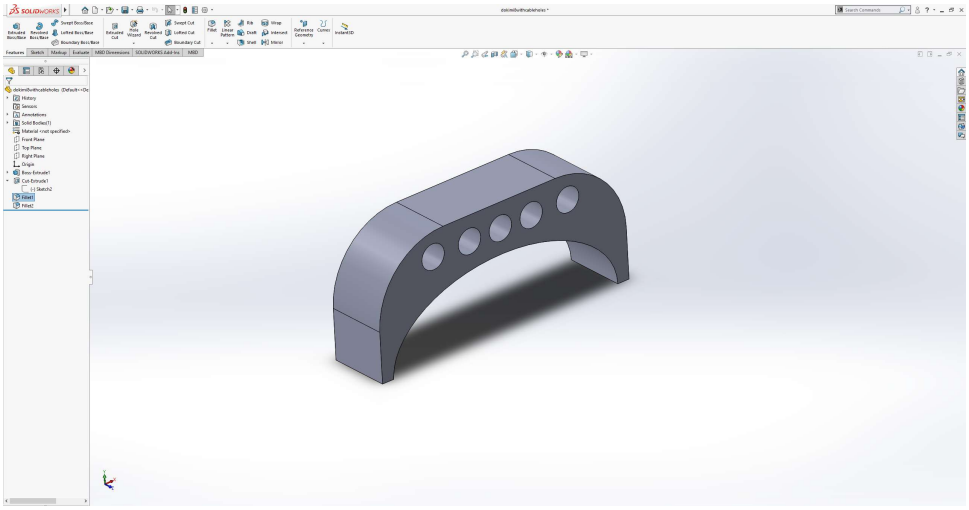

Figure 4.19: Driving cables part

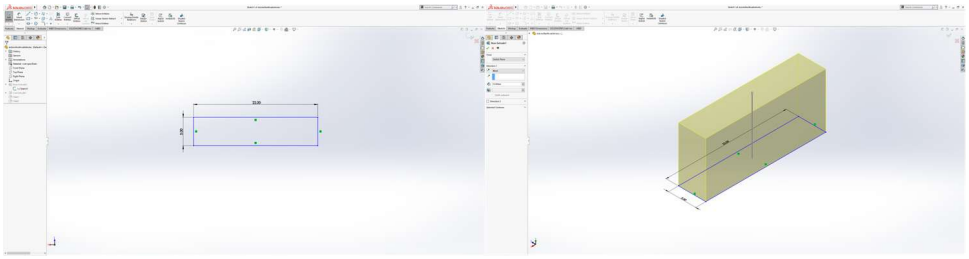

Figure 4.20: Step 1: Create the 2D sketch and Extrude it

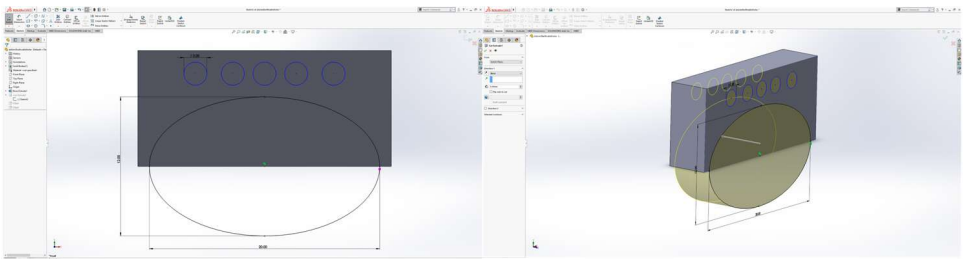

Figure 4.21: Step 2: Create holes for the finger and the cables/strings

the haptic feedback. Additionally, we designed an ellipse with a 6mm radius on the minor radius and 10mm on the major radius. Then, we used the "Extruded-Cut" command the create the holes as shown in Figure 4.21.

- **Step 3: Fillet the edges of the 3D model**

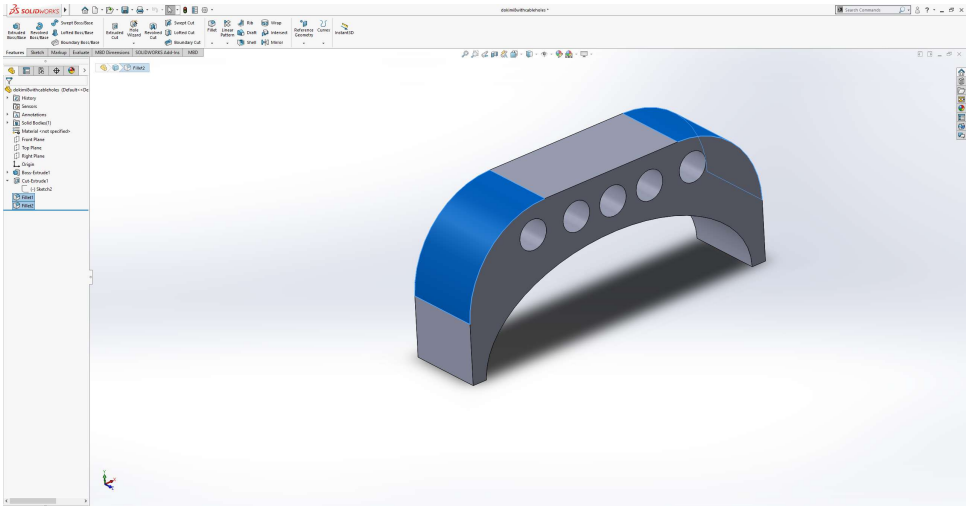

Figure 4.22: Step 3: Fillet the edges of the 3D model

The final step was to make the edges of the 3D model smoother, thus we used the "Fillet" command in the "Features" tab.

4.2.4 Servo braking mechanism

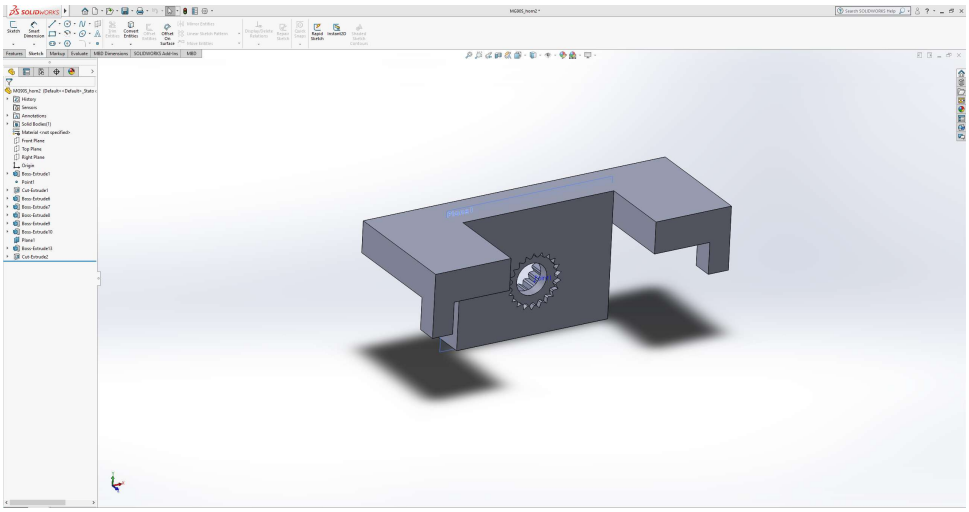

Figure 4.23: Servo breaking mechanism part

To design the servo-breaking part for the kinesthetic feedback, we followed these steps:

- **Step 1: Create a 2D sketch and Extrude it**

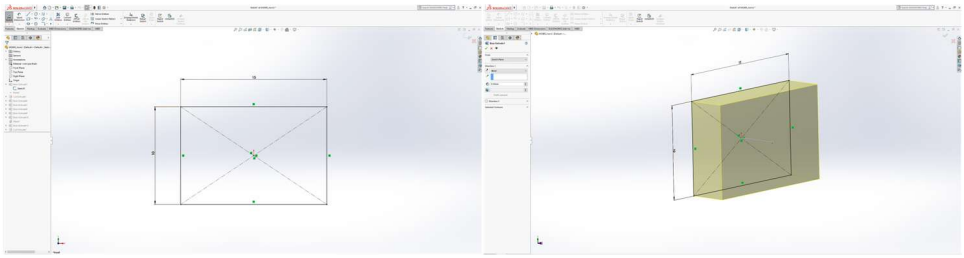

Figure 4.24: Step 1: Create a 2D sketch and Extrude it

In Step 1 we created a rectangle sketch as shown in 4.24, with dimensions 15mm x 10mm. Then, we extruded this sketch by 4mm to create the 3D model.

- **Step 2: Create a hole specific for the servo motor output shaft**

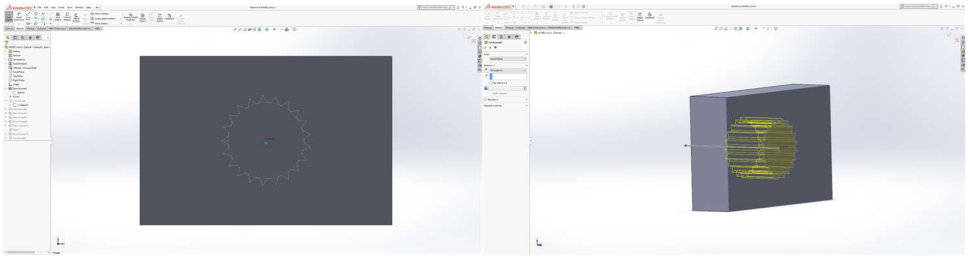

Figure 4.25: Step 2: Create a hole specific for the servo motor output shaft

After creating the 3D model, the following step was to create a hole specifically to fit the servo motor output shaft. This was done by sketching a circle using the "Sketch" command from the CommandManager and adding small triangles around the perimeter of the circle. Then by selecting the "Extruded-Cut" command, we managed to create a hole in the 3D model.

- **Step 3: Create the blocking parts' bases**

In Step 3, we created a base on each side of the 3D model in order to extend it for the blocking part mechanism. More specifically, we designed a rectangle with dimensions 7mm x 3.5mm through the "Sketch" command and we extruded it for 10mm using the "Extruded Boss/Base" option from the "Features" tab.

- **Step 4: Create the blocking parts**

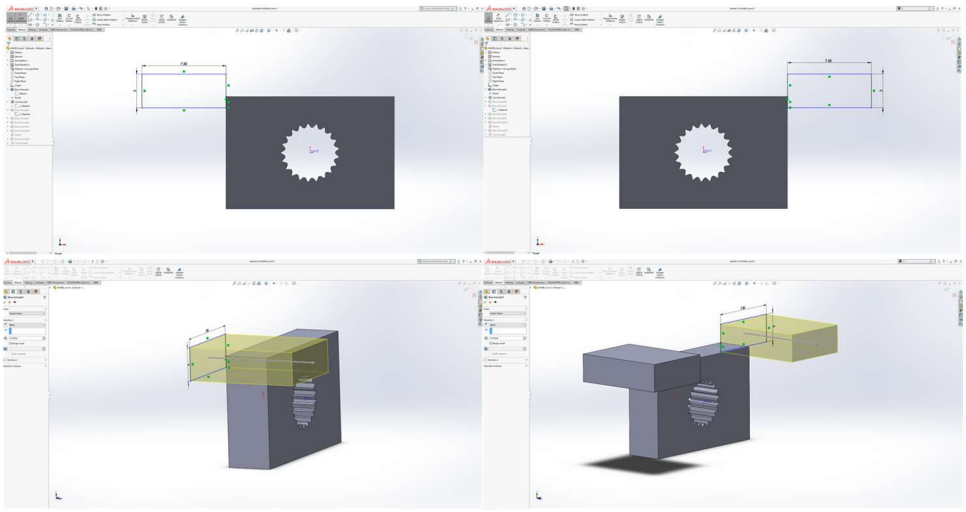

Figure 4.26: Step 3: Create the blocking parts' bases

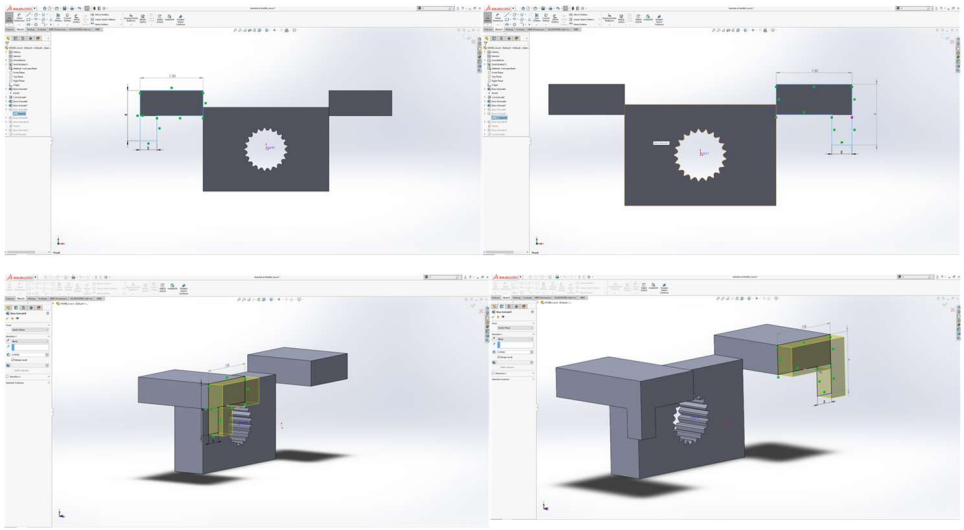

Figure 4.27: Step 4: Create the blocking parts

After creating the base on each side, the next step was to create the blocking parts on these bases. This was done by creating the shape shown in Figure 4.27 and extruding it by 2mm.

- **Step 5: Create a part to support the side parts**

  In order to make the 3D model more difficult to break under tension we added an extra part, that connects the two bases and creates a stronger bond between them.

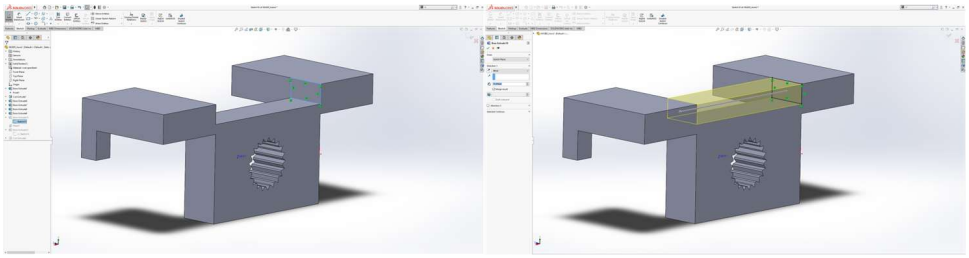

Figure 4.28: Step 5: Create a part to support the side parts

4.2.5 Softness Level Controller & Hard Object Controller

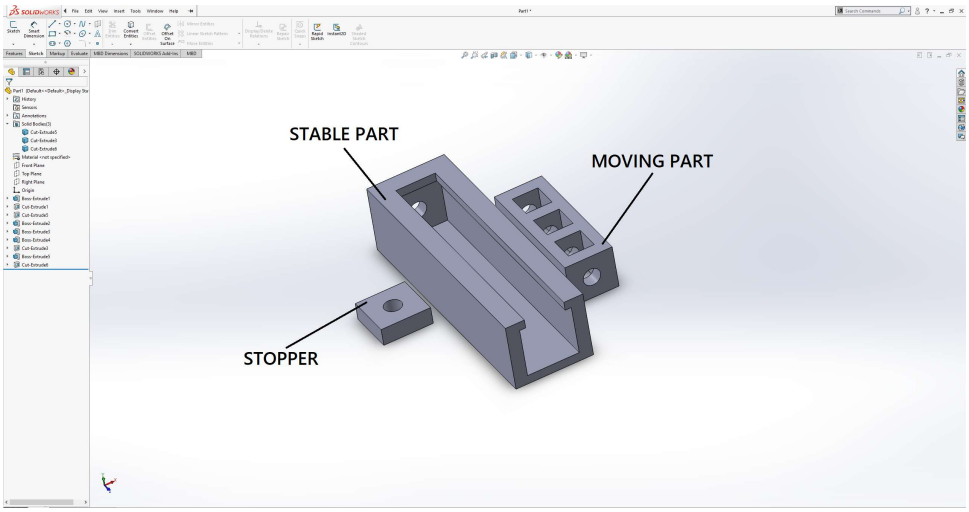

Figure 4.29: Softness Level Controller & Hard Object Controller

The Softness Level Controller and the Hard Object Controller consist of 3 three parts. The parts are the following:

- **Stable part**

  The stable part of this 3D assembly is shown in Figure 4.29. To create it, we first drew a 28mm x 6mm rectangle using the "Sketch" command. Then we extruded it for 1mm to create the base of the 3D model. After creating the base, we sketched a Π shape on the edges of the base and extended them for 6.75mm to create the walls of the 3D model. Finally, we sketched a slightly bigger Π so that the moving part, which we will discuss below, can't move out of the stable part.

- **Moving part**

  The moving part of the design shown in Figure 4.29 was designed to move freely inside the stable part. To design it, we drew a 15mm x 5.5mm rectangle and extruded it for 5mm. Then we created three square holes using the "Extruded-Cut" command. These holes are used so that the servo-breaking mechanism can block the movement of the moving part.

- **Stopper for the moving part**

  Finally, after placing the moving part inside the stable part we had to block it from moving outside of it in the side that is empty. Thus, we created a rectangle with 5.75mm x 5.5mm dimensions using the "Sketch" command and extruded it by 2mm. This 3D model is glued on the empty side of the stable part.

Figure 4.30: Creality Ender 3 V2 3D printer

After designing each of the 3D models, the subsequent phase in our research and development process was to make these models come into reality through 3D printing. The utilization of a 3D printer, the Creality Ender 3 V2[**?**], gave us the capability to transform our digital creations from the virtual world of computer-aided design(CAD) into physical tangible objects with precision and accuracy.

After an exhaustive exploration of different materials, we selected PLA+ as our primary material of choice over alternatives such as ABS, ASA, and standard PLA.

Firstly, PLA+ offers excellent printability characteristics on the Creality Ender 3 V2, including minimal warping and adhesion to build platforms. This provides consistency and reliability, which are important in research and development.

Secondly, exhibits adequate mechanical strength and durability, making it suitable for prototyping. This material has enhanced toughness compared to standard PLA, which makes it more suitable for applications where resistance and structural integrity are crucial.

Moreover, PLA+ is more environmentally friendly than materials like ABS and ASA, which emit potentially harmful fumes during the printing phase. PLA+ is created from renewable resources and is biodegradable.

Cost-effectiveness also played an important role in our material selection, since it offers a good balance between affordability and performance.

4.3 Components used for the Haptic Glove

4.3.1 Arduino Micro

To facilitate control and coordination of all implementation components, the use of a microcontroller was necessary. A microcontroller is a compact computing device, housing a processor, memory, and configurable input/output peripherals. In the context of this thesis, the Arduino Micro was the selected microcontroller board. The Arduino Micro, even though it is smaller in size compared to other Arduino boards, works well with our haptic glove demands. While it doesn't offer the same level of processing power as larger Arduino variants, it excels in our implementation needs to be compact and lightweight.

The Arduino Micro is based on Atmel's ATmega32U4 microcontroller. This board is equipped with essential components, making it easy to program and operate either through a USB connection to a computer or an external power supply. It offers a rich set of features, including 20 digital inputs/outputs, of which 7 support PWM outputs, 12 analog inputs, a micro USB port for programming and communication, a power supply input, a 16 MHz crystal oscillator, an ICSP(In-Circuit Serial Programming) header, and a reset button.

In addition to the hardware, the Arduino ecosystem provides an Integrated Development Environment(IDE) which we analyzed in Chapter 3. Within the Arduino IDE, each sketch(program containing the code) consists of two functions: "setup()" and "loop()". The "setup()" function

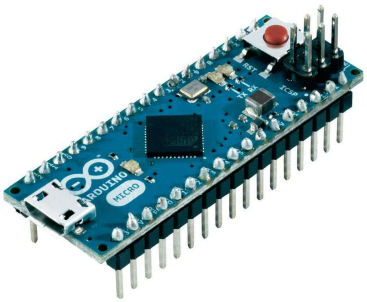

Figure 4.31: Arduino Micro microcontroller

runs once at the start of the sketch and is used for initializing variables, inputs, outputs, and libraries. The "loop()" function runs continuously, managing the board's operations until it is powered off.

Furthermore, essential functions utilized in this thesis include "pinMode()", "delay()", and "analogWrite()". The "pinMode()" is used for configuring a pin as either an input or an output, with the syntax "pinMode(pin, mode)" where "pin" is the Arduino pin number, and "mode" can be "INPUT" or "OUTPUT". Typically, this function is employed in the "setup()" section. The "analogWrite()" generates an analog value, often a PWM signal, on a pin and accepts two parameters: the pin's name and an integer value(0-255) determining the duty cycle(0 for fully off and 255 for fully on). Lastly, the "delay()" function pauses the program for a specified duration, with the syntax "delay(ms)" where "ms" indicates the pause duration in milliseconds.

4.3.2 Vibration Motors

One of the most important aspects of the tactile feedback is vibrations. This was achieved through 5 coin vibration motors. The options for the vibration were either Linear Resonant Actuators or Eccentric Rotating Mass motors, each having its advantages and disadvantages.

While Linear Resonant Actuators(LRAs) offer several advantages over ERM(Eccentric Rotating Mass) motors, including significantly lower power consumption(50% less), independence of amplitude from frequency for more complex waveforms, reduced noise, and longer lifespan, there were practical considerations that led to the choice of ERM motors. The primary factor was that the LRAs require an AC(Alternative Current) signal for operation, which needs a more complex circuit compared to the simpler control requirements of ERM motors. This complex

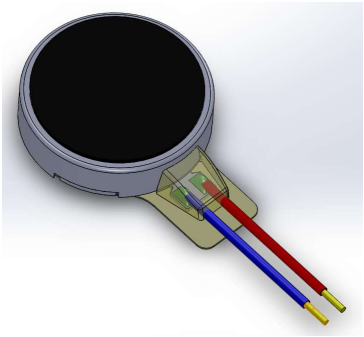

Figure 4.32: The coin vibration motors

circuit will also add more weight to the implementation making it heavy and cumbersome. Additionally, there were budget constraints, as LRAs tend to be more expensive and less available in the market.

The following table shows the specifications of the chosen ERMs.

Specification	Value
Voltage [V]	3
Frame Diameter [mm]	10
Body Length [mm]	3.4
Weight [g]	1.2
Voltage Range [V]	2.5 – 3.8
Rated Speed [rpm]	12000
Rated Current [mA]	75
Start Voltage [V]	2.3
Start Current [mA]	85
Terminal Resistance [Ohm]	75
Vibration Amplitude [G]	0.8

Table 4.1: The coin vibration motor specifications

4.3.3 WS-MG90S Micro Servos

Another important aspect of haptics is enriched tactile feedback (in the form of normal indentation, and lateral skin stretch) and kinesthetic feedback. This was achieved with 7 WaveShare MG90S micro servo motors(4 for the tactile feedback and 3 for the kinesthetic feedback).

The WS-MG90S is a small-sized and lightweight servo motor, which makes it suitable for our implementation where space is limited. It typically provides a torque of around 2kg/cm at 4.8V, which means it can exert a force of 2 kilograms to rotate any object attached to the servo. The operating voltage for this servo motor is between 4.8V to 6V. The speed of the

Figure 4.33: The MG90S servo motor

WS-MG90S servo is 0.11s per 60 degrees of rotation at 4.8V. Also, it has a rotation range of 180 degrees. The WS-MG90S is controlled using PWM signals and it usually comes with a three-wire connector: power(red), ground(brown), and PWM(orange).

The following table shows the specifications of the servo motors.

Specification	Value
Gear	Metal
Dimension [mm]	22.8 x 12.2 x 28.5
Operating Voltage [V]	4.8 – 6
Weight [g]	12.2
Rotating Angle	180°
Speed [s/60°]	0.11(4.8V), 0.09(6V)
Dead Band [us]	5
Torque [kg/cm]	2.0(4.8V), 2.8(6V)

Table 4.2: The MG90S servo motor specifications

4.3.4 PCA9685-Servo Driver

To drive the servo motors, Arduino offers the Servo library, but each servo will consume a pin and some Arduino processing power. The proposed solution in this thesis suggests using the PCA9685 16-Channel 12-bit PWM/Servo Driver to control servo motors instead of relying solely on the Arduino Servo Library. The PCA9685 is capable of controlling up to 16 servos simultaneously over the I2C communication protocol using only two pins. It also features an onboard PWM controller.

The figure illustrates the control input pins on each side of the chip, and this flexibility allows users to chain multiple PCA9685 boards together to control more than 16 servos. Here

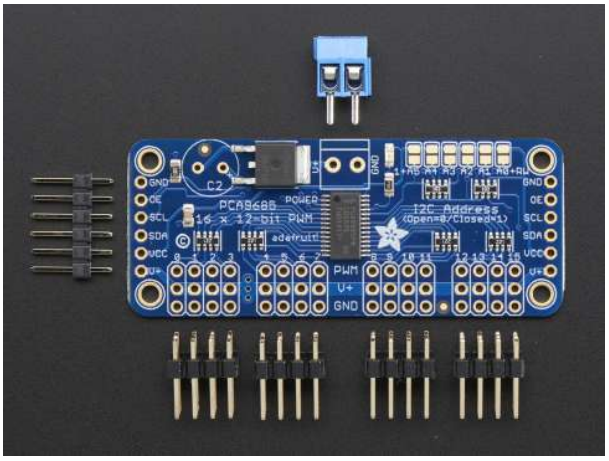

Figure 4.34: The PCA9685 16-Channel 12-bit PWM/Servo Driver

is an overview of the control input pins:

- **GND**, which serves as both the power and signal ground pin.
- **OE**, which can be used to disable all outputs quickly when set high.
- **SCL**, which is the I2C clock pin and is connected with the microcontrollers I2C clock line.
- **SDA**, which is the I2C data pin and is connected with the microcontrollers I2C data line.
- **VCC**, which is the logic power pin.
- **V+**, which supplies power to the servos and can be left unconnected if power is provided through the blue terminal in the middle of the chip.

Additionally, there are 16 output ports on the bottom side of the chip, each featuring three pins: V++, GND, and PWM output. Although each PWM output is independent, they must all operate at the same PWM frequency.

After soldering the pin headers into the designated positions, the Arduino is connected to the PCA9685 as follows:

- **GND → GND**
- **5V → VCC**

- **SDA → SDA**
- **SCL → SCL**

Please note that this connection powers the breakout board itself, but to provide power to the servos, an external 5V source should be connected to the blue terminal.

The servos are connected to the output ports on the bottom side of the PCA9685 board, with the brown wire connected to GND, the red wire to V+, and the orange wire to the PWM input. If the user wants to connect more than 16 servos, he should chain more PCA9685(up to 62), but he needs to assign each board a unique address. This is accomplished by soldering to bridge the address jumpers on the upper right edge(board 0: 00000, Board 1: 000001, etc.).

4.3.5 Motor Driver Circuit

In the context of this thesis, the ERM motors used had specific electrical characteristics, including a startup current draw of 85mA and an operating current draw of 75mA. These current requirements exceeded the output current capacity of individual Arduino pins, which is limited to 40mA. To overcome this limitation, a driver circuit was necessary, and the major component in this circuit was a transistor. Transistors are semiconductor devices used for electronic signal switching or amplification. There are two main types: Bipolar Junction Transistors(BJTs) and Field Effect Transistors(FETs).

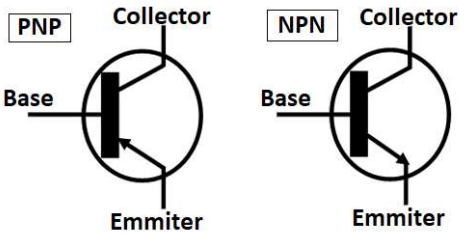

Figure 4.35: Bipolar Junction Transistor

Bipolar Junction Transistors have three terminals- the Emitter, Base, and Collector. They come in two configurations: NPN and PNP. NPN BJTs are made by sandwiching a p-type material between two n-type materials, while PNP BJTs are made with an n-type material between two p-type materials. BJTs are current-controlled devices, where the current flowing

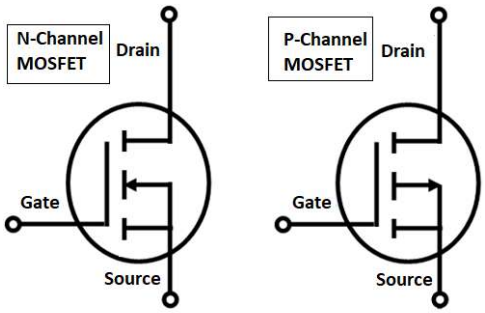

Figure 4.36: Metal Oxide Semiconductor Field Effect Transistor

into the Base terminal controls the current flow from Collector to Emitter(NPN) or from Emitter to Collector(PNP).

Field Effect Transistors also have three terminals, the Source, Gate, and Drain. The two main types of FETs are Junction Field Effect Transistors(JFETs) and Metal-Oxide-Semiconductor FETs(MOSFETs). MOSFETs are divided into two categories, the N-Channel MOSFET and the P-Channel MOSFET. N-MOSFETs have a p-type body with two n-type regions (Source and Drain) adjacent to the Gate, while P-MOSFETs have an n-type body with two p-type regions adjacent to the Gate. FETs are voltage-controlled devices, where applying voltage at the Gate controls the current flow between the Source and the Drain. If the voltage is negative, the MOSFET operates in depletion mode, and if it's positive, it operates in enhancement mode.

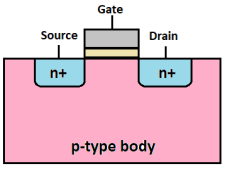

Figure 4.37: N-type MOSFET

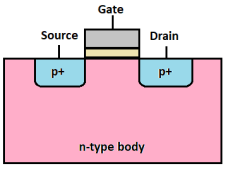

Figure 4.38: P-type MOSFET

In this thesis, the choice was made between BJTs and MOSFETs, with the selection of an n-type MOSFET(in its saturated mode), specifically the 2N7000. N-MOSFETs were preferred because they work well with 2V and higher turn-on-voltage(Vgs).

After choosing the N-MOSFET, additional components were needed to complete the circuit effectively:

- To protect the MOSFET from voltage spikes generated by the motor coils, a Schottky

Figure 4.39: The 2N7000 N-Channel MOSFET

diode (1N5817) was added in parallel across the motor terminals. Schottky diodes allow current flow from anode to cathode and are able to handle a maximum DC blocking voltage of 20V with a maximum average forward rectified current of 1A.

Figure 4.40: The Schottky Diode 1N5817

- To ensure the MOSFET remains fully off when there is no signal, a pull-down resistor(50kΩ) was used, reducing the quiescent current.
- To suppress high-frequency electromagnetic noise generated by the motor, an EMI(Electromagnetic Interference) suppression capacitor was added. Capacitors ranging from 10 to 100pF were suitable for this purpose, as they are small enough not to interfere with the PWM signal but large enough to limit voltage spikes.

The complete circuit that is used in this thesis to drive the ERM motors looks like this:

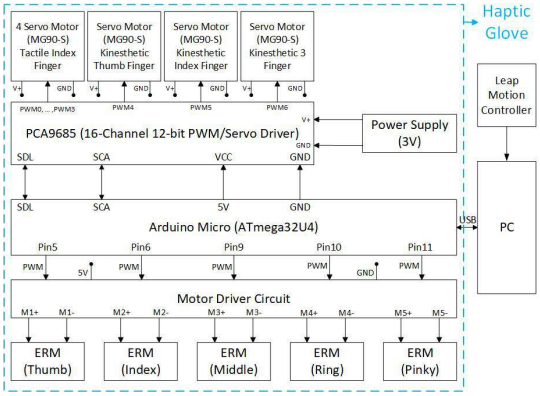

Figure 4.41: Mechatronic system block diagram with pinout information

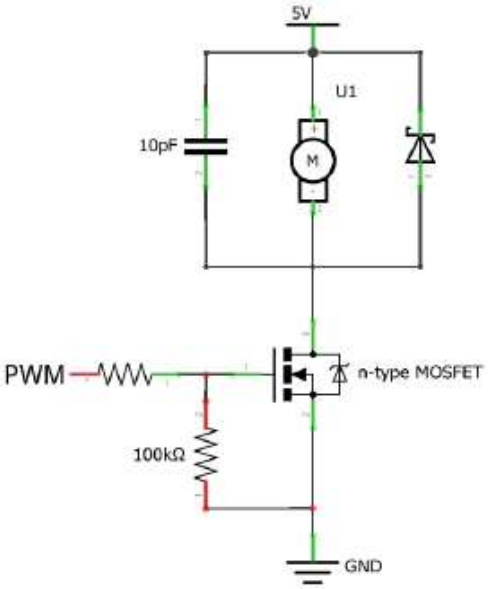

Figure 4.42: The Motor Driver Circuit

4.4 Mechanical Design

We incorporated a total of seven servo motors, each weighing 10.2 grams, into our design with the aim of ensuring that our device does not impede users' hand movements and remains lightweight. Among these, four are strategically positioned around the wrist to control the motion of the index finger's moving platform. The remaining servo motors are situated on the back of the hand and are responsible for providing kinesthetic feedback through a system of strings. To enhance functionality, we introduced the 3D-printed components described above. The 3D-printed part equipped on each fingernail weighs 2 grams. Also, we added four driving cable parts on each finger, each one weighing 1 gram.

4.4.1 Tactile Feedback

For enhancing tactile feedback our design incorporates a system consisting of individual moving platforms for each fingertip, namely the thumb, middle, ring, and pinky fingers. The key element in our approach is the use of moving platforms in conjunction with servo motors and strings. We utilized four strings, each tethered to one of the platform's corners. When a virtual object is

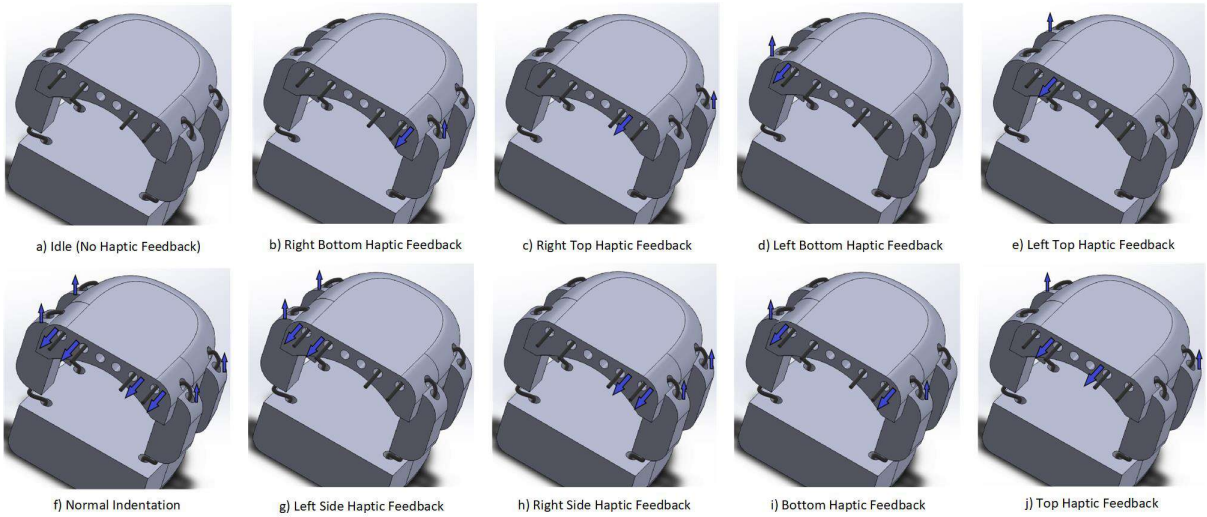

Figure 4.43: The strings that are pulled in each case to simulate location-based tactile feedback to the index finger

encountered, the corresponding servo motor activates and moves these strings. This movement of the strings pulls the platform in a precise manner against the fingertip and simulates a sensation of normal indentation. In the case of the index finger, we had a different approach. It is equipped with a moving platform connected to four servo motors, each dedicated to one of the four corners of the platform. The distribution of these strings across the platform's corners is strategically designed to simulate the complex sensations of real-world touch. Each string's attachment point corresponds to a specific region of the fingertip, enabling the system to provide feedback in the following ways:

- **Normal Indentation**, which is achieved when all four strings are simultaneously pulled, recreating the sensation of pressing on a surface.

- **Lateral Skin Stretch**, where the users can perceive different directions through shear forces applied to the fingertip, enhancing the perception of object shapes and textures.

- **Surface Curvature Display**, where the users can distinguish between round bumps, pointy bumps, and holes, adding depth and realism to the virtual interactions.

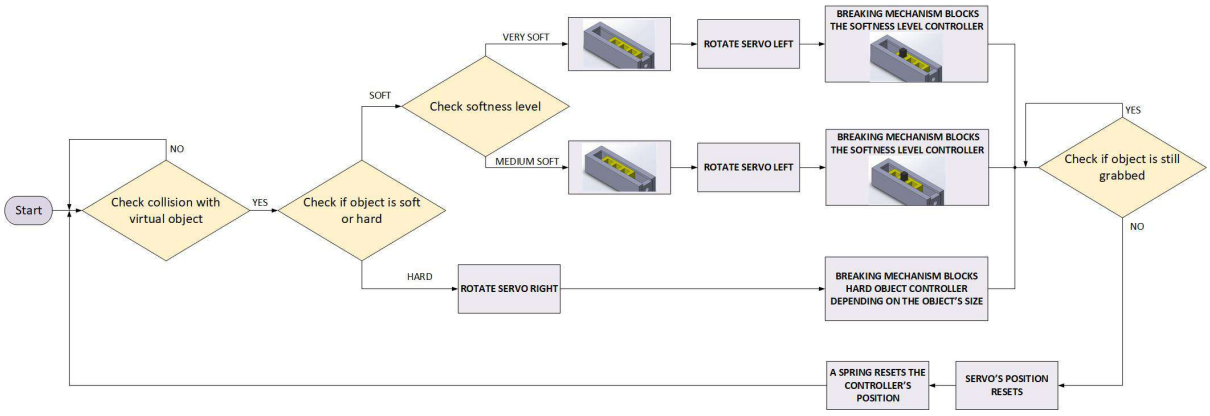

Figure 4.44: Flowchart of the kinesthetic feedback

4.4.2 Kinesthetic Feedback

When our haptic glove comes into contact with a virtual object it employs a mechanism to deliver kinesthetic feedback to the user. This feedback system is facilitated by strings connected to a 3D-printed structure, which is under the control of a servo motor. With our design, three servo motors are strategically deployed to provide kinesthetic feedback for different parts of the hand: one for the thumb, one for the index finger, and one for the remaining three fingers. This setting is designed with weight efficiency in mind, as it combines the feedback for three fingers into one actuator. When the user interacts with a virtual object, our system evaluates the object's properties. First, it determines whether the object is soft or hard. If the object is categorized as soft, the system further assesses the level of softness. Based on these evaluations, the servo motor controls the movement of the servo-breaking mechanism. Then the servo-breaking mechanism can be rotated left to block the softness level controller or right to block the hardness object controller. The softness level controller is linked to the fingers through a flexible string that can stretch and mimic the elasticity of soft objects. This provides a sensation of softness in two distinct levels. Objects with higher softness levels are associated with smaller colliders, causing the hand grip to close more in order to interact with them effectively. On the other hand, for hard objects, the system utilizes a normal string to connect the fingers to the hard object controller. Depending on the size of the object, the servo-breaking mechanism restricts the controller. The 3D-printed blocks that house these controllers are robust and capable of withstanding approximately 26.4N of force before yielding to tension. The tension

in the strings generates an effective force of 192.6N, significantly limiting the movement of the user's fingers. This restriction offers resistance, delivering convincing kinesthetic feedback that enhances the realism of the virtual interaction.

4.5 Front-End Implementation

In this section, we will provide a detailed analysis of the Unity scenes employed in our experiments. These scenes serve as the primary environment for the implementation.

4.5.1 Experiment 1: Softness/Hardness Perception

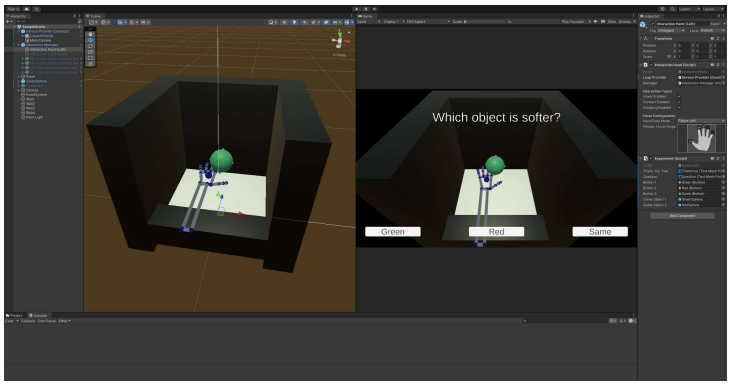

Figure 4.45: Experiment 1 Unity scene

The Unity scene in the first experiment contains the following game objects:

- 'ServiceProvider' by Leap Motion: This game object includes the 'Capsule Hands' representing hand movements and the 'Main Camera' for rendering the scene. These components are used for hand tracking and visualization.
- 'GreenSphere' and 'RedSphere': These game objects are grasped by the user in order to determine which one is softer. They also contain the 'InteractionBehaviour' from Leap Motion, which allows the user to manipulate them through the Leap Motion Controller.
- 'InteractionManager': This game object manages hand interactions through 'Interaction Hands', enabling effective Leap Motion integration. In this experiment, the 'Interaction Hand' contains the main script of our implementation, which controls the buttons and display visibility, and enables/disables the two spheres that the user interacts with.

- 'Walls': These game objects are used to block the two spheres from escaping the scene.

The primary objective of this experiment was to investigate users' perception of the softness or hardness of virtual objects. In the Softness/Hardness Experiment Unity scene, users were presented with pairs of spheres. Each sphere possessed an attribute value, which could randomly be either 0, 1, or 2. These values corresponded to the perceived softness of objects, categorizing them as soft, medium soft, or hard, respectively. Users encountered pairs of spheres representing each softness level, and these pairings were presented three times during the experiment. For each pairing, users had a limited time of 10 seconds to interact with and manipulate each of the two spheres. After the interaction with a pair of spheres, users were asked to provide feedback regarding their perception of the objects. Specifically, they had to report, using three buttons, whether they perceived the first or the second sphere as softer, or if they perceived that both spheres had the same softness level.

4.5.2 Experiment 2: Lateral Skin Stretch Perception

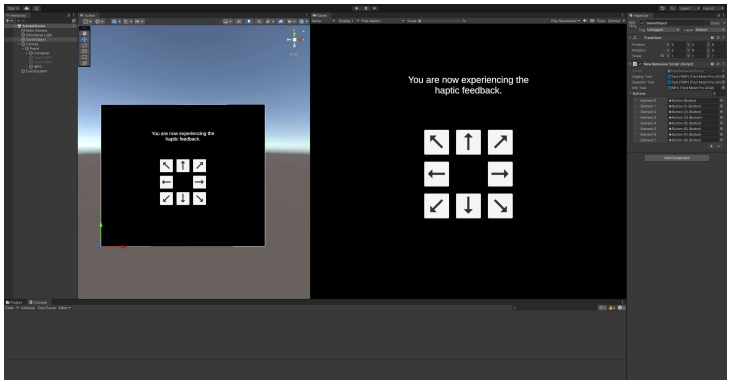

Figure 4.46: Experiment 2 Unity scene

The Unity scene for the second experiment consists of the following game objects:

- 'GameObject': This is an empty game object, which contains the script that controls the experimental procedure. More specifically, it controls the visibility of the buttons and the display text, generates random numbers each one representing one of the 8 directions, and sends the appropriate commands to the Arduino using serial communication.
- 'Canvas': This game object contains 8 buttons, one for direction as shown in Figure 4.46, and a display text that changes based on the experiment phase.

The main objective of the lateral skin stretch perception experiment was to investigate how users perceived directional feedback provided through the manipulation of a moving platform. In the second experiment, each participant was exposed to stimuli generated by the moving platform. The moving platform was equipped with four strings, which were manipulated by pulling one or more of these strings. These manipulations provided the users with specific directional information. Each direction was presented to the users three times, and the order of presentation was randomized to minimize bias. After each exposure to the haptic cues(that lasted for 3 seconds), users were presented with a set of 8 buttons, each corresponding to a specific direction. Then, they were asked to report which of the eight directions they felt they had experienced. This process was repeated until all possible directions had been presented three times. In the second phase of the experiment, vibration feedback was introduced. Users received tactile feedback through vibrations applied to their index fingertips while experiencing the directional cues from the moving platform. Then, the same procedure of reporting the feedback was carried out.

4.5.3 Experiment 3: Surface Geometry Perception

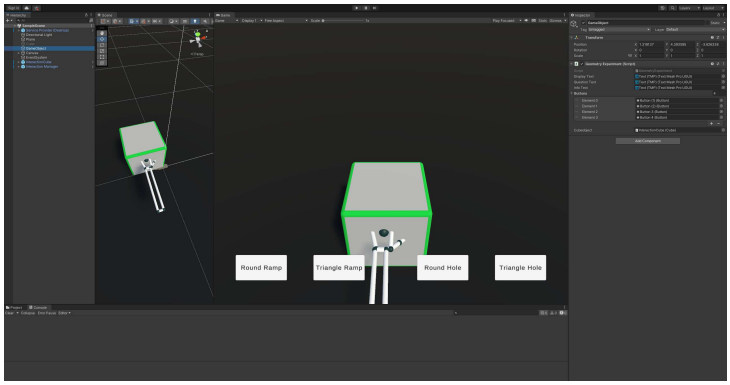

Figure 4.47: Experiment 3 Unity scene

The Unity scene for the third experiment comprises the following key game objects:

- 'ServiceProvider' by Leap Motion: This game object includes the 'Capsule Hands' representing hand movements and the 'Main Camera' for rendering the scene. These components are used for hand tracking and visualization.

- 'GameObject': This empty game object contains the main script, which controls the visibility of the buttons, sends commands to the Arduino using serial communication, and presents haptic cues from a list in random order.
- 'Canvas': The 'Canvas' contains UI elements, including the four buttons shown in Figure 4.46, along with a display text.
- 'InteractionCube': This interactive cube allows users to interact with it using the Leap Motion Controller.
- 'InteractionManager': This game object manages hand interactions through 'Interaction Hands', enabling effective Leap Motion integration.

The primary aim of the surface geometry perception experiment was to investigate how participants perceive different surface geometries through tactile feedback. In this experiment, participants were instructed to move their hands horizontally from left to right inside a box. As they moved their index finger within this space, they encountered tactile stimuli generated by a moving platform. These tactile feedback cues were developed to simulate four surface geometries: a round hole, a triangular hole, a round bump, and a triangular bump. After the exposure to each surface geometry stimulus, users were presented with a set of response options. They were instructed to select which of the four geometries they felt. Then, the experiment was repeated with the addition of vibration feedback. The entire process was repeated until each haptic feedback cue had been presented three times and in random order.

4.6 Back-End Implementation

4.6.1 Experiment 1 Arduino Code

In the first part of the code(before the setup() function, we include two libraries, "Wire.h" and "Adafruit_PWMServoDriver.h". The "Wire.h" library provides essential functions for I2C(Inter-Intergrated Circuit) communication. I2C is a two-wire serial communication protocol commonly used for connecting microcontrollers to various sensors, devices, and modules. Key functions provided by this library include 'begin()', 'write()', and more. This code is used for communicating with the Adafruit PWM Servo Driver module via I2C to send commands for servo control. The "Adafruit_PWMServoDriver.h" library is specific to Adafruit products

and is designed for controlling servo motors using the Adafruit PWM Servo Driver board. Key functions in this library are 'begin()', 'setPWM()', etc.

After the libraries, we include some global variables. With the 'char contact[20]', we declare a character array named 'contact' with a length of 20. This array is used to store incoming data received from serial communication. It can hold up to 20 characters. The line "Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();" declares an instance of the 'Adafruit_PWMServoDriver' class named 'pwm'. It is initialized using the 'Adafruit_PWMServoDriver()' constructor. In the case of the experiment with vibration enabled, we add the "const int motorPin = 10", which defines a constant integer named 'motorPin' and assigns it the value 10. This constant represents the digital pin on the Arduino to which the finger's vibration motor is connected.

After the addition of the global variables, we add some definitions. With "#define SERVOMIN 100", we define a preprocessor directive to create a constant named 'SERVOMIN' with a value of 100. It is described as the minimum pulse length count out of 4096 and is used as the minimum position for the servo motors. With "#define SERVOMAX", we define 'SERVOMAX' as 500, which is the maximum position for the servo motors. Then, we add the line "define USMIN 600", which is the constant that represents the minimum microsecond length based on the minimum pulse of 150. It is related to the timing and pulse width for the servo control. Similarly, the line "#define USMAX 2400" defines the maximum microsecond length based on the maximum pulse of 2400. Finally, with the line "#define SERVO_FREQ 50" we set the PWM frequency for the servo motors to 50Hz. This frequency affects the speed and precision of the servo movement.

```
#include <Wire.h>
#include <Adafruit_PWMServoDriver.h>

char contact[20];
Adafruit_PWMServoDriver pwm = Adafruit_PWMServoDriver();
const int motorPin = 10;

#define SERVOMIN  100 // This is the 'minimum' pulse length count (out of 4096)
#define SERVOMAX  500
#define USMIN  600 // This is the rounded 'minimum' microsecond length based on the minimum pulse of
#define USMAX  2400
#define SERVO_FREQ 50
```

Listing 4.1: The global section of the code

In the "setup()" function, we initialize the communication interface, configure the PWM driver for servo control, and set up the initial states for both servos and the motor pin. More specifically, we first include the "Serial.begin(9600)", which initializes serial communication with a baud rate of 9600 bits per second. A baud rate of 9600 provides a good balance between speed and reliability. Then, we add "pwm.begin()", which initializes the PWM driver, making it ready to control the servo motors. The initialization process sets up the communication between the Arduino and the PWM driver board. The next step was to add the "pwm.setOscillatorFrequency(27000000)", where we set the oscillator frequency to 27MHz and generated a stable PWM signal. Moreover, we add the "pwm.setPWMFreq(SERVO_FREQ)", which sets the PWM frequency for the servo motors to 50Hz. The PWM frequency determines how fast the PWM signal repeats its cycle, thus in our approach, the signal repeats 50 times per second. In the next lines, we set the initial positions for the servo motors. More specifically, for the thumb finger, we use "pwm.setPWM(0,0,195)", for the index finger we use "pwm.setPWM(0,0,380)", and for the remaining three fingers we use "pwm.setPWM(0,0,305)". All the initial positions are different because of the different positioning of the servo motors on top of the hand. After the initialization of the servo position, the next step is to set up five digital pins as output pins. These pins are used to control the vibration motor(one on each fingertip).

```
void setup()
{
Serial.begin(9600);
pwm.begin();
pwm.setPWMFreq(50);
pwm.setOscillatorFrequency(27000000);
pwm.setPWMFreq(SERVO_FREQ);
delay(10);
pwm.setPWM(0, 0, 195); //fat
pwm.setPWM(1, 0, 380); //index
pwm.setPWM(2, 0, 305); //triple
pinMode(11, OUTPUT);
pinMode(10, OUTPUT);
pinMode(9, OUTPUT);
pinMode(6, OUTPUT);
```

```
pinMode(5, OUTPUT);
}
```

Listing 4.2: The "setup()" section of the first experiment's code

In the "loop()" function of the first experiment, we continuously read and interpret commands from the serial interface and adjust the position of the servo motors and the vibration motor based on the received commands. At the start, with "int n1=10" we declare an integer variable 'n1' and initialize it with the value 10. Then, we add the "Serial.readBytesUntil(n1, contact, 4)", which reads data from the serial interface until it encounters a newline character(") or reads four characters. The received byte(character) is stored in the 'contact' area, which is used to hold the command received from Unity. The following code block processes the received command stored in the 'contact' array and it uses 'strcmp' to compare the received command with predefined commands('left', 'righ', 'lero'). If 'contact' is equal to 'left', we set the position of the three servo motors so that they move left to block the softness level controller. If 'contact is equal to 'righ', we set the position of the three servo motors so that they move right to block the hard object controller. Whenever we move the servos left or right we also enable the vibration motors to provide feedback on touch using the 'analogWrite(pinNubmer,25)'. Finally, if 'contact' is 'lero', we set both the position of the three servo motors back to their initial position and the vibration motors to 0.

```
void loop()
{
int nl=10;
Serial.readBytesUntil(nl,contact,4);
// Check if data is received from Unity

  if(strcmp(contact,"left")==0){
    pwm.setPWM(0, 0, 160); //fat
    pwm.setPWM(1, 0, 300); //index
    pwm.setPWM(2, 0, 250); //triple
    analogWrite(11, 25);
    analogWrite(10, 25);
    analogWrite(9, 25);
    analogWrite(6, 25);
    analogWrite(5, 25);
  }
```

```
  if(strcmp(contact,"righ")==0){
    pwm.setPWM(0, 0, 260); //fat
    pwm.setPWM(1, 0, 460); //index
    pwm.setPWM(2, 0, 380); //triple
    analogWrite(11, 25);
    analogWrite(10, 25);
    analogWrite(9, 25);
    analogWrite(6, 25);
    analogWrite(5, 25);
  }

  if(strcmp(contact,"lero")==0){
    pwm.setPWM(0, 0, 195); //fat
    pwm.setPWM(1, 0, 380); //index
    pwm.setPWM(2, 0, 305); //triple
    analogWrite(11, 0);
    analogWrite(10, 0);
    analogWrite(9, 0);
    analogWrite(6, 0);
    analogWrite(5, 0);
  }
}
```

Listing 4.3: The "loop()" section of the first experiment's code

4.6.2 Experiment 2 Arduino Code

Similarly to the first experiment, we follow the same steps in the "setup()" function for the second and third experiments. The difference is in the lines from "pwm.setPWM(0, 0, 500)" to "pwm.setPWM(3, 0, 500)", where we set the initial positions for the servo motors that control the index finger's moving platform. Additionally, in the case of the experiments with vibration enabled, we only use one pin with the "pinMode(motorPin, OUTPUT)", which configures the pin connected to the vibration motor as an output pin. Setting it as an output pin means it will be used to send an electrical signal to control the speed of the vibration motor using PWM.

```
void setup()
{
Serial.begin(9600);
pwm.begin();
pwm.setOscillatorFrequency(27000000);
```

```
pwm.setPWMFreq(SERVO_FREQ);
pwm.setPWM(0, 0, 500);
pwm.setPWM(1, 0, 500);
pwm.setPWM(2, 0, 500);
pwm.setPWM(3, 0, 500);
pinMode(motorPin, OUTPUT);

}
```

Listing 4.4: The "setup()" section of the second and third experiment's code

In the "loop()" function of the second experiment, we continuously read and interpret commands from the serial interface and adjust the position of the servo motors and the vibration motor based on the received commands. Similarly to the first experiment, with "int n1=10" we declare an integer variable 'n1' and initialize it with the value 10. Then, we add the "Serial.readBytesUntil(n1, contact, 1)", which reads data from the serial interface until it encounters a newline character(") or reads one byte. The following code block processes the received command stored in the 'contact' array and it uses 'strcmp' to compare the received command with predefined commands(i.e. '1', '2', ..., '9'). If a match is found, the code within the corresponding 'if' block is executed. More specifically the commands are the following:

- When the received command is 1, we want to move the top left region of the moving platform. Thus we set servo 1 to the pulling position with 'pwm.setPWM' set to 100 and the rest servos to the neutral position with 'pwm.setPWM' set to 500.

- When the received command i 2, we want to move the top region of the moving platform. Thus we set both servos 1 and 2 to the pulling position and the rest servos to the neutral position.

- When the received command is 3, we want to move the top right region of the moving platform. Thus we set servo 2 to the pulling position and the rest servos to the neutral position.

- When the received command is 4, we want to move the left region of the moving platform. Thus we set both servos 0 and 1 to the pulling position and the rest servos to the neutral position.

- When the received command is 5, we want to move the right region of the moving platform. Thus we set both servos 2 and 3 to the pulling position and the rest servos to the neutral position.
- When the received command is 6, we want to move the bottom left region of the moving platform. Thus we set servo 0 to the pulling position and the rest to the neutral position.
- When the received command is 7, we want to move the bottom region of the moving platform. Thus we set both servos 0 and 3 to the pulling position, The rest 2 servos were set to the neutral position.
- When the received command is 8, we want to move the bottom right region of the moving platform. Thus, we set servo 0, 1, and 2 to 500(neutral position) and we controlled servo 3 to move to 100(pulling position).
- When the received command is 9, all servo motors are reset to a neutral position with 'pwm.setPWM' by setting them to 500.

After processing the received command, the "loop()" function continues to the next iteration, waiting for the next command.

```
void loop()
{
  int nl=10;
  Serial.readBytesUntil(nl,contact,1);
   //top-left
   if(strcmp(contact,"1")==0){
    pwm.setPWM(0, 0, 500);
    pwm.setPWM(1, 0, 100);
    pwm.setPWM(2, 0, 500);
    pwm.setPWM(3, 0, 500);
    analogWrite(motorPin, 25);
   }
   //top
   if(strcmp(contact,"2")==0){
    pwm.setPWM(0, 0, 500);
    pwm.setPWM(1, 0, 100);
    pwm.setPWM(2, 0, 100);
    pwm.setPWM(3, 0, 500);
```

```
 analogWrite(motorPin, 25);
}
//top-right
if(strcmp(contact,"3")==0){
 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 500);
 analogWrite(motorPin, 25);
}
//left
if(strcmp(contact,"4")==0){
 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);
 analogWrite(motorPin, 25);
}
//right
if(strcmp(contact,"5")==0){
 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 100);
 analogWrite(motorPin, 25);
}
//bottom-left
if(strcmp(contact,"6")==0){
 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);
 analogWrite(motorPin, 25);
}

//bottom
if(strcmp(contact,"7")==0){
 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 100);
```

```
    analogWrite(motorPin, 25);
   }
   //bottom-right
   if(strcmp(contact,"8")==0){
    pwm.setPWM(0, 0, 500);
    pwm.setPWM(1, 0, 500);
    pwm.setPWM(2, 0, 500);
    pwm.setPWM(3, 0, 100);
    analogWrite(motorPin, 25);
   }
  //reset values
   if(strcmp(contact,"9")==0){
    pwm.setPWM(0, 0, 500);
    pwm.setPWM(1, 0, 500);
    pwm.setPWM(2, 0, 500);
    pwm.setPWM(3, 0, 500);
    analogWrite(motorPin, 0);
   }

}
```

Listing 4.5: The "loop()" section of the second experiment's code

4.6.3 Experiment 3 Arduino Code

In the "loop()" function of the third experiment, we continuously read and interpret commands from the serial interface and adjust the position of the servo motors and the vibration motor based on the received commands. Similarly to the first and second experiments, with "int n1=10" we declare an integer variable 'n1' and initialize it with the value 10. Then, we add the "Serial.readBytesUntil(n1, contact, 1)". The following code block processes the received command stored in the 'contact' array and it uses 'strcmp' to compare the received command with predefined commands(i.e. '1', '2', '3', '4'). If a match is found, the code within the corresponding 'if' block is executed. More specifically the commands are the following:

- When the received command is 1, we want to provide the round curve haptic feedback. Thus, we first move the right region of the moving platform by setting servo 3 and 4 to 100. Then, we delay the process by 1 sec and set all servos to 100 to provide normal indentation. To complete the haptic feedback for the round curve, we incline the ramp to

the opposite side, simulating the curve sensation. We do this by setting servo 1 and 2 to 100. This provides users with the perception of a curved surface. Finally, we reset all the servo positions back to 500.

- When the received command is 2, we want to provide the triangular curve haptic feedback. Thus, we first move the right region of the moving platform by setting servo 3 and 4 to 100. Following the initial feedback phase, we smoothly transition to a steeper triangular curve sensation. To achieve this, we set servos 3 and 4 back to their default position of 500 while simultaneously setting servos 1 and 2 to 100. This adjustment provides users with a more pronounced curve sensation. After delaying for 2 more seconds, we reset all the servo positions back to 500.

- When the received command is 3, we want to provide the round hole haptic feedback. Thus, we first move the left region of the moving platform by setting servo 1 and 2 to 100. Then, we delay the process by 1 sec and set all servos to 100 to provide normal indentation. After delaying for 1 more second the next step is to ascent to the opposite side, in order to create the hole, by setting the servo 3 and 4 to 100. Finally, we reset all the servo positions back to 500.

- When the received command is 4, we want to provide the triangular hole haptic feedback. Thus, we first move the left region of the moving platform by setting servo 1 and 2 to 100. Then, we delay the process by 1 sec, set the servos 1 and 2 back to 500 and servos 3 and 4 to 100 in order to provide a more steep curve. After delaying for 2 more seconds, we reset all the servo positions back to 500.

After processing the received command, the "loop()" function continues to the next iteration, waiting for the next command.

```
int nl=10;
  Serial.readBytesUntil(nl,contact,1);

   //round bump
   if(strcmp(contact,"1")==0){
    pwm.setPWM(0, 0, 500);
    pwm.setPWM(1, 0, 500);
    pwm.setPWM(2, 0, 100);
```

```
 pwm.setPWM(3, 0, 100);

 delay(1000);

 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 100);

 delay(1000);

 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 delay(1000);

 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 return;
 }

//triangle bump
if(strcmp(contact,"2")==0){
 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 100);

 delay(1000);

 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 delay(2000);
```

```
 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 return;
 }

//round hole
if(strcmp(contact,"3")==0){
 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 delay(1000);

 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 100);

 delay(1000);

 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 100);

 delay(1000);

 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 return;
 }

//triangle hole
```

```
if(strcmp(contact,"4")==0){
 pwm.setPWM(0, 0, 100);
 pwm.setPWM(1, 0, 100);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 delay(1000);

 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 100);
 pwm.setPWM(3, 0, 100);

 delay(2000);

 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);

 return;
 }


 if(strcmp(contact,"0")==0){
 pwm.setPWM(0, 0, 500);
 pwm.setPWM(1, 0, 500);
 pwm.setPWM(2, 0, 500);
 pwm.setPWM(3, 0, 500);
 }
```

Listing 4.6: The "loop()" section of the third experiment's code

4.6.4 Experiment 1 Unity Code

The first step after setting up the variables is to generate a timestamp by using the command 'System.DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss")'. It formats the date and time as a string, which includes the year, month, day, hour, minute, and second, separated by hyphens and underscores. Then, we create a file path by concatenating the formatted timestamp with the string 'experiment_data_' and appending '.csv'. After the creation of the file path, we create

a serial port with the name 'COM6' and a baud rate of 9600 bits per second. Finally, we set a timeout for the serial port to 1 ms, which means that if no data is received within 1 ms of attempting to read from the serial port a timeout exception occurs.

```
string timestamp = System.DateTime.Now.ToString("yyyy-MM-dd_HH-mm-ss");
filePath = "experiment_data_" + timestamp + ".csv";
serialPort = new SerialPort("COM6", 9600);
serialPort.Open();
serialPort.ReadTimeout = 1;
```

Listing 4.7: Create a file to store experiment's data and open serial communication for the first experiment

In Listing 4.8, we set up the initial state of the game by generating combinations, shuffling them, setting initial game object values, hiding UI elements, and adding button listeners. More specifically, we use nested foreach loops to generate all possible combinations of integers from the 'combination' array(which contains the values 0, 1, and 2). For each combination of 'i' and 'j', we create a new integer array containing these values and add it to the 'combinationList'. So, 'combinationList' contains all combinations of pairs of 0, 1, and 2. Additionally, we initialize the 'combinationCounts' dictionary with keys in the format of 'i,j'(i.e. '0,0') and set their values to 0. This dictionary is used to keep track of how many times each combination is encountered. After populating the list with all the possible combinations, we shuffle the order of elements within the list. This is done by iterating through the list and swapping each element with a randomly selected element from the list. Then, we call the "SetValues()" method(which is explained thoroughly below) with the values from the first combination in 'combinationList'. This sets the initial state of the two spheres. The next lines are used to initially hide the buttons and the question from the user. Also, we add three button click listeners that when clicked, invoke the "OnButtonClick()" method with the argument indicating which button was clicked(1, 2, or 3). In the final line of Listing 4.8, we use the 'File.WriteAllText' to create or clear a CSV file specified by the 'filepath' variable. The file will have a header line with 'Button, Value1, Value2' to indicate the columns.

```
// Populate combinationList with all possible combinations
```

```
foreach (int i in combinations)
{
    foreach (int j in combinations)
    {
        combinationList.Add(new int[] { i, j });
        combinationCounts[i + "," + j] = 0;
    }
}

// Shuffle combinationList for random order
for (int i = 0; i < combinationList.Count; i++)
{
    int randomIndex = Random.Range(i, combinationList.Count);
    int[] temp = combinationList[i];
    combinationList[i] = combinationList[randomIndex];
    combinationList[randomIndex] = temp;
}

// Set initial values for gameObject1 and gameObject2
SetValues(combinationList[index][0], combinationList[index][1]);

// Reveal buttons
button1.gameObject.SetActive(false);
button2.gameObject.SetActive(false);
button3.gameObject.SetActive(false);
question.gameObject.SetActive(false);

// Add button listeners
button1.onClick.AddListener(() => OnButtonClick(1));
button2.onClick.AddListener(() => OnButtonClick(2));
button3.onClick.AddListener(() => OnButtonClick(3));

// Create or clear the file
File.WriteAllText(filePath, "Button,Value1,Value2\n");
```

Listing 4.8: Create a list with all the possible combinations for the first experiment

The code in the Listing 4.9 controls the behavior of the game objects, it checks the conditions related to object grasping, adjusts collider sizes, sends commands through a serial port, and handles the UI element visibility. At the start, we create a conditional logic for UI ele-

ments(button visibility and thank you message) that control their visibility. If the user grasps only the first sphere or if he doesn't grasp any, the logic hides the UI elements. Then, we use 'GameObject.Find' to find the two spheres and store them in 'checkedObj1' and 'checkedObj2'. In the next lines, we implement the core function of our implementation. More specifically, we check the values of 'a'(which corresponds to the first sphere softness/hardness) and 'b'(which corresponds to the second sphere softness) and we perform the following actions based on their values:

- if 'a' is 0(which is the value for medium soft), we set the sphere collider radius to default(0.5f) and check if the user grasped the sphere. If the user is grasping the sphere, we invoke a method in order to disable the sphere after 10 sec. Also, we send the command 'left' via serial communication to the Arduino. If the user stops grasping the sphere for the duration of 10 sec before disabling the object, we send the command 'lero' to rest the servo motor position.
- if 'a' is 1(which is the value for soft), we set the sphere collider radius to 0.25f. If the user is grasping the sphere, we invoke the method that disables the game object after 10 sec. We also send the command 'left' to the Arduino.
- if 'a' is 2(which is the value for hard), we set the sphere collider radius to 0.5f. If the user is grasping the sphere, we invoke the same method as in the previous cases and we send the command 'righ' with 'serialPort.Write'.

Similarly, we perform the same actions for the different values of 'b'.

```
// Update is called once per frame
void Update()
{
    if (grasped1 && !grasped2)
    {
        button1.gameObject.SetActive(false);
        button2.gameObject.SetActive(false);
        button3.gameObject.SetActive(false);
        question.gameObject.SetActive(false);
    }
    else if (!grasped1 && !grasped2)
```

```
{
    button1.gameObject.SetActive(false);
    button2.gameObject.SetActive(false);
    button3.gameObject.SetActive(false);
    question.gameObject.SetActive(false);
}

if (thank)
{
    thankYouText.gameObject.SetActive(true);
    thankYouText.text = "Thank you for participating!";
}
GameObject checkedObj1 = GameObject.Find("GreenSphere");
GameObject checkedObj2 = GameObject.Find("RedSphere");

if (a == 0)
{
    SphereCollider collider1 = gameObject1.GetComponent<SphereCollider>();
    if (collider1 != null)
    {
        float originalRadius = 0.5f; // Adjust to the original radius
        collider1.radius = originalRadius;
    }
    if (interactionHand.graspedObject != null
    &&  interactionHand.graspedObject.gameObject == checkedObj1 && !sentData)
    {
        Debug.Log("Touch 1 but medium soft");
        grasped1 = true;
        Invoke("DisableObject1", 10.0f);
        serialPort.Write("left");
        sentData = true;
    }
    else if (!interactionHand.isGraspingObject && sentData)
    {
        sentData = false;
        serialPort.Write("lero");
        Debug.Log("Sent data: -1 (reset servo)");
    }
}
else if (a == 2)
{
```

```
        SphereCollider collider1 = gameObject1.GetComponent<SphereCollider>();
        if (collider1 != null)
        {
            float originalRadius = 0.5f; // Adjust to the original radius
            collider1.radius = originalRadius;
        }
        if (interactionHand.graspedObject != null
        && interactionHand.graspedObject.gameObject == checkedObj1 && !sentData)
        {
            Debug.Log("Touch 1 but hard");
            grasped1 = true;
            Invoke("DisableObject1", 10.0f);
            serialPort.Write("righ");
            sentData = true;
        }
        else if (!interactionHand.isGraspingObject && sentData)
        {
            sentData = false;
            serialPort.Write("lero");
            Debug.Log("Sent data: -1 (reset servo)");
        }
    }
    else if (a == 1)
    {
        // Modify the collider size of gameObject1 when a equals 1
        SphereCollider collider1 = gameObject1.GetComponent<SphereCollider>();
        if (collider1 != null)
        {
            float originalRadius = 0.25f; // Adjust to the original radius
            collider1.radius = originalRadius;
        }
        if (interactionHand.graspedObject != null
        && interactionHand.graspedObject.gameObject == checkedObj1 && !sentData)
        {
            Debug.Log("Touch 1 soft");
            grasped1 = true;
            Invoke("DisableObject1", 10.0f);
            serialPort.Write("left");
            sentData = true;
        }
        else if (!interactionHand.isGraspingObject && sentData)
```

```
        {
            sentData = false;
            serialPort.Write("lero");
            Debug.Log("Sent data: -1 (reset servo)");
        }
    }

    if (b == 0)
    {
        SphereCollider collider2 = gameObject2.GetComponent<SphereCollider>();
        if (collider2 != null)
        {
            float originalRadius = 0.5f; // Adjust to the original radius
            collider2.radius = originalRadius;
        }
        if (interactionHand.graspedObject != null
        && interactionHand.graspedObject.gameObject == checkedObj2 && !sentData)
        {
            Debug.Log("Touch 2 but medium soft");
            grasped2 = true;
            Invoke("DisableObject2", 10.0f);
            serialPort.Write("left");
            sentData = true;
        }
        else if (!interactionHand.isGraspingObject && sentData)
        {
            sentData = false;
            serialPort.Write("lero");
            Debug.Log("Sent data: -1 (reset servo)");
        }
    }
    else if (b == 2)
    {
        SphereCollider collider2 = gameObject2.GetComponent<SphereCollider>();
        if (collider2 != null)
        {
            float originalRadius = 0.5f; // Adjust to the original radius
            collider2.radius = originalRadius;
        }
        if (interactionHand.graspedObject != null
        && interactionHand.graspedObject.gameObject == checkedObj2 && !sentData)
```

```
        {
            Debug.Log("Touch 2 but hard");
            grasped2 = true;
            Invoke("DisableObject2", 10.0f);
            serialPort.Write("righ");
            sentData = true;
        }
        else if (!interactionHand.isGraspingObject && sentData)
        {
            sentData = false;
            serialPort.Write("lero");
            Debug.Log("Sent data: -1 (reset servo)");
        }
        }
    else if (b == 1)
    {
        SphereCollider collider2 = gameObject2.GetComponent<SphereCollider>();
        if (collider2 != null)
        {
            float originalRadius = 0.25f; // Adjust to the original radius
            collider2.radius = originalRadius;
        }
        if (interactionHand.graspedObject != null
        && interactionHand.graspedObject.gameObject == checkedObj2 && !sentData)
        {
            Debug.Log("Touch 2 but soft");
            grasped2 = true;
            Invoke("DisableObject2", 10.0f);
            serialPort.Write("left");
            sentData = true;
        }
        else if (!interactionHand.isGraspingObject && sentData)
        {
            sentData = false;
            serialPort.Write("lero");
            Debug.Log("Sent data: -1 (reset servo)");
        }
    }
```

Listing 4.9: Send commands via serial communication depending the object softness or hardness

Then, we added the "SetValues()" method, which sets random values to the attributes of the two spheres.

```
void SetValues(int value1, int value2)
{
    gameObject1.GetComponent<GraspedObject>().attribute = value1;
    gameObject2.GetComponent<GraspedObject>().attribute = value2;

    if (value1 == 0)
    {
        a = 0;
    }
    else if (value1 == 2)
    {
        a = 2;
    }
    else if (value1 == 1)
    {
        a = 1;
    }

    if (value2 == 0)
    {
        b = 0;
    }
    else if (value2 == 2)
    {
        b = 2;
    }
    else if (value2 == 1)
    {
        b = 1;
    }
}

}
```

Listing 4.10: The "SetValues()" method for the first experiment

The code in the Listing 4.11 is the method "OnButtonClick()", which gets executed when a

button is clicked in the Unity application. At the start, we create a string 'line' that contains the button number and the values from the current combination(from 'combinationList') separated by commas. It also appends this 'line' to a file specified by the 'filePath'. Then we reset the values that indicate if the objects are being grasped, hide several UI elements, and reactivate the first sphere after a button press. The next step is to increment the count for the current combination in the 'combinationCounts' dictionary. The key for the dictionary is a string formed by concatenating the values of the current combination. The next step is to check if the count for the current combination is greater than or equal to 3(because we want to check each combination 3 times). If so, we remove the current combination from the 'combinationList'. If all combinations have been checked, we end the experiment and activate a thank you message. However, if there are more combinations to show, we choose a random index within the range of available combinations and assign it to the 'index' variable. Finally, we call the "SetValues()" method to set new attribute values for the two spheres based on the next combination in 'combinationList'.

```
void OnButtonClick(int button)
{
    // Record button press and values in the file
    string line = button + "," + combinationList[index][0] + "," + combinationList[index][1] + "\n";
    File.AppendAllText(filePath, line);
    grasped1 = false;
    grasped2 = false;
    button1.gameObject.SetActive(false);
    button2.gameObject.SetActive(false);
    button3.gameObject.SetActive(false);
    question.gameObject.SetActive(false);
    gameObject1.SetActive(true);

    // Increment count for current combination
    combinationCounts[combinationList[index][0] + "," + combinationList[index][1]]++;

    // If current combination has been shown 3 times, remove it from combinationList
    if (combinationCounts[combinationList[index][0] + "," + combinationList[index][1]] >= 3)
    {
        combinationList.RemoveAt(index);
    }
```

```
    // If all combinations have been checked, show a text and end the experiment
    if (combinationList.Count == 0)
    {
        thank = true;
        return;
    }

    // Choose a random index for next combination
    index = Random.Range(0, combinationList.Count);

    // Set new values for gameObject1 and gameObject2
    SetValues(combinationList[index][0], combinationList[index][1]);
}
```

Listing 4.11: The "OnButtonClick()" method for the first experiment

4.6.5 Experiment 2 Unity Code

At the start, we declare the following variables:

- 'numbersList', which is a list of integers initialized with the values from 1 to 8 and represents a list of available numbers.
- 'numberCount', which is an integer array with a length of 8. It was used to count how many times each number had been selected.
- 'randomNumber' which is an integer variable that was used to store a randomly selected number.
- 'filePath', which is a string variable that stores the path to a CSV file.
- 'sp', which is a SerialPort object configured to communicate over the COM6 port at a baud rate of 9600.

Then at the "Start()" method, we construct the 'filePath', using the 'Application.dataPath', which appends a timestamp to the filename that stores the results of the experiment. Then we open a serial port for communication and we send the character '9' to Arduino in order to reset the servo motor position. Additionally, we call the "ShowRandomNumber()" method and set a text to "Which orientation did you feel?".

```
private List<int> numbersList = new List<int> { 1, 2, 3, 4, 5, 6, 7, 8 };
private int[] numberCount = new int[8];
private int randomNumber;
private string filePath;
private SerialPort sp = new SerialPort("COM6", 9600);

private void Start()
{
    filePath = Application.dataPath + "/selectedNumbers_"
    + System.DateTime.Now.ToString("yyyyMMdd_HHmmss") + ".csv";
    sp.Open();
    sp.Write("9");
    ShowRandomNumber();
    questionText.text = "Which orientation did you feel?";
}
```

Listing 4.12: Initialization of variables, create a file to store experiment's data and open serial communication for the second experiment

After the "Start()" method we create the "OptionSelected()" method, where we track and record the user's selections in response to randomly displayed haptic feedback cues. More specifically, we increment the count of the 'randomNumber' in the 'numberCount' array, which tracks how many times each haptic feedback has been selected. Then, with the if condition we check whether the current haptic feedback has been selected three or more times. If the condition is true, we remove the current haptic feedback from the list. With the 'using (StreamWriter streamWriter = File.AppendText(filePath))', we open a file, where we write the button pressed and the haptic feedback separated by commas. Finally, if all haptic feedback cues have been presented, we display an appropriate message and call the "ShowRandomNumber()" method.

```
public void OptionSelected(int buttonNumber)
{
    numberCount[randomNumber - 1]++;
    if (numberCount[randomNumber - 1] >= 3)
    {
        numbersList.Remove(randomNumber);
    }
```

```
    using (StreamWriter streamWriter = File.AppendText(filePath))
    {
        streamWriter.WriteLine(buttonNumber + "," + randomNumber);
    }

    if (numbersList.Count == 0)
    {
        displayText.text = "All numbers have been selected 3 times.";
        return;
    }
    ShowRandomNumber();
}
```

Listing 4.13: The "OptionSelected()" method for the second experiment

The "ShowRandomNumber()" method is responsible for presenting random haptic feedback to the user. More specifically, we select a random haptic cue from the list and assign it to the 'randomNumber' variable. Then, we disable the buttons and texts. Finally, we send the random haptic feedback cue to the Arduino using serial communication and we call the co-routine "WaitAndEnableButtons()".

```
private void ShowRandomNumber()
{
    randomNumber = numbersList[Random.Range(0, numbersList.Count)];
    displayText.text = "Number: " + randomNumber;
    questionText.text = "Which orientation did you feel?";

    // Disable buttons
    foreach (Button button in buttons)
    {
        button.interactable = false;
        questionText.gameObject.SetActive(false);
        infoText.gameObject.SetActive(true);
    }
    sp.Write(randomNumber.ToString());
    StartCoroutine(WaitAndEnableButtons());
}
```

Listing 4.14: The "ShowRandomNumber()" method for the second experiment

The co-routine "WaitAndEnableButtons()" is used to introduce a delay of 3 seconds(by using the IEnumerator) during which the buttons are disabled, and the character '9' is sent to the Arduino in order to reset the servos.

```
IEnumerator WaitAndEnableButtons()
{
    yield return new WaitForSeconds(3f);
    sp.Write("9");
    foreach (Button button in buttons)
    {
        button.interactable = true;
        questionText.gameObject.SetActive(true);
        infoText.gameObject.SetActive(false);
    }
}
```

Listing 4.15: The "WaitAndEnableButton()" method for the second experiment

4.6.6 Experiment 3 Unity Code

The main difference between the "Start()" method of the second and the third experiment is the 'numberList' initialization which has the values '1, 2, 3, 4'. These values represent the four different haptic cues presented in this thesis. Additionally, in the third experiment, our reset command sent to Arduino is '0' instead of '9'. Also, we changed the question text.

```
private List<int> numbersList = new List<int> { 1, 2, 3, 4 };
private int[] numberCount = new int[8];
private int randomNumber;
private string filePath;
private SerialPort sp = new SerialPort("COM6", 9600);
public Cube cubeobject;
private bool stopflag = false;

private void Start()
{
    filePath = Application.dataPath + "/selectedNumbers_"
    + System.DateTime.Now.ToString("yyyyMMdd_HHmmss") + ".csv";
```

```
    sp.Open();
    sp.Write("0");
    foreach (Button button in buttons)
    {
        button.interactable = false;
        questionText.gameObject.SetActive(false);
        infoText.gameObject.SetActive(false);
    }
    questionText.text = "Which type of surface geometry did you feel?";
}
```

Listing 4.16: Initialization of variables, create a file to store experiment's data and open serial communication for the third experiment

The code provided is part of the "Cube.cs" script and serves two crucial functions. Firstly, it implements the "OnTriggerEnter()" method, responsible for detecting collisions between the user's finger and the cube, and subsequently setting the variable 'entered' to 'true' upon collision detection. Secondly, it includes the "OnTriggerExit()" method, which monitors when the user's finger exits the cube's collision zone and promptly updates the 'entered' variable to 'false'. These functions are essential for tracking user interactions with the cube, enabling the application to respond accurately to touch or collision events, and providing a reliable mechanism for determining when the cube is in contact with the user's finger or not.

```
private void OnTriggerEnter(Collider other)
{
    if( other.gameObject.name == "Contact Fingerbone6")
    {
        entered = true;
        Debug.Log("Entered");

    }
}

private void OnTriggerExit(Collider other)
{
    if (other.gameObject.name == "Contact Fingerbone6")
    {
        entered = false;
```

```
        Debug.Log("Exited");
    }
}
```

Listing 4.17: The "OnTriggerEnter()" and OnTriggerExit()" methods in "Cube.cs" script

The check in the "Update()" method, which examines the 'cubeobject.entered' condition, serves as a conditional trigger mechanism to control the execution of the experiment. When 'cubeobject.entered' is true, it invokes the "ShowRandomNumber()" method and sets the 'stopflag' to false. This mechanism prevents redundant or repeated calls to "ShowRandomNumber()".

```
private void Update()
{
    if (cubeobject.entered && !stopflag)
    {
        ShowRandomNumber();
        stopflag = true; // Set the stopflag to true to prevent repeated calls
    }
}
```

Listing 4.18: The "Update()" method in the third experiment

Similarly to the second experiment, the "OptionSelected()" method is responsible for updating the counts of the selected haptic cues, removing numbers that have been selected three times, and logging user selection to a file. However, in the third experiment, we made some modifications. More specifically, we reset the 'stopflag' in order to control the execution of the "ShowRandomNumber()" method. Also, we include a foreach iteration to control the visibility of the buttons.

```
public void OptionSelected(int buttonNumber)
{
    numberCount[randomNumber - 1]++;
    if (numberCount[randomNumber - 1] >= 3)
    {
```

```
            numbersList.Remove(randomNumber);
        }
        using (StreamWriter streamWriter = File.AppendText(filePath))
        {
            streamWriter.WriteLine(buttonNumber + "," + randomNumber);
        }

        if (numbersList.Count == 0)
        {
            displayText.text = "All numbers have been selected 3 times.";
            return;
        }

        // Reset the stopflag to allow ShowRandomNumber to be called again
        stopflag = false;
        foreach (Button button in buttons)
        {
            button.interactable = false;
            questionText.gameObject.SetActive(false);
            infoText.gameObject.SetActive(false);
        }
}
```

Listing 4.19: The "OptionSelected()" method for the third experiment

The only differences between the "ShowRandomNumber()" method in the second and third experiments are the question text, which is set to 'Which type of surface geometry did you feel?', and using the same foreach iteration to disable the buttons.

```
private void ShowRandomNumber()
{
    randomNumber = numbersList[Random.Range(0, numbersList.Count)];
    displayText.text = "Number: " + randomNumber;
    questionText.text = "Which type of surface geometry did you feel?";

    // Disable buttons
    foreach (Button button in buttons)
    {
```

```
        button.interactable = false;
        questionText.gameObject.SetActive(false);
        infoText.gameObject.SetActive(false);
    }

    sp.Write(randomNumber.ToString());
    StartCoroutine(WaitAndEnableButtons());
}
```

Listing 4.20: The "ShowRandomNumber()" method for the third experiment

The "WaitAndEnableButtons()" method is the same as in the second experiment with the only difference being in the command sent to Arduino.

```
IEnumerator WaitAndEnableButtons()
{
    yield return new WaitForSeconds(3f);
    sp.Write("0");
    foreach (Button button in buttons)
    {
        button.interactable = true;
        questionText.gameObject.SetActive(true);
        infoText.gameObject.SetActive(true);
    }
}
```

Listing 4.21: The "WaitAndEnableButtons()" method for the third experiment

Chapter 5

Evaluations & Results

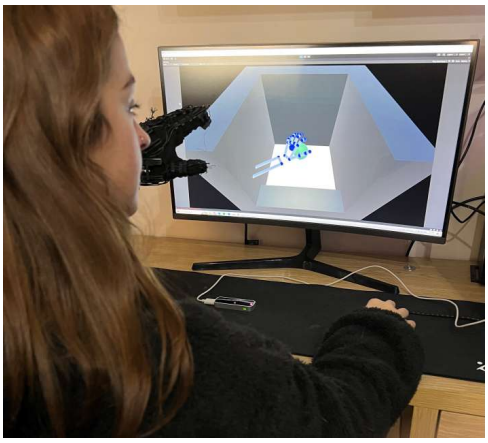

Figure 5.1: Experimental setup for our implementation

5.1 Introduction

We perform a set of experiments to evaluate the effectiveness of haptic feedback. These experiments involve evaluating the ability to distinguish the softness or stiffness of objects, as well as assessing the lateral skin stretch on the fingertip and how well users can identify the geometry of a surface.

5.1.1 Apparatus

We utilized a 27-inch Samsung CJG50 display, boasting a resolution of 2560 x 1440 and a rapid refresh rate of 144 Hz. To monitor the user's hand and finger movements, we employed the

Leap Motion Controller, which has a system consisting of two cameras and three infrared LEDs that trace infrared light with a wavelength measuring 850 nanometers. The applications were executed on a desktop computer featuring an AMD Ryzen 7-3700X CPU, 16 GB of RAM, and a solitary Radeon RX-5700XT GPU.

5.1.2 Participants

In the research, 18 individuals participated, including 7 females, with an average age of 26.67 years (SD 2.68), all of whom had either normal vision or vision corrected to normal. Participants were positioned in front of the screen, as depicted in Figure 5.1. Each participant wore the haptic glove and engaged with virtual objects to become acquainted with the tracking capabilities of the Leap Motion Controller. Additionally, they wore headphones to block out external auditory distractions. Subsequently, the experiment sequence was initiated. On average, it took approximately 27.5 minutes for each participant to complete the three experiments and the Leap Motion Controller assessment.

5.2 Results and Discussion

5.2.1 Performance Metrics: Softness/Hardness Perception

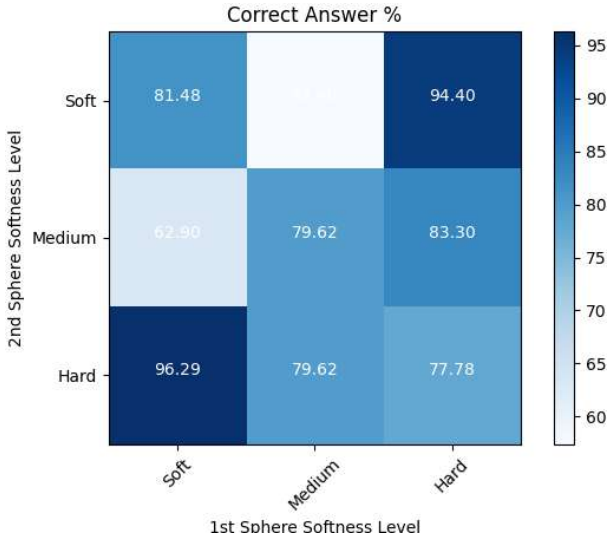

Figure 5.2: The percentage of correct answers on each combination

Findings from Experiment 1, which focused on assessing the perception of softness and hardness, demonstrated better accuracy when users were tasked with distinguishing between a soft and a hard sphere. Success rates for these conditions were 96.29% and 94.4%, respectively (see 5.2). However, when the softness levels of the two spheres were set to medium soft and soft, the success rate notably declined to 62.9% and 57.4%. These results suggest that users encountered challenges in detecting subtle variations in softness.

5.2.2 Performance Metrics: Lateral Skin Stretch Perception

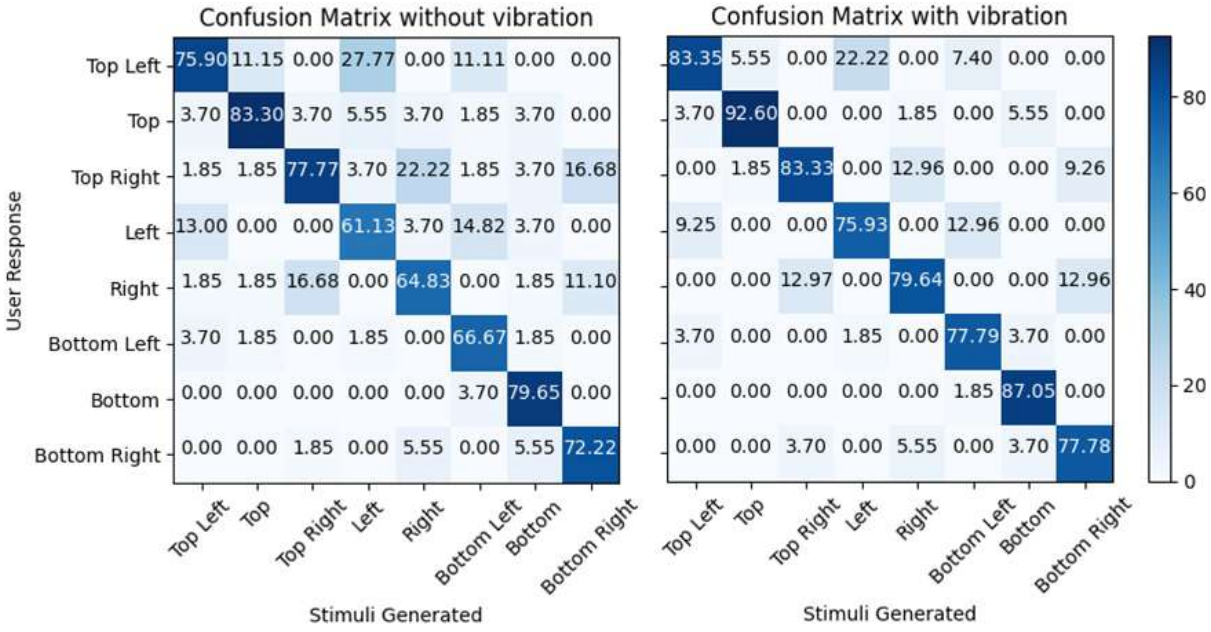

Figure 5.3: Confusion matrices of 2nd experiment with and without vibration on fingertip

As depicted in Figure 5.3, the outcomes from Experiment 2, which focused on perceiving lateral skin stretch, revealed that participants successfully identified the platform's orientation, with success rates exceeding 61.13% depending on the specific orientation. When vibration was introduced to the fingertip, these success rates increased to above 75.93%. These findings indicate that the addition of vibration to the fingertip enhanced participants' ability to distinguish the platform's orientation. In Figure 5.4, we summarized the results by categorizing them into the following groups:

- Grouping top left, left, and bottom left orientations as "left."

- Grouping top right, right, and bottom right orientations as "right."
- Keeping top orientation as "top."
- Keeping bottom orientation as "bottom."

When the eight platform orientations were condensed into four categories, it became evident that participants successfully distinguished the primary orientation of the platform, achieving success rates exceeding 87.05% with vibration and 79.65% without vibration. Thus, these findings indicate that users could reliably identify whether the platform was oriented as top, left, right, or bottom, but had greater difficulty recognizing the precise corner orientation. Notably, the inclusion of fingertip vibration also led to improved results.

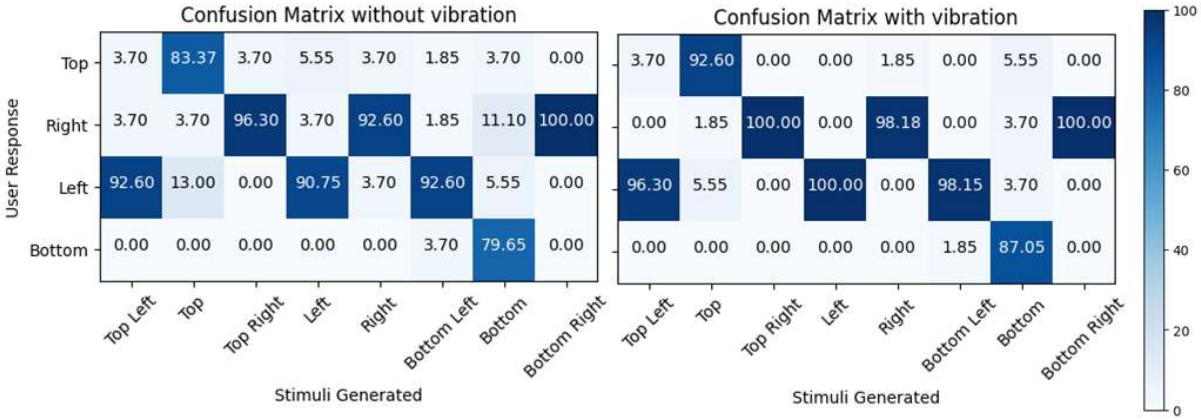

Figure 5.4: Confusion matrices of the simplified orientations with and without vibration

5.2.3 Performance Metrics: Surface Geometry Perception

As shown in Figure 5.5, the results of the third experiment, which focuses on surface geometry perception, indicated that participants effectively differentiated between various surface geometries, achieving success rates exceeding 81.48% without vibration and 87.04% with vibration. These findings imply that participants were capable of distinguishing subtle differences in surface geometry, which was further enhanced when vibrations were introduced.

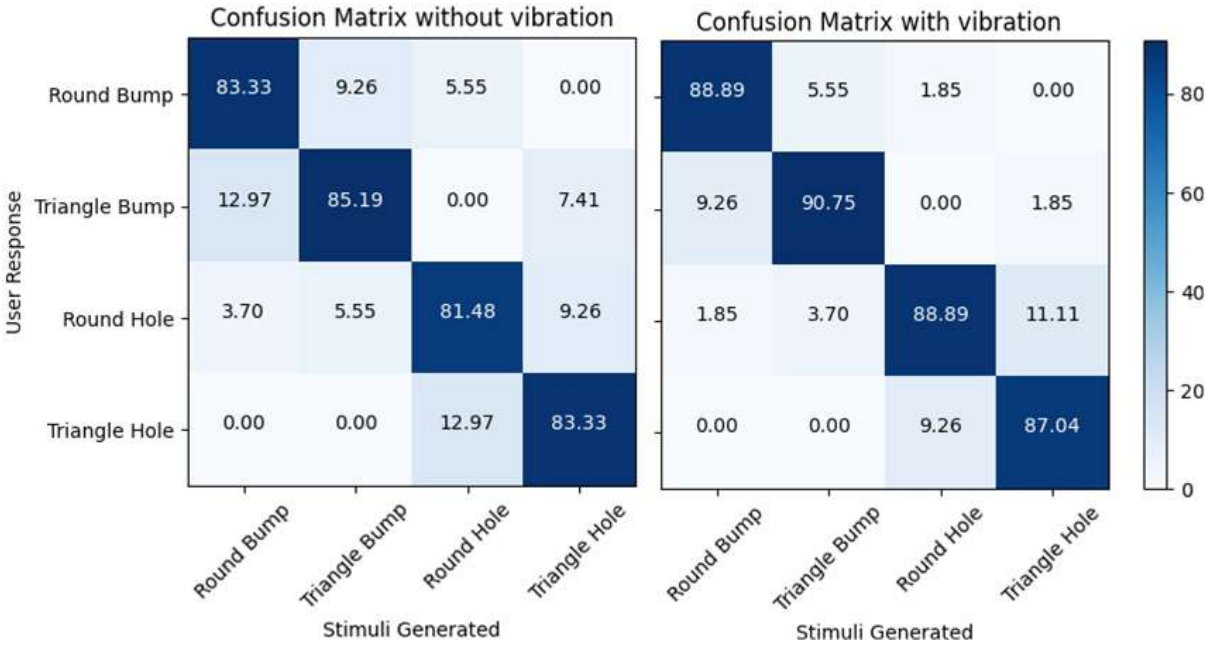

Figure 5.5: Confusion matrices of the 3rd experiment with and without vibration on fingertip

Chapter 6

Conclusion, Limitations & Future Work

6.1 Limitations and Intuitions

We introduce an innovative wearable haptic feedback device capable of delivering tactile sensations through vibration, simulating normal indentation, and replicating surface geometries. Our design employs four servo motors that connect to a mobile platform positioned on the fingertip, offering eight-directional movement. What sets our haptic glove apart is its focus on being lightweight and compact, distinguishing it from previous systems. Additionally, our system provides kinesthetic feedback, enabling users to experience the varying softness levels of objects and interactions with objects of different hardness levels. To gauge user perception, we conducted three formal experiments assessing object softness, hardness, directional cues, and surface geometry recognition.

Our analysis indicates that incorporating vibration in the second and third experiments enhanced participants' ability to distinguish directional cues and recognize surface geometries. However, our first experiment brought to light a limitation of our haptic device. Users encountered notable challenges when attempting to perceive minor differences in the softness of virtual objects. This limitation became particularly evident when users were asked to differentiate between objects that were soft and medium soft. In such cases, the success rate in accurately identifying differences in softness decreased. This limitation is crucial to address, as real-world scenarios often involve subtle variations in softness. An important limitation we encountered was related to the learning curve associated with the Leap Motion Controller. Participants

who were not familiar with the device required additional time to become accustomed to its operation and to effectively track their hand movements. This learning curve can pose a significant challenge, especially in applications where user-friendliness and rapid deployment are important.

Furthermore, participants expressed that the haptic glove was comfortable and easy to wear. Following the successful completion of the experiments, they conveyed that the sensory feedback was unique, allowing them to genuinely experience the sensation of interacting with both soft and hard objects. Their astonishment was particularly notable during the geometry perception experiment, as they were able to perceive various shapes.

6.2 Future Work

In the future, we aim to transition to a wireless design. Eliminating the need for constraining cables offers users freedom of movement within virtual environments. This advancement not only enhances user mobility but also reduces potential safety hazards associated with tangled cords. The wireless design should aim for seamless connectivity and a robust battery life, ensuring uninterrupted haptic experiences.

Addressing the limitations of the Leap Motion Controller, future iterations of the haptic glove will integrate cutting-edge tracking technologies. These technologies should provide precise and robust hand-tracking capabilities, reducing the learning curve for users and simplifying setup processes. The tracking solution will be self-contained within the device, eliminating the need for external tracking hardware. We will also explore computer vision systems, and machine learning algorithms, that can deliver reliable and user-friendly tracking.

While our current implementation utilizes Eccentric Rotating Mass (ERM) motors for vibration feedback, the transition to Linear Resonant Actuators (LRAs) holds immense promise. LRAs offer a higher degree of control and precision in generating vibrations. By integrating LRAs, the haptic device can deliver a more diverse range of vibrations, closely simulating real-world tactile sensations. This enhancement is particularly valuable in scenarios where accurate feedback is critical, such as medical simulations.

Recognizing the diverse range of hand and finger sizes among users, future haptic glove designs will prioritize customization and inclusivity. Offering multiple sizing options ensures that the device fits securely and comfortably on various hand shapes. This approach not only enhances user comfort but also optimizes the effectiveness of the haptic feedback.

Achieving the highest level of perceptual accuracy in simulating object softness is important. Future iterations of the haptic device will incorporate separate servo motors for controlling softness and hardness independently. This separation ensures that the device can accurately convey even the subtlest differences in softness levels.

Bibliography

[1] J. Kreimeier, S. Hammer, D. Friedmann, P. Karg, C. Bühner, L. Bankel, and T. Götzelmann. Evaluation of different types of haptic feedback influencing the task-based presence and performance in virtual reality. In Proceedings of the 12th ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '19), pages 289–298. ACM, 2019. 1

[2] E. Whitmire, H. Benko, C. Holz, E. Ofek, and M. Sinclair. Haptic Revolver: Touch, Shear, Texture, and Shape Rendering on a Reconfigurable Virtual Reality Controller. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18), pages 1–12. ACM, 2018. 1

[3] Q. -Z. Ang, B. Horan and S. Nahavandi. Multipoint Haptic Mediator Interface for Robotic Teleoperation. In IEEE Systems Journal, vol. 9, no. 1, pages 86-97. IEEE, 2015. 1

[4] R. M. Pierce, E. A. Fedalei and K. J. Kuchenbecker. A wearable device for controlling a robot gripper with fingertip contact, pressure, vibrotactile, and grip force feedback. 2014 IEEE Haptics Symposium (HAPTICS), pages 19-25. IEEE, 2014. 1

[5] C. Pacchierotti, A. Tirmizi, G. Bianchini and D. Prattichizzo. Enhancing the Performance of Passive Teleoperation Systems via Cutaneous Feedback. In IEEE Transactions on Haptics, vol. 8, no. 4, pages 397-409. IEEE, 2015. 1

[6] D. D'Auria, F. Persia and B. Siciliano. A Low-Cost Haptic System for Wrist Rehabilitation. 2015 IEEE International Conference on Information Reuse and Integration, pages 491-495. IEEE, 2015. 1

[7] A. Gupta and M. K. O'Malley. Design of a haptic arm exoskeleton for training and rehabilitation. In IEEE/ASME Transactions on Mechatronics, vol. 11, no. 3, pages 280-289. IEEE, 2006. 1

[8] Optimo Arm, https://roboligent.com/ 1

[9] HapticVR, https://fundamentalsurgery.com/ 2

[10] M. Solazzi, A. Frisoli and M. Bergamasco. Design of a novel finger haptic interface for contact and orientation display. 2010 IEEE Haptics Symposium, pages 129-132. IEEE, 2010. 2

[11] S. B. Schorr and A. M. Okamura. Fingertip Tactile Devices for Virtual Object Manipulation and Exploration. In Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems (CHI '17), pages 3115–3119. ACM. 2017. 2

[12] J. Iqbal, N. G. Tsagarakis, and D. G. Caldwell. Four-fingered lightweight exoskeleton robotic device accommodating different hand sizes. Electron. Lett., 51: pages 888-890. 2

[13] K. Tadano, M. Akai, K. Kadota and K. Kawashima. Development of grip amplified glove using bi-articular mechanism with pneumatic artificial rubber muscle. 2010 IEEE International Conference on Robotics and Automation, pages 2363-2368. IEEE, 2010. 2

[14] F. Chinello, C. Pacchierotti, M. Malvezzi and D. Prattichizzo. A Three Revolute-Revolute-Spherical Wearable Fingertip Cutaneous Device for Stiffness Rendering. In IEEE Transactions on Haptics, vol. 11, no. 1, pages 39-50. IEEE, 2018. 2

[15] S. Nisar, M. O. Martinez, T. Endo, F. Matsuno and A. M. Okamura. Effects of Different Hand-Grounding Locations on Haptic Performance With a Wearable Kinesthetic Haptic Device. In IEEE Robotics and Automation Letters, vol. 4, no. 2, pages 351-358. IEEE, 2019. 2

[16] X. Gu, Y. Zhang, W. Sun, Y. Bian, D. Zhou, and P. O. Kristensson. Dexmo: An Inexpensive and Lightweight Mechanical Exoskeleton for Motion Capture and Force Feedback in VR. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16), pages 1991-1995. ACM, 2016. 7

[17] HaptX Gloves DK2, https://haptx.com/ 8

[18] M. Hosseini, A. Sengül, Y. Pane, J. De Schutter and H. Bruyninck. ExoTen-Glove: A Force-Feedback Haptic Glove Based on Twisted String Actuation System. In 27th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), pages 320-327. IEEE, 2018. 8, 9

[19] R. Hinchet, V. Vechev, H. Shea, and O. Hilliges. DextrES: Wearable Haptic Feedback for Grasping in VR via a Thin Form-Factor Electrostatic Brake. In Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology (UIST '18), pages 901–912. ACM, 2018. 9

[20] M. Roumeliotis and K. Mania. Development of a Wearable Embedded System providing Tactile and Kinesthetic Haptics Feedback for 3D Interactive Applications. In SIGGRAPH Asia 2021 Posters (SA '21 Posters), Article 8, pages 1–2. ACM, 2021. 10

[21] M. Efraimidis and K. Mania. Wireless Embedded System on a Glove for Hand Motion Capture and Tactile Feedback in 3D Environments. In SIGGRAPH Asia 2020 Posters (SA '20), Article 8, pages 1–2. ACM, 2020. 10, 11

[22] F. Chinello, M. Malvezzi, D. Prattichizzo and C. Pacchierotti. A Modular Wearable Finger Interface for Cutaneous and Kinesthetic Interaction: Control and Evaluation. In IEEE Transactions on Industrial Electronics, vol. 67, no. 1, pages 706-716. IEEE, 2020. 11, 12

[23] D. Leonardis, M. Solazzi, I. Bortone and A. Frisoli. A 3-RSR Haptic Wearable Device for Rendering Fingertip Contact Forces. In IEEE Transactions on Haptics, vol. 10, no. 3, pages 305-316. IEEE, 2017. 12

[24] F. H. Giraud, S. Joshi and J. Paik. Haptigami: A Fingertip Haptic Interface With Vibrotactile and 3-DoF Cutaneous Force Feedback. In IEEE Transactions on Haptics, vol. 15, no. 1, pages 131-141. IEEE, 2022. 13

[25] S. V. Salazar, C. Pacchierotti, X. de Tinguy, A. Maciel and M. Marchal. Altering the Stiffness, Friction, and Shape Perception of Tangible Objects in Virtual Reality Using Wearable Haptics. In IEEE Transactions on Haptics, vol. 13, no. 1, pages 167-174. IEEE, 2020. 13, 14

[26] D. Prattichizzo, F. Chinello, C. Pacchierotti and M. Malvezzi. Towards Wearability in Fingertip Haptics: A 3-DoF Wearable Device for Cutaneous Force Feedback. In IEEE Transactions on Haptics, vol. 6, no. 4, pages 506-516. IEEE, 2013. 14

[27] M. Gabardi, M. Solazzi, D. Leonardis and A. Frisoli. A new wearable fingertip haptic interface for the rendering of virtual shapes and surface features. In 2016 IEEE Haptics Symposium (HAPTICS), pages 140-146. IEEE, 2016. 15

[28] A. Girard, M. Marchal, F. Gosselin, A. Chabrier, F. Louveau, A. Lécuyer. HapTip: Displaying Haptic Shear Forces at the Fingertips for Multi-Finger Interaction in Virtual Environments. Frontiers in ICT, vol. 3, 2016. 15, 16

[29] H. Benko, C. Holz, M. Sinclair, and E. Ofek. NormalTouch and TextureTouch: High-fidelity 3D Haptic Shape Rendering on Handheld Virtual Reality Controllers. In Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16), pages 717–728. ACM, 2016. 16

[30] I. Choi, E. W. Hawkes, D. L. Christensen, C. J. Ploch and S. Follmer. Wolverine: A wearable haptic interface for grasping in virtual reality. 2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 986-993. IEEE, 2016. 17

[31] I. Choi, H. Culbertson, M. R. Miller, A. Olwal, and S. Follmer. Grabity: A Wearable Haptic Interface for Simulating Weight and Grasping in Virtual Reality. In Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology (UIST '17), pages 119–130. ACM, 2017. 17, 18

[32] I. Choi, Eyal Ofek, H. Benko, M. Sinclair, and C. Holz. CLAW: A Multifunctional Handheld Haptic Controller for Grasping, Touching, and Triggering in Virtual Reality. In Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18), Paper 654, pages 1–13. ACM, 2018. 18

[33] K. G. Sreeni, K. Priyadarshini, A. K. Praseedha, S. Chaudhuri. Haptic Rendering of Cultural Heritage Objects at Different Scales. In: Isokoski, P., Springare, J. (eds) Haptics: Perception, Devices, Mobility, and Communication. EuroHaptics 2012. Lecture Notes in Computer Science, vol 7282. Springer, 2012. 19

[34] S. Krumpen, R. Klein, and M. Weinmann. Towards Tangible Cultural Heritage Experiences—Enriching VR-based Object Inspection with Haptic Feedback. J. Comput. Cult. Herit. 15, 1, Article 19. 2021. 20

[35] J. Ma, L. Sindorf, I. Liao, and J. Frazier. Using a Tangible Versus a Multi-touch Graphical User Interface to Support Data Exploration at a Museum Exhibit. In Proceedings of the Ninth International Conference on Tangible, Embedded, and Embodied Interaction (TEI '15), pages 33–40. ACM, 2015. 20

[36] X. Ji, X. Liu, V. Cacucciolo, Y. Civet, A. El, S. Cantin, Y. Perriard, H. Shea. Untethered Feel-Through Haptics Using 18-μm Thick Dielectric Elastomer Actuators. In Adv. Funct. Mater. 2021. 21

[37] J. Lu, Z. Liu, J. Brooks, and P. Lopes. Chemical Haptics: Rendering Haptic Sensations via Topical Stimulants. In The 34th Annual ACM Symposium on User Interface Software and Technology (UIST '21), pages 239–257. ACM, 2021. 21, 22