José Miguel Mendes Canelha

Programmable Vibrotactile Sensations For Virtual Reality

1 2 9 0 UNIVERSIDADE D COIMBRA

1 2 9 0

UNIVERSIDADE D
COIMBRA

José Miguel Mendes Canelha

Programmable Vibrotactile Sensations For Virtual Reality

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Professor Jorge Cardoso and presented to Faculty of Sciences and Technology / Department of Informatics Engineering.

March 2021

Faculty of Sciences and Technology

Department of Informatics Engineering

Programmable Vibrotactile Sensations for Virtual Reality

José Miguel Mendes Canelha

Dissertation in the context of the Master in Informatics Engineering, Specialization in Software Engineering, advised by Prof. Jorge Cardoso and presented to the Faculty of Sciences and Technology / Department of Informatics Engineering.

March 2021

1 2 9 0

UNIVERSIDADE D
COIMBRA

This page is intentionally left blank.

Acknowledgments

First of all, I would like to begin by expressing my deep gratitude to Professor Jorge Cardoso for the opportunity to participate in this exciting project and especially for the availability that he always had during the whole semester. Secondly, I would like to extend my gratitude to André Perrotta, not only for the opportunity to participate in the TherTactExo project but also for his availability to clarify any doubts that arose.

To all my colleagues that I now have the pleasure to call friends, a big thank you for everything. Without you, this journey would not be the same.

Finally, I want to thank from the bottom of my heart my parents, grandmother, and sister for always supporting me during this long journey. It is to you that I dedicate this work. Thank you!

This page is intentionally left blank.

Abstract

Virtual reality has been growing in diverse areas in the recent past, ranging from games to the medical field. Simultaneously with this growth, the field of haptics stands out. Haptic displays are emerging as powerful interaction support for improving the realism of virtual worlds. In most virtual reality applications, auditory and visual systems have a more significant presence, whereas the tactile system is somewhat absent. With haptic devices, these applications can incorporate tactile feedback, bringing the user the ability to touch and feel objects in a virtual environment, creating many new and exciting applications.

Virtual reality has also been prominent in rehabilitation processes for different pathologies such as stroke, spatial memory loss, and spinal cord injuries. We emphasize spinal cord injuries within these pathologies, one of the most significant clinical challenges of today.

In 2016, a group of researchers presented a study that resulted in neurological improvements through a 12-month intensive training using brain-machine interfaces. Today, the TherTactExo team proposes modifications to reduce the training time through a new protocol that includes virtual reality associated with different sensory stimuli.

This project aims to develop a vibrotactile pattern editor and virtual reality components to assist the TherTactExo team in their research. Besides, this project also intends to provide tools to help projects involving virtual reality experiences with vibrotactile feedback. We started by researching the state of the art, where we analyzed existing editors. After the analysis, we commenced the process of building our system. We designed an architecture based on a Middleware that allows vibrotactile feedback to be expressed directly from the editor as well as from virtual reality components. This process resulted in the Vibrotactile Editor's development, which introduces features that allow the user to create highly customizable vibrotactile patterns through a built-in waveform editor and the multi-channel timeline. We also present the components developed for virtual reality, namely the Unity and A-Frame components, enabling the user to create virtual experiences where user interaction with these components expresses vibrotactile feedback.

Keywords

Vibrotactile Sensations, Virtual Reality, Haptics, Graphical Editor, A-Frame, Unity.

This page is intentionally left blank.

Resumo

A realidade virtual tem vindo a crescer nos últimos anos, estendendo-se de áreas de jogos a áreas como medicina. Junto deste crescimento, destaca-se o campo das sensações hápticas. Os dispositivos hápticos surgem como um mecanismo de suporte de interacção para melhorar o realismo dos mundos virtuais. Na maioria das aplicações de realidade virtual, os sistemas auditivos e visuais têm uma presença mais significativa, enquanto que o sistema táctil está ligeiramente ausente. Com os dispositivos hápticos, as aplicações de realidade virtual podem incorporar feedback táctil, oferecendo ao utilizador a capacidade de tocar e sentir objectos num ambiente virtual, dando origem à criação de um grande número de aplicações novas e interessantes.

A realidade virtual também tem sido proeminente nos processos de reabilitação de diferentes patologias, tais como o AVC, perda de memória, e lesões da medula espinal. Aqui, damos ênfase às lesões da medula espinal, um dos maiores desafios clínicos da actualidade.

Em 2016, um grupo de investigadores apresentou um estudo que resultou em melhorias neurológicas através de um treino intensivo de 12 meses, utilizando interfaces cérebro-computador. Hoje, a equipa TherTactExo, propõem modificações para reduzir o tempo de treino através de um novo protocolo que inclui a realidade virtual associada a diferentes estímulos sensoriais.

Este projecto visa desenvolver um editor de padrões vibrotácteis e componentes de realidade virtual para ajudar a equipa TherTactExo na sua investigação. Além disso, este projecto pretende também disponibilizar ferramentas para ajudar projectos que envolvam experiências de realidade virtual com feedback vibrotáctil. Começámos por pesquisar o estado da arte, onde analisámos os editores existentes. Após a sua análise, iniciámos o processo de construção do nosso sistema. Concebemos uma arquitectura baseada num Middleware que permite que o feedback vibrotáctil seja expresso directamente a partir do editor, bem como através dos componentes de realidade virtual. Este processo resultou no desenvolvimento do Vibrotactile Editor, um editor que apresenta características que permitem ao utilizador criar padrões vibrotáteis altamente personalizáveis através de um editor de forma de onda integrado e de uma "timeline" multicanal. Apresentamos também os componentes desenvolvidos para a realidade virtual, nomeadamente os componentes Unity e A-Frame, que permitem ao utilizador criar experiências virtuais onde a interacção do utilizador com estes componentes exprime o feedback vibrotáctil.

Palavras-Chave

Sensações Vibrotácteis, Realidade Virtual, Hápticos, Editor Gráfico, Unity, Aframe.

This page is intentionally left blank.

Contents

This page is intentionally left blank.

Acronyms

API Application Programming Interface. 2, 81, 82, 87

BCI Brain-Computer Interface. xv, 6–8

CSS Cascading Style Sheets. 29, 33, 62

DGS Direcção Geral de Saúde. 71, 79

EEG Electroencephalography. 7, 8

ERM Eccentric Rotating Mass. 13

GUI Graphical User Interface. xv, xvii, 26–29, 59, 62, 66

HTML HyperText Markup Language. 29, 35, 62, 66, 83

LRA Linear Resonant Actuator. 13

MEG Magnetoencephalography. 8

MRI Magnetic Resonance Imaging. 8

PWM Pulse Width Modulation. 14

REST Representational State Transfer. 62

SCI Spinal Cord Injury. 1, 6, 10, 15

SDLC Software Development Life Cycle. 37

SUS System Usability Scale. xvi, 71–73, 75, 77, 78

SVG Scalable Vector Graphics. 62

UI User Interface. 29, 78

URI Uniform Resource Identifier. 62

VR Virtual Reality. xvi, 1–3, 15, 26, 35, 36, 57, 69

WAV Waveform Audio File. 21, 58, 62, 81

This page is intentionally left blank.

List of Figures

List of Tables

This page is intentionally left blank.

Chapter 1

Introduction

In 2016, a group of researchers proposed and demonstrated a study that induces partial neurological recovery in paraplegic patients through long-term training with a brain-machine interface-based gait protocol [1]. Here, the patients were exposed to intense, immersive VR training, enhanced visual-tactile feedback, and walking with a brain-controlled exoskeleton. The overall integration of these multiple stimuli increased the subject's neural activity.

Today, researchers are working on an improvement to the previous study, named the the TherTactExo project. This project consists of an upgrade to the previous research reported in Donati [1]. The main goal is to reduce the duration of a training protocol to achieve partial neurological recovery of sensorimotor function in Spinal Cord Injury (SCI) patients through an increased number of feedback stimuli coherent with the subjects' intention during the SCI rehabilitation. To accomplish this objective, the TherTactExo team defined a new rehabilitation protocol, detailed in section 2.5.

1.1 Motivation

This dissertation aims to assist the TherTactExo by providing a tool for creating programmable vibrotactile sensations for VR environments. In these virtual reality experiences, programming vibrotactile sensations prove to be difficult as there are no high-level toolkits available to work with the TherTactExo project's hardware specifications. Therefore, it is necessary to develop a straightforward-to-use editor capable of creating different vibrotactile sensations. The editor must allow users to visualize the vibration pattern over time, test and calibrate the vibration patterns on the hardware, and export it into a machine-and-human-readable format. The resulting tactile feedback will be expressed through a feedback sleeve composed of several actuators placed on the patient's arm during the VR experience.

1.2 Challenges

Every project has its challenges. Being able to participate in such an ambitious project carries many challenges. Understanding the background work and the theoretical foundations behind its implementation it is utterly essential.

Virtual Reality associated with Rehabilitation or merely Virtual Rehabilitation has many benefits and consequentially many challenges. For example, when comparing to the "tradi-

tional rehabilitation", Virtual Rehabilitation can be presented in different formats, bringing interactivity and motivation to the patient. Therefore, the first challenge is to develop interactive and motivating virtual environments. Furthermore, these challenges are common to Virtual Rehabilitation projects in general [2]. The first is clinical acceptance, which is conditioned on proved medical efficacy. The VR interfaces pose another challenge. The current standard VR interfaces are not designed as medical equipment, meaning that they cannot accommodate the different patients. Finally, and one challenge that fits directly in our project's context is the shape, format, and weight of the haptic feedback equipment. The patients must be able to interact with virtual environments naturally, without usability constraints.

The next challenge is to deliver software according to the best practices of development. Presenting a software product must always follow software development norms and standards. We also intend that this software be used not only by researchers in the medical area but also for users with fewer computer skills. Therefore, it is imperative to build an easy to learn, consistent, and efficient editor. As a consequence of this challenge, one rises. The fact that there are few editors similar to what we plan to develop makes designing one a challenge because there are few guiding examples. The same challenges apply to the development of the Application Programming Interface (API). There is no guiding example to help in the development. Also, the API design must be flexible to not only be used by the delineated technologies but also for a wide range of applications.

Finally, we intend to develop the editor directly for the TherTactExo team. Considering that the team is currently working with diverse platforms, another challenge is to build the editor to interact with all the future implemented systems.

1.3 Objectives

Our work aims to help the TherTactExo team researchers create vibrotactile feedback by developing a vibrotactile feedback editor to be used in the current training protocol. Thus, the objectives of this project are:

- Develop a vibration pattern editor. This editor should be flexible enough to allow different numbers and configurations of actuators, allow users to visualize the vibration pattern over time, test and calibrate the vibration patterns on the hardware, and export the pattern into a machine- and human-readable format.
- Develop VR components for A-Frame[1] and Unity to control the vibration actuators according to pre-configured vibration patterns. We need to develop components for the scenarios already defined in the rehabilitation protocol, which are environments that experience "walking on the pavement", "sand" or "grass".
- Develop VR applications that showcase the work done in the two previous objectives. We will create three virtual reality scenarios:
    1. A stone environment: for the experience of "walking on the pavement".
    2. A grass environment: for the experience of "walking on grass".
    3. A sand environment: for the experience of "walking on sand".

[1] A web framework for building virtual reality applications

1.4 Contributions

The main contributions from this projects are:

- Vibrotactile Editor that assists the TherTactExo team in creating VR experiences with tactile feedback
- VR components for vibrotactile feedback that will be used to build the final VR experience by the TherTactExo team
- Editor usability evaluation
- A-Frame component API usability evaluation
- Open-source software for the Vibrotactile Editor
- Open-source software for the Middleware
- Open-source software for the A-Frame component
- Open-source software for the Unity component

1.5 Document Structure

This document is structured as follows:

- **Chapter 1 - Introduction**: This chapter presents the overview of the work to be done, including the motivation for its development, challenges, contributions, and objectives.
- **Chapter 2 - Background**: This chapter presents the background knowledge essential to the further reading of this document.
- **Chapter 3 - State of the art**: This chapter presents the existent platforms available in the market as well as the technologies to be used.
- **Chapter 4 - Methodology and Work Plan**: This chapter presents the followed methodology for developing software and the work plan for both semesters.
- **Chapter 5 - Requirements**: This chapter presents the requirements for the project.
- **Chapter 6 - Architecture**: This chapter presents the architecture designed for the project.
- **Chapter 7 - The Vibrotactile Editor System**: This chapter describes the Vibrotactile Editor System done during this dissertation.
- **Chapter 8 - Evaluation**: This chapter describes the evaluations done during this dissertation.
- **Chapter 9 - Conclusion**: This chapter describes an overview of this dissertation.

This page is intentionally left blank.

Chapter 2

Background

In this chapter, we briefly introduce the reader to the fundamental concepts that we will be addressing during the present document. Given that this dissertation aims at providing a tool for an ongoing project, there is also the need to provide background knowledge of what is already implemented and how the interaction with it will be carried out.

2.1 Rehabilitation through Virtual Reality

The use of virtual reality in different healthcare applications is increasing. We found various virtual reality applications such as rehabilitation of spatial memory [3], rehabilitation of patients after a stroke [4], rehabilitation of physical activity in patients admitted to the intensive care unit [5], and rehabilitation of patients with spinal cord injuries [1], which is the motivation of this work. A significant reason for the growth in the inclusion of virtual reality in this context is the entertainment it brings to the participants, leading to a greater willingness to participate in the rehabilitation process [6]. In [5], the authors evaluated the level of activity a Nintendo Wii console can provide and the patient's satisfaction. The results were that 86% of the participants would like to play the videogame in future physical rehabilitation sessions.

Application of virtual reality can be classified as immersive, semi-immersive, non-immersive [6]:

- Immersive: Refers to the extent the user is integrated into a virtual world experience that is extensive, surrounding, inclusive, vivid, and matching [7].
- Semi-immersive: Refers to the extent to which the user interacts with the virtual worlds through a surrounding screen [4].
- Non-immersive: Refers to the extent to which the user interacts with the virtual world through a screen or monitor [4].

Rehabilitation associated with virtual reality shows promises in patients' recovery. The rehabilitation process shows improvement in isolation and when complemented with conventional rehabilitation [4]. Besides, these studies also found that patients are more entertained and are more willing to participate in the rehabilitation process when incorporated with virtual reality [5, 6].

2.1.1 Spinal Cord Injury Rehabilitation

Spinal Cord Injury (SCI) is a neurological condition associated with many life-changing limitations, and rehabilitation remains a major clinical challenge. Motor recoveries after neurological injury mostly depend on maximizing neuroplasticity. Neuroplasticity can be defined as "the ability of the nervous system to respond to intrinsic or extrinsic stimuli by reorganizing its structure, function, and connections"[8].

The brain is a component in the nervous system that controls the various functions of the human body. It constitutes what is designated as the central nervous system, along with the spinal cord and nerves. The nervous system is also composed of the peripherical nervous system, consisting of sensory neurons, ganglia, and nerves that connect to the central nervous system.

We can view the central nervous system as the command center of the body. Functionally, the nervous system can be divided into the somatic and the autonomic components. The autonomic component regulates specific body processes, such as blood pressure and the rate of breathing. The somatic part consists of nerves that connect the brain and spinal cord with muscles and sensory receptors in the skin [9]. Therefore, damage in the spinal cord causes permanent changes in strength, sensations, and body functions below the injury zone. The severity of the injury can be classified as complete or incomplete, depending on the loss of sensory and motor functions.

Virtual reality is primarily used in the rehabilitation of this pathology in protocols that "used games to provide stimuli that encourage movements to improve motor function, balance, aerobic function, and pain" [10]. There are several types of methodologies and protocols associated with the rehabilitation of patients with spinal cord injuries. However, they all are "commonly used to induce or facilitate processes of neural regeneration and plasticity, which might lead to significant functional recovery after SCI" [10].

2.2 Brain-computer Interfaces

A Brain-Computer Interface (BCI) is a device that measures and captures the activity of the central nervous system and translates the signals transmitted by the brain into artificial output signals of a computer system [11]. BCI's are a technology that grows great interest, not only in the scientific community but also in the lay public. The principal reason for this stems from the fact that this technology has the potential to restore motor behaviors in severely handicapped patients and also to control limb prostheses in amputees patients [12].

A BCI framework or application is essentially composed of six steps in a closed-loop process. Figure 2.1, presents the whole process.

Measuring brain activity is where the transmitted brain signal is acquired, amplified, and digitalized. Two distinct methods for signal-acquisition result in different categories of BCIs. They are categorized by using invasive or non-invasive methodologies, which we are described later in this chapter.

The first phase is preprocessing the signal. In this phase is where we acquire the "brain-data" that will be used to detect the subject signals. Usually, when using EEG, this data tends to contain a lot of noise and needs to be filtered out [14].

Next, we have the feature extraction. One of the significant challenges in BCI platforms

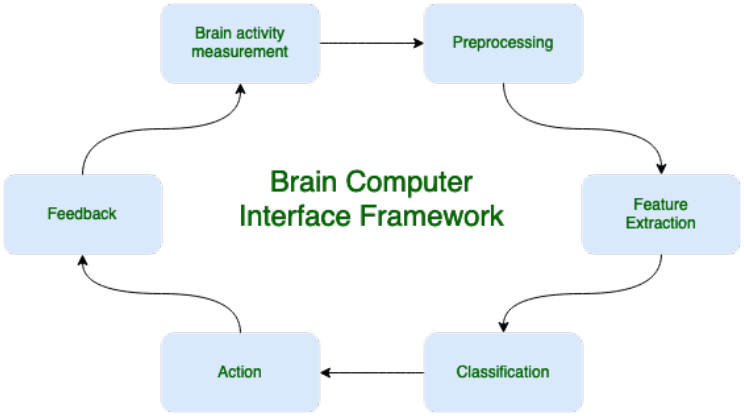


Figure 2.1: BCI Framework (redrawn from [13])

is developing algorithms for translating neural activity. One known drawback is trying to "decipher the subject's voluntary intentions, and decisions through measurements of the joined electrical activity of the neuronal populations"[12]. Feature extraction is all about translating the neural activity into a feature or a command. It is significantly essential in the BCI systems due to its role in the precise representation of the classification stage at specifying mental states [15]. The digitalized signal is subjected to a variety of feature extraction procedures, such as spatial filtering, voltage amplitude measurements, spectral analyses, or single-neuron separation. Also, BCIs can use signal features that are in the time domain or the frequency domains. Once extracted these signal features, a translation algorithm translates these signal features into orders that carry out the user's intent. These algorithms might use linear methods, such as classical statistical analysis, or non-linear methods, such as neural networks [16]. In present-day BCIs, numerous techniques are applied during the feature extraction phase. For example, in [17], wavelet-based feature extraction algorithms were introduced. This algorithm, as described in [15], decomposes a signal, allowing us to "analyze the different frequency bands, with different resolutions. The signal decomposition is obtained by employing two sets of functions, called scaling and wavelet functions, which are associated with low-pass filters and high-pass filters." Also, in [15], it is provided a description and a comparison of feature extraction algorithms and their impact in the classification stage.

In the classification phase, the classifiers translate the extracted features into actions/device commands. Classifications of mental states vary according to the BCI system's design. For this reason, a broad spectrum of classification methods is used to design BCIs systems, such as Linear Discriminant Analysis, Support Vector Machine, and Hidden Markov Model [13].

Finally, after determining the user's intents through the specific signal features, these are translated into actions that operate a device. "Success depends on the interaction of two adaptive controllers, user, and system. The user must develop and maintain a good correlation between his or her intent and the signal features employed by the BCI. The BCI must select and extract features that the user can control and translate those features into device commands correctly and efficiently" [16].

2.2.1 Non-invasive BCI's Systems

In non-invasive systems, the modulation of the brain signals are recorded from the surface of the head using Electroencephalography (EEG) [18]. These systems are commonly

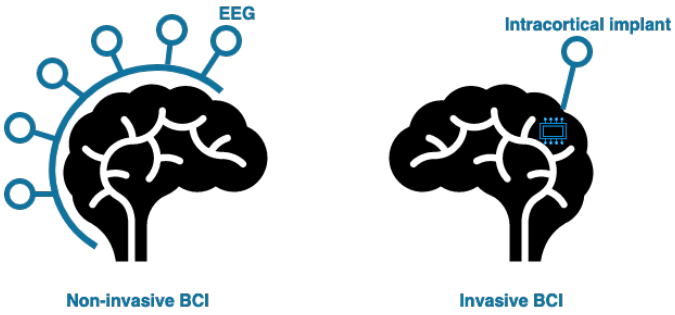


Figure 2.2: Invasive and Non-invasive BCIs

named EEG-Based BCI. An EEG is a method that detects electrical activity in the brain using small electrodes attached to the scalp (Figure 2.2) [19]. This method provides a simple solution to get the brain recordings and requires relatively low-priced equipment. Other non-invasive methods are based on neuroimaging technologies [15], such as Magnetoencephalography (MEG), where signals are "obtained directly from the magnetic fields produced by the brain activity" [20], and Magnetic Resonance Imaging (MRI), where "functional MRI signals reflect brain activity indirectly by measuring the oxygenation of blood flowing near active neurons" [20].

2.2.2 Invasive BCI's Systems

In invasive systems, the devices to capture signals are inserted directly in the human brain. The recordings of the brain activity are measured based on methods using epidural, subdural, or intracortical electrodes (Figure 2.2) [16]. This methodology provides better neural signals with high potential for further improvement; however, it carries the risk associated with an invasive surgical procedure to place the electrodes [12]. There is a higher threshold for adopting this methodology instead of a non-invasive one. It should be used when they can provide communication superior to that offered by non-invasive methods or other problems that impede non-invasive methods [16].

2.3 Exoskeletons

The concept and research on exoskeletons arose around the 1960s when General Electric developed the first exoskeleton prototype [21]. Designated as Hardiman, this powered exoskeleton was designed to allow the user to carry heavy loads with ease. However, it was too heavy to accomplish its purpose [21]. Figure 2.3 illustrates the Hardiman concept, as described in [21].

An exoskeleton is an external mechanical structure, with joints and links that simulate those of the human body, respectively. Worn by the user, it is an external wearable framework that facilitates, allows, assists and enhances physical activity through mechanical interaction with the human body [22].

Taking into account the numerous possibilities of this technology, countless experiments with exoskeletons were conducted. As a result, a broad spectrum of applications related to exoskeletons has emerged with industry, military, and medicine standing out as the most common.

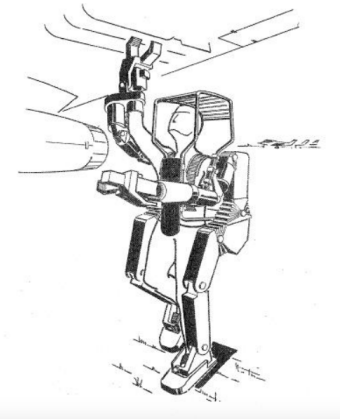


Figure 2.3: The hardiman project [21]

2.3.1 Industry and Military Applications

In industry and military sectors, exoskeletons are being developed to enhance the ability of a healthy human. In the industrial area, the purpose of an exoskeleton is to increase, amplify, or reinforce the performance of the worker, often targeting specific body components. [23].

In situations where the work is mostly physical - lifting and handling heavy materials -, the aim is to increase the productivity gain, improve the quality of work and reduce the risk of work-related muscle injuries [23]. Figure 2.4 illustrates a leg exoskeleton designed for people who engage in load-carrying activities.

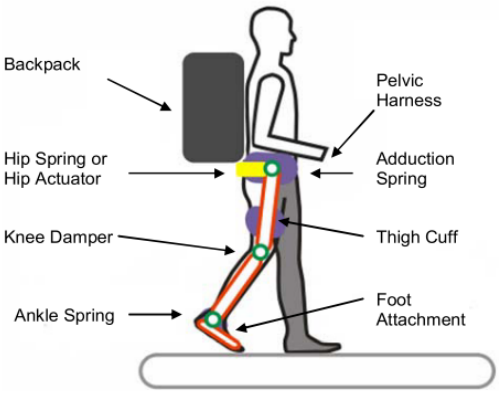


Figure 2.4: Leg exoskeleton components [24]

In the military area, the same paradigm is followed. Exoskeletons aim to enhance the military's physical capacity and resistance and increase load capacity without mobility loss [24].

2.3.2 Medical Applications

In medicine, exoskeletons are commonly used for rehabilitation purposes associated with a wide range of injuries, such as sports injuries, spinal cord injuries, and strokes. The treatment for these conditions relies on extensive physiotherapy procedures that require a high level of one to one attention from the therapist [25]. With exoskeletons, a single

therapist can administer rehabilitation training with little physical labor due to mechanical assistance that the exoskeletons provide [26].

For example, the exoskeleton used in [1] was developed in the neurorehabilitation paradigm for restoring locomotion in patients suffering from SCI. In [26], a lower limb orthosis (Figure 2.5) was designed to assist during motor rehabilitation in patients suffering from neuronal injuries through the re-learning of gating patterns. Exoskeletons can assist in the rehabilitation of other pathologies, as is the case of strokes. Strokes can cause severe long-term disabilities, such as abnormal muscle activation and coordination, muscle weakness, and dexterity, and precision loss [27]. In [28], an exoskeleton was designed to aid primarily hemiplegic[1] stroke patients with loss of function in the upper extremities, more specifically to simulate a rehabilitation therapy session to achieve finger extension to open up the palm, in order to resume motor activities and maintain daily activities.

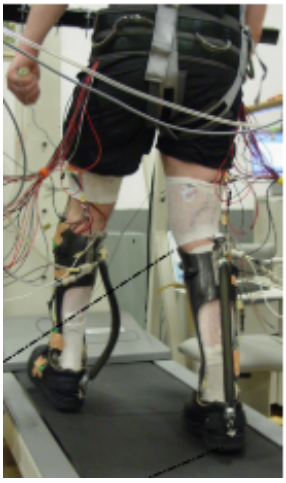


Figure 2.5: Powered Ankle-Foot Orthosis [26]

2.4 Haptics

Haptics comes from the Greek *haptikós*, meaning "able to grasp or perceive." It refers to the ability to identify and perceive the properties of objects through our sense of touch and, more specifically, on active touch [29].

Haptic perception is fundamental to interact with the environment successfully. For example, in the past decade, prosthetic devices have advanced remarkably, replicating nearly the degrees of freedom of the human hand [30]. However, the use of these advanced devices has been limited, due primarily to the feedback deficit. The primary reason for this drawback is the feedback deficit from sensory receptors in the skin [30]. To overcome this difficulty, researchers put their efforts into creating sensors that provide tactile and kinesthetic feedback. To achieve this end, it is crucial to study the fundamentals of the human haptic system. Here, we summarize some terminologies and fundamental concepts essential to the further reading of this document and provide relevant information about designing haptic interfaces.

[1] total paralysis of one side of the body

2.4.1 Human Haptic System

The human haptic system is bidirectional in the sense that the information it receives is a function of hand movements used to explore the environment [30]. It consists of the mechanical, sensory, motor, and cognitive components, as illustrated in Figure 2.6. The tactile sensing information from the hand can be divided into two classes. These classes delineate the features of the haptic sense. The first is denominated as tactile information. Tactile information refers to the stimulation of sensors in the skin known as mechanoreceptors. Kinesthetic information completes the tactile sensing. It refers to the sense of position and motion of the limbs that comes from sensors in muscles, tendons, joints, and the forces generated by muscles [30]. The obtained sensory information from the sensors in the skin and muscles is then transmitted to the brain, where it is processed in distinct cortical areas to give raise to our haptic experiences.

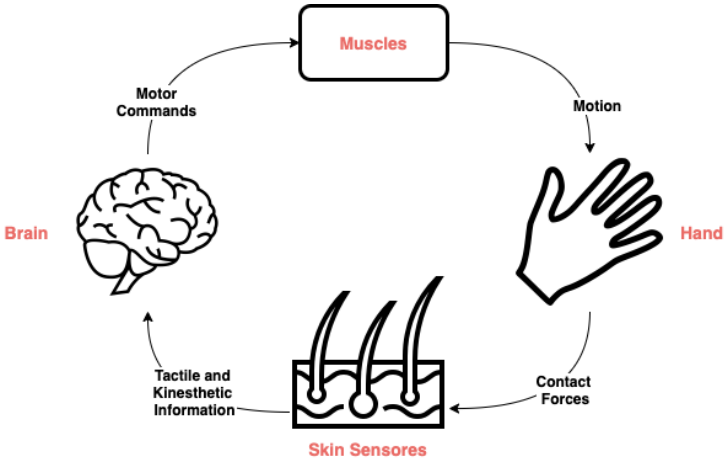


Figure 2.6: Human haptic perception [31]

2.4.2 Tactile and Haptic Interfaces

Using the skin and touch for communication, goes back to the early nineteenth century with the development of braille for the visually impaired [30]. Since then, a variety of tactile displays have been developed. However, tactile and haptic devices have a well-defined distinct definition. Haptic interfaces enable a user to make contact with an object in a real or virtual environment, to feel properties such as weight or stiffness, and to manipulate the object directly [30]. Tactile interfaces provide vibrotactile, electrotactile, or static tactile only to the skin whereas haptic interfaces engage both the tactile and kinesthetic senses [30].

2.4.3 Vibrotactile Interfaces

We can find devices that are part of our daily lives in the broad scope of tactile interfaces. In areas like gaming, for example, the vibration generated by the handheld console controller. Phone vibrations are indicating an incoming call or a reminder for an approaching event. Therefore, and based on the input delivered to the skin, the tactile interfaces can be divide into three broad categories: vibration, static pressure, and lateral skin stretch [30]. However, given that the focus of the current work is on vibrotactile interfaces, we will not be detailing the remaining categories and solely focus on vibration.

Vibrotactile interfaces stimulate the skin by delivering pressure through motors or actuators that convert electrical energy into mechanical displacement on the skin [30]. According to [32], numerous features of vibrotactile stimuli can be modulated to send information over this sensory channel. The physiological perception of the stimuli can differ from individual to individual due to the sensitivity of one's skin. Therefore, several parameters were defined to produce different patterns of vibration [32]:

- Frequency: the main spectral component of the periodic stimulus;
- Intensity: the strength of stimulation;
- Timbre: the complexity of the stimulation waveform;
- Duration: the time length of the "on" time or an elementary stimulation;
- Spatial location: the single body part or the pattern of parts that are stimulated;

Adjusting the values given to each one of these parameters, we can create a wide number of different stimuli patterns giving the user different feedback for different activities.

2.4.4 Vibrotactile Patterns

A vibrotactile pattern is a sequence of vibrotactile stimuli of different durations to create a temporal pattern [30]. Figure 2.7 illustrates a vibrotactile pattern with one actuator and varying intensity levels during a time period.

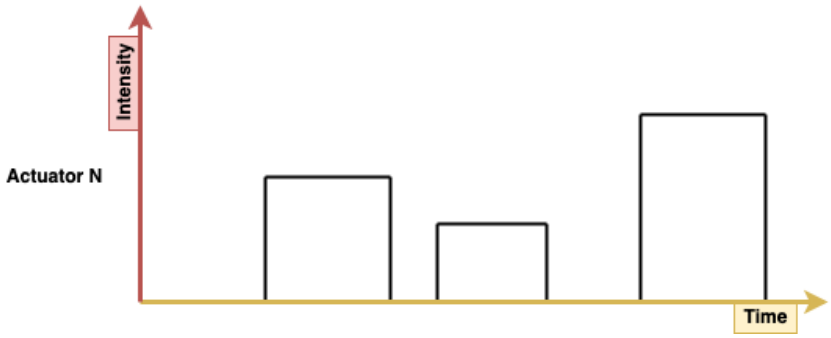


Figure 2.7: Vibrotactile Pattern

Understanding vibrotactile patterns and how to create them is a fundamental part of the present work.

A haptic system (Figure 2.8) is usually composed of three principal components [33]. The first is an input that can be from the user or a system event. Depending on the input, the processor makes the decision to send one or more vibration patterns to the actuator(s). Finally, the actuators produce the output through its vibration properties.

In this system, it is worth highlighting and specifying the type of actuators since their vibration properties can give rise to different haptic perceptions. Actuators are mechanical or electro-mechanical mechanisms that present controlled and occasionally limited movements or positioning, which are operated electrically, manually, or by various fluids such as air or hydraulic [34].

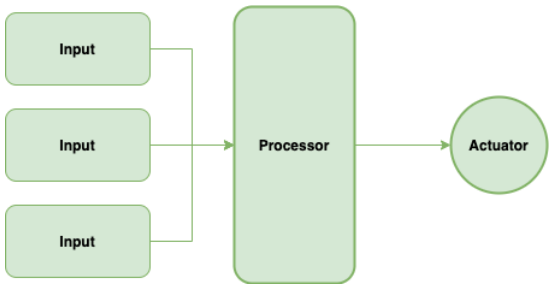
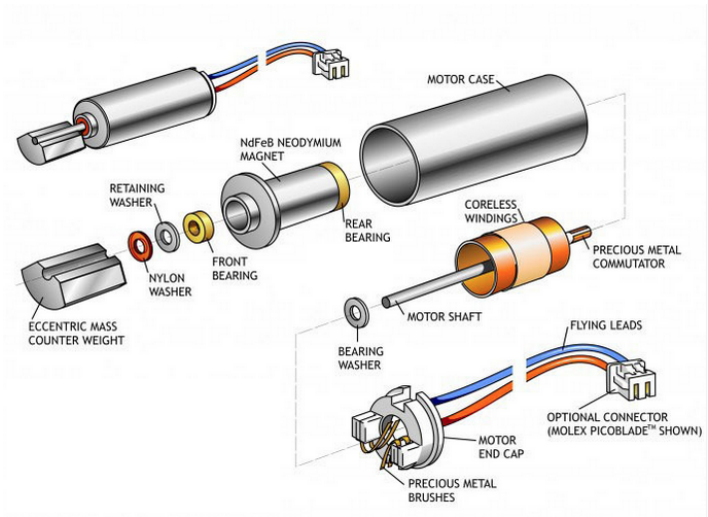
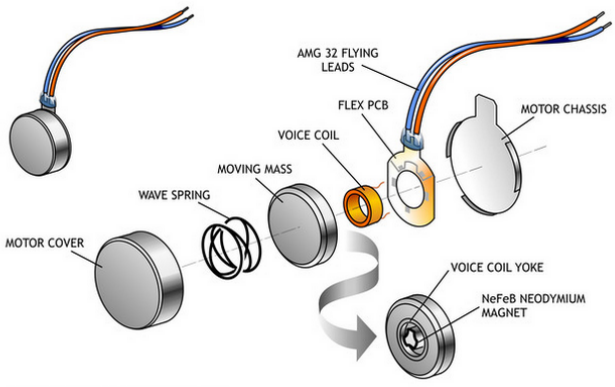


Figure 2.8: Haptic System adapted from [33]

In haptics, the most common actuators are the Eccentric Rotating Mass (ERM) and Linear Resonant Actuator (LRA). The LRA has a linear motion and the ERM a rotary motion. The linear actuators convert the energy into straight-line motions, usually applied to applications that have push and pull functions. The vibration is produced when the LRA pushes the mass up and down through a magnetic coil enhanced by a spring. Rotary actuators convert energy to provide rotary motion [35]. The ERM actuators produce vibration by rotating an unbalanced weight around the motor shaft. This rotation of the shaft causes the spinning of the irregular mass simulating vibration effects [35].



(a) ERM motor



(b) Linear resonant actuator

Figure 2.9: Vibration motors. Retrieved from `https://www.precisionmicrodrives.com/vibration-motors`

There are also have the piezoelectric actuators, although their application is comparatively smaller than ERM or LRA. These actuators generate motion through the piezoelectric effect, a property that causes the material to squeeze or stretch when an electric signal is applied [36].

Creating Vibrotactile Patterns

We can represent vibrotactile patterns in different waveforms. Figure 2.7 shows a representation of a vibrotactile pattern as square wave. The challenge of creating an editor for vibrating patterns involves knowing how to re-create an analog waveform by digital means. To give a better understanding, we pretend to provide the user with the ability to create any waveform. Analog signals are continuous in both time and amplitude, giving the user higher freedom of manipulation to achieve the representation of the waveform they desire. We can obtain these signals through the Pulse Width Modulation (PWM) technique.

PWM is a technique used by microcontrollers to re-create an analog signal by digital means [37]. In most control applications, analog signals range continuously over a voltage range between 0 and 5V. A PWM signal consists of two key elements: the duty cycle and frequency. The duty cycle is the percentage of time the signal is on during a period of time [37]. The time the signal stays on is called the pulse width. The frequency determines how fast the PWM completes a cycle. With PWM is possible to simulate all the voltages between 0 and 5V. This is accomplished by changing the portion of the time the signal spends on versus the time that the signal spends off [37], that is, manipulating the duty cycle.

To give a concrete example, the graphic in Figure 2.10, shows four different duty cycles. The first one represents a voltage of 0V during the period of time. Next, we have a representation of a 25% duty cycle. The average voltage can be calculated by taking the maximum voltage and multiply by the duty cycle [38]. Therefore, with a 25% duty cycle would yield 0.25 x 5V = 1.25V. Selecting a duty cycle of 50% would yield 2.5V, and, finally, a 100% duty cycle would yield the maximum voltage, 5V.

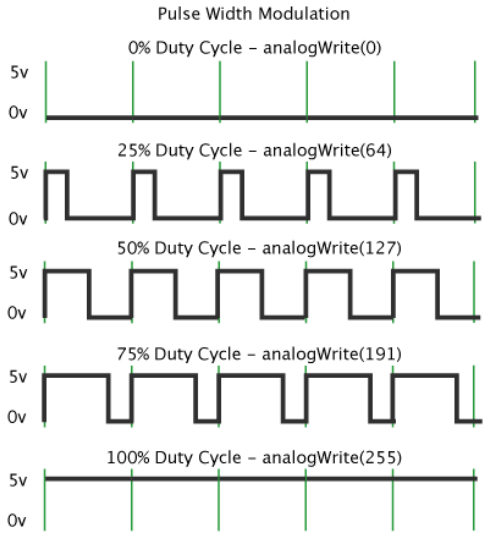


Figure 2.10: PWM Duty Cycle. Retrieved from [37]

2.5 TherTactExo

This section describes the TherTactExo's project.

The TherTactExo project aims to reduce the previous training protocol capable of achieving partial neurological recovery of sensorimotor functions in patients with spinal cord injuries [1] by adding thermal feedback and dynamic virtual reality.

With these new additions, thermal feedback will be correlated with the virtual reality scenario's alterations and tactile feedback. In this new training protocol, patients will be trained with richer sensory experiences to fulfill the time reduction goal, yet always coherent with their volition.

2.5.1 Trainning Protocol

The TherTactExo's training protocol is a refined version from the training protocol described in [1] by introducing dynamic Virtual Reality (VR) scenarios and thermal feedback that will match the users' virtual experiences. The protocol consists of three phases.

In the first phase, the patient is required to modulate his/her neural activity by imagine movements of their own legs to indicate commands, such as "start" or "stop" walking. In SCI, the patient's brain may not be able to generate these simple commands due to the loss of sensations and movement. Therefore, the goal of this first phase is to guarantee that the patient's brain can make such commands many years after the SCI. The commands generated will then be translated to control the gaiting of the avatar legs. Gaiting refers to the way of an individual natural walking. In this phase, the patient will be in an orthostatic position[2] on a stand-in-table. Thermal and tactile feedback is expressed in the patient's arm through the feedback sleeve. Visual and auditory feedback is delivered through VR. Figure 2.11 illustrates the first phase.

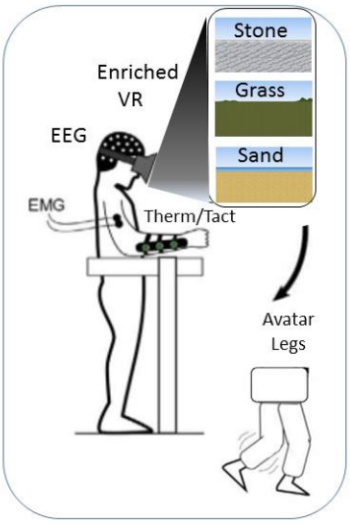


Figure 2.11: First training phase

During the second phase, the patient is using an exoskeleton in a passive gaiting mode – robotically assisted walk without neuronal control – while receiving all the feedback previously mentioned (Tactile, Visual, Auditory, and Thermal (TVAT)), as illustrated in

[2]Upright standing position

Figure 2.12. In other words, the patient is wearing the exoskeleton in passive mode, that is, the exoskeleton simulates walking. At the same time, tactile stimulation is given in the forearm following the rolling robotic feet on the ground. This phase consists of regular rehabilitation training and is aimed at improving muscle strength, bowel function, and getting the patient's cardiovascular system used to the orthostatic position.

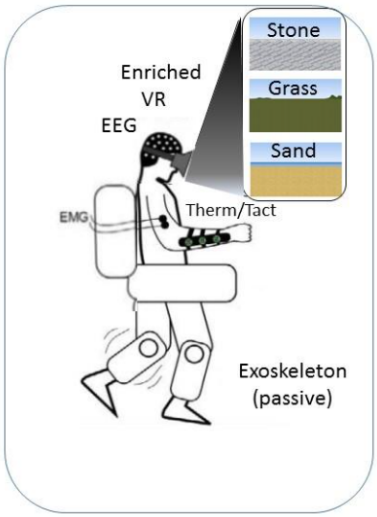


Figure 2.12: Second training phase

Finally, in the third phase, as demonstrated in Figure 2.13, the patient will actively control the exoskeleton movement using neural modulation of their EEG. At this point, the patient's lower limbs will move as a result of patients' intention (first phase training), and a continuous stream of tactile feedback information will be given to the upper arms.

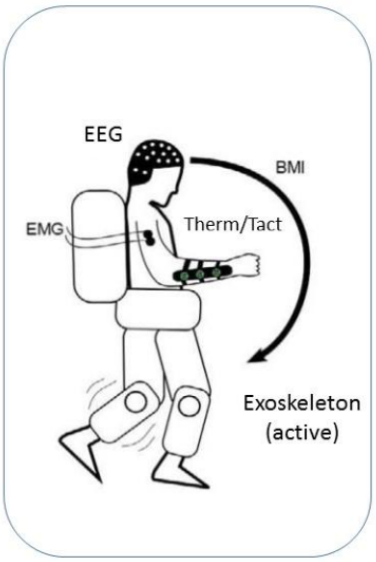
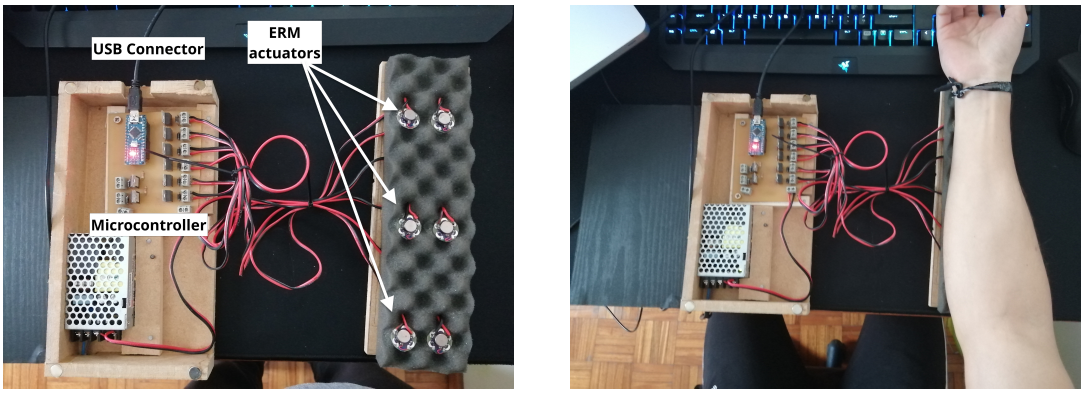


Figure 2.13: Third training phase

2.5.2 Feedback Sleeve

The feedback sleeve (Figure 2.14) is connected to a microcontroller, which is responsible for sending information to the actuators embedded in the sleeve. The microcontroller is

connected to the computer via USB and communicates through the board's serial port. Figure 2.15 shows a diagram that illustrates the interactions between the computer and the feedback sleeve.



(a) Components (b) Testing vibration

Figure 2.14: Prototype Sleeve

Currently, the feedback sleeve is composed only by actuators. The thermal component will not be used during this work nevertheless, it is described in this section to give a full perspective of the system. The prototype we are currently using for testing is illustrated in Figure 2.14

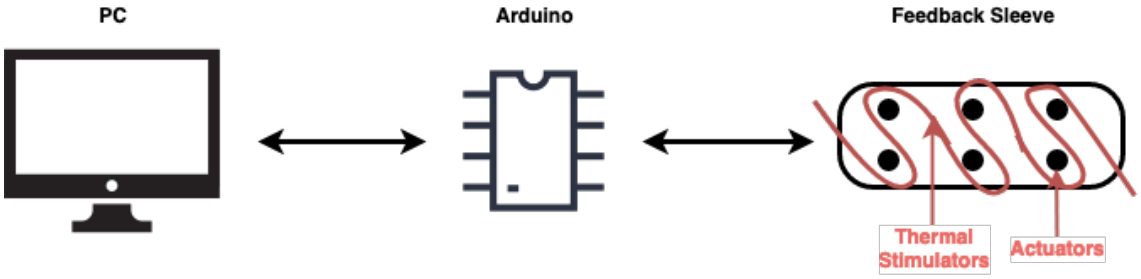


Figure 2.15: Feedback sleeve prototype

In Figure 2.14a, we can observe that the prototype is composed of six ERM actuators embed in an acoustic foam material. The prototype uses an Arduino Nano to control these ERM actuators, connected to a computer via USB. On the right side, in Figure 2.14b, it is demonstrated how to perform a test to feel the output of a vibration pattern in the user's arm. The user lays its arm over the motors and perceives its vibration pattern.

This page is intentionally left blank.

Chapter 3

State of the Art

This chapter presents the research result of the current state of the art on vibrotactile editors and an analysis of possible solutions for programming graphical user interfaces. We also describe complementary technologies related to the project.

3.1 Vibrotactile Software

3.1.1 Macaron

Macaron is a web-based effect editor for creating vibrotactile patterns. Previously vibrotactile effect libraries, available to designers, did not have an option for effect customization. Those library elements were generally opaque in construction and immutable. For this reason, Macaron was developed to allow the customization o vibrotactile effects, through the manipulation of signal parameters such as frequency, amplitude, and waveform [39]. In Figure 3.1 below, it is presented the Macaron Editor.

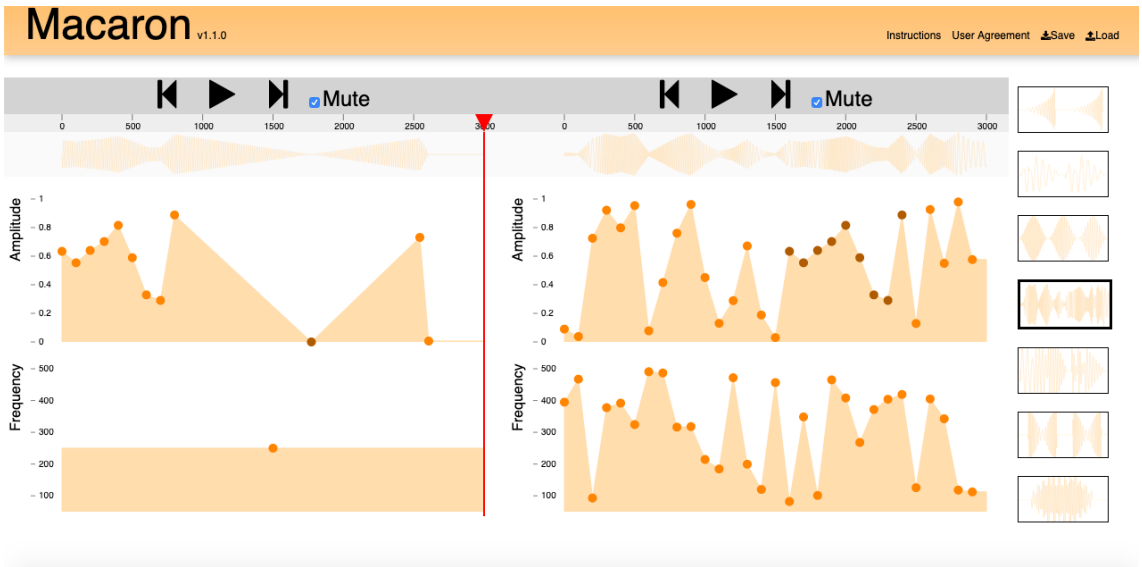


Figure 3.1: Macaron Editor retrieved from [39]

On the right side of the editor, the user can select examples from the available frames. These examples are immutable, however, it is possible to crop a selection and drag it to

the left side of the editor. On the left side, the user can manipulate freely the waveform. Macarron editor implemented familiar concepts such as tracks, envelopes, and keyframes. These concepts are standard in a broad spectrum of editors, concretely video and audio. The tracks are used to provide the user with perceptual parameters such as the amplitude and frequency. Using keyframes, the user can easily manipulate the waveform to a specific design.

Macarron editor presents a set of unique features. It gives the user freedom to manipulate the waveform. Features such as drag-and-drop, realtime playback, undo and redo, copy and paste from selected examples, a set of preset examples, are the principal assets of this software. It brings value in terms of usability and fast-reaching the main objective: generating a vibrotactile pattern.

3.1.2 VibViz

Vibviz is an interactive tool for end-user to access a large and diverse set of vibrotactile stimuli organized in different schemes: Sensory and Emotional View, Physical View, and Metaphor & Usage Example View [40]. Following, in Figure 3.2 is a snapshot of the VibViz library.

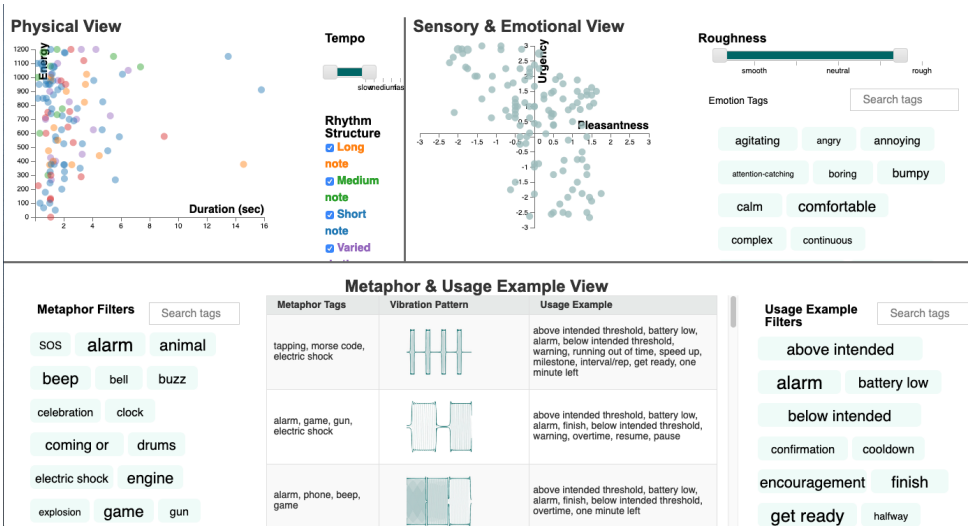


Figure 3.2: VibViz library. Snapshot from *https://www.cs.ubc.ca/~seifi/VibViz/main.html*

VibViz provides to the users a way to assist in waveform customization or manipulation by creating a vast library divided into three views. Each of these views contains a screen where the user can observe the vibration representations graphically. Also, each vibration has associated different characteristics. These characteristics were synthesized through research on tactile language and vibrotactile effects, and are the following:

- Emotional: represents the emotional interpretation of the vibration.
- Usage Examples: Types of events in which a vibration fits.
- Metaphors: Familiar examples resembling the vibration's feel.

Although VibViz does not allow the user to manipulate a waveform directly, this vast library of examples is an asset. It gives the user the possibility to replicate a vibration

pattern by presenting a visual representation. The filters and the navigation itself is very intuitive. The users can quickly narrow the search for a pattern. Additionally, the vibrotactile effects used in the VibViz library are available for use, in Waveform Audio File (WAV) format.

3.1.3 posVibEditor

The posVibEditor is a graphical authoring tool of vibratory patterns for multiple vibration motors or actuators. This editor supports the customization of the vibration patterns through graphical editing. It also supports a multi-channel vibration pattern design as well as a drag-and-drop paradigm. Figure 3.3 shows the posVibeEditor's interface.

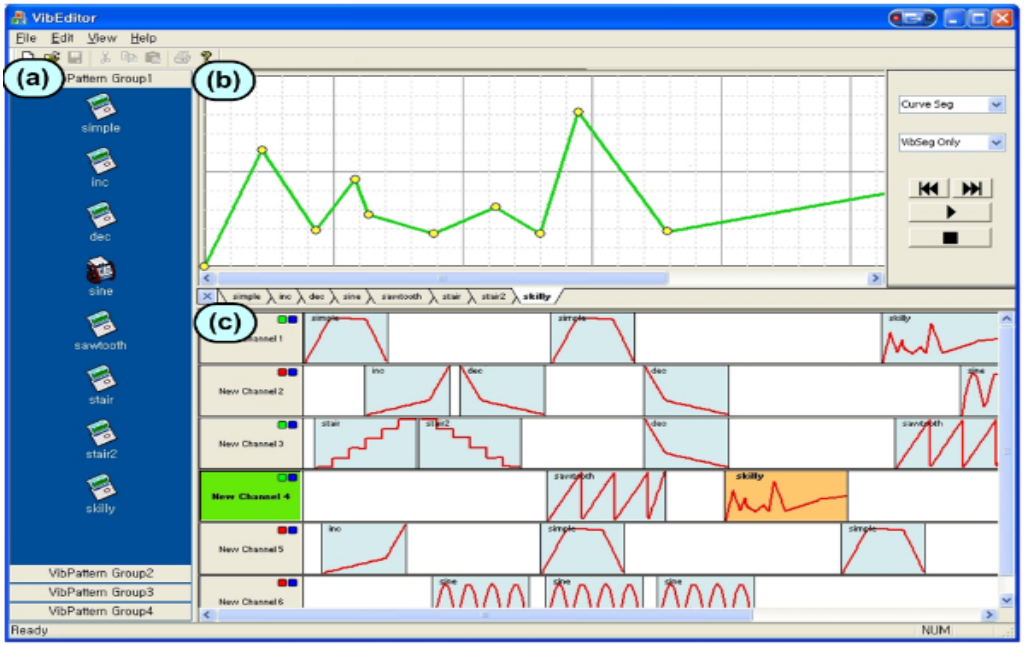


Figure 3.3: The posVibeEditor retrieved from [41]

The vibration pattern editor provides the user interface for designing vibration and playing them on the spot. The multi-channel interface allows the design and test of simultaneous vibration for multiple actuators. Finally, the panel is a workspace where the user can manage and group the formerly designed vibration patterns [41].

The posVibeEditor has unique features when comparing to the already analyzed works. It allows the user to save the vibration pattern in a database using XML formats. It permits the user to reuse previously designed patterns and saves the recently created. The multi-channel interface can assign different vibration patterns to individual actuators and can also chronologically handle its activation.

3.1.4 Hapticon Editor

One of the first interfaces for creating vibrotactile patterns was Hapticon Editor. Before the name of vibrotactile sensations became the standard, the programmed forces applied to the user through a haptic interface were also referred to as *hapticons* [42]. Below, in Figure 3.5, it is represented the graphical interface of the Hapticon Editor.

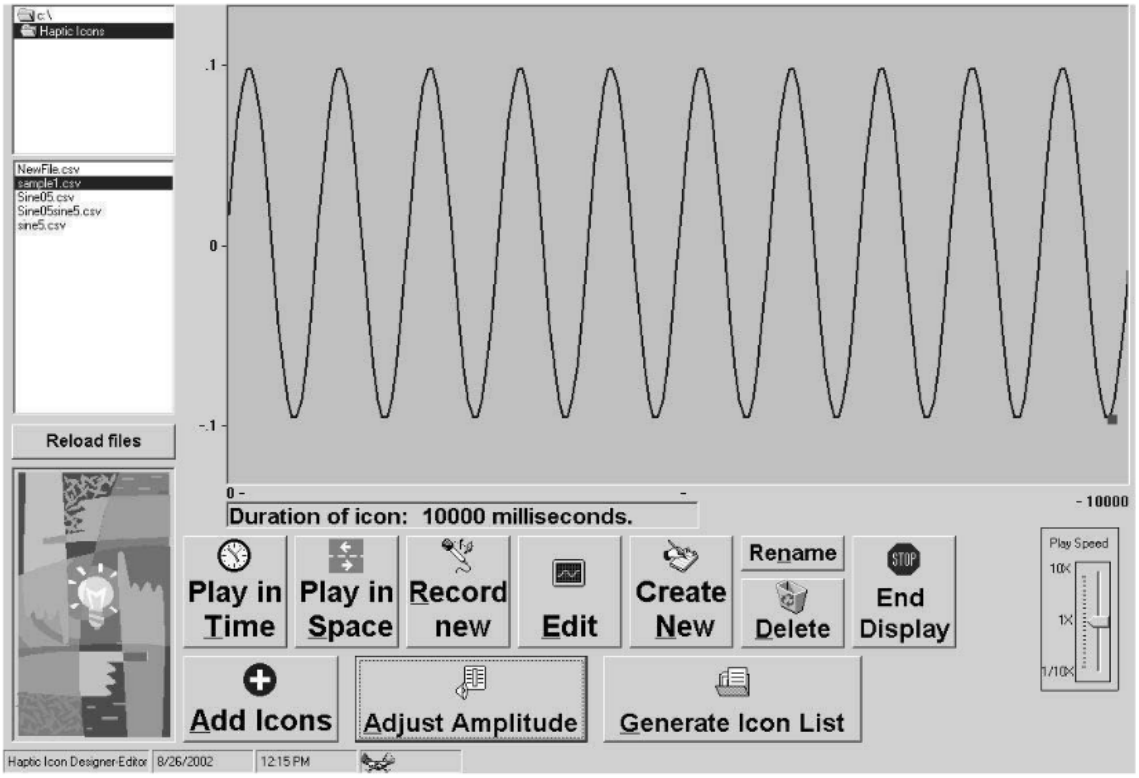


Figure 3.4: Hapticon Editor retrieved from [42]

The Hapticon Editor was developed for a haptic display with a single degree of freedom. It allows the user to create a new file for designing the waveforms. These files can be saved and are displayed on the left panel of the editor. The editor provides seven simple waveforms with varying amplitudes, frequencies, and durations. The user can customize the waveform by concatenating the types of wave functions. Also, the user can select the dots - similar to the keyframes in the Macarron Editor - and move the wave up or down.

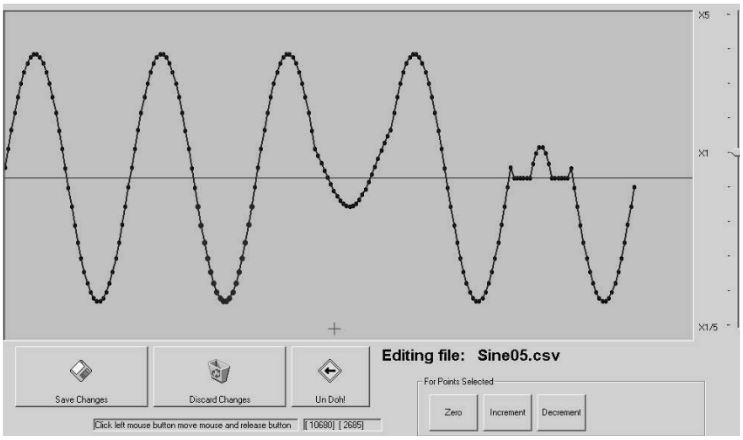


Figure 3.5: Hapticon Creator Screen retrieved from [42]

Overall, the Hapticon Editor has most of the features desired for this set of editors. It provides an effortless way to create a waveform and later saving it for reuse. However, this editor is focused on a single haptic interface. As a consequence, it lacks a multi-channel timeline. Also, the editor does not have a preset of vibration pattern examples, relevant for inexperienced users in waveform customization.

3.1.5 VITAKI

VITAKI is a vibrotactile prototyping toolkit for Eccentric Rotating Mass actuators. This platform was developed to facilitate the prototyping and testing procedures of new vibrotactile interaction techniques for Virtual Reality and video games [43].

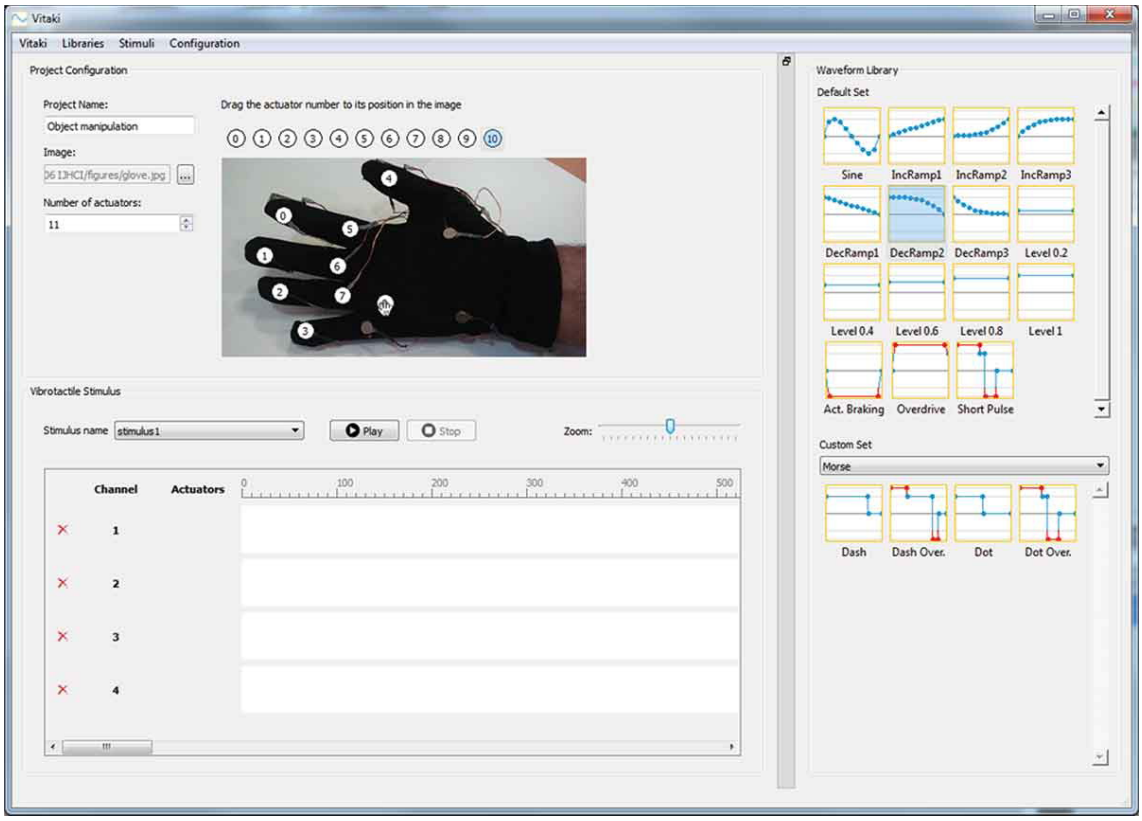


Figure 3.6: VITAKI Editor [43]

The VITAKI platform allows the user to adjust the controller's parameters and create complex vibrotactile stimuli over multiple actuators. The user can also choose a picture that represents the device used. After that, the user can set the number of the actuators and rearrange them according to the user's preferred spatial position. In the right panel is present a default set of waveforms where the user can drag and drop to the desired channel or actuator. These waveforms can be edited by the user through the waveform editor (Figure 3.7), saving them, creating a custom set for future use.

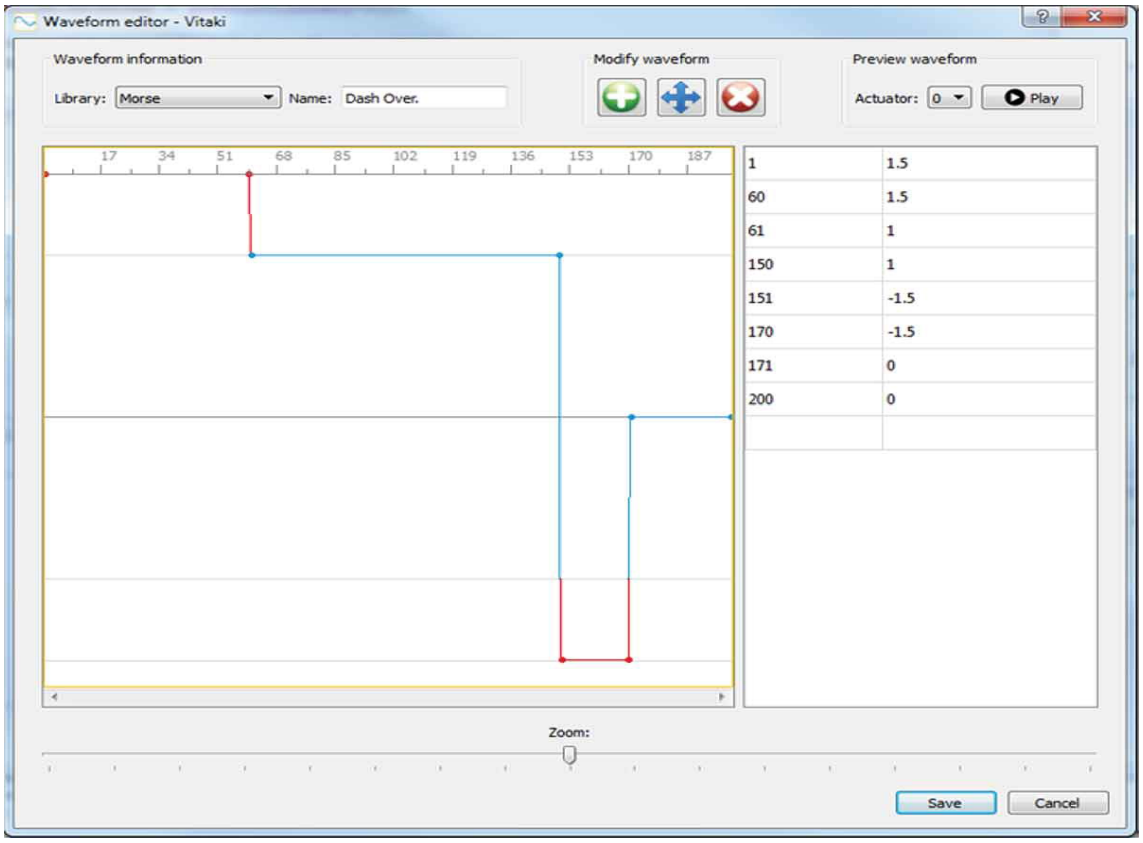


Figure 3.7: Waveform Editor [43]

Finally, VITAKI also was a configuration dialog (Figure 3.8) that allows communication with a particular hardware device. Here, it is possible to identify the number of actuators available and change the configuration according to the present parameters.

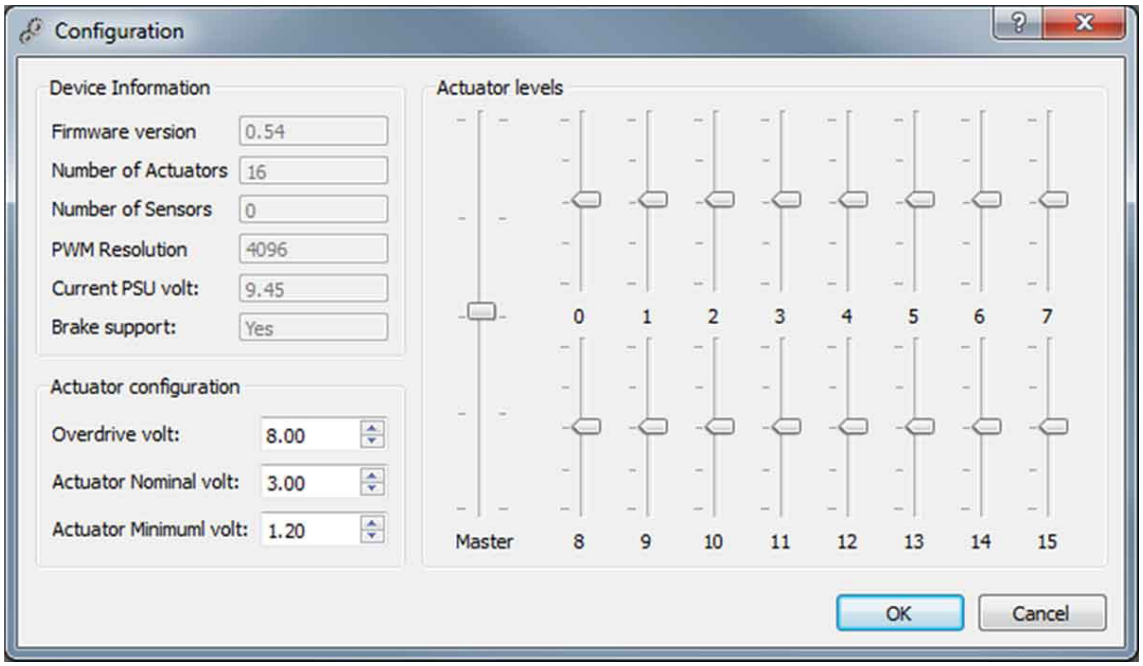


Figure 3.8: Configuration Dialog in the VITAKI platform [43]

3.1.6 Comparative Analysis

All of the reviewed editors provided some unique features essential to include when creating vibrotactile patterns. Next, we identified the characteristics essential to have in a vibration pattern editor and performed a comparative analysis between the studied editors, detailed in Table 3.1.

Feature	Macaron	posVibeEditor	VibViz	Hapticon Editor	VITAKI	Vibrotactile Editor
Built-in waveform editor	Yes	Yes	No	Yes	Yes	Yes
Free waveform design manipulation	Yes	No	No	No	No	Yes
Visual representation of the device	No	No	No	No	Yes	Yes
Used actuator	-	Vibration motor	-	Force knob	Vibration motor	Vibration motor
Actuators supported	-	ERM	-	DC motor	ERM/LRA	ERM/LRA
Internal player	Yes	Yes	Yes	Yes	Yes	Yes
Multichannel timeline	No	Yes	No	No	Yes	Yes
Multiple actuators	No	Yes	No	No	Yes	Yes
Sample library	Yes	Yes	Yes	No	Yes	Yes
Open-source	Yes	No	Yes	No	No	Yes
Software available	Yes	No	No	No	No	Yes
High level VR components	No	No	No	No	No	Yes
API	No	No	No	No	Yes	Yes

Table 3.1: Comparative analysis between the studied editors

The first feature that emerges and is familiar to all editors is the built-in waveform editor. This is an essential feature to have as it allows drawing the vibrotactile patterns.

Also, in the built-in waveform editor, it is essential to give the user the freedom to design vibrotactile patterns as needed without concatenating various functions representing waveforms, as is the case in most of the editors.

Presenting the user with a visual representation of the device they are working on helps them envision possible configurations and how many possible ways they can express the vibrotactile feedback. Besides VITAKI, this feature is not included in any of the remaining editors.

The following two characteristics concern the type of actuators that are used and supported by the editors. An editor that has more flexibility in the kind of actuators supported becomes more appealing to future users.

In addition to creating the vibrotactile patterns, the editor should also be able to reproduce the vibration directly on the feedback device. This is a must-have feature as we can see it is available in all the editors.

A timeline with multiple channels allows the definition of numerous parameters, including the start time for each pattern, which actuators and vibration patterns are associated with and contained within the channel. Besides these features, it allows the user to create much

more complex vibrotactile patterns.

The editor must also be able to support multiple actuators. When creating more complex projects involving more than an actuator, it is imperative to the editor to be flexible to handle a finite number of actuators.

Almost all the editors have a library of examples. VibViz stands out in this regard, presenting a large set of samples. The examples facilitate engagement, discover different vibration pattern types, and help construct new ones.

Aside from Macaron, most editors are not open source and do not make the software available. Making the software available helps us understand its behavior and identify aspects that can be improved.

Finally, none of the editors provides high level VR components and, except for VITAKI, no other presents an API for communication with external applications.

In conclusion, in this dissertation, we aim to develop an editor capable of meeting all the enumerated features in Table 3.1. A common characteristic among the reviewed editors leans on working with single degrees of freedom actuators, meaning that the waveform manipulation only reflects on the frequency vibration of a specific actuator. The objective here is to develop an editor capable of customizing both the intensity and the frequency of the actuator's vibration. Also, to have an extensive library of vibration patterns categorized according to several parameters as in the VibViz example. One of the possible solutions is to integrate the available files provided by the VibViz researchers and accessible for free use to accomplish these features. In the next section, we will be evaluating the technologies available for developing the editor, starting by reviewing graphical user interfaces.

3.2 Graphical User Interfaces

A Graphical User Interface (GUI) is a type of user interface that enables the user to interact with electronic devices via graphical controls or widgets [44]. In this section, we will be listing and analyzing different types of GUI toolkits. To develop the graphical editor for programming the vibrotactile sensations, a review of the features provided by all the solutions available is essential for the definitive selection of the GUI framework.

Most of the GUIs have a preset of standard features and are coupled with a particular programming language. The most important type of controls are listed below:

1. Buttons, radio buttons, and check-boxes
2. Canvas
3. Text entries, labels, and messages
4. Menus, spin-boxes, and scrollbars

Of these list items, canvas, in particular, it is an asset for the user to create its pattern through freely drawn geometric shapes. Also, we pretend that the GUI is available for a variety of different systems. This is solvable by opting for cross-platform GUI frameworks or opting for web-based user interface frameworks. Therefore, this section is divided into two main categories: Cross-platform and web-based GUI framework solutions.

3.2.1 Cross-platform Interfaces

A cross-platform system or applications are systems or applications that can run in multiple types of platforms or operating environments [45]. Here, we present a list of solutions to existing cross-platform GUIs.

TkInter

TkInter is a GUI programming toolkit for Python language. It is the standard GUI toolkit, and it is available in the standard Python package. This library offers all the types of controls stated at the beginning of this section. Also, since it is coupled with the Python language, it can be used alongside with *matplotlib* library to create automatic charts [46]. Figure 3.9 illustrates a GUI example using the *matplotlib* library.

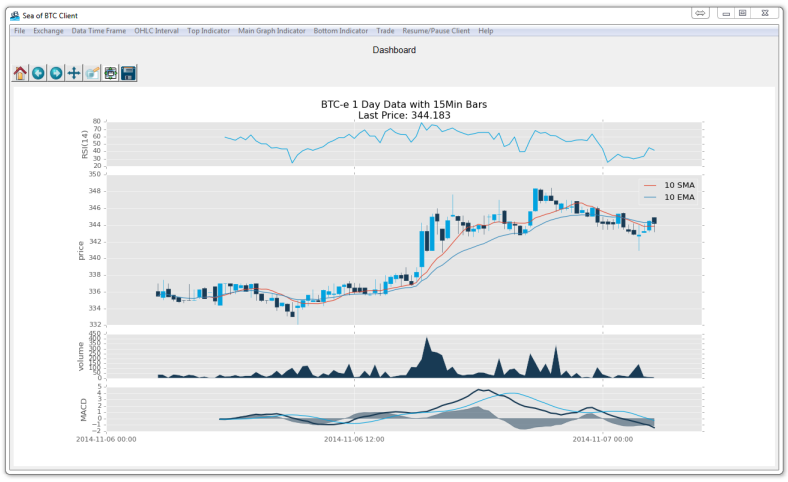


Figure 3.9: TkInter GUI example [47]

Kivy

Also associated with Python language, Kivy is an open-source library for rapid development of applications that make use of innovative user interfaces [48]. Kivy brings a more modern look to its components when compared to Tkinter. Like Tkinter, it can also use all the python libraries and use *matplotlib* to integrate charts into the user interface. However, since Kivy is not included with Python, it requires a new package installation. One of the significant advantages of Kivy's are developing for touch interfaces as it can natively use most inputs, protocols and devices [48].

Swing

Java Swing is a GUI programming toolkit for Java. Swing was developed to give a more refined set of GUI components than the previous GUI programming toolkit, the Abstract Window Toolkit [49]. Coupled with the Java GUI programming, it offers all the main GUI features listed previously. Figure 3.10 illustrates a GUI example developed in Java Swing with the principal features highlighted.

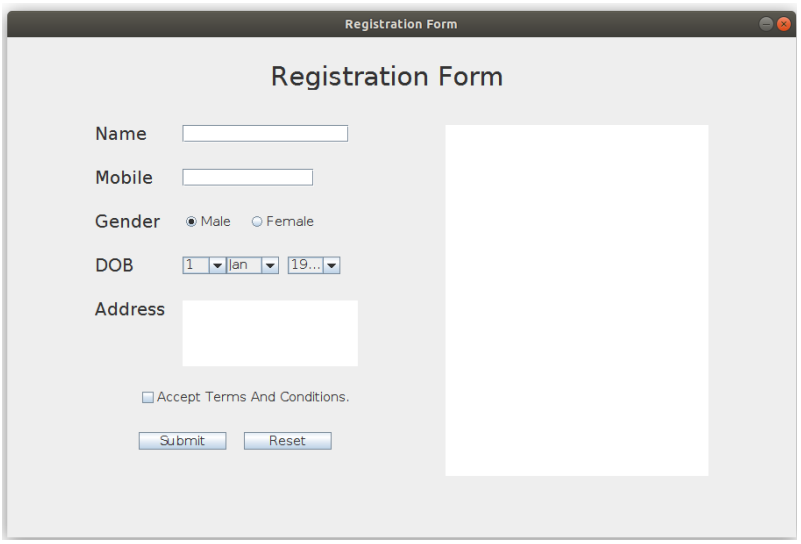


Figure 3.10: Java Swing GUI example [50]

JavaFx

Also written in the Java language, JavaFX is a set of graphics and media packages that enables developers to design, create, test, debug, and deploy rich client applications that operate consistently across diverse platforms [51]. The first version was released in 2008 and first developed by Sun Microsystems until being acquired by the Oracle Corporation. JavaFX has UI components based on Swing since its initial release arrived much earlier than JavaFX. However, this is not a shortcoming. JavaFX provides rich GUI components with an advanced look and feel and has upcoming new rich UI components. Figure 3.11 illustrates an example of a GUI designed in JavaFX.

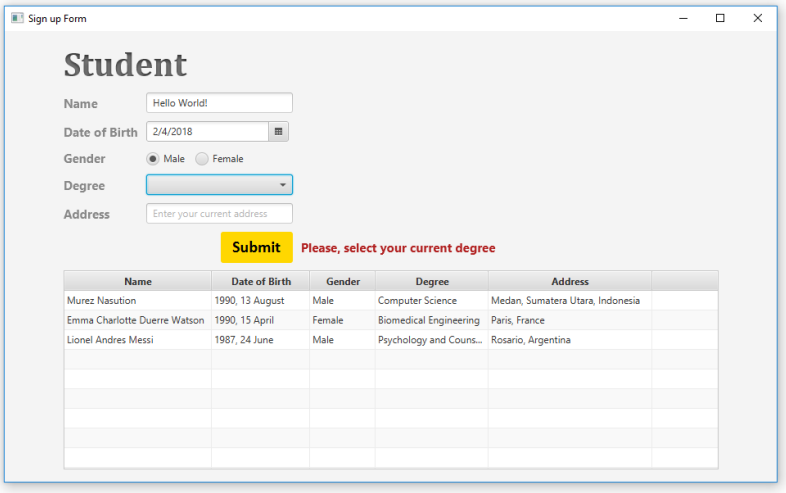


Figure 3.11: JavaFX GUI example [52]

ElectronJs

ElectronJS is an open-source framework created by Cheng Zhao. Formerly known as Atom Shell, ElectronJs is now being developed by GitHub. The first version was initially released in May 2016. It enables the development of desktop GUI applications using components

originally conceived to designing web applications. Nowadays, there are a significant number of applications developed with Electron, including Atom, GitHub Desktop, Slack, Discord, WhatsApp Desktop, and Visual Studio Code. Currently, the stable version is 8.1.1 [53]. For example, in Figure 3.12 is presented an image of the GitHub Desktop application developed with Electron.

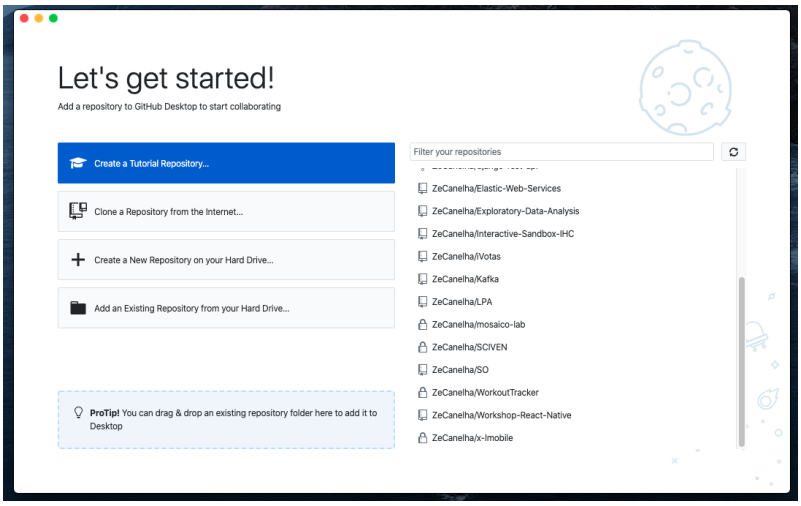


Figure 3.12: Screenshot of GitHub Desktop developed with Electron

3.2.2 Web-based User Interfaces

A web-based GUI refers to the interaction between the user and software running on a web server, where the browser acts as the user interface [44]. The advantage of using web user interfaces is that they are platform-independent and require no installation. Also, it requires no manual updates. With the growth of web development, various open-source frameworks for creating user interfaces are available for integration. When analyzing platforms that enable UIs development, we look at two distinct groups:

- Frameworks that allow us to organize the User Interface (UI) with pre-defined UI elements (Cascading Style Sheets (CSS) Frameworks).
- Frameworks that allow us to couple between frameworks from the previous point with application logic (Development Frameworks).

We analyzed these groups separately, starting with the frameworks for UI organization.

Due to the vast diversity of existing frameworks for building user interfaces, the chosen criteria was based on popularity, date of the initial release, and last stable version date. The popularity metric was measured recurring to GitHub repository stars.

Bootstrap

Bootstrap is currently the most popular front-end framework to design graphical user interfaces. Created by Mark Otto and Jacob Thornton, Bootstrap is an open-source framework for the development of interface components using HyperText Markup Language (HTML), CSS, and JavaScript. It is a well-documented framework and provides the user with several free to use templates. From version 3.0, Bootstrap follows the mobile-first design paradigm,

prioritizing the responsive design by default. Currently, the stable version is 4.1, and it is compatible with most of the modern web browsers [54]. Using the resources given in the official documentation, Figure 3.13 illustrates an example of a GUI dashboard template, showcasing nearly all the features we intend for our editor.

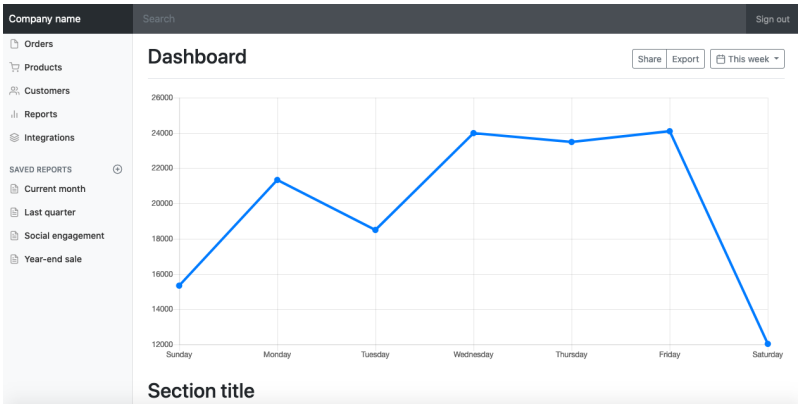
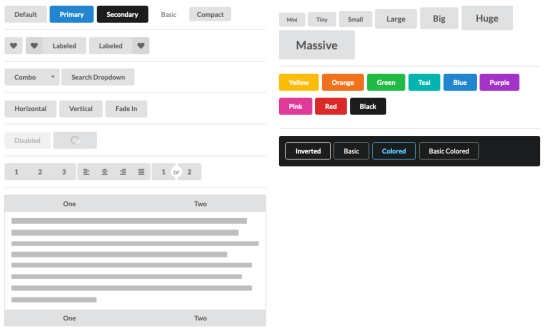


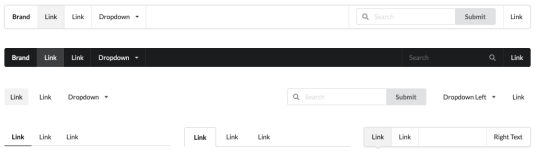
Figure 3.13: Dashboard Template for Bootstrap [54]

Semantic UI

Semantic UI is a modern front-end development framework, powered by LESS and jQuery. It is designed for theming and enables the developer to create websites having more than fifty user interface elements, numerous CSS variables, and based on the flexbox model. On top of that, the elements are built with EM values for a responsive design. This allows the creating of a multiple device web platform. Finally, Semantic is integrated with popular frameworks such as React, Angular, Meteor, and Ember. Currently, it is being used by diverse applications, including Snapchat, Kmong, and Accenture [55].



(a) Semantic UI elements



(b) Semantic UI elements

Figure 3.14: Example of UI elements available in SemanticUI [55]

Bulma

Inspired by Bootstrap, Bulma is a new open-source CSS framework based on the flexbox model. As no JavaScript is required, it easily integrates with any JavaScript environment such as Vanilla, Angular, React, and Vue. Bulma can be integrated into any project with a single CSS file [56].

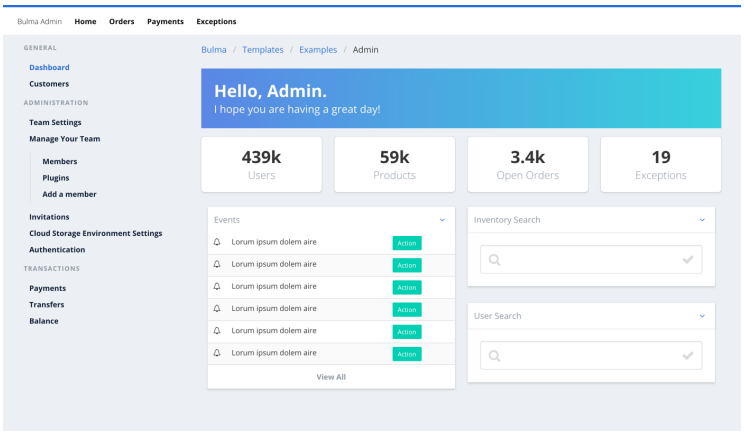


Figure 3.15: Example of UI designed with Bulma [56]

Foundation by Zurb

Foundation is a responsive front-end framework. Foundation provides a responsive grid and HTML and CSS UI components, templates, and code snippets, including typography, forms, buttons, navigation, and other interface elements, as well as optional functionality provided by JavaScript extensions [57]. In Figure 3.16 are illustrated some examples of Foundation's UI components.

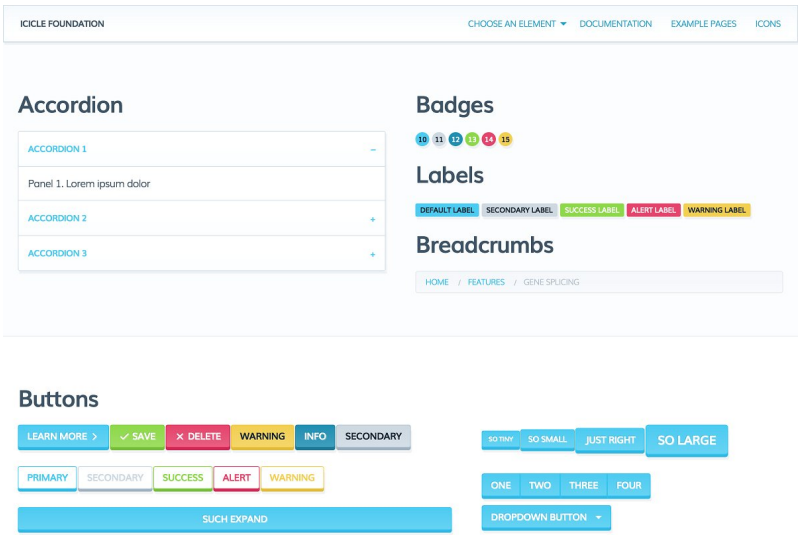


Figure 3.16: Example of UI elements from Foudation framework [57]

We conclude here the analysis of the frameworks of the first group. Next, we present the frameworks from the second group:

React

Launched in 2015 by Facebook, React is an open-source JavaScript framework for creating user interfaces. First created by Jordan Walk, a software engineer working for Facebook, React's first appearance was on Facebook's news feed in 2011 [58].

VueJs

Created by Evan You, an ex-engineer of Google, first released in February 2014, Vue is an open-source Javascript framework for client-side programming, focused on creating single-page applications and user interfaces [59].

Angular

Angular is a platform for building mobile and desktop web applications based on TypeScript [60]. It was created by Google, first released in September 2016.

3.2.3 Comparative Analysis

Cross-platform Technologies Comparison

In the analysis of the technologies for the development of cross-platform interfaces we have not found a linear approach to compare these technologies with each other. All of them present the characteristics that we are looking for to develop our graphical interface.The differences found are mainly the design of the components, where newer technologies provide components with a better look and feel, and the directly associated programming language. Kivy and TkInter are coupled with Python, Swing and JavaFX with Java. ElectronJs on the other hand is coupled with JavaScript but can use the CSS Frameworks listed in the last section.

Web-based Technologies Comparison

Most of the frameworks analyzed in the web-based GUIs section are trending frameworks used in many applications. There is no specific metric when it comes to making a comparative analysis. We used the same criteria for the evaluation as the criteria used for the CSS framework selection:

1. Popularity
2. Date of the initial release
3. Date of the last stable version

Popularity is essential to the extent that comes with increased support from the community. There is a higher probability of the same issue being resolved in community forums in case of doubts. The date of the initial release can be related to the date of the last stable version. It reveals framework growth or stagnation. We want to develop a user-friendly editor, consistent and appealing to the user. The look and feel of the components must be rich, must be intuitive, and overall must be familiar. To achieve this goal, newer frameworks or old frameworks with consistent growth are the most desirable solution. Figure 3.17 presents a popularity chart based on GitHub stars for each CSS Framework repository.

Next, we assemble the release and the last stable version date in a chart to visualize how each framework has evolved and if it is in continuous evolution.

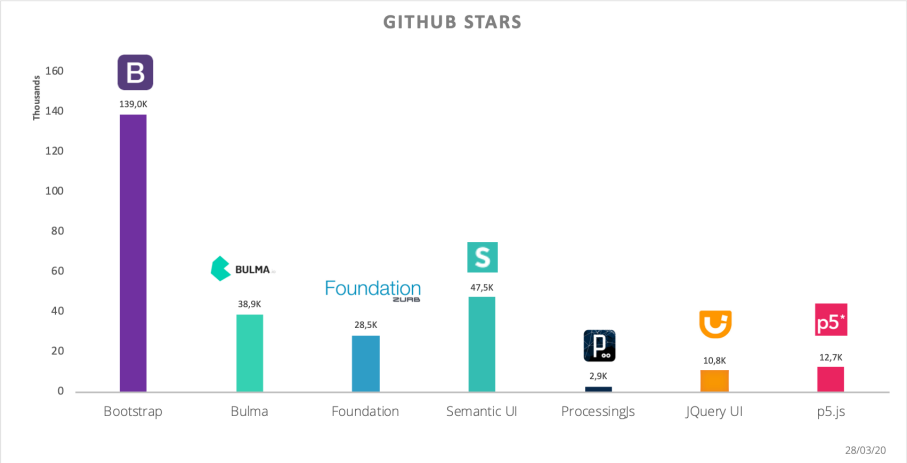


Figure 3.17: CSS frameworks popularity based on GitHub stars

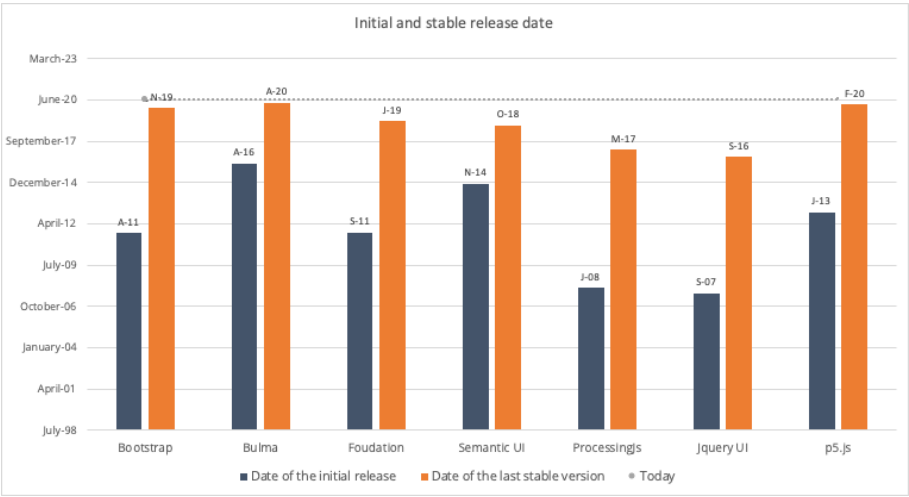


Figure 3.18: Initial release and last stable version date of each CSS framework

Similar to the CSS frameworks analysis, we adopted the same approach for the development frameworks. Figures 3.19 shows the frameworks' popularity according to the stars on GitHub, and Figure 3.20 the framework release and stable version date.

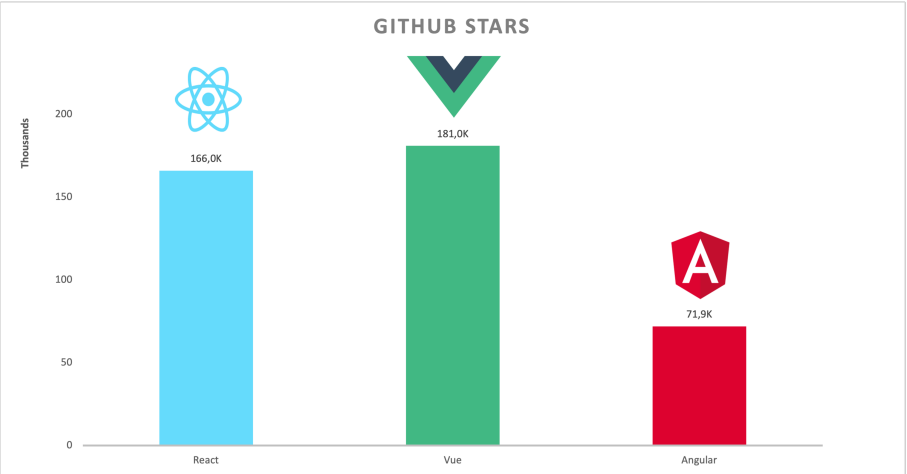


Figure 3.19: Development frameworks popularity based on GitHub stars

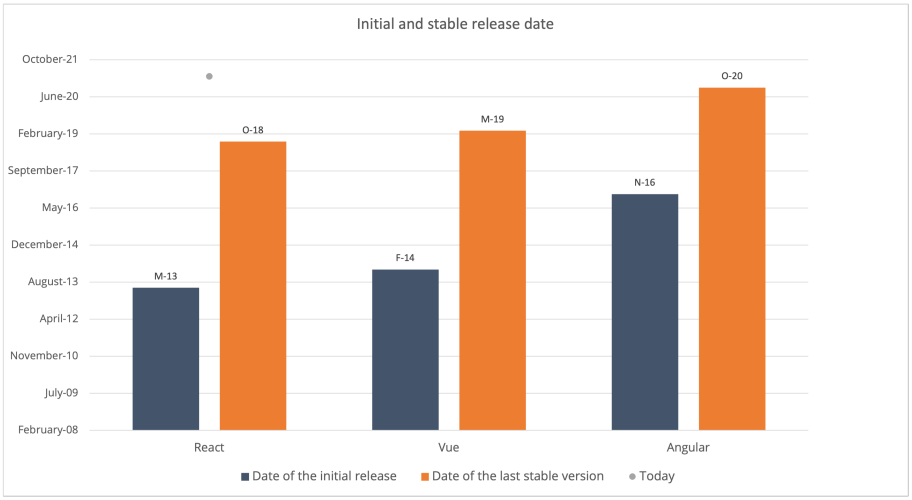


Figure 3.20: Initial release and last stable version date of each development framework framework

Final Conclusions

All the analyzed frameworks enable the development of a well-defined GUI. Within the technologies for the development of cross-platform interfaces, we highlighted ElectronJs for its flexibility and integration with CSS frameworks that provide us with more freedom to design UI elements and, consequently, the GUI.

Within the technologies for Web-based GUI development, as far as CSS frameworks are concerned, Bootstrap is the most widely used framework by developers, evidenced by the popularity of its GitHub repository. In development frameworks, React, and Vue stand out. These frameworks are among the most frequently found in web-based GUI development.

That said, the first decision we made was to select the type of GUI we wanted. The decision made was to develop a Web-based GUI. A significant reason behind this choice was the diversity of frameworks to organize the UI. We also take into consideration that the A-Frame component is developed for virtual reality experiences on the web. Once we made this decision, the next choice was the CSS framework. We decided to go with Bootstrap, given its popularity, well-structured documentation, and web design responsiveness. To develop the interface, we opted for React. Although Vue is more popular, this was heavily influenced by the author's previous experiences with the React framework.

In conclusion, the GUI will be web-based developed using React combined with Bootstrap.

3.3 Technologies

3.3.1 Arduino

Arduino is an open-source electronics platform based on easy-to-use hardware and software [61]. Arduino boards are used in a vast broad of applications. These boards are capable to read inputs and turn it into an output. For example, a user presses a button, the board detects the event and lights a specific LED light.

There are several other microcontrollers available in the market but Arduino stands out as it is inexpensive, has published open-source software, and hardware and has a clear programming environment. We will be using Arduino to be a bridge between the feedback sleeve and the Vibrotactile Editor.

3.3.2 Unity

Developed by Unity Technologies, Unity is a powerful known cross-platform game engine released in June 2005. Unity is an application that enables the user to create games and experiences [62]. Figure 3.21, shows the Unity3D interface.

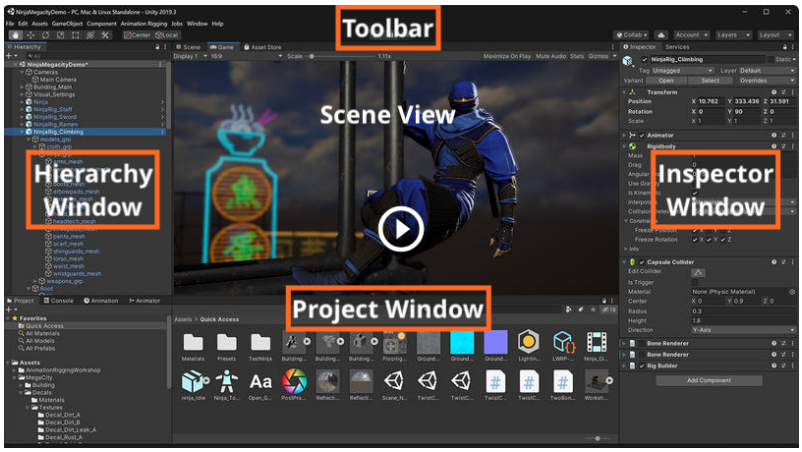


Figure 3.21: Snapshot of the Unity Interface retrieved from [63]

The project window displays the library of available assets as well as imported assets to the project. Scene view permits the developers to have visual navigation and editing capabilities for the created scene. On the right side is the Inspector Window. It enables the inspection of all the editable properties of a selected object. As the name suggests, the Hierarchy Window shows a hierarchical representation of the available objects in the scene. Finally, the Toolbar Window contains the essential tools to manipulate the scene and its objects. We will be using Unity to create components for virtual reality. The objective is to integrate the resulting vibration patterns created previously with the Vibrotactile Editor with a Unity component.

3.3.3 A-Frame

A-Frame is a recent open-source web framework to build virtual reality experiences. Released in 2015 and developed initially by the Mozilla VR team, A-Frame is a JavaScript framework that enables developers to design 3D and VR experiences with HTML. A-Frame

was developed to be an effortless way to build VR content and has grown to be one of the largest VR communities. It supports most of the VR headsets, and it is platform-independent, meaning that still woks in standard desktops and smartphones when the user lacks VR headsets [64]. Figure 3.22 showcases a virtual reality experience developed with A-Frame.

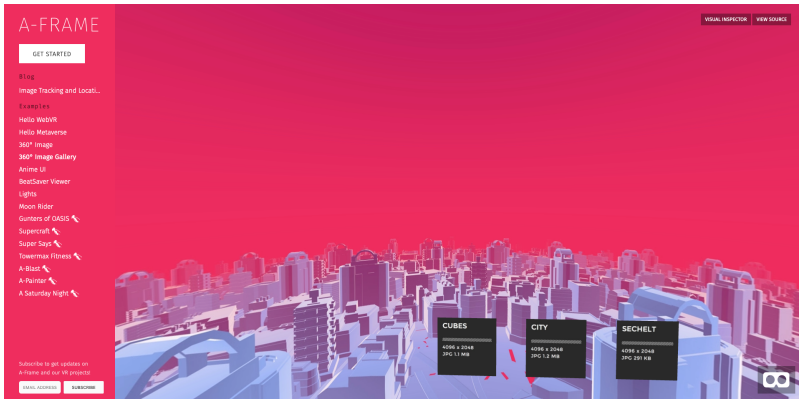


Figure 3.22: Snapshot of one A-Frame example retrieved from [64]

Here, the same objective as in Unity is applied. The A-Frame framework will be used to develop a component capable of handling a vibration pattern exported from the Vibrotactile Editor.

Chapter 4

Methodology and Work Plan

In this chapter, we explain the methodology and the work plan adopted. The first section describes the software development methodology, followed by the definition of the work plan for both semesters and, finally, the risk management.

4.1 Methodology

For this project and considering the possibility of changes in the requirements, the methodology adopted was the modified version of Royce's Waterfall Model [65]. Figure 4.1 shows a representation of this model.

In choosing an Software Development Life Cycle (SDLC), we reviewed several models before selecting the modified waterfall version.

Among the models analyzed, we start by examining the **incremental** model. In this model, product development and delivery are planned in increments. In each increment, a set of functionalities is chosen to be implemented. In theory, more features are developed than in the previous increment [66]. We discarded this model simply because it does not fit a short-term project, where the components to be implemented are relatively small.

Next, we addressed the **spiral** model, which is intended for long-running projects focusing on identifying risks early on. This model is based on successive iterations that start with the design goals and end with the client reviewing the progress so far and the future direction [66]. That said, we also eliminated this alternative since it did not fit our requirements.

Finally, we looked at **agile** models. These models focus on delivering the product in small incremental builds generated in small iterations lasting between one and three weeks. Agile methodologies are more suitable for teams that work in several organized areas: planning, requirements analysis, design, development, and testing [67].

The waterfall model presented is composed of six phases.

- **Requirements & Analysis**: To gather the requirements, the first step towards this objective was to study the state of the art. We identified the key components necessary to compete with the existing applications and features that may add value to our editor by surveying existing vibrotactile patterns. To validate the requirements specification, we wrote user stories to promote the discussion about the system's functional requirements. Through meetings with the advisor and members of the TherTactExo team, we did the analysis and validation of the system requirements.

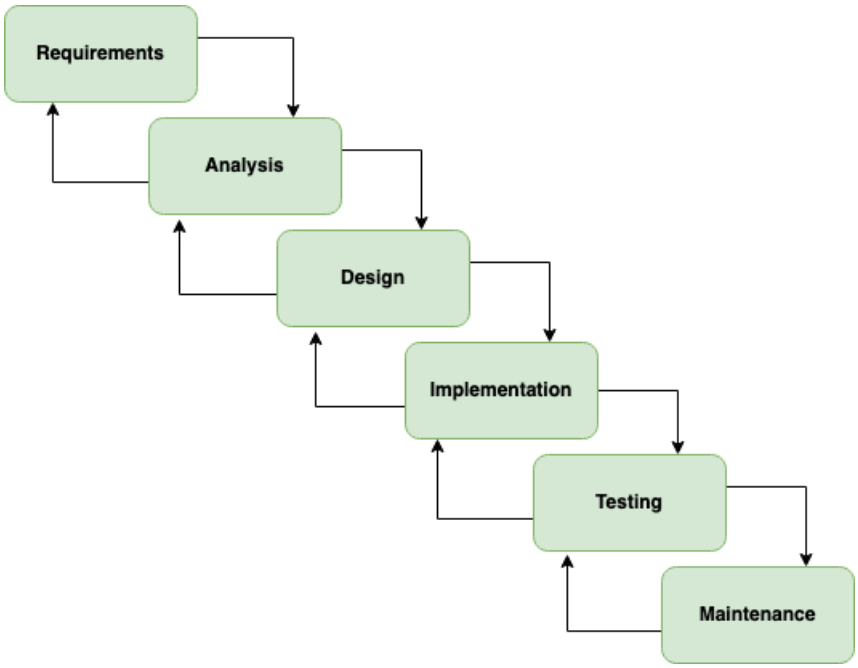


Figure 4.1: Royce's Modified Waterfall Model

- **Design**: We defined the system software architecture, data models, and visual prototypes in the design phase. After identifying the requirements, we modeled the system architecture through diagramming, such as context diagrams, container diagrams, and UML diagrams. Finally, we created a first visual prototype by designing wireframes, an excellent way of presenting both the design and user interactions with the system.

- **Implementation & Testing**: Defined the requirements and the software architecture, in the implementation phase, we developed the functional requirements, guided by the previously created visual prototype. We concluded the development of the functionalities and proceeded with the testing phase. In this phase, we validated all the work done and evaluated the usability of the editor.

- **Maintenance**: After the release of the product, maintenance occurs by delivering new versions to fix eventual issues or add new features. This phase is not applicable in this dissertation context.

Within the models analyzed, the modified waterfall model is not a perfect model for this project, but it is the one that suits it better. As with all models, it has its drawbacks, the most significant being the swift change of requirements and having to go back through the model's steps [66]. However, it is something that, to some extent, we have come to expect. Yet, although the requirements are not entirely defined, the examples we have studied provide a reasonably solid idea of the possibilities. Thus we do not anticipate the necessity for many further iterations.

One example we can give, during the project, after developing the Vibrotactile Editor, the testing phase occurred. When analyzing the results during the testing phase, we found that some of the implemented features were not perceptible to the user. Consequently, this caused us to rethink the design and go back several steps to reconsider an alternative design and development, based on the conclusions reached during the testing phase.

4.2 Work Plan

To present the work plans for both semesters, we defined two Gantt charts.

4.2.1 First Semester

During the first semester the work done was above all to study and research the dissertation subject and related works. After collecting the information, we proceeded to elicit the project's requirements and to prototype it. In the final stages, we developed the first prototype of the Vibrotactile Editor. In this first semester, there was no significant deviation between the planned and realized work. In the Gantt chart, shown in Figure 4.2, we illustrate the planned and achieved work.

Task	Start Date	End Date	Duration
Background	12/02/20	18/03/20	35
Study of haptic sensations	12/02/20	18/03/20	35
State of the art	18/03/20	03/04/20	16
Study of similar applications	18/03/20	27/03/20	9
Study of the targeted frameworks	27/03/20	03/04/20	7
Requirements Elicitation	03/04/20	14/06/20	11
Prototyping	14/04/20	20/06/20	67
Architecture definition	14/04/20	24/04/20	10
Creation of a visual prototype	24/04/20	02/05/20	8
Implementation	02/04/20	20/06/20	49
Intermediary report writing	12/02/20	25/06/20	134

Table 4.1: Work plan defined for the first semester

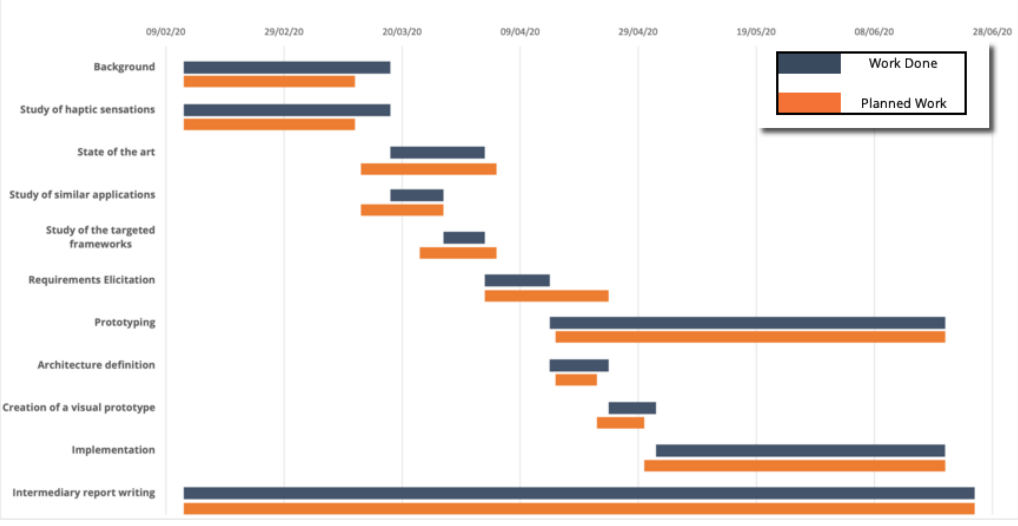


Figure 4.2: Gantt Chart for the first semester

4.2.2 Second semester

The second semester was focused on developing the Vibrotactile Editor, the components for virtual reality, and testing them. In the Gantt chart, shown in Figure 4.3, we illustrate the planned and achieved work.

Task	Start Date	End Date	Duration
Vibrotactile Editor Implementation	10/07/20	30/09/20	82
Development of the Vibrotactile Editor	10/07/20	31/08/20	52
Development of the Vibrotactile Middleware	10/09/20	30/09/20	20
Vibrotactile Editor Usability Testting	30/09/20	15/10/20	15
Development of VR components	04/10/20	16/12/20	73
Development of the A-Frame component	06/10/20	27/10/20	21
Development of the Unity component	27/10/20	18/11/20	22
VR Components Usability Testing	20/11/20	10/12/20	20
Vibrotactile Editor Implementation - 2nd Iteration	-	-	-
Vibrotactile Editor Usabiltity - Testing 2nd Round	-	-	-
Final report writting	01/08/20	20/01/21	172

Table 4.2: Work planned for the second semester

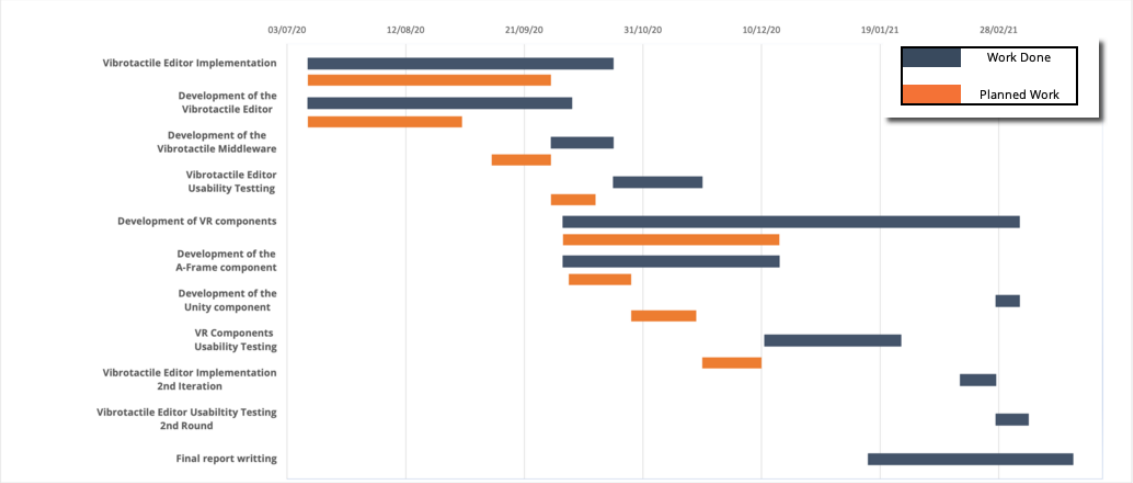


Figure 4.3: Gantt Chart for the second semester

We noticed several delays to what was initially planned during the second semester, beginning with the editor's development conclusion. Also, we changed the order of specific tasks. After developing the A-Frame component, we decided to start with its usability test and, later, to develop the component for Unity.

One of the most significant setbacks during this second semester was the usability testing, which took longer than planned for two reasons. The first reason was the pandemic situation caused by COVID-19, which made it necessary to change the workplace, requiring us to work remotely, bringing some communication and productivity difficulties. Furthermore, it did not allow the tests to be conducted as planned. The second reason was due to the author's inexperience in estimating the duration of the usability tests.

Thus, by the end of November, we decided to postpone the final delivery of the dissertation, and, accordingly, we adapted the work plan in such a way as to accomplish the objectives proposed in this dissertation.

4.3 Risk Management

In software engineering, risk management involves all aspects of the program phases as they relate to each other, from the idea to the conclusion [68]. Risk identification and risk assessment should be done as early as possible in order to minimize possible negative impacts on the project.

Risk management processes involve five essential steps [69]:

1. Identifying the risks: Understand if any factors can negatively affect the project.

2. Assessing the risks: List the risks in order to certain attributes. These attributes are commonly the impact of occurrence and the degree of risk certainty.

3. Planning the response: Modify the project plan to adjust for the risk or create a mitigation plan if the risk can be eliminated or mitigated.

4. Monitoring the risks: Continue to re-evaluate the risks and update the risk profile according to their evolution during the project development.

5. Document lessons learned: Learn from the risk management process.

4.3.1 Risk Description

Following, we describe the project's risks.

ID	R1
Title	Estimations
Description	Lack of planning experience can lead to bad effort estimations
Impact	High
Probability	Medium
Mitigation Plan	Help of the advisor to validate the estimations

ID	R2
Title	Thesis' topic
Description	Lack of knowledge about the thesis' topic
Impact	Medium
Probability	Medium
Mitigation Plan	In-depth study on the topic and its technological application

ID	R3
Title	Microcontrollers
Description	Lack of knowledge on how microcontrollers work and how they communicate with other devices
Impact	High
Probability	Low
Mitigation Plan	Study and explore microcontrollers

ID	R4
Title	Editor design
Description	The presented design does not correspond to the user's expectations.
Impact	High
Probability	Low
Mitigation Plan	Study in-detail the available designs. Get periodic feedback from the team that will use the editor.

ID	R5
Title	Pandemic situation
Description	The worldwide pandemic COVID-19 brought restrictive measures that may affect productivity and work completion.
Impact	High
Probability	Medium
Mitigation Plan	Report delays and problems encountered. Analyze alternative solutions that deliver similar results.

Chapter 5

Requirements

Before starting to develop a software project, one of the most critical, knowledge-intensive, recognized software development activities is the requirements elicitation [70]. It is crucial to understand what the system is intended to do. In most cases, poor elicitation leads to project failure. Thus, this process must be well defined, employing the best requirements elicitation techniques that can be applied to the project.

In this chapter, we outline the requirements for the components we will design. In section 5.1, we present the detailed process of requirements elicitation for the Vibrotactile Editor. In section 5.2, we describe the requirements elicitation process for the Virtual Reality components.

5.1 Requirements for Vibrotactile Editor

Regarding the Vibrotactile Editor requirements, first, we explain the problem definition and the system objective. Following, we describe the requirements elicitation process as well as the resulting requirements.

5.1.1 Problem Definition and System Objective

By researching the state of the art in vibrotactile editors, we identify that few editors fully accomplish what we pretend in the field of haptics: creating vibrotactile patterns and testing the effect of haptic feedback delivered to the users with ease and in a suitable way. For this reason, we seek to develop a tool capable of creating vibrotactile feedback efficiently. Therefore, we assemble a set of characteristics necessary to achieve this purpose. The editor will have three main features:

- **A dedicated area for customization**: As explained in section 2.4.4, we can represent vibrotactile patterns in different waveforms. Therefore, we need to give the user liberty to create and modify these vibration patterns by providing means to manipulate typical signal parameters beginning with time, intensity/amplitude, and waveforms.
- **Timeline**: We need a proper way to represent the instants of time when the actuators are activated, the associated vibration pattern, and a way to support the design and test of simultaneously controlled vibration patterns for various vibration actuators.

The solution found to have these features is to have a multi-channel timeline. Each channel can contain one or more vibrotactile patterns and be associated with different actuators.

- **Library**: We need to provide the user with an environment rich in examples to facilitate engagement, vibrotactile patterns discovery, and customization. Also, give the user a feature where he can save custom patterns, provide a brief description, and an example of its usage.

With this background, we then advanced to the requirements elicitation. Through an iterative process involving techniques such as the creation of user stories – to promote discussion about the characteristics of the system- and prototyping – design of wireframes to help in the visualization of system features- we gathered a set of required functionalities for the system.

5.1.2 User Stories

To gather the requirements, we opted to use user stories. User stories are short, simple description of a feature, written from the perspective of the end-user, used to gather the system functionalities quickly, and without the need to create a formal requirements document. The user stories usually have the following template:

As a <type of user>, I want <some goal> so that <some reason>

After we defined the first version of the user stories, these served as an initial discussion about the editor's functionalities. The initial discussion was sole with the advisor, where we reviewed the user stories and made the necessary changes. After this first iteration, we took this result to a meeting with one of the TherTactExo researchers, which resulted in the final version of the user stories, defined in Table 5.1.

ID	As a	I want to	So that
1	User	Be able to upload a image of the device	I can create different spatial configurations
2	User	Be able to assign the number of actuators	I can create specific patterns
3	User	Be able to assing a microcontroller to the project	I can know the thresholds of the patterns
4	User	Create different types of waveforms	I can create different patterns
5	User	Create waveforms in both frequency and intensity	I can create different sensations
6	User	Be able to trigger an actuator according to a timeline	I can create different sensations
7	User	Be able to create patterns for each acuator individually	I can create different sensations
8	User	Be able to save a pattern	I can reuse it in future designs
9	User	Have preset examples of patterns	I can create waveforms easily
10	User	Be able to save the project configuration	I can continue to work later

11	User	Be able to export a pattern	It can be used in other applications
12	User	Be able to preview the animation of the pattern	I can check the correct activation of the actuators
13	User	Have default project configurations	I can focus solely on creating patterns

Table 5.1: User Stories

5.1.3 Functional Requirements

With the user stories defined, the transition to functional requirements was made with ease. This approach allows the combination of user stories in specific tasks, which are a better guideline for the development process. Furthermore, to consider the priority of the requirements, we also used the moSCoW method for prioritization [71], which consists of the distribution of the requirements within the following categories:

- Must Have: Requirements which the project guarantees to deliver.
- Should Have: Requirements that are important, but not obligatory for the success of the project
- Could Have: Requirements that are desirable but not required for the project
- Won't Have: Requirements which the project team has agreed will not be included in this version of the project.

Similar to the user story process, the functional requirements were discussed and validated with the advisor. The final requirements are described in Table 5.2 and are composed of an ID, following by the description, and finally its priority.

ID	Description	Priority
FR1	The user can edit and save project configurations such as the device image, number of actuators, and the hardware controller at any time.	Must Have
FR2	The user can freely manipulate the waveform and adjust the different parameters	Must Have
FR3	The user can assign different patterns to multiple channels in the timeline	Must Have
FR4	The user can assign one or more actuators to a channel in the timeline	Must Have
FR5	The user can reproduce any channel in the timeline to preview the activation of the actuators	Should Have
FR6	The user can add or remove channels to and from the timeline	Must Have

FR7	The user can save a custom pattern in the editor library	Must Have
FR8	The user can import any pattern from the editor library to the waveform editor	Must Have
FR9	The user can search for project configurations and import to the current project	Should Have
FR10	The user can create patterns in the editor and export them to Unity or A-Frame format	Must Have
FR11	The user can place the actuators on the device image in order to visualize and create different spatial configurations	Must Have
FR12	The user can export the vibrotactile project to audio format	Should Have

Table 5.2: Functional Requirements for the Vibrotactile Editor

After the Vibrotactile Editor usability test with a participant from the TherTactExo project, he proposed that the editor should export the project to audio format to make it more flexible and general to all microcontrollers. From this suggestion emerged another functional requirement, the FR12.

5.1.4 Wireframes and Quality Attributes

Along with user stories, we used the wireframes to communicate with the team to promote discussion about the requirements of the system. In Figure 5.1, we present the "Initial Screen" wireframe, designed during the requirements elicitation process. The remaining wireframes can be viewed in the Appendix A.1

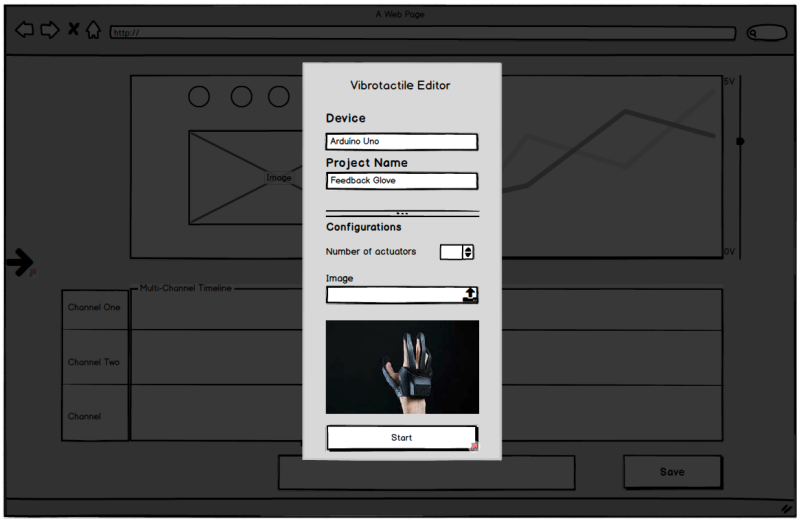


Figure 5.1: Initial Screen

Finally, the quality attributes, also called as "Non-Functional Requirements," are characteristics that the system must have in addition to functionality. Table 5.3 describes the quality attributes required for the system.

Quality Attribute	Description	Priority
Usability	The GUI must be designed in an efficient, simple way, to facilitate the interactions and to maximize the user experience.	Must Have
Interoperability	The system must be designed to facilitate the integration with external systems and provide an API according to the current design standards.	Must Have

Table 5.3: Quality Attributes

5.2 Requirements for Virtual Reality Components

During the final stages of the Vibrotactile Editor's development, we initiated the elicitation process to ensure that the integration was thought through properly. The two components' requirements are analogous; only the development technologies change; therefore, we have gathered it all in one single section.

The methodology followed for the elicitation of requirements was more straightforward than the Vibrotactile Editor, given its lower complexity.

From a user perspective, interacting with these components during virtual reality experiences, the component's purpose will be to transmit vibrotactile feedback that matches the experience. These virtual reality experiences will be conducted by other individuals, who need to program the vibrotactile components to match the coupled elements.

Therefore, another perspective that we took during this process was the programmer's perspective. The first requirement was obvious. The component would have to express the vibrations exported from the Vibrotactile Editor. To not be dependent on the editor, we studied a standard set of simple functions used in vibrotactile feedback devices. From this analysis, we came up with vibrations corresponding to sinusoidal and sawtooth waves. However, from the programmer's viewpoint, to programmatically create vibrations with a higher degree of freedom without using the editor was impossible. That said, we identified this further final requirement. To give the freedom to create more complex vibrations programmatically within the component.

To summarize, we describe the functional requirements in Table 5.4.

ID	Description	Priority
FR1	The user receives vibrotactile feedback when interacting with the elements associated with the vibrotactile components	Must Have
FR2	The programmer can associate vibrations exported via the Vibrotactile Editor to the vibrotactile components.	Must Have
FR3	The programmer can create simple vibrations through simple pre-defined functions.	Must Have
FR4	The programmer can create complex vibrations programmatically within the component.	Must Have

Table 5.4: Functional Requirements for the VR components

This page is intentionally left blank.

Chapter 6

Architecture

In this chapter, we report on all the details of the system architecture. In the section 6.1, we present the model selected for the architecture documentation. In section 6.2, we present the system architecture. Following, in section 6.3, we describe the chosen data model. Finally, in section 6.4, we provide the communication protocol with the prototype feedback sleeve developed by the TherTactExo team.

6.1 Software Architecture Documentation Model

Before defining the system architecture, we first decided on a model that would adequately document the entire system architecture. Therefore, we chose the C4 model. The C4 model features a set of hierarchical diagrams where we can visualize different levels of abstraction. From its definition, the C4 model is "an 'abstraction first' approach to diagramming software architectures, based upon abstractions that reflect how software architects and developers think about and build software" [72].

In this section, we present the principal concepts of this model. Although C4 does not dictate a particular notation, the most common notation (Figure 6.1), and used in the following section, is as follows:

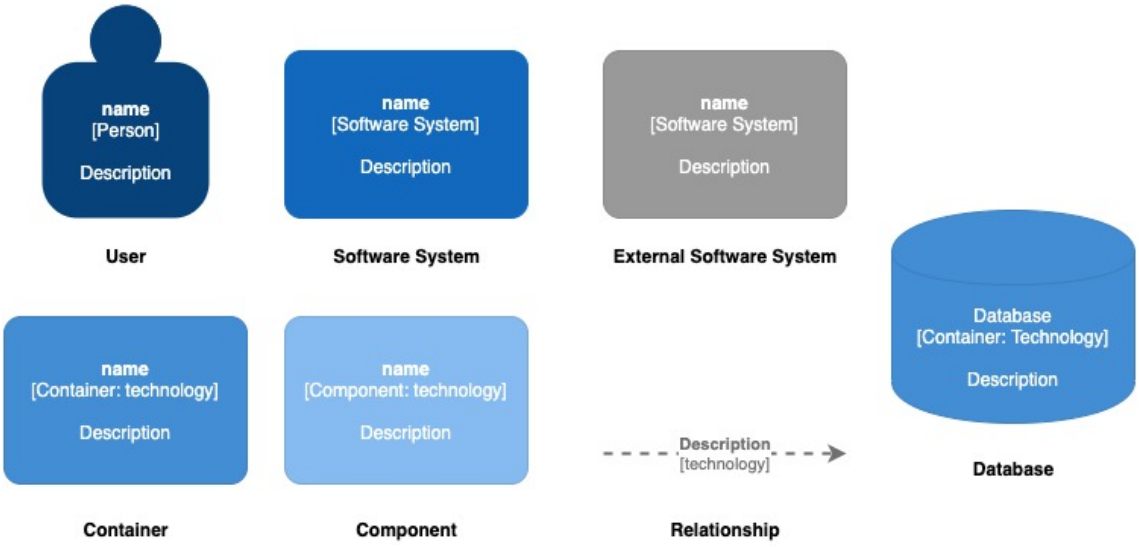


Figure 6.1: C4 Model notations

With the introduction to the notation used, we now define the different levels of abstraction offered by this model [72]:

- **Context Diagram**: Provides a starting point for diagramming a software system. It shows how the system is built, its interactions with users and external software systems.
- **Container Diagram**: Shows the high-level shape of the software architecture and how responsibilities are distributed across it. It also specifies the major technology choices and how the containers communicate with each other
- **Component Diagram**: Expands an individual container to to further identify the significant building blocks and their interactions.
- **Code**: The code shows how an individual component is implemented. However, in this model, this level of abstraction is optional.

That being said, we conclude the introduction to the C4 model. In the next section, we use the C4 model to present the architecture of the system.

6.2 System Architecture

For organizational reasons, we divide this section to match the C4 model levels of abstraction, detailed in the previous section.

6.2.1 System Context View

As we stated earlier, the first level of abstraction in the C4 model concerns the context diagram. This diagram presents the interactions with users and external software systems. Thus, we understand that our system will mainly interact with three external systems, namely:

1. A-Frame: Uses the Vibrotactile Editor to express vibrotactile feedback to the feedback device.
2. Unity: Uses the Vibrotactile Editor to express vibrotactile feedback to the feedback device.
3. Arduino: Provides a connection between the Vibrotactile Editor and the feedback device

Finally, we have the users that interact with the system. These are unregistered users that can access the services provided by the Vibrotactile Editor. In conclusion, Figure 6.2 shows the system context diagram.

6.2.2 System Containers View

Having the starting point in the context diagram and understanding how the system works more abstractly, the next step is to provide the high-level technical components and communication protocols between them. In Table 6.1, we describe the identified containers:

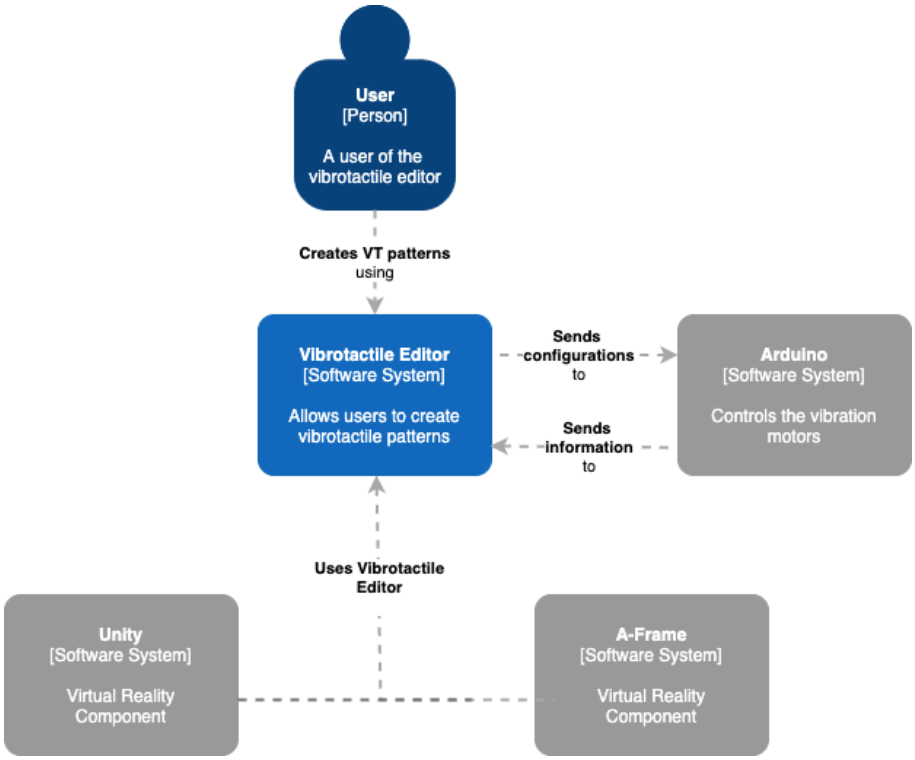


Figure 6.2: Level 1: System context diagram

Container	Description
Application	Client application that allows the users to create vibrotactile projects.
Middleware	Application layer acting as a mediator between different applications and exposes a set of different services
Database	Container that stores all the data within the Vibrotactile Editor, such as patterns and projects.

Table 6.1: Application Containers

Once we identified the components, the next step was to choose the technologies for their development and the definition of the communication protocols between each container. Table 6.3 describes the selected technologies for each container's development, and in Table 6.3, the communication protocols between them.

Container	Technology	Description
Application	ReactJs	JavaScript library for developing user interfaces. Already detailed in chapter 3.
Middleware	Node.js	Node.js is a open-source, cross-platform JavaScript run-time that allows developers write server-side scripts outside the browser [73].
Database	MongoDB	MongoDB is a document database with the scalability and flexibility to querying and indexing as the developer needs [74].

Table 6.2: Container Technologies

Protocol	Description
HTTPS	HTTPS is a secure communication protocol that protects the integrity and confidentiality between of data between the user's and the browser [75].
JSON	JSON is a standard text-based format for representing structured data based on JavaScript object syntax, typically used to transporting data in web applications [76].
Serialport	Serial communication interface to connect a serial device to the computer and capable of transmitting one bit at a time [77].

Table 6.3: Communication Protocols used between the containers

After all the relevant details were covered, we designed the container diagram, illustrated in Figure 6.3.

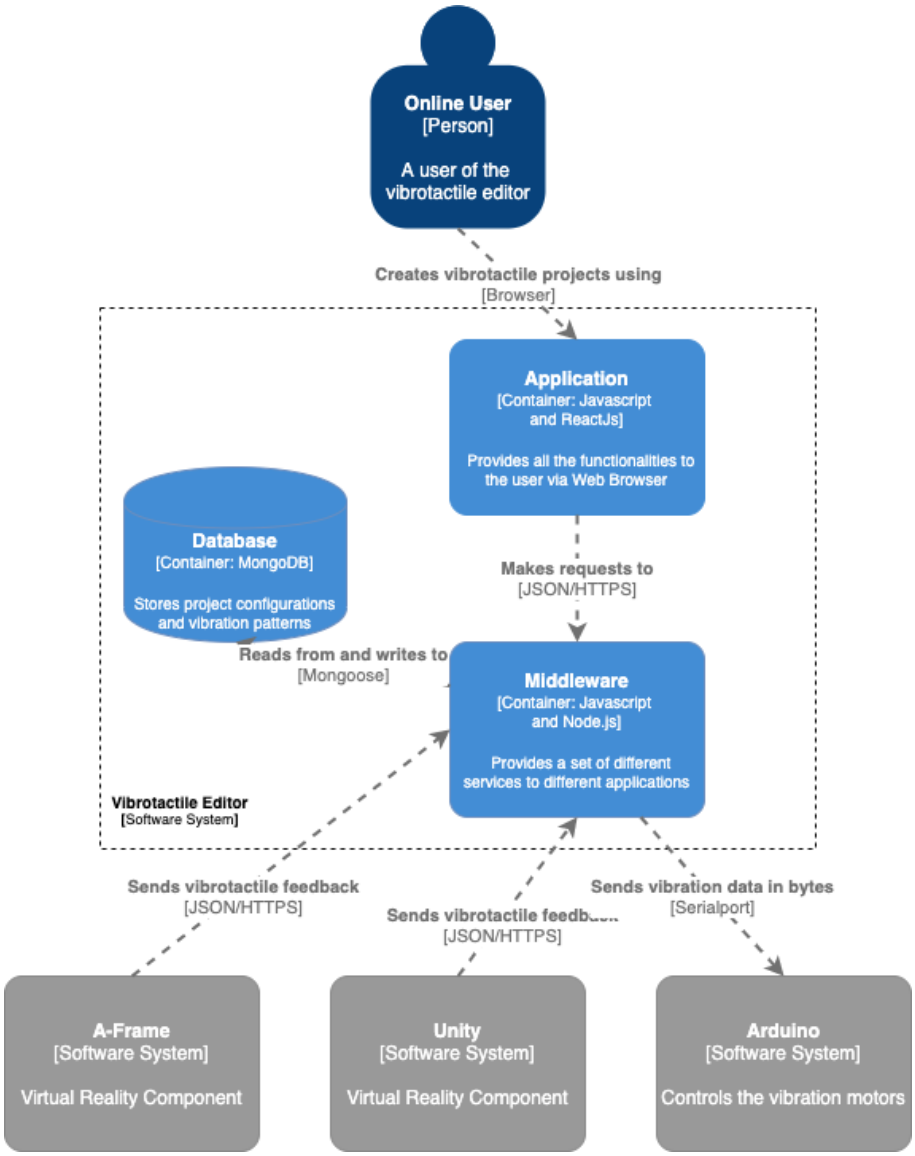


Figure 6.3: Level 2: The container diagram

6.2.3 System Components View

To describe the component diagram it is necessary to expand the containers that compose the system. However, our emphasis is on the Application and Middleware containers as they are the system's fundamental containers.

The first expansion details the Application components. Here, we distinguish between two types of components:

- **Controllers**: These components are used to access, modify and update data. For example, controllers know how to query and alter data via a RESTfull API [78].
- **React Components**: These components are independent pieces of the user interface that describe what should appear on the screen [79].

That said, the application container is composed of the components detailed in the diagram in Figure 6.4.

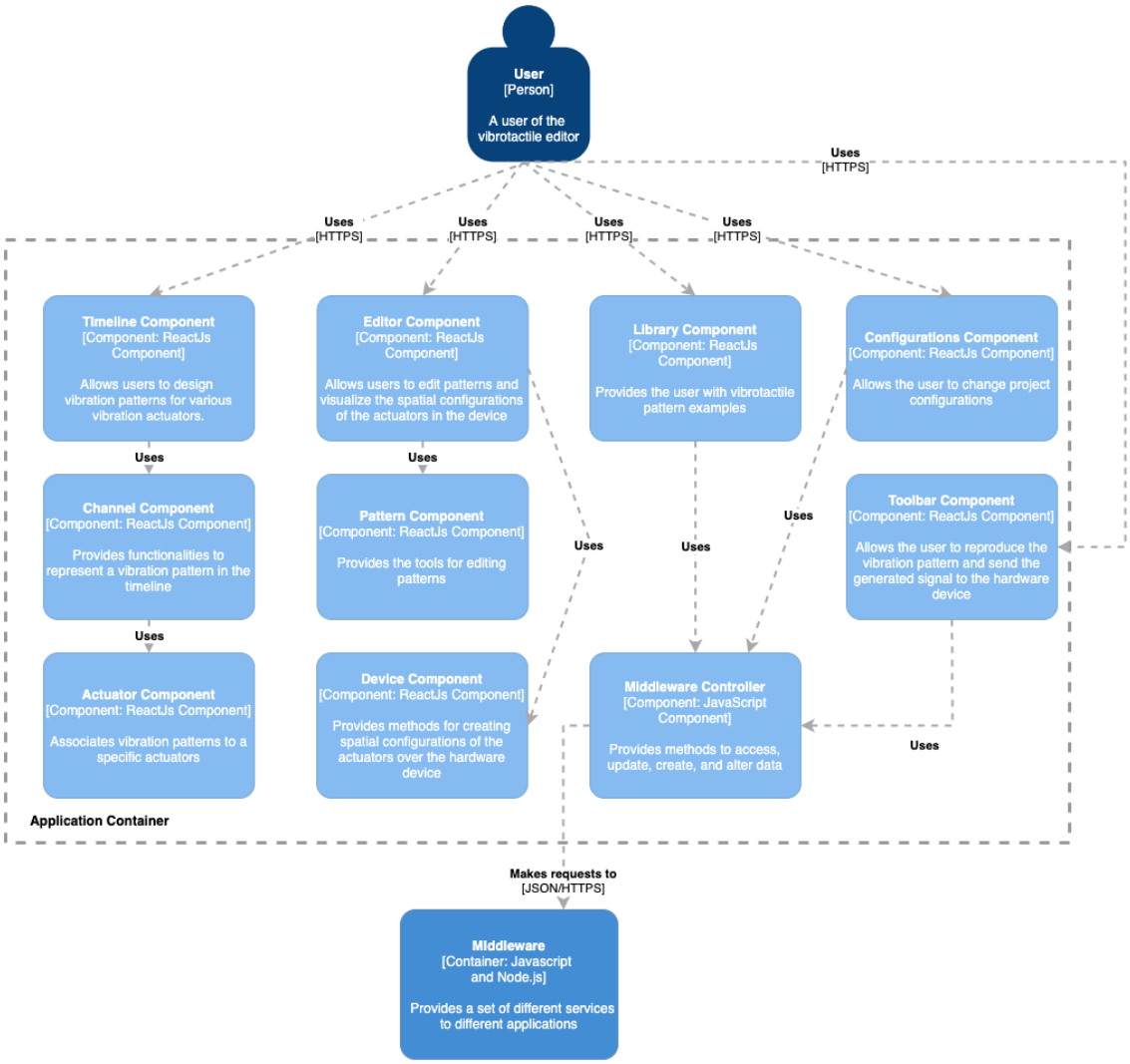


Figure 6.4: Level 3: The component diagram for the Application Container

Regarding the middleware, the components diagram (Figure 6.5) has the following structure:

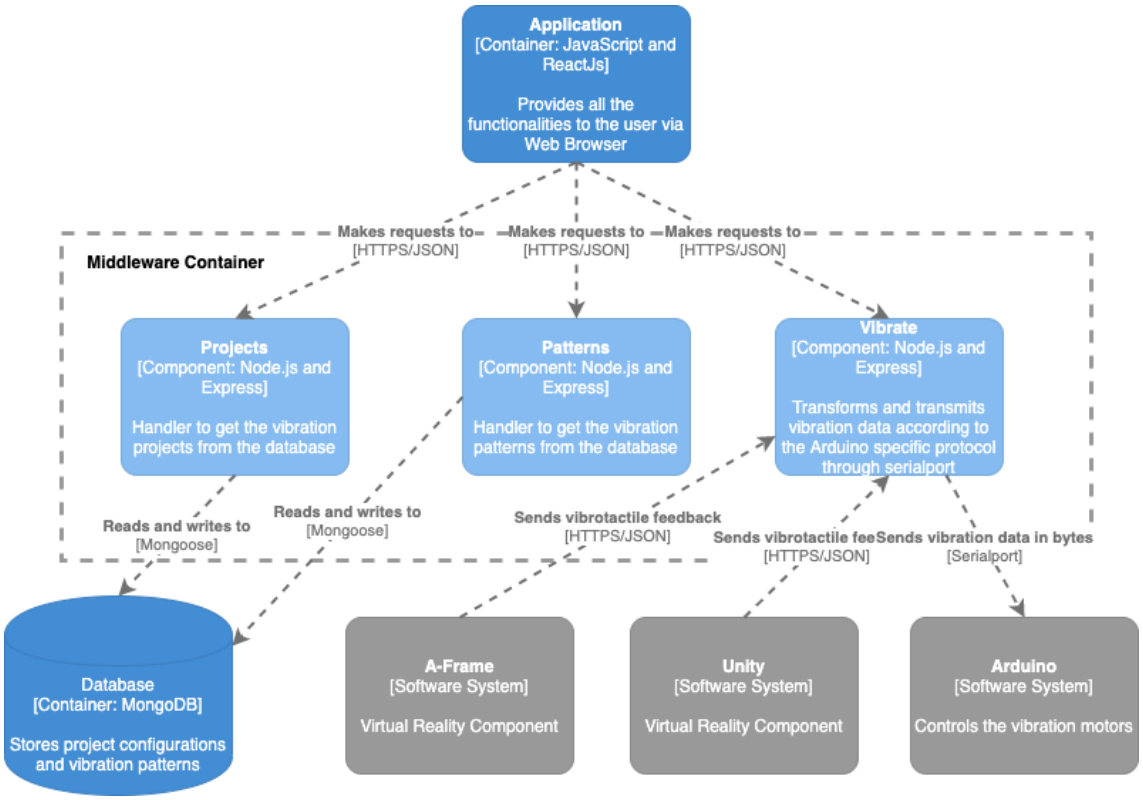


Figure 6.5: Level 3: The component diagram for the Middleware Container

6.3 Data Model

The database is responsible for storing and securing all data regarding the projects and patterns created in the editor. For this project, we chose MongoDB as the technology to develop the database. In this section, we present the design process of the selected data model.

Considering the use of the application and the technologies used for its development, we did not find the necessity to use a relational database. Since MongoDB is a database based on documents [74] that follow a structure analogous to a JavaScript object, this decision seemed natural to take.

That said, we designed the following data model, illustrated in Figure 6.6:

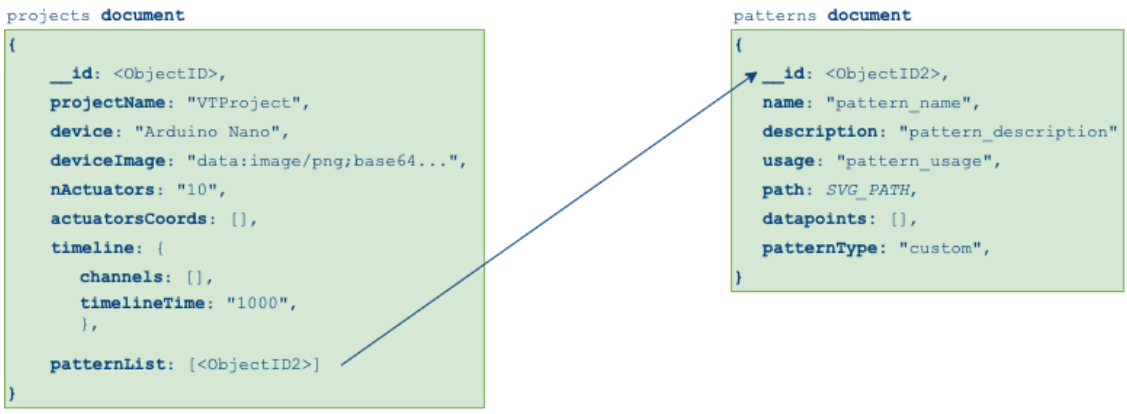


Figure 6.6: Data Model

6.4 Communication Protocol

The feedback device we are working with uses six actuators. Each actuator vibrates at an intensity between 1 to 255. This implies that any vibration scale has to be translated into this interval of values. That said, the Arduino reads the vibration data from the serial port. As already mentioned, communication via the serial port occurs bit by bit. Therefore we defined seven messages, listed in Table 6.4, to read the bits.

Message No.	Description
1	Sends the 0-bit value indicating that the next six messages are the intensity values for each actuator.
2	Intensity value between [1,255] for the first actuator.
3	Intensity value between [1,255] for the second actuator.
4	Intensity value between [1,255] for the third actuator.
5	Intensity value between [1,255] for the fourth actuator.
6	Intensity value between [1,255] for the fifth actuator.
7	Intensity value between [1,255] for the sixth actuator.

Table 6.4: Communication protocol messages

The first message that arrives has a value of zero bits, indicating that the following six messages will be the actuators' vibration values. After reading all the last six messages, the following message arrives at the microcontroller according to the set sampling frequency value.

In summary, the communication protocol is as described bellow:

1. Message No. 1 arrives.
2. Arduino knows the following six messages contain the vibration intensity for the six actuators.
3. Message No. 2 to 7 arrive with intensity values between [1,255].
4. Next available message will arrive according to the sample rate defined.
5. Repeat.

This page is intentionally left blank.

Chapter 7

The Vibrotactile Editor System

This chapter describes the details about the implementation of the system developed. During this dissertation, we developed three components: the Vibrotactile Editor, the Middleware, and the VR components. Section 7.1 describes the Vibrotactile Editor's implementation details. Section 7.2 details the Middleware implementation. Finally, in section 7.3, we present the implementation details of the VR components.

7.1 Vibrotactile Editor

The Vibrotactile Editor was the primary outcome that resulted from this project. The editor introduces features that allow the user to create highly customizable vibrotactile patterns through a built-in waveform editor and the multi-channel timeline. In this section, we describe in detail the process, procedure, and decisions we made during the implementation of the Vibrotactile Editor.

7.1.1 Graphical User Interface

In this subsection, we present the final results of Vibrotactile Editor . The development of the editor was divided into two parts. We conducted the Vibrotactile Editor GUI usability tests between this interval of time, marking the first part of the development and beginning the second. During the first iteration, we developed most of the functionalities. In the second iteration, we implemented new emerging features, and, above all, we adjust the editor according to the results of the usability tests. A significant challenge faced during this software development stage involved creating an editor that any user could operate effortlessly with a minimum of experience and knowledge.

In Figure 7.1, we show the final version of the Vibrotactile Editor. We first introduce the editor in its entirety, and then we describe the core components.

The first component we want to detail is the pattern editor. In the pattern editor, located in the top-right corner, we can create vibrotactile patterns by inserting data points represented in the image by the blue circles. Each of these data points represents a key-value pair. The first value refers to time and the second to intensity. The d3.js library is responsible for handling the vector that corresponds to the pattern. The user interacts with the editor through the predefined functions for adding and removing data points. Besides, we have also defined keyboard shortcuts corresponding to the same features to facilitate

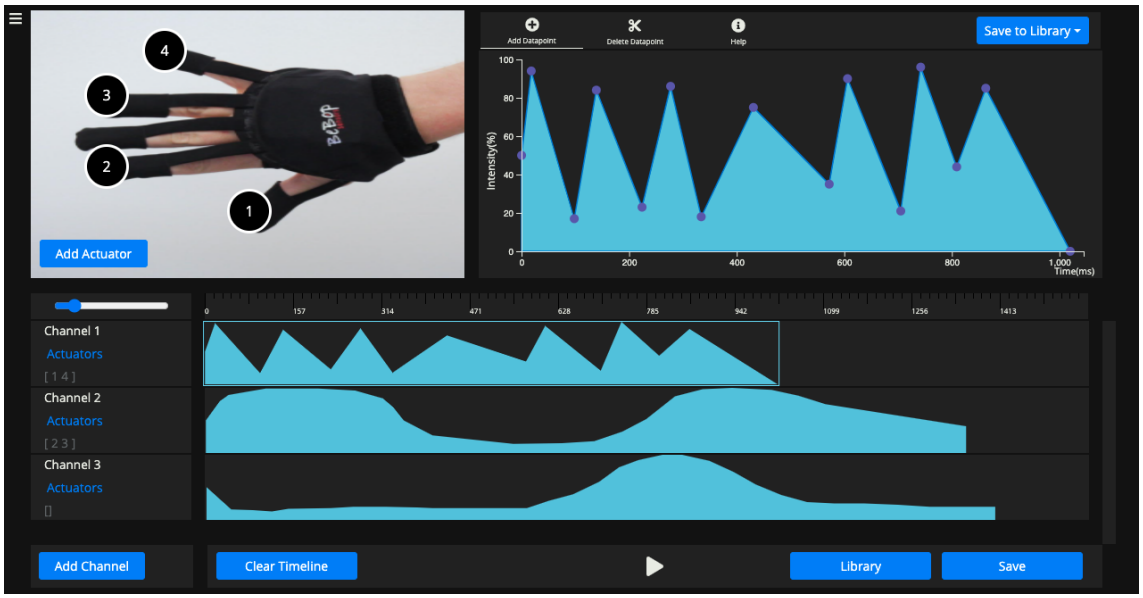


Figure 7.1: Vibrotactile Editor GUI

interaction. Finally, the pattern editor's last feature enables the user to save the designed pattern into the pattern library.

The component on the top-left corresponds to the image of the vibrotactile device that will be used throughout the project. This component aims to envision how the vibrotactile feedback will be replicated by the actuating motors depending on their position in the prototype. Here, the user can add and remove actuators and create the spatial configuration of choice.

In the timeline component, placed in the middle of the Vibrotactile Editor, the user can create a more complex vibrotactile pattern by adding different patterns into multiple channels. In each of these channels, we can associate various actuators with the contained patterns. This functionality is done through user interaction with the "Actuators" text button. The associated actuators are displayed at the beginning of each channel, underneath the channel name and the text button. In each project, the timeline has three predefined channels. However, it is possible to associate and remove more channels as needed by the user. This functionality is available in the toolbar component placed at the bottom of the Vibrotactile Editor.

In the toolbar, the user can use the functionalities to access the library of examples, save the project, add channels to the timeline and finally reproduce the project on the vibrotactile feedback hardware. In the example library, illustrated in Figure 7.2, the user can search and add patterns directly to a timeline channel of choice.

Finally, in the save project menu (Figure 7.3a), besides effectively saving the project, we can export the project in two formats. The first format is used in the middleware, in the Unity and A-Frame components. We explain this process in the individual sections for each of the components. The second format corresponds to exporting the project to audio, precisely the WAV format. Exporting the project to audio is handled with the Web Audio API's support and the *audioEncoder* dependency. The user defines the sample rate in the export menu (Figure 7.3b) - within the values accepted by *audioEncoder* - and the file name. This operation results in audio files equivalent to each timeline channel.

For illustration, Figures 7.4a and 7.4b show an example of a Vibrotactile Editor project

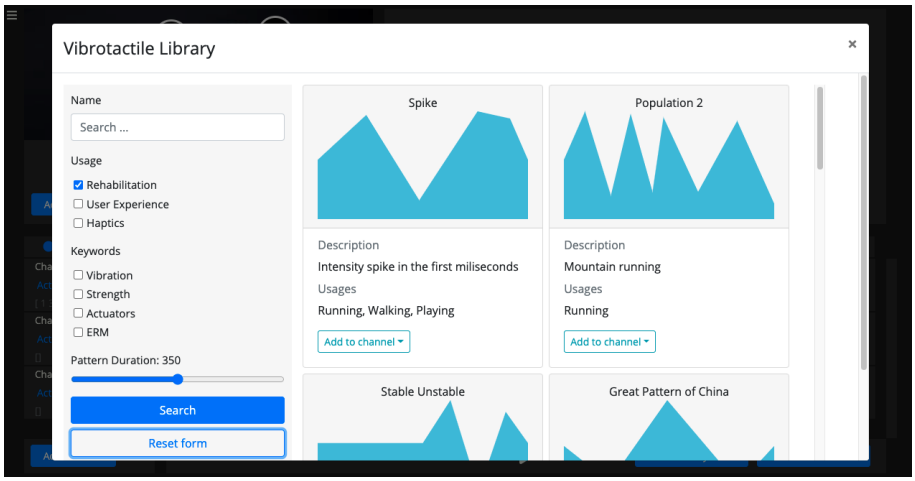
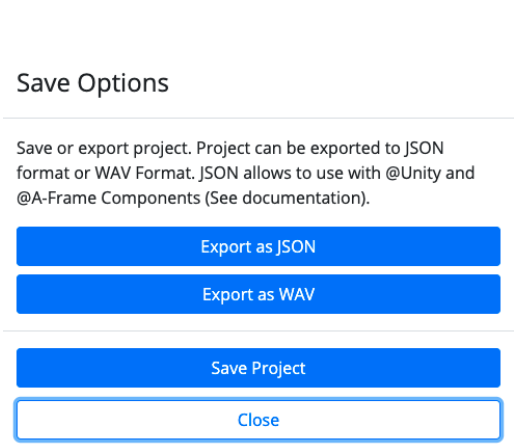
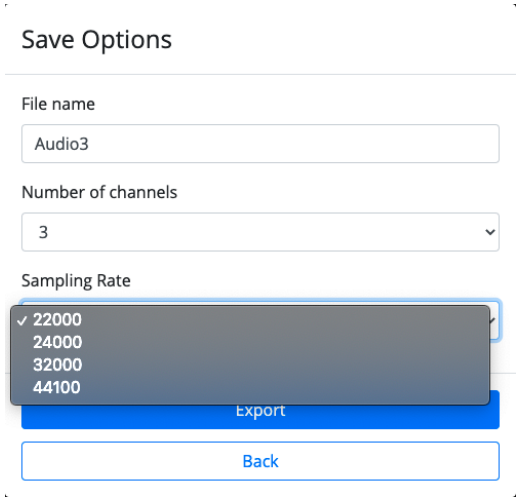


Figure 7.2: Vibrotactile Editor library component



(a) Save Menu



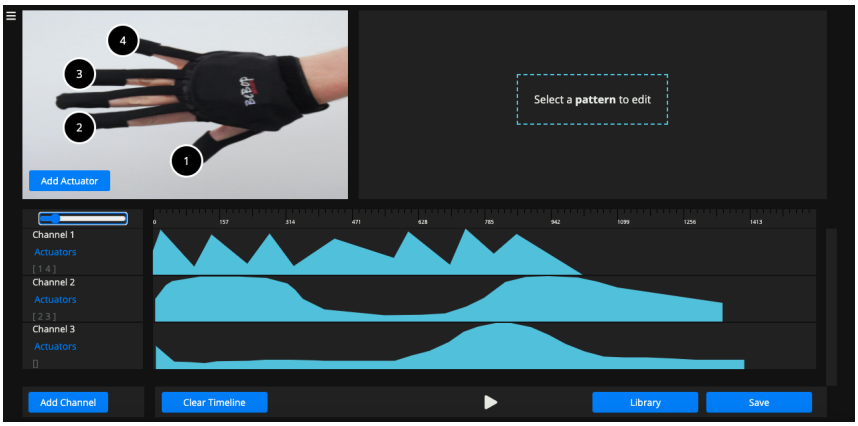
(b) Export to audio menu

Figure 7.3: Vibrotactile Editor save menu component

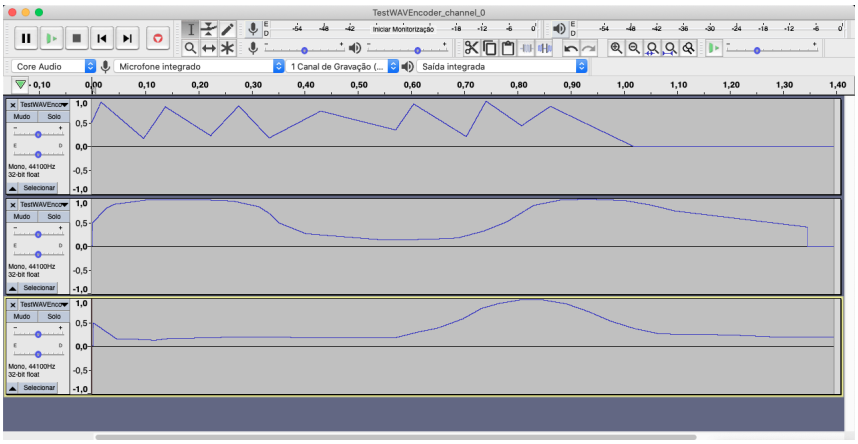
and the resulting audio files in the audio processing software Audacity [80]. The export to audio format feature was introduced to provide more flexibility and widen the editor's communication with other microcontrollers that do not follow the same transmission protocol. The audio files facilitate sampling and can be used efficiently to create another communication protocol with any other microcontroller type.

To finalize this section of results, we are left with the component that allows the loading or initiation of a project. In Figure 7.5, we have an overview of the component that allows these operations to be performed. For a new project, the user defines the microcontroller used, the project name, the prototype image, and, finally, the number of actuators. All these fields are validated first in the editor but are also subject to validation in the Middleware.

Having described and demonstrated the Vibrotactile Editor GUI, now, we describe the implementation details.



(a) Vibrotactile Project



(b) Audacity Project

Figure 7.4: Resulting audio files exported from the Vibrotactile Editor

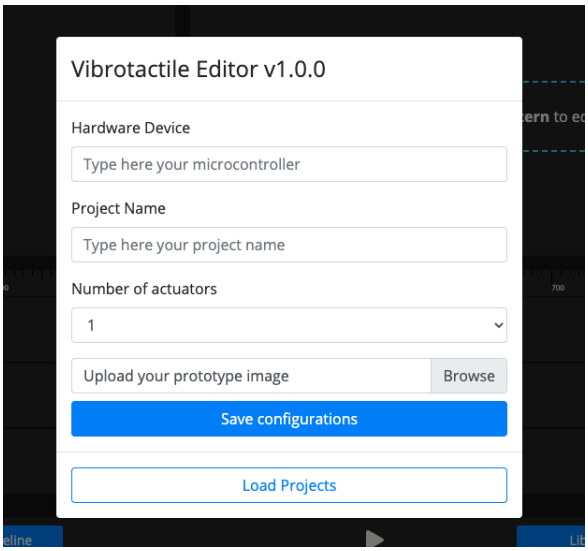


Figure 7.5: Vibrotactile Editor configurations component

React-Redux Flow

In React, the data follows from parent components to child components [81]. In a medium or large-sized application, keeping track of the application data may be complex. Therefore,

we decide to use a state-managing library for managing and updating the application state, as the Redux framework [82].

The Redux framework serves as a centralized store for state that needs to be used across the entire application, with rules ensuring that the state can only be updated predictably [82]. To facilitate the description of how the data flows in Redux, in Figure 7.6 below, we illustrate the whole process step by step:

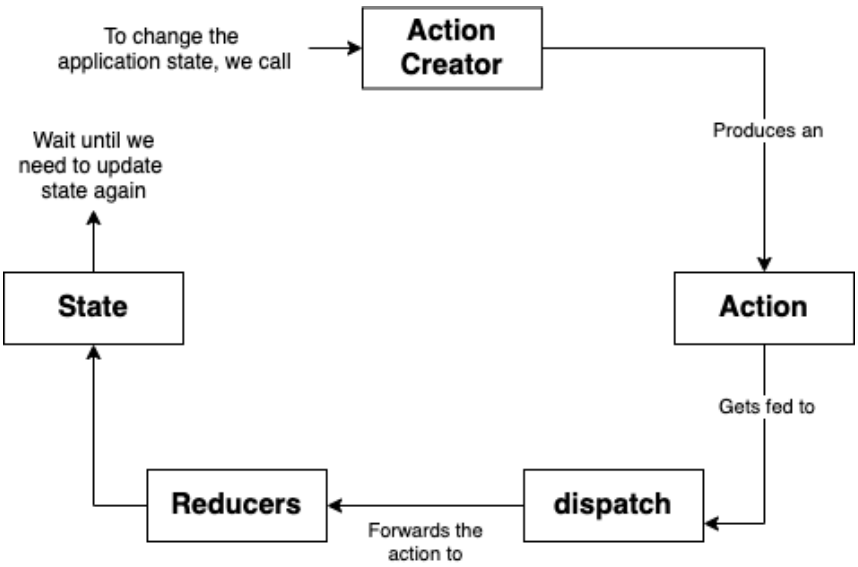


Figure 7.6: Data flow in Redux adapted from [83]

The state of the application is kept in an object tree which we call **store**. To be able to change the application's state tree, we need to create an **action**. This action is produced in the **Action Creator** and contains information about how we want to change our central state's data [83]. In the Listing 7.1, we can see an example of an action that indicates it wants to add another datapoint to the pattern that we are editing.

```
{
  action: {
    type: "PATTERN_UPDATE_DATAPOINTS",
    payload: {index: patternIndex, datapoints: _datapoints}
  }
}
```

Listing 7.1: Action

Having the action created, now we need to **dispatch** it to the store. Before it reaches the store, the dispatch forwards the action to the **reducer**. The reducer specifies how the state gets updated in response to the action received [82]. Through reducer functions, we can calculate a new state based on the old state and the action. After calculating the result, the reducer sends the updated data to the state.

To access the store, we need to create a React component that has a store reference. This component is labeled as the **Provider** component. Furthermore, a React component can only listen and make changes to the store when wrapped with the **connect** method, which communicates with the Provider to enable these operations. When the store updates, all wrapped components will update.

From a software engineering perspective, the connect method is an excellent example of

the observer pattern [84]. The observer pattern is a software design pattern that defines a one-to-many dependency between objects so that when one object changes state, all its dependencies are notified and updated automatically [85].

Dependencies

Apart from the technologies already mentioned, we used some dependencies to fulfill the application's requirements. In the following Table 7.1, we list the primary dependencies used in developing the graphical interface, followed by a concise description.

Dependencies	Description
d3.js	D3.js is a JavaScript library for producing dynamic, interactive data visualization in web browsers using primarily HTML, CSS, and Scalable Vector Graphics (SVG) [86].
react-bootstrap	React-bootstrap is a JavaScript library to replace Bootstrap without unneeded dependencies [87].
audio-encoder	Audio Encoder is a JavaScript library to encode audio buffers, created from raw data using the Web Audio API [88], from the browser to WAV format [89].
file-saver	FileSaver.js is a JavaScript library for saving files in the client side [90].
yup	Yup is a JavaScript schema builder for value parsing and validation [91].
formik	React tool to handle form submissions, validation and error messages [92].

Table 7.1: Dependencies used on the GUI implementation

7.2 Middleware

The Middleware refers to the server part of the platform. It manages all the business operations and exposes the services to the clients. Also, it is responsible for the communication with the database and the microcontroller.

The server was developed in Node.js, concretely using the Express framework. Express is a web application framework that provides a set of features for web applications [93]. These features include minimalistic code, robust routing, adopting Representational State Transfer (REST) service standards, and best practices. Furthermore, it is cross-platform, so it is not limited to one operating system [94].

Within these enumerated characteristics, we have routing. Routing refers to the definition of Uniform Resource Identifier (URI)s and how they respond to client requests [94]. Having defined a route, it decides how it proceeds according to the incoming requests. These requests arrive from methods. In our middleware, we registered three routes, described in Table 7.2. In the first column, we have the designated route, and in the following columns, the accepted HTTP methods.

Route	GET	POST	PUT	DELETE
/projects	Yes	Yes	Yes	Yes
/patterns	Yes	Yes	Yes	Yes
/vibrate	No	Yes	No	No

Table 7.2: Defined routes

The first registered route is associated with getting (**GET**), creating (**POST**), updating (**PUT**), and deleting (**DELETE**) **projects**. Requests arriving at this route are handled according to the associated HTTP verb. The processing of these requests takes place in the handler function. A GET call to the projects route returns all available projects. However, it is possible to retrieve only one project by concatenating its identifier in the request. For example, we have GET */projects:id*, where the *id* parameter is the specific project's identifier. To store a new project, we use the POST verb. We define the default encoding for the body of the request as *application/json* with the same structure as the *projects* document, defined in the data model section, in chapter 6.

As with the HTTP POST method, the PUT method follows the same guidelines. However, we use PUT when updating an existing project. Thus it is necessary to provide the project identifier in the route. The HTTP DELETE method deletes a specific project and requires specification of the project identifier in the route as well. In the diagram in Figure 7.7, we illustrate the process of obtaining a project.

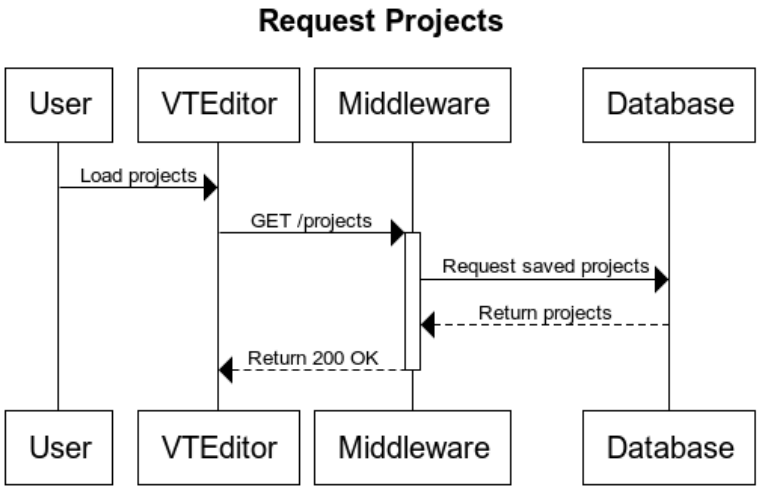


Figure 7.7: Sequence diagram for loading projects

That said, the following routes behave similarly, so we focused only on the differences.

The second route allows you to get, create, update and delete **patterns**. This route is in many ways analogous to the previous one. The difference is concerned with the data schema transmitted in the POST method's body. Here, we use the same structure as the *patterns* document.

Finally, the **vibrate** route, only available with the HTTP POST method, is where the entire process, from receiving the data, coming from the editor or the components, to sending the vibrotactile feedback to the microcontroller, is carried out.

To accomplish this, we had to develop a solution to handle the different types of data received. The first decision was to define a flexible and easily manipulated data schema that would be adaptable to both the editor and the other components communicating with this route. The resulting schema is presented in Listing 7.2 below:

```
{
    "samplingRate": 50,
    "numberOfActuators": 6,
    "channels": [
        {
            "patterns": [
```

```
7                   {
8                       "datapoints": [
9                           {
10                              "intensity": 50,
11                              "time": 0
12                          },
13                          {
14                              "intensity": 50,
15                              "time": 175
16                          },
17                          {
18                              "intensity": 50,
19                              "time": 350
20                          }
21                      ],
22                      "startingTime": 0
23                  },
24                  {
25                      "ramp": {
26                          "initialIntensity": 20,
27                          "finalIntensity": 100
28                      },
29                      "startingTime": 450,
30                      "duration": 1000
31                  }
32              ],
33              "actuators": [
34                  0,
35                  5
36              ]
37          }
38      ]
39  }
```

Listing 7.2: Data scheme for the vibrate route

The first two of fields give us the sampling frequency and the number of actuators in the project. Then we have the channels where each contains information about the patterns contained and which actuators are associated. The patterns can be a collection of points, such as those from the editor or specific vibrotactile feedback functions. These functions are available only in the components, and we will explain them in their respective sections.

With this data schema, we can handle all requests in a simple and flexible way. Finally, we transform and transmit the vibration data through the serial port following the microcontroller's protocol, presented earlier in Chapter 6.

We tested all these routes during the development phase, using the Postman software [95], before beginning with the usability tests.

Dependencies

To conclude this section, as in the previous one, we list the dependencies used during development and their description.

Dependencies	Description
express	Node.js framework for developing web applications.
body-parser	Node.js body parsing middleware. Extracts the body of an incoming request and exposes it in an easier way to handle [96].
mongoose	Mongoose is a MongoDB object modeling tool designed to work in an asynchronous environment [97].
serialport	JavaScript package to access serial ports [77].

Table 7.3: Middleware dependencies list

7.2.1 Database

The database is the place where we store all the necessary information from the application. From project settings to new customizable vibrotactile patterns, everything is in the database. Out of a diverse selection of databases, we decided to develop our own as a document-oriented, non-relational database. Within this spectrum, we chose the MongoDB database. One of the reasons for this choice was the flexibility to create collections of data. An advantage when system requirements emerge or evolve, and database adaptation is more straightforward.

We integrated the database with the server from the dependency mentioned in the previous section, *mongoose*. For this integration, it is only necessary to define the connection to the database on the server. When creating the connection, it is required to supply some parameters such as the port, location, and database name. All operations related to changing data in the database and creating data schemas were executed on the middleware through the mongoose dependency.

7.3 Virtual Reality Components

This section was divided into two subsections. Subsection 7.3.1 contains the implementation details of the A-Frame component, and subsection 7.3.2 the implementation details of the Unity component.

7.3.1 A-Frame Component

A-Frame is a framework for creating web-based virtual reality experiences. One of this project's objectives was to develop a component to be used later in virtual reality tests with vibrotactile feedback. Thus, in this section, we report the development details of the developed A-Frame, labeled A-Frame Vibrotactile Component, available at `https://github.com/ZeCanelha/aframe-vibrotactile-component` and in Appendix B.1.

One of the biggest challenges observed during the development of the component was the specification of the methods and the behavior itself. We questioned whether the component

would only use vibrations exported from the Vibrotactile Editor or it would also implement inherent functions that programmers could use in a simple way to create vibrotactile feedback. We concluded that one option would be to expose an API with the functions available for use in the A-Frame context. Following, we describe all the methods developed as well as examples of their usage. Finally, we present a simple model of a virtual reality experience where we transmit vibrotactile feedback to the user according to the virtual ground he/she is walking on.

A-Frame Vibrotactile Component API

The A-Frame component features four methods of creating vibrotactile feedback. Before beginning to detail each of these methods individually, it is necessary to understand how this feedback is transmitted. As we previously discussed, the Vibrotactile Editor application's middleware communicates with the microcontroller, which subsequently sends the feedback information to the feedback prototype. Like the GUI, the reproduction of the vibration is handled through the *vibrate* route. Here, we adapted all the inherent methods of the component to the protocol followed in that same middleware route. Having said this, we now continue with the enumeration and description of the available A-Frame Vibrotactile component methods.

Vibrotactile method

A component is defined as "JavaScript modules that can be mixed, matched, and composed onto entities to build appearance, behavior, and functionality" [98] in A-Frame. Besides, we can use it declaratively in the HTML or programmatically with JavaScript. Therefore, to use the vibrotactile component, it is necessary to associate it with an entity. In short, "entities are placeholder objects to which we plug-in components to provide them appearance, behavior, and functionality" [98]. That said, it was necessary to register the component and assign a name - **vibrotactile** - to use declaratively in the DOM, attached to an entity. In the Table 7.4, we describe the component properties followed by an usage example, illustrated in Figure 7.8.

Property	Description	Default Value
src	Path to the vibration file exported from the Vibrotactile Editor	none
event	A-Frame event that triggers the vibration.	none

Table 7.4: Vibrotactile component properties

```
<a-scene>
  <a-box vibrotactile="src: vibrations.json; event: mouseenter;"></a-box>
</a-scene>
```

Figure 7.8: Basic usage example of the Vibrotactile component

In the example given, we have the usage of the Vibrotactile component in the HTML interface. In the A-Frame scene, we have an entity that represents a box. In this box, when the user passes with the cursor - the gaze reference point in A-Frame - the attached vibration, defined by the vibration file *vibrations.json*, is triggered.

In the following description, we present the functions the user can use programmatically from the Vibrotacile component. We designed these functions to enable users to create

simple but common vibrations, such as vibrations with sinusoidal behavior or a sawtooth wave, often called a ramp.

Vibrotactile Sin and Ramp methods

The Sin and Ramp methods share a panoply of properties that we describe in the Table 7.5 below.

Property	Description	Default Value
samplingRate	Sampling rate in milliseconds	5
numberOfActuators	Number of actuators	6
actuators	The specific actuators to perform the vibration	[0,1,2,3,4,5]
startingTime	Time in milliseconds to start the vibration	0
duration	Duration of the vibration in milliseconds	1000

Table 7.5: Common properties between the two methods

The properties listed in the Table 7.5 give the user the possibility to adjust some of the vibrotactile feedback characteristics. For example, it is possible to set the sinusoidal vibration for five seconds for the first two actuators. That said, to create a sinusoidal vibration, we need to use a set of parameters, as listed in Table 7.6.

Property	Description	Default Value
sin	A JavaScript object containing the sine wave properties	
sin.amplitude	Sine amplitude value between [0,1]	1
sin.frequency	Sine frequency value	5
sin.phase	Sine phase value	0
options	Common parameters	Common parameters default values

Table 7.6: Sin method properties

The properties defined for this method are those required to designate a sine wave, the amplitude, frequency, and phase. To exemplify this method, we use the example in Figure 7.9. In this example, we defined the amplitude and the desired frequency. Since we did not set the phase, we assumed the default value. Next, we define the common properties, and finally, we run the method.

```
<script>
  AFRAME.registerComponent("vibrotactile-example", {
    init: function () {
      var sin = {
        amplitude: 0.8,
        frequency: 10,
      };
      var options = {
        samplingRate: 10,
        actuators: [0, 1],
        numberOfActuators: 6,
        startingTime: 0,
        duration: 1500,
      };
      var vibrotactile = this.el.components.vibrotactile;
      vibrotactile.sin(sin, options);
    },
  });
</script>
```

Figure 7.9: Sin method usage example

To define a vibration behaving as a single sawtooth wave, it is necessary to specify the initial and final intensities. According to the given values, we either establish the vibration as an ascending ramp or a descending ramp. There is also the possibility to create a constant vibration just by setting equal intensities. The ramp method's properties are detailed in Table 7.7, followed by an example (Figure 7.10) of the ramp's method use in a very similar manner as the sine.

Property	Description	Default Value
ramp	A JavaScript object containing the ramp wave properties	
ramp.initialIntensity	Initial vibration intensity value [0,100]	0
ramp.finalIntensity	Final vibration intensity value [0,100]	100
options	Common parameters	Common parameters default values

Table 7.7: Ramp method properties

```
<script>
  AFRAME.registerComponent("vibrotactile-example", {
    init: function () {
      var ramp = {
        initialIntensity: 25,
        finalIntensity: 10,
      };
      var options = {
        samplingRate: 5,
        actuators: [3, 4, 5, 6],
        numberOfActuators: 6,
        startingTime: 0,
        duration: 500,
      };
      var vibrotactile = this.el.components.vibrotactile;
      vibrotactile.ramp(ramp, options);
    },
  });
</script>
```

Figure 7.10: Ramp method usage example

The remaining methods follow a similar approach as the one previously described. To avoid making this a repetitive description, the remaining methods can be accessed in Appendix B.1.

A-Frame Component Virtual Reality Experience

After the development of the Vibrotactile component, we developed simple virtual reality experiments for testing purposes (Figure 7.11).

The first was a simple scenario with a set of A-Frame primitives that reproduced the associated vibration through events such as "mouseenter" and "click". The second experiment developed shows a scenario with different floor types (Figure 7.12), such as grass, sand, and stone. In this experiment, we have combined specific vibrations for each of these elements. Concretely, the ramp method for the grass floor, the sin method for the sand floor, and, finally, the customVibrations method for the stone floor. To simulate the walking motion, we use the keyboard arrows to move. In this case, as we "walked", the component sent vibrotactile feedback to the feedback prototype according to the user's position.

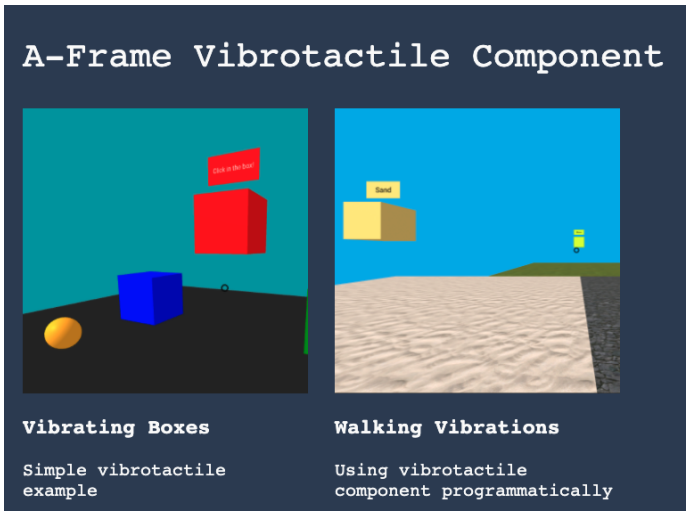
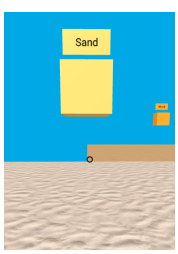
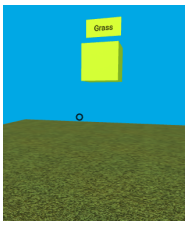


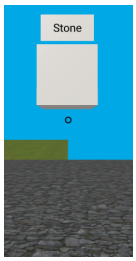
Figure 7.11: Experiences developed



(a) Sand floor



(b) Grass floor



(c) Stone floor

Figure 7.12: A-Frame Vibrotactile componente VR experience

7.3.2 Unity Component

The last of the project's objectives, the Unity component, also permits us to transmit vibrotactile feedback in virtual reality experiences. The development of this component became a challenge given the inexperience with the software. Moreover, as with the A-Frame component, doubts emerged about how it would operate and its functionalities.

In contrast to A-Frame, Unity features a very elaborate GUI. Therefore, we took advantage of it during the development of the component. Unity displays the component's properties visually in what is called the "Inspector". That said, we concluded that it might be an excellent option to expose the available methods and their properties in the Inspector window.

As mentioned earlier, the behavior of the Unity component is similar to that of the A-Frame component. The main methods that are part of the Unity component are the same as in A-Frame, namely ramp, sin, and customVibrations. Vibrotactile feedback is also sent through the middleware through the *vibrate* registered route. That said, we now detail the implementation of the unity component.

The Inspector in Unity exposes all the public properties of a component. However, it is not always enough. To achieve better results, we created a customizable editor: "A custom editor is a separate script which replaces this default layout with any editor controls that you choose [62]". With these considerations in hand, we developed the component with an editor that visually exposes all available methods (Figure 7.13), making it easier to create vibrotactile feedback.

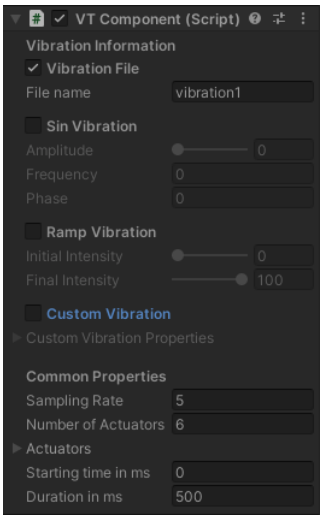
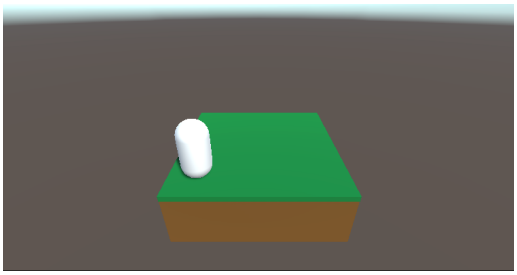


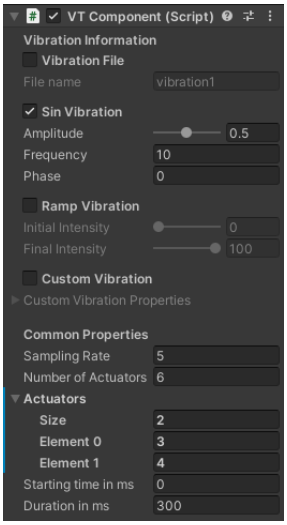
Figure 7.13: Unity vibrotactile component

In Figure 7.13 we can select which method we intend to associate with this component. In this case, we set the *Vibration File* method that takes the vibration file, exported from the Vibrotactile Editor, as an argument. This is an example of how to associate a particular method with a component.

However, it remains to be defined how the vibration is activated. Previously, in the A-Frame component, we found several types of triggers, from mouse events to solutions that involved checking a user's distance from the ground. In Unity, activating vibration is also made slightly simpler. Using the *onCollisionEnter()* method, we activate the vibration defined in the component's editor. As an example, in Figure 7.14, we have a user represented by the capsule (Figure 7.14a). In the terrain cubes, we associate our vibration component (Figure 7.14b). When the user is detected colliding with the terrain, the vibration defined is activated, sending vibrotactile feedback to the user.



(a) Unity scene example



(b) Vibrotactile component editor panel associated with the terrain

Figure 7.14: Unity vibrotactile component usage example

Chapter 8

Evaluation

This chapter describes the evaluation of the Vibrotactile Editor GUI, as well as the evaluation of the VR components.

8.1 Vibrotactile Editor Usability Testing

In this section, we describe in detail the usability evaluations of the Vibrotactile Editor. We started by identifying the usability factors we wanted to evaluate. Afterward, we created a test plan according to the objectives identified. We then performed usability testing with ten participants and analyzed the results. We used the results to improve the editor's user interface, and then we conducted a second round of usability testing with 5 participants.

Making a system usable can vary from a different number of views of what usability is. In this context, we want to measure usability based on the user performance view: how the user interacts with the system in a controlled environment [99]. Here, the system's usability is the product of the types of users, the tasks they perform, and the environment in which they work.

To summarize the Vibrotactile Editor usability test planning, we divided it into 4 phases, as shown in Figure 8.1. The first phase was the usability test planning, where we identified the tasks, users, metrics, and objectives. The second phase consisted of the execution of the usability test sessions. The usability test sessions started with a brief introduction to the project followed by the execution of the tasks, which were recorded. Afterward, users completed the System Usability Scale (SUS) questionnaire, and we interviewed them about their experience using the Vibrotactile Editor. In the third phase, we made a detailed analysis of the user interaction with the editor through the records of their interaction, notes, and questionnaire responses. Finally, we discussed the results and possible ways to fix the usability issues that were uncovered.

To conclude the planning section and taking into account the COVID-19 pandemic, we have established a strict and specific protocol (Figure 8.2) to respect all the safety rules imposed by Direcção Geral de Saúde (DGS), during the realization of the usability testing.

With this usability test, we intend to:

- Determine design inconsistencies and usability problems within the user interface
- Test the application in a controlled environment

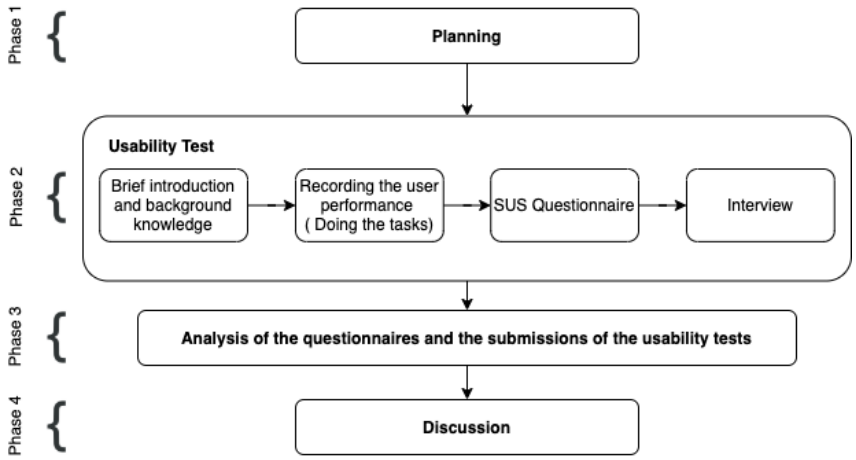


Figure 8.1: Methodology Approach

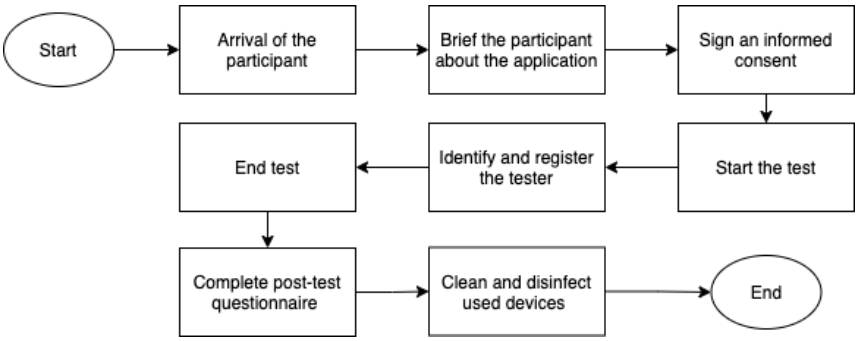


Figure 8.2: Usability Test Protocol

To measure the usability of the system, we identified the following metrics. To evaluate the user's performance in each task, we reviewed two possibilities: Task Completion and Task Efficiency. Task Completion relates to the completion of a task. Task Efficiency defines as in [100]:

TE = Quantity * Quality / 100

Where quantity is the proportion of the task completed, and Quality is the proportion of the goal achieved. To simplify the subjective value of the quality of the task performed, we assign a weight of 100% for a goal achieved and 50% for a goal achieved with help or with errors in accuracy and navigation. However, these metrics are only for evaluating and comparing participants' results. The purpose of this usability test is to find design inconsistencies and usability problems within the user interface.

Besides, to collect the system's usability rating, we chose the SUS questionnaire. This questionnaire comprises ten statements, scored on a 5-point Likert scale of strength of agreement, as follows:

1. I think that I would like to use this system frequently.
2. I found the system unnecessarily complex.
3. I thought the system was easy to use.
4. I think that I would need the support of a technical person to be able to use this system.

5. I found the various functions in this system were well integrated.
6. I thought there was too much inconsistency in this system.
7. I would imagine that most people would learn to use this system very quickly.
8. I found the system very cumbersome to use.
9. I felt very confident using the system.
10. I needed to learn a lot of things before I could get going with this system.

We presented the ten statements to the participants through a "Google Form" (see Appendix C.1).

The SUS provides a point estimate measure of usability and satisfaction, the SUS score. We can use it as a basis for comparison with other user interfaces.

The next step was to identify the group of users who would participate in this test. We identified the following groups that we would be able to reach:

1. Students without experience/knowledge in creating vibrotactile sensations.
2. Researchers from the TherTactExo project.

In the first usability test we targeted the first group and performed the test with eight participants.

8.1.1 Tasks

We defined a list of tasks to test all the relevant features of the editor. However, the tasks defined are too extensive to be documented in full in this subsection. Therefore, we transcribe the first task and summarize the remaining tasks. The full version is available in Appendix C.2.

> **Task 1** Create a new project with the name of your choice, where the device will be an Arduino Nano connected to 6 vibration actuators. Finally, upload the prototype image from the file system, available at Desktop/VTEditorTest/64-teste.jpg.

Tasks summarized:

1. Create a new project
2. Add and edit a new pattern
3. Create a spatial configuration
4. Associate actuators to created pattern
5. Save the project
6. Use the examples library
7. Associate actuators to imported pattern

8. Add a new channel
9. Add an edit a new pattern
10. Save the new pattern
11. Search for the saved pattern
12. Clear the timeline
13. Create timeline with pattern configurations
14. Add intervals to timeline

We tested these tasks in the first version of the editor, illustrated in Figure 8.3.

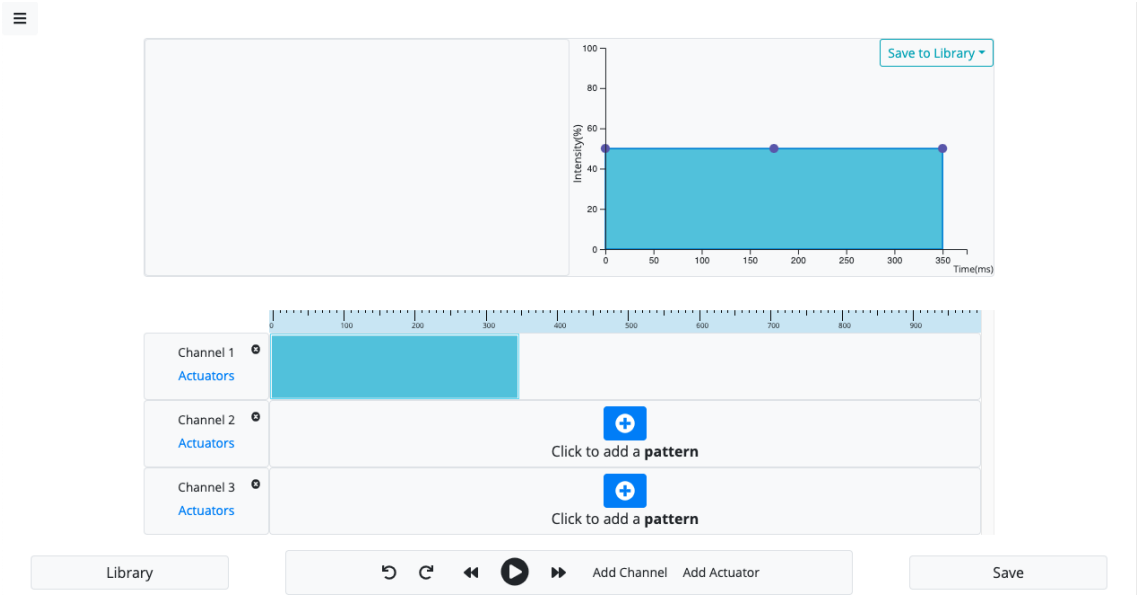


Figure 8.3: Vibrotactile Editor GUI for the first usability test

8.1.2 Procedure

The following procedure was adapted from the usability test plan template available in [101].

Participants took part in the usability test at Lab G5.3 in the Department of Informatics Engineering. A computer with the application and supporting software was used in a typical office environment, as shown in Figure 8.4. The participant's interaction with the application was observed by the facilitator and screen-recorded with audio. Both facilitator and participants followed the established COVID-19 safety rules.

The facilitator briefed the participants on the application and assured that what was being evaluated was the application, not the participant. Participants signed an informed consent that acknowledges: the participation is voluntary, that participation can cease at any time, and allows the recording of the session and subsequent treatment of the gathered data.

The facilitator instructed the participant to 'think aloud' so that a verbal record of the interaction with the application exists. The facilitator observed and took notes about user behavior, comments, and system actions.

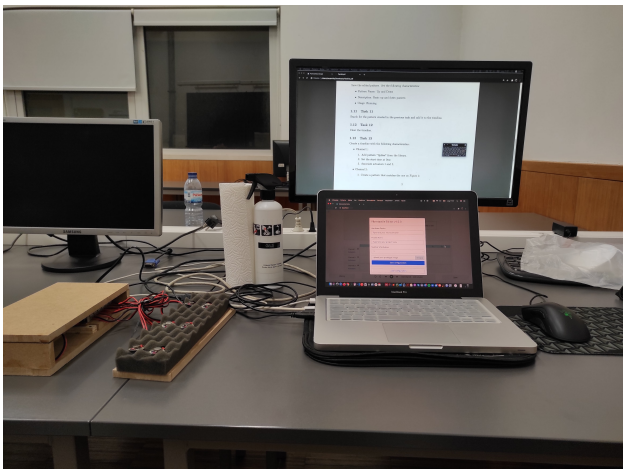


Figure 8.4: Lab environment

The facilitator instructed the participant to begin the test, informing that exploratory behavior outside the task flow should not occur until after task completion. Time measurement began when the participant started the task.. After all tasks were completed, the participant was asked to fill in the post-test SUS questionnaire.

8.1.3 Results

In Table 8.1, we present the success rate in executing the tasks and the main difficulties. The table organizes as follows: the first column contains the requested task; the second column summarizes the notes on the task performed; finally, the third column the number of users who completed the task, without errors. In the chart depicted in Figure 8.5, we present the efficiency of each task.

Task	**Notes**	**N° of users who successfully completed the task**
#1 Create a new project	-	8
#2 Add and edit a new pattern	Noticeable difficulties in finding the feature requested	0
#3 Create a spatial configuration	-	8
#4 Associate actuators to created pattern	Functionality not intuitive or perceptible	1
#5 Save the project	-	8
#6 Use the examples library	Navigation, location, accuracy problems.	2
#7 Associate actuators to imported pattern	-	7
#8 Add new channel	-	8
#9 Add an edit a new pattern	-	8
#10 Save the new pattern	Accuracy problems	1
#11 Search for the saved pattern	-	8
#12 Clear the timeline	-	6
#13 Create timeline with pattern configurations	-	6
#14 Add intervals to timeline	-	8

Table 8.1: Vibrotactile Editor usability tests results: task execution success and main dificulties

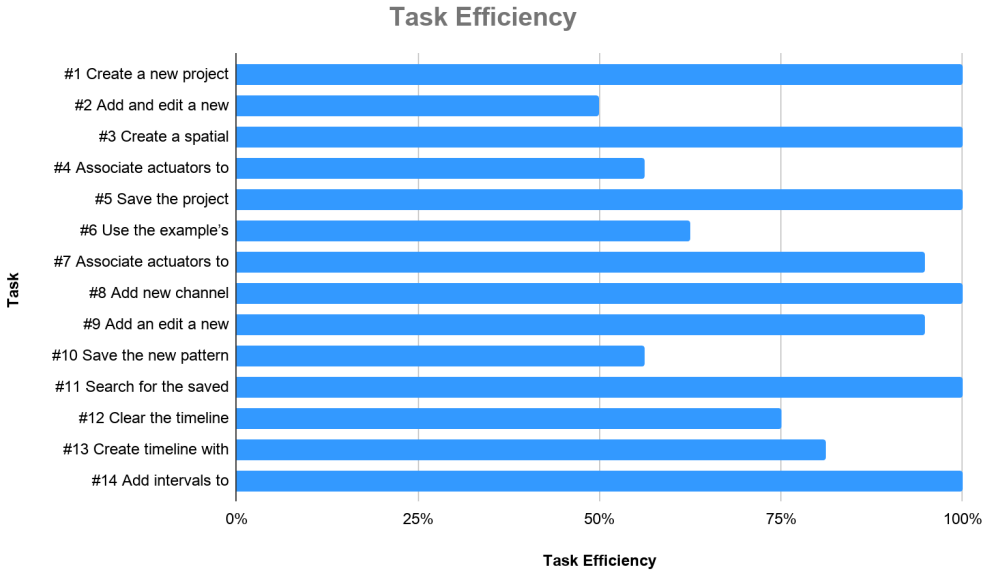


Figure 8.5: Task efficiency

Difficulties observed

During the study, we noticed that sometimes participants had difficulties completing some of the tasks. For simplicity, here, we address the tasks by their number.

In task #2, we asked the participants to add and edit a vibrotactile pattern to match a given pattern image. This task's purpose was to see if the editing functionalities (add, drag, and delete a keyframe) were quickly found and used. These features were inspired by editing software using the same keyboard shortcuts. We defined the "double-click" to add and "shift-click" to delete a keyframe. We observed that adding keyframes was not easily found. Some participants moved the mouse around the editor, searching for a function, and others tried to right-click in the editor area and over the keyframes. In the end, we conclude that dragging was the only direct and perceptible feature.

In task #4, we asked participants to associate an actuator to the channel. We observed difficulties in performing this task: participants wandered around the editor to find the functionality for quite some time before the moderator intervened.

Another task that brought confusion to the participants was task #6. We requested participants to add a new pattern from the library. However, some participants' first intuition was to add a default new pattern, clicking the button in the timeline channel. However, the participants realized that they were not achieving the right result and corrected their mistakes promptly.

Finally, in task #10, we asked the participants to save the created pattern. The first action of the majority of the participants was to select the save project button. Therefore, we assume the placement of the solicited functionality is confused.

There were also tasks that, from our viewpoint, would pose problems but which participants successfully executed without a problem. The first was task #9, where we asked the user to edit a pattern according to a given example. The purpose of this task was to discover the expansion function of the time axis. Although some users noticed the example had a more extended time scale than that shown in the editor, we thought this posed a problem

since no visual indication exists, such as an indicator to expand the axis. With this task, we intended to observe the participants' behavior and whether such an indicator would be necessary. However, the functionality proved to be easily discoverable.

The other task that we considered that might cause difficulties was #13. Our concern was to know if the user would move the timeline patterns according to the task description's time. However, we observed that most users had no difficulties executing the task, even though not all participants completed it thoroughly.

Sugestions

After completing the post-questionnaire, we interviewed participants to gather suggestions for improvement from their point of view. Regarding task #2, where participants had more difficulties, a general opinion was to have a visible toolbar to facilitate the editing of vibrotactile patterns. Other participants pointed out that tasks #6 and #10 could be improved by adding more vivid colors to the buttons. Finally, in task #12, we asked the user to clear the timeline. Some suggested adding a button to remove all the content at once would be more efficient.

SUS Questionnaire

In this section, we analyzed the SUS questionnaire results (Figure 8.6). The SUS statements alternate between positive and negative aspects of the system. Odd-numbered statements represent the negative, and even-numbered statements the positive. To calculate the SUS score, we need to follow the following rules:

- For odd-numbered statements: subtract one from the participant response.
- For even-numbered statements: subtract the participant response from five.
- Add up the converted responses for each participant and multiply that total by 2.5.

The SUS score classifies from 0-100. In a study of 2,324 applications of the SUS questionnaire [102] the average SUS score was found to be 70.14. That same study [102] categorized the evaluated systems into: cell phone equipment, customer premise equipment, graphical user interfaces, interactive voice response systems, internet-based Web pages and applications.

The GUIs category (208 systems analysed) had a mean score of 75.24 with a standard deviation of 20.77. The Vibrotactile Editor obtained a score of 80, meaning it was scored above average in the specific GUI category as well as overall.

8.1.4 Discussion

The usability testing of the Vibrotactile Editor showed encouraging results in this first iteration. The values obtained through the usability test show a total Task Efficiency of 83,6%. Acknowledging the participants' suggestions and the faults observed, we can be assured that the next iteration can show more promising results. One of the objectives of this test was to determine design inconsistencies and usability problems. We achieved this

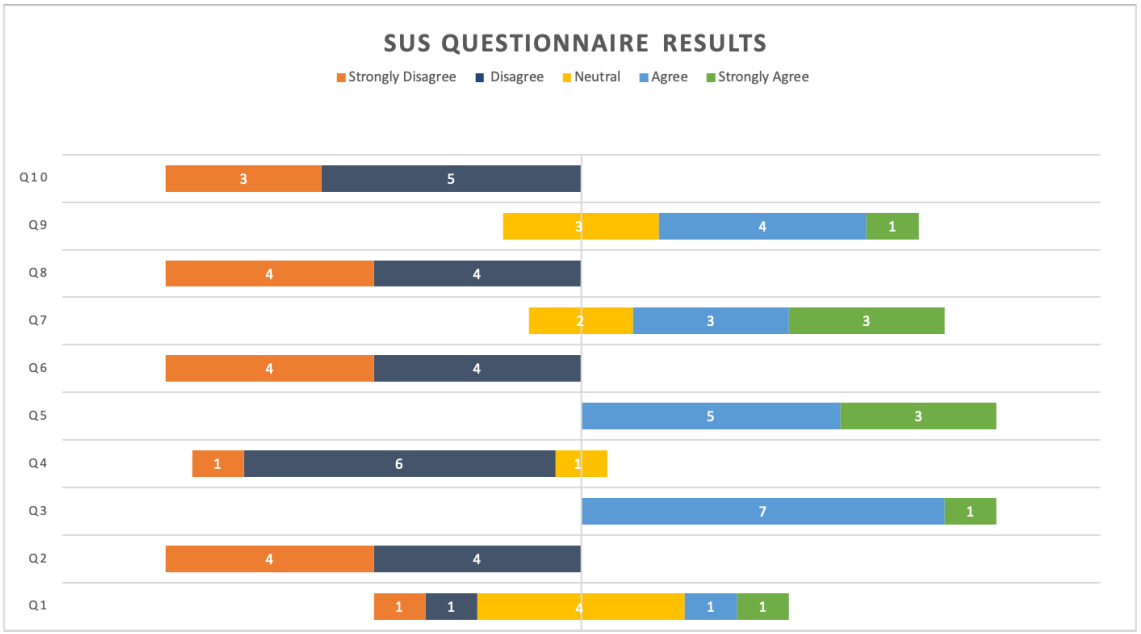


Figure 8.6: SUS results

objective, discovering the inconsistencies and pointing out the most relevant flaws in the editor's design.

During the participants' test, we notice the Vibrotactile Editor's learning curve was initially steep as it is required to become familiar with new concepts, background knowledge, and, principally, the editor's functionalities. Some of the practical difficulties were due to a lack of empirical knowledge and design flaws, as is Task 4, where the functionality requested was not noticeable. In task 6, on the other hand, we concluded that the difficulties were only due to design problems: the position and projection of the UI control element associated with the requested functionality.

After the initial learning phase, solving more advanced tasks became relatively simple, and the learning curve flattened. We can observe this behavior in task #13, where we ask the participants a combination of tasks #2, #4, and #6. As we discussed in the results section, these tasks score the lowest task efficiency value. However, when doing the #13, the participants had no difficulty. The Task Efficiency value and observations show that there was an acquisition of knowledge throughout the usability test. Regarding the participants' suggestions, some were in line with the solutions we have in mind for their resolution. This feedback was crucial to be able to view the editor from a user perspective. Besides, it also promoted discussion on how we could solve the raised issues.

Finally, the result of SUS shows that users found the Vibrotactile Editor satisfactory and easy-to-use. The result obtained was above the average of the numerous systems analyzed by [102]. Moreover, within the context of GUIs, the editor was also on the upside. However, this result is not significantly higher than the reported average. This shows that there are still factors that can be improved.

8.1.5 Second Round

To avoid a repetitive description of the process, we summarize the procedure involved, which is very similar to the previous one. Subsequently, we present the results obtained

along with their discussion.

In this usability test, we had 4 + 1 participants. In other words, we had four participants from the first group and one from the TherTactExo project. Regarding the first group participants, when recruiting, we focused on having two participants who had already participated in the first round and two new entrants. By doing this, we wanted to analyze whether the observed faults recurred in both new participants and repeaters, with the focus on the former.

Since we had one participant from the TherTactExo project in this iteration, we chose to run different tests.

Due to the COVID-19 pandemic and the DGS rules during this time, we conducted the second round of testing remotely. In short, we asked the participants from the first group to share their screen at the time of the test. Previously, we gave a brief introduction to the application and basic concepts. Besides, we also sent out an informed consent, similar to the first round of testing. By the end of the test, we discussed the changes and inquired about them with the participant.

For the TherTactExo project member test, we allow the participant to freely explore the application due to his knowledge and experience in the area. This approach intends to identify the limitations of the application in a future user's hands.

Figure 8.7, shows the Vibrotactile Editor GUI where we conducted the second round of usability testing. In this version, we incorporated all the feedback obtained in an effort to improve it.

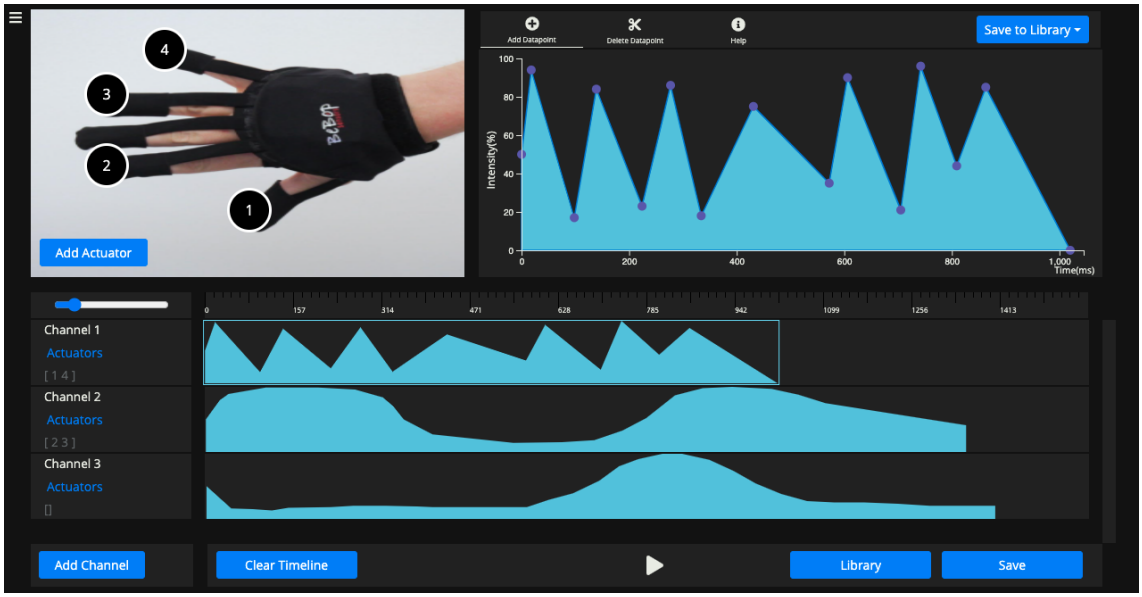


Figure 8.7: Vibrotactile Editor GUI for the second usability test

In comparison to the first round, the most resonant change is the color scheme. The change was suggested by a participant and further designed to emphasize and express elevation of certain features by contrasting them with darker surfaces [103, 104].

We also expanded the size of the timeline and the pattern editor. In the pattern editor, we created a toolbar where we indicate the available editing options. In the timeline, besides the size modification, we visually show which actuators are associated with each channel. Finally, we changed the position of the "Add Actuator", "Add Channel", and "Library" buttons to positions that seemed appropriate, according to the conclusions obtained in the

previous round, and included a button to clear the timeline.

Results and Discussion

Table 8.2 presents the success rate in executing the tasks and the main difficulties in the tasks that proved to be more difficult in the previous round. We omitted the remaining tasks' results as they were analogous and did not provide relevant information for analysis and discussion.

Task	Notes	Nº of users who successfully completed the taks
#2 Add and edit a new pattern	Users used the toolbar as intended	4
#4 Associate actuators to created pattern	Feature more noticeable and gives better understanding after first use	3
#6 Use the examples library	Library button is more eye-catching	4
#10 Save the new pattern	Save button is more obvious	3

Table 8.2: Vibrotactile Editor usability second round tests results

In task #2, we have improved the efficiency of the task significantly. Following the users' suggestions, we created a toolbar. The toolbar allowed the user to find the editor's features easily and promptly. Also, we added an information button where keyboard shortcuts are available. Finally, we indicate the coordinates as the user hovers the mouse over the editor for better accuracy in creating patterns.

In task #4, we noticed that one of the new participants had difficulty understanding what we required. We had already concluded the task's fault could be related to a lack of knowledge in this specific area. The design changes we made were minimal, as we did not have a more direct solution. However, we now provide an immediate visual indication to the user, telling which actuators are currently on that channel. After performing this task and having the new visual information, we noticed that the users better understood the feature's purpose.

In Task #6, we found a substantial improvement. All the participants in this task proceeded easily and quickly to the button which opened the example library. We conclude that the change of position and the increased exposure by changing the color of the button to a primary color made a significant improvement.

Finally, in Task #10, we also noticed a slight improvement in approaching the "Save to Library" functionality. However, we perceived some hesitation in one of the participants. As before, the participant address the save project button first. However, he ended up using the correct functionality.

Regarding the results reported by the participant from the TherTactExo project, we got very positive feedback, with some remarks for improving the application. The shortcomings reported involved changing the design of some elements. In the Table 8.3, we detail the feedback received.

In addition to the feedback received on the design aspects, we also received one improvement suggestion to enhance the application to make it more compatible with distinct hardware controllers. The Vibrotactile Editor was developed specifically for the protocol explained in chapter 6. This protocol, originated as part of the TherTactExo team's work,

UI Element	Suggestions
Actuators	The function to remove an actuator may appear after a "click" made on the relevant actuator. With a mouse hover, the remove button should be placed more on the top edge.
Keyboard shortcuts	The keyboard shortcuts can be changed to match the audio editors' standards. One mouse "click" to add a datapoint and a "control + click" to remove.

Table 8.3: Notes on the feedback received

is intended to control the prototypes they create. So one suggestion for improvement would be able to export the projects to another type of format. The format suggested by the participant was to transform the pattern data into audio files. These files, precisely the WAV format, facilitates sampling and adaption to other hardware controlers.

The second round of tests, although with fewer participants, served to consolidate the results obtained previously. From the feedback received, we developed an editor that matched a future user's expectations. The tasks that presented the most problems in the first iteration of the Vibrotactile Editor were solved. Where previously no participants had completed the task, as in task #2, all participants found success in this second round of testing. Not just in this task, but there was also a significant improvement in the remaining tasks. Finally, having very positive feedback from the project participant helps confirm the participants' positive results without experience and knowledge in this area. Furthermore, it validates our work developed during this dissertation.

8.2 A-Frame Vibrotactile Component API Usability Testing

In this section, we describe the usability evaluation of the A-Frame Vibrotactile Component. In this particular evaluation, we focused on the usability of the Application Programming Interface (API) in the A-Frame context. To do so, we develop a test plan consisting of two parts: an introductory tutorial to the A-Frame framework to train the participants, and an online usability test for evaluating the API. We decided to include an introductory tutorial due to the highly probable lack of knowledge and experience of the participants who would take part in this test regarding the A-Frame framework and to acquire sufficient experience to evaluate the component's API, abstracting from the A-Frame framework's possible complexity. Although it is JavaScript-based, its usage is very particular, oriented to 3D development. Therefore, we ensure that users have acquired the necessary experience to perform the usability test with this approach and that possible difficulties are mostly attributable to our API rather than to A-Frame itself.

An API's usability is a qualitative attribute that indicates how easy it is, for developers, to learn and use an API in a certain context [105]. We can consider different usability characteristics [105] when evaluating the usability of a programmatic API, such as:

- Learnability
- Efficiency
- Understandability
- Effectiveness
- Readability

- Satisfaction
- Debbugability
- Productivity
- Reusability
- Abstraction
- Expressiveness
- Unambiguousity

However, with this usability test, we focused on evaluating the following four characteristics of API's usability:

1. Learnability
2. Readability
3. Understandability
4. Abstraction

Learnability measures the capability of software to be learned by its developers with ease [100]. The readability of an API measures the level to which a code written with the API is readable by its users, such that they can follow it logically [105]. Understandability measures how well a user can understand the code without confusion [100]. We want the users to be able to understand what the API offers by just reading its documentation and its examples. To reinforce this characteristic, we also evaluate the API's abstraction. In [106], API abstraction is defined as the ability to guarantee that programmers can use the API with proficiency without requiring specific knowledge or assumptions concerning its implementation details.

We focus on these four characteristics as they are among the most frequently evaluated in primary studies [105], alongside efficiency and effectiveness. However, we did not cover later two characteristics. Efficiency defines how much software enables its users to use the right amount of resources to complete a task [100], effectiveness defines how much software enables its users to complete their tasks correctly [100]. We deemed these characteristics less relevant because we were not comparing our solution with other possible solutions and measuring efficiency and effectiveness in absolute terms would be difficult.

API usability evaluation methods can be performed using two main types of studies [105]:

1. Analytic methods: the object of study is the API specification. In these methods, the main focus is analyzing the technical specifications described in the API documentation where API reviewers give their feedback about the API design.
2. Empirical methods: the object of the study is focused on how developers use an API in practice.

Analytic methods can take two different approaches for evaluation: Metrics, where the API design is studied through evaluation software and the result is compared with predefined thresholds, or Reviews, where experts study the design of the API and its documentation.

Empirical methods, on the other hand, can be task-based usability tests where participants perform one or more predefined tasks using the API, controlled experiments which compare outcomes of different tasks using different APIs, or surveys and repository mining, which focus on the study of developer's past experiences using an API.

In this evaluation, we opted for conducting an empirical method using the task-based usability test approach, given that it provides us with users' feedback on a single existing and new API (we did not have different APIs to compare, and the proposed API is new without an existing user base).

After consolidating all the information, we developed the usability test based on tasks where the user will use methods provided by the developed API.

8.2.1 Procedure

For this usability test, we recruited nine participants, most of them students in the computer engineering department. The participants were contacted, some via direct contact, others via instant messaging applications. We mainly communicated via the internet during this usability test, explaining the entire procedure to each participant individually.

The first assignment to be completed would be the A-Frame tutorial, which we made available online at `https://github.com/ZeCanelha/AFrame-Tutorial` (see Appendix C.3). We ask the participants to complete the tutorial in no more than a week. If not possible, we asked them to mark the tutorial's completion date.

Afterward, we explained that the next assignment was to complete a set of tasks in which the A-Frame component was evaluated. We made the task list available for online assessment in a shared document and sent the link to each participant.

The evaluation consisted of four tasks. In each task, we asked the users to use the different methods and functionalities of the API. We design them to cover all aspects of the component usability. In A-Frame, the components are encapsulated JavaScript code modules that we can use declaratively in HTML as attributes of the elements [64]. Additionally, programmers can access the inherent methods of a component from any other A-Frame component. We intend the first two tasks to explore the HTML interface, whereas the last two explore its internal methods. We also invited the users to leave suggestions in the code as comments as they completed the tasks.

Finally, we requested the users to send their solutions through email and fill a questionnaire regarding usability, available in Appendix C.5.

Given the length of each task, we transcribe only the first task. However, to give the reader context for the results section, we describe each task with a summary of the objective and a figure illustrating the scenario under consideration. The tasks in their entirety are available in Appendix C.4.

Tasklist description

Task 1

Make changes in the example Task 5 provided, to match the following:

Use the Vibrotactile component to create a scene where there is vibration feedback

(the vibration is defined by the vibrations1.json available in the directory) when the cursor intercepts the toasted yellow <a-box> in the scene (Figure 8.8).

Since we do not have a vibration device to receive vibrotactile feedback, confirm the following output in the browser console:

*Vibrations from *vibration file* attached with success*

If any doubts arise, refer to the vibrotactile component documentation.

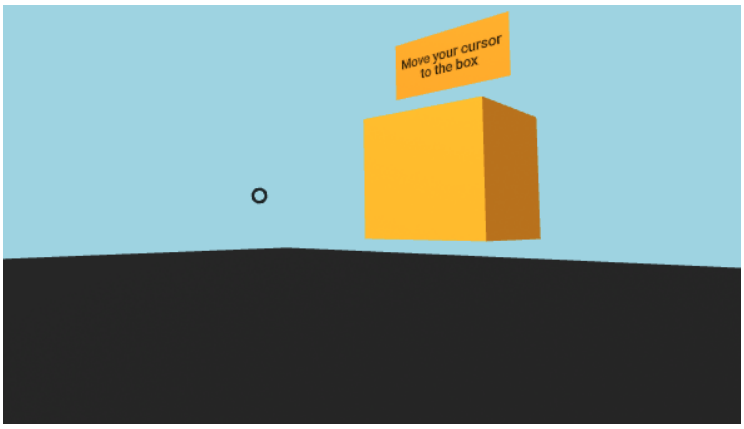


Figure 8.8: A-Frame Scenario - Task 1

In the **second task**, we asked the participants to continue exploring the HTML interface. In this task, the objective was to use two components simultaneously in the same entity. Specifically, we intended to associate a vibration and an animation component, and on animation complete, the vibration should trigger. We used an empty scenario with background only where the user had to define the entity in a given position.

For the **third task**, we requested the participants to use JavaScript methods of the vibrotactile component. In this task, we present a scenario with three entities (Figure 8.9). Each of these entities was to be associated with a vibration function explicit in the task statement. By doing this, we intend to explore the ability of users to use the API programmatically.

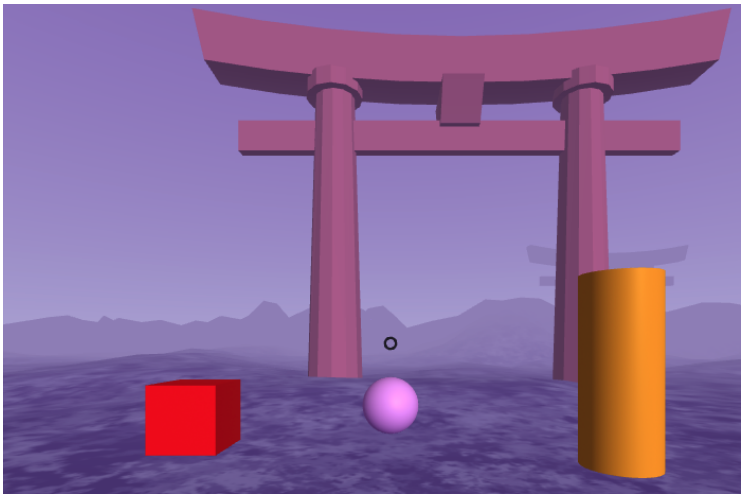


Figure 8.9: A-Frame Scenario - Task 3

Finally, in the **fourth task**, the participant was asked to create a Vibrotactile GPS. In this

task, the participant had to walk towards a given destination (Figure 8.10). As he/she was getting closer, distinct vibrotactile feedback was transmitted within predefined thresholds. In this task, we requested the participants more autonomous use of the A-Frame framework and extensive use of the component's API.

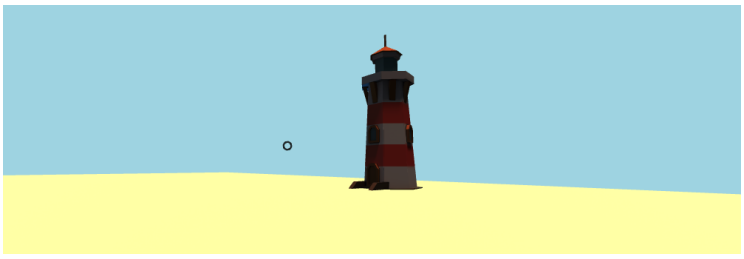


Figure 8.10: A-Frame Scenario - Task 4

Usability questionnaire description

The questionnaire was based on the structured interview presented by [107] and was as follows:

Questions regarding **understandability**:

1. Do you find that the API types map to the domain concepts in the way you expected?
2. Do you feel you had to keep track of information not represented by the API to solve the tasks?
3. Does the code required to solve the tasks match your expectations?

Questions regarding **abstraction**:

1. Do you find the API abstraction level appropriate to the tasks?
2. Did you need to adapt the API (inheriting from API classes, overriding default behaviors, providing non-API types) to meet your needs?
3. Do you feel you had to understand the underlying implementation to be able to use the API?

Questions regarding **learnability**:

1. Once you performed the first two tasks, was it easier to perform the remaining tasks?
2. Do you feel you had to learn many classes and dependencies to solve the tasks?

Questions regarding **readability**:

1. Do you find the API understandable, accessible, and readable?
2. Do you find yourself able to use the API logically?

Figure 8.11 summarizes the A-Frame component API usability test plan. The first phase was dedicated to planning. The second phase started with the tutorial and ended with the conclusion of the usability test. The third and fourth phase was dedicated to the analysis and discussion of the results obtained.

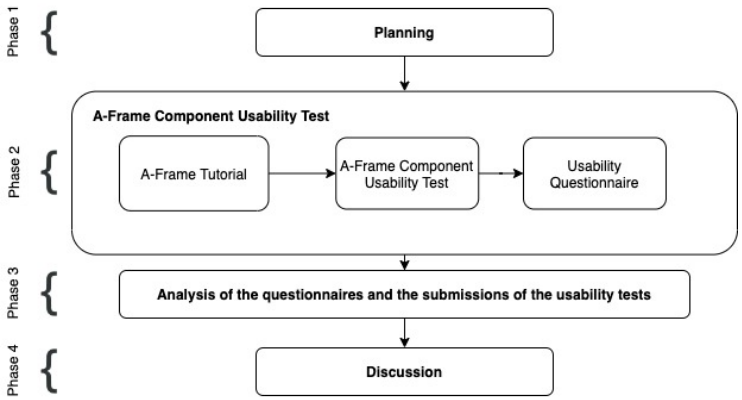


Figure 8.11: Methodology Approach

8.2.2 Results

Usability Test

As this usability test was conducted online, we indicate the expected output for the user to validate his / her answer in each task. In the table that follows, we have the results of the practice test. The first column relates to the actual task. The second column shows the expected output. Finally, in the third column, we indicate the number of users who completed the task.

Task	Expected Output	Nº of users who successfully completed the task
#1 Attach the vibrotactile component to an entity	*Vibrations from *vibration file* attached with success*	9
#2 Use the vibrotactile component with others	Vibrations from *vibration file* attached with success	9
#3 Use the inherent methods of the vibrotactile component programmatically	An object containing the method's name followed by the parameters specified in the task	8
#4 Vibrotactile GPS	An object containing the method's name followed by the parameters specified in the task	5

Table 8.4: A-Frame Component API usability tests results

Upon analyzing the submissions regarding the first task, we found this task to be relatively simple, and all users were successful in its execution. We intended to verify two key factors through this task: importing an external component to the A-Frame environment and the proper use and understanding of the Vibrotactile component.

Regarding the second task, all users have completed this task. In two of the codes submitted, we verified that the animation component parameters were not strictly the ones defined in the statement, which does not influence the expected result.

As we mentioned earlier, the first two tasks corresponded to using the component through the HTML interface. In the third task, we started the questions concerning the component's programmatic use. Reviewing the submitted codes, we verified that most users completed the task successfully. The errors found were mainly related to JavaScript itself rather than the use of the component. Of the two participants who did not complete this task correctly, the mistake is common. We wanted the vibrations to be activated according to specific interactions between the entities and the user. One of these interactions would be when

the user intercepted the entity with the cursor (the WebGL gaze reference point). Here, we expected the "mouseenter" event. However, the participant set as the event parameter the string "intercept," as visible in Figure 8.12, thus not activating the entity's vibration.

```
this.onIntercept = this.onIntercept.bind(this);
sphere.addEventListener("intercept", this.onIntercept);
```

Figure 8.12: JavaScript related error

Finally, in the last task, we noticed that a significant portion of the participants did not initiate the task's resolution at all. We are aware that the task required more effort and autonomy from the participant with the A-Frame. The conclusion we can draw from this might fall on the extent or the understanding of the requested task. The participants who did had some questions on particular aspects of the problem. One recurring doubt was how to calculate the distance between two objects in the A-Frame environment. Since this was a question slightly outside the evaluation context, we assisted the participants who asked. Given the feedback on this question, we evaluated it and concluded that the implications for assessing the component's use were practically non-existent. Other than this hindrance, these five participants used the requested methods correctly, with the proper parameters.

Usability Questionnaire

In this section, we analyzed the usability questionnaire results, available in Figure 8.13. The first group of questions is related to the API's **understandability**. The consensus among the participants was that the API was perceivable and fulfilled the participant's expectations. The second group concerns API's **learnability**. Based on the responses, we conclude that the participants' opinion is unanimous. After the API usage in the first tasks, further use becomes more effortless. Also, we can conclude the participants feel that it is not necessary to learn extra dependencies to use the API. Regarding the third question group, we evaluated the API's **abstraction**. We can safely assume that users felt that the level of abstraction was appropriate for completing the tasks. Besides, each of the methods' declaration and documentation was sufficient for the participants to understand how each function works without understanding the underlying implementation. Finally, the last group of questions concerns the **readability** of the API. The participants' consensus was that the API was accessible, readable, understandable, and above all, all participants could use the API logically.

8.2.3 Discussion

After analyzing the results obtained, we can objectively conclude that the developed component presents positive results. For this usability test, we took a different approach to ensure that the participants had the background knowledge and experience required through an introductory tutorial. This approach enabled the evaluation to be focused only on the API, avoiding the A-Frame's possible complexities.

However, the usability questionnaire results stand out somewhat compared to the results obtained from the usability test. In the first three tasks, we achieved excellent results. Yet, these results were to be expected, especially in the first and second tasks. We wrote the documentation for the vibrotactile component in such a manner that a practical example followed all information regarding a method. And in these two tasks, reading the

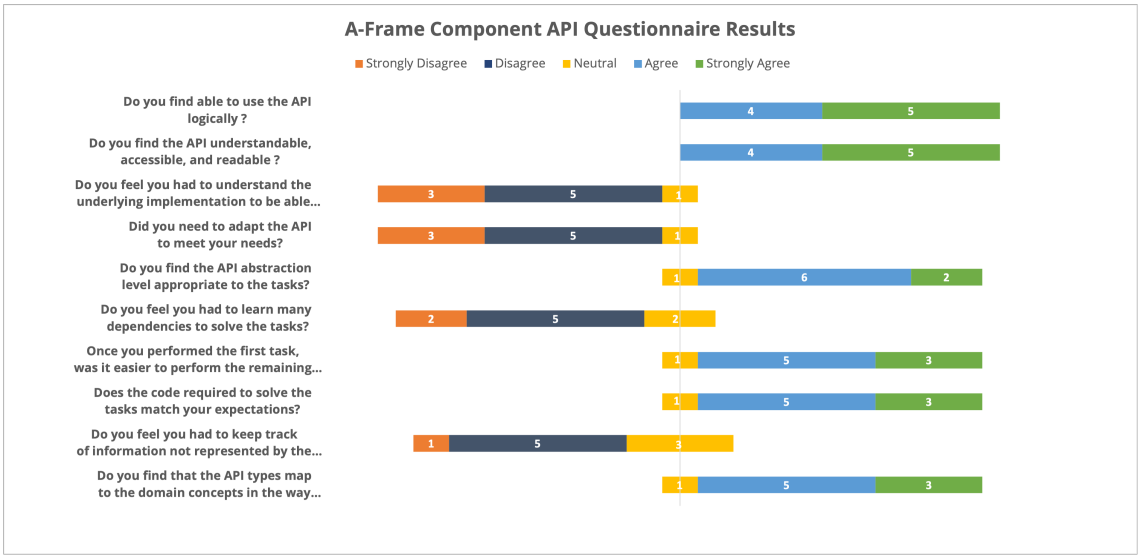


Figure 8.13: A-Frame component API usability questionnaire results

documentation and understanding the provided example was sufficient for solving the first tasks. In the last tasks, the resolution was not quite so linear. In the third task, we obtained good results. The participants understood what was asked of them and solved it accurately, with the correct methods and parameters. Although there were some errors, as we pointed out in the results section, these had no impact on the final result. During the component development, we had several iterations about the optimal way to state the parameters. This was one of the primary aspects we wanted to validate, given that each method takes a vast number of parameters.

Nevertheless, we cannot confidently state that this is proved entirely. The discrepancies between the results of the test and the usability questionnaire leave us somewhat uncertain. On the one hand, we have sufficient data to support that the results are positive overall. On the other hand, we cannot help but feel that the participants' disinterest in the last task is reflected in such positive results in the usability questionnaire. Possible explanations for the results obtained may relate to an attempt to help a colleague in his work, which in reality, turned out not to be helpful. We expected the participants to be fully engaged in the evaluation. Seeing that almost half chose not to perform the last task demonstrates that our preparation failed to captivate participants to explore the A-Frame and API in a challenge that would test their skills.

Chapter 9

Conclusion

To conclude this dissertation, we review all the work done during this academic year, detailing the primary contributions. Finally, we give an outlook on the future work that remains to be done.

9.1 Overview and Contributions

This dissertation was developed in the context of the TherTactExo project, which focuses on rehabilitating patients with spinal cord injuries. Through a new specific protocol involving virtual reality and feedback from several sensory stimuli, the main goal is to reduce the duration of a training protocol to achieve partial neurological recovery of sensorimotor function in SCI patients. Within these stimuli, we introduce the field of haptics, specifically, vibrotactile feedback. The work developed in this dissertation was intended to build an editor that would facilitate the creation of vibrotactile feedback.

We developed the Vibrotactile Editor and the primary contributions are:

- The creation of highly customizable vibrotactile patterns through a built-in waveform editor and the multi-channel timeline
- Provides a Middleware that enables the Virtual Reality components to create and express vibrotactile feedback
- Provides a simple, intuitive, and easy-to-use GUI to allow any user to create vibrotactile feedback

We tested our work in a study conducted with eight participants to evaluate and improve the editor. The usability test provided us an insight into how a user works with the Vibrotactile Editor. Besides, the participants' suggestions also proved to be a valuable contribution to the improvement of the editor. We incorporated all the results from the usability test into a second version of the Vibrotactile Editor and conducted a second round usability test. The results obtained were significantly better. We also received the validation and approval of the work developed by a TherTactExo team member, a future user of the Vibrotactile Editor.

In this dissertation, we present a set of functional tools for assistance in the TherTactExo project. The Vibrotactile Editor is already being used in the discrimination tests conducted at the moment. They are also considering using the Unity component to develop the virtual

reality experiments. However, this work is not restricted to the TherTactExo project exclusively. Other projects working with virtual reality and haptic feedback have, in this work, the necessary tools to support their developments.

9.2 Future Work

Future work on the Vibrotactile Editor System should focus on improving it further. One of the possible improvements is to adapt the nature of the feedback that we can create–for example, adapting the editor to create thermal feedback patterns. Other possibilities may address the different types of microcontrollers available on the market. As described in chapter 6, we developed the Vibrotactile Editor to meet a predefined communication protocol. Therefore, another improvement should be developing functionalities or establishing a protocol that enables integration with multiple microcontrollers.

Another point for improvement in future work is to upgrade the A-Frame component and its API. In the developed work, we introduced a set of functions to express vibrotactile feedback. However, it can be expanded to incorporate all sorts of waveforms. Re-running the A-Frame Component API usability tests to obtain more concrete results should be another point to consider in future work. In a similar direction, a usability test of the Unity component should also be conducted.

References

[1] Ana R. C. Donati, Solaiman Shokur, Edgard Morya, Debora S. F. Campos, Renan C. Moioli, Claudia M. Gitti, Patricia B. Augusto, Sandra Tripodi, Cristhiane G. Pires, Gislaine A. Pereira, Fabricio L. Brasil, Simone Gallo, Anthony A. Lin, Angelo K. Takigami, Maria A. Aratanha, Sanjay Joshi, Hannes Bleuler, Gordon Cheng, Alan Rudolph, and Miguel A. L. Nicolelis. Long-Term Training with a Brain-Machine Interface-Based Gait Protocol Induces Partial Neurological Recovery in Paraplegic Patients. *Scientific Reports*, 6(1):30383, 9 2016.

[2] C G. Virtual rehabilitation–benefits and challenges. *Methods Inf Med*, 42(February 2003), 2003.

[3] Shachar Maidenbaum, Ansh Patel, Elisabeth Stein, and Joshua Jacobs. Spatial Memory Rehabilitation in Virtual Reality - Extending findings from Epilepsy Patients to the General Population. *International Conference on Virtual Rehabilitation, ICVR*, 2019-July, 2019.

[4] Marta A. Montalbán and Oscar Arrogante. Rehabilitation through virtual reality therapy after a stroke: A literature review. *Revista Científica de la Sociedad de Enfermería Neurológica (English ed.)*, 52(C):19–27, 2020.

[5] Tamires Teixeira Gomes, Debora Stripari Schujmann, and Carolina Fu. Rehabilitation through virtual reality: Physical activity of patients admitted to the intensive care unit. *Revista Brasileira de Terapia Intensiva*, 31(4):456–463, 2019.

[6] Tyler Rose, Chang S. Nam, and Karen B. Chen. Immersion of virtual reality for rehabilitation - Review. *Applied Ergonomics*, 69(January):153–161, 2018.

[7] Mel Slater, Vasilis Linakis, Martin Usoh, and Rob Kooper. Immersion, presence and performance in virtual environments. pages 163–172, 1996.

[8] Steven C. Cramer, Mriganka Sur, Bruce H. Dobkin, Charles O'Brien, Terence D. Sanger, John Q. Trojanowski, Judith M. Rumsey, Ramona Hicks, Judy Cameron, Daofen Chen, Wen G. Chen, Leonardo G. Cohen, Christopher Decharms, Charles J. Duffy, Guinevere F. Eden, Eberhard E. Fetz, Rosemarie Filart, Michelle Freund, Steven J. Grant, Suzanne Haber, Peter W. Kalivas, Bryan Kolb, Arthur F. Kramer, Minda Lynch, Helen S. Mayberg, Patrick S. McQuillen, Ralph Nitkin, Alvaro Pascual-Leone, Patricia Reuter-Lorenz, Nicholas Schiff, Anu Sharma, Lana Shekim, Michael Stryker, Edith V. Sullivan, and Sophia Vinogradov. Harnessing neuroplasticity for clinical applications. *Brain*, 134(6):1591–1609, 2011.

[9] Central nervous system (cns) functions, parts, and locations. `https://www.emedicinehealth.com/anatomy_of_the_central_nervous_system/article_em.html`. [Online; accessed April-2020].

[10] Amanda Vitória Lacerda De Araújo, Jaqueline Freitas De Oliveira Neiva, Carlos Bandeira De Mello Monteiro, and Fernando Henrique Magalhães. Efficacy of Virtual Reality Rehabilitation after Spinal Cord Injury: A Systematic Review. *BioMed Research International*, 2019, 2019.

[11] N.J. Hill and J.R. Wolpaw. Brain–Computer Interface. In *Reference Module in Biomedical Sciences*. Elsevier, 1 2016.

[12] Mikhail A. Lebedev and Miguel A.L. Nicolelis. Brain–machine interfaces: past, present and future. *Trends in Neurosciences*, 29(9):536–546, 9 2006.

[13] R. Bousseta, I. El Ouakouak, M. Gharbi, and F. Regragui. EEG Based Brain Computer Interface for Controlling a Robot Arm Movement Through Thought. *IRBM*, 39(2):129–135, 4 2018.

[14] A beginner's guide to brain-computer interface and convolutional neural networks. `https://towardsdatascience.com/a-beginners-guide-to-brain-computer-interface-and-convolutional-neural-networks-9f35bd4a` [Online; accessed April-2020].

[15] L. Vega-Escobar, A.E. Castro-Ospina, and L. Duque-Munoz. Feature extraction schemes for BCI systems. In *2015 20th Symposium on Signal Processing, Images and Computer Vision (STSIVA)*, number September, pages 1–6. IEEE, 9 2015.

[16] Dennis J. McFarland and Jonathan R. Wolpaw. Brain-computer interfaces for communication and control. *Communications of the ACM*, 54(5):60–66, 2011.

[17] Vladimir Bostanov. BCI competition 2003 - Data sets Ib and IIb: Feature extraction from event-related brain potentials with the continuous wavelet transform and the t-value scalogram. *IEEE Transactions on Biomedical Engineering*, 51(6):1057–1061, 2004.

[18] Hubert Cecotti. Spelling with non-invasive Brain-Computer Interfaces - Current and future trends, 1 2011.

[19] Eeg (electroencephalogram). `https://www.mayoclinic.org/tests-procedures/eeg/about/pac-20393875`. [Online; accessed April-2020].

[20] What is magnetoencephalography (meg)? `http://ilabs.washington.edu/what-magnetoencephalography-meg`. [Online; accessed April-2020].

[21] Ralph S. Mosher. Handyman to Hardiman. In *SAE Technical Papers*, 2 1967.

[22] Joel C. Perry, Jacob Rosen, and Stephen Burns. Upper-limb powered exoskeleton design. *IEEE/ASME Transactions on Mechatronics*, 12(4):408–417, 2007.

[23] John Howard, Vladimir V. Murashov, Brian D. Lowe, and Ming Lun Lu. Industrial exoskeletons: Need for intervention effectiveness research. *American Journal of Industrial Medicine*, 63(3):201–208, 2020.

[24] Conor James Walsh, Daniel Paluska, Kenneth Pasch, William Grand, Andrew Valiente, and Hugh Herr. Development of a lightweight, underactuated exoskeleton for load-carrying augmentation. *Proceedings - IEEE International Conference on Robotics and Automation*, 2006(May):3485–3491, 2006.

[25] N. G. Tsagarakis and Darwin G. Caldwell. Development and control of a 'soft-actuated' exoskeleton for use in physiotherapy and training. *Autonomous Robots*, 15(1):21–33, 2003.

[26] Gregory S. Sawicki, Antoinette Domingo, and Daniel P. Ferris. The effects of powered ankle-foot orthoses on joint kinematics and muscle activation during walking in individuals with incomplete spinal cord injury. *Journal of NeuroEngineering and Rehabilitation*, 3, 2006.

[27] Evan A. Susanto, Raymond K.Y. Tong, Corinna Ockenfeld, and Newmen S.K. Ho. Efficacy of robot-assisted fingers training in chronic stroke survivors: A pilot randomized-controlled trial. *Journal of NeuroEngineering and Rehabilitation*, 12(1):18–21, 2015.

[28] Yang-Kun Ou, Yu-Lin Wang, Hua-Cheng Chang, and Chun-Chih Chen. Design and Development of a Wearable Exoskeleton System for Stroke Rehabilitation. *Healthcare*, 8(1):18, 2020.

[29] James Minogue and M. Gail Jones. Haptics in education: Exploring an untapped sensory modality. *Review of Educational Research*, 76(3):317–348, 2006.

[30] Lynette A. Jones. *Haptics*. The MIT Press, 2018.

[31] Mandayam A. Srinivasan and Cagatay Basdogan. Haptics in virtual environments: Taxonomy, research status, and challenges. *Computers and Graphics (Pergamon)*, 21(4):393–404, 1997.

[32] Febo Cincotti, Laura Kauhanen, Fabio Aloise, Tapio Palomäki, Nicholas Caporusso, Pasi Jylänki, Donatella Mattia, Fabio Babiloni, Gerolf Vanacker, Marnix Nuttin, Maria Grazia Marciani, and José Del R. Millán. Vibrotactile Feedback for Brain-Computer Interface Operation. *Computational Intelligence and Neuroscience*, 2007:1–12, 2007.

[33] Adding and improving haptics. `https://www.precisionmicrodrives.com/haptic-feedback/adding-and-improving-haptics/`. [Online; accessed April-2020].

[34] What is an actuator? `https://www.thomasnet.com/articles/pumps-valves-accessories/types-of-actuators/`. [Online; accessed April-2020].

[35] Precision Microdrives. Vibration motors. `https://www.precisionmicrodrives.com/vibration-motors/`. [Online; accessed 22-April-2020].

[36] Haptic actuators: Comparing piezo to erm and lra. `https://blog.piezo.com/haptic-actuators-comparing-piezo-erm-lra`. [Online; accessed 22-April-2020].

[37] Pwm. `https://www.arduino.cc/en/tutorial/PWM`. [Online; accessed 22-April-2020].

[38] NI. What is a pulse width modulation (pwm) signal and what is it used for? `https://knowledge.ni.com/KnowledgeArticleDetails?id=kA00Z0000019OkFSAU&l=pt-PT`. [Online; accessed 22-April-2020].

[39] Oliver S. Schneider and Karon E. MacLean. Studying design process and example use with Macaron, a web-based vibrotactile effect editor. *IEEE Haptics Symposium, HAPTICS*, 2016-April:52–58, 2016.

[40] Hasti Seifi, Kailun Zhang, and Karon E. MacLean. VibViz: Organizing, visualizing and navigating vibration libraries. *IEEE World Haptics Conference, WHC 2015*, pages 254–259, 2015.

[41] Jonghyun Ryu and Seungmoon Choi. PosVibEditor: Graphical authoring tool of vibrotactile patterns. *HAVE 2008 - IEEE International Workshop on Haptic Audio Visual Environments and Games Proceedings*, (October):120–125, 2008.

[42] M. J. Enriquez and K. E. MacLean. The hapticon editor: A tool in support of haptic communication research. *Proceedings - 11th Symposium on Haptic Interfaces for Virtual Environment and Teleoperator Systems, HAPTICS 2003*, pages 356–362, 2003.

[43] Jonatan Martínez, Arturo S. García, Miguel Oliver, José P. Molina, and Pascual González. VITAKI: A Vibrotactile Prototyping Toolkit for Virtual Reality and Video Games. *International Journal of Human-Computer Interaction*, 30(11):855–871, 2014.

[44] Graphical user interface. `https://www.omnisci.com/technical-glossary/graphical-user-interface`. [Online; accessed April-2020].

[45] What is cross-platform software. `https://medium.com/@hakanasal51/what-is-cross-platform-software-38ee57b7304a`. [Online; accessed April-2020].

[46] tkinter - python interface to tcl/tk. `https://docs.python.org/3/library/tkinter.html`. [Online; accessed 29-March-2020].

[47] Tkinter gui example. `https://pythonprogramming.net/static/images/tkinter/tkinter-gui-tutorial-with-python-3.png`. [Online; accessed 29-March-2020].

[48] kivy. `https://kivy.org/#home`. [Online; accessed 29-March-2020].

[49] Java swing tutorial. `https://www.javatpoint.com/java-swing`. [Online; accessed 29-March-2020].

[50] Java swing gui example. `https://media.geeksforgeeks.org/wp-content/uploads/20190828174750/Registration-Form-unfilled.png`. [Online; accessed 29-March-2020].

[51] Javafx. `https://openjfx.io/`. [Online; accessed 29-March-2020].

[52] Javafx - first app with modern graphical user interface. `https://steemit.com/utopian-io/@murez-nst/mureztutorial010`. [Online; accessed 29-March-2020].

[53] Electron. `https://www.electronjs.org/`. [Online; accessed 25-April-2020].

[54] Bootstrap - the most popular html, css, and js library. `https://getbootstrap.com/`. [Online; accessed 25-April-2020].

[55] Semantic ui. `https://semantic-ui.com/`. [Online; accessed 04-April-2020].

[56] Bulma: Free, open source, and modern css framework based on flexbox. `https://bulma.io/`. [Online; accessed 04-April-2020].

[57] Foundation. `https://get.foundation/`. [Online; accessed 04-April-2020].

[58] React – a javascript library for building user interfaces. `https://reactjs.org/`. [Online; accessed Aug-2020].

[59] Vue.js. `https://vuejs.org/`. [Online; accessed Aug-2020].

[60] Angular - one framework. mobile desktop. `https://angular.io/`. [Online; accessed Aug-2020].

[61] Arduino. `https://www.arduino.cc`. [Online; accessed 22-April-2020].

[62] Unity manual. `https://docs.unity3d.com/Manual`. [Online; accessed 15-March-2021].

[63] Unity - editor tools. `https://unity.com/pt/releases/2019-3/editor-tools`. [Online; accessed Aug-2020].

[64] A-frame. `https://aframe.io`. [Online; accessed March-2020].

[65] Predrag Matković and Pere Tumbas. A Comparative Overview of the Evolution of Software Development Models. *International Journal of Industrial Engineering and Management (IJIEM)*, 1(4):163–172, 2010.

[66] Marco Viera. Project managment - lifecycles, 2019.

[67] Sdlc - agile model. `https://www.tutorialspoint.com/sdlc/sdlc_agile_model.html`. [Online; accessed 04-April-2020].

[68] S.M. Avdoshin and E.Y. Pesotskaya. Software risk management. *2011 7th Central and Eastern European Software Engineering Conference (CEE-SECR), Software Engineering Conference in Russia (CEE-SECR), 2011 7th Central and Eastern European*, pages 1 – 6, 2011.

[69] Risk analysis and management. `https://www.pmi.org/learning/library/risk-analysis-project-management-7070`. [Online; accessed 04-April-2020].

[70] A. M. Hickey and A. M. Davis. Requirements elicitation and elicitation technique selection: model for two knowledge-intensive software development processes. In *36th Annual Hawaii International Conference on System Sciences, 2003. Proceedings of the*, pages 10 pp.–, Jan 2003.

[71] Moscow prioritization. `https://www.productplan.com/glossary/moscow-prioritization/`. [Online; accessed April-2020].

[72] C4 model. `https://c4model.com/`. [Online; accessed 12-June-2020].

[73] Node.js. `https://nodejs.dev`. [Online; accessed Aug-2020].

[74] Mongodb. `https://www.mongodb.com/`. [Online; accessed 11-March-2021].

[75] Https. `https://developers.google.com/search/docs/advanced/security/https`. [Online; accessed Aug-2020].

[76] Working with json. `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON`. [Online; accessed Aug-2020].

[77] Serialport. `https://www.computerhope.com/jargon/s/seriport.htm`, note="[Online; accessed Aug-2020]".

[78] Working with json. `https://developer.mozilla.org/en-US/docs/Learn/JavaScript/Objects/JSON`. [Online; accessed Mar-2021].

[79] A model view controller pattern for react. `https://blog.testdouble.com/posts/2019-11-04-react-mvc/`. [Online; accessed Mar-2021].

[80] Audacity. https://www.audacityteam.org/. [Online; accessed 11-March-2021].

[81] Sanchit Aggarwal. Modern Web-Development Using ReactJS | Document Object Model | Model–View–Controller. *International Journal of Recent Research Aspects*, 5(1):133–137, 2018.

[82] Redux framework. https://redux.js.org/. [Online; accessed 11-March-2021].

[83] React-redux flow, terminologies, and example. https://dev.to/bouhm/react-redux-flow-terminologies-and-example-104b. [Online; accessed 11-March-2021].

[84] An obsession with design patterns: Redux. https://engineering.zalando.com/posts/2016/08/design-patterns-redux.html?gh_src=22377bdd1us. [Online; accessed 11-March-2021].

[85] Fernando J. Barros. Design patterns.

[86] Data-driven documents. https://d3js.org/. [Online; accessed 11-March-2021].

[87] React bootstrap. https://react-bootstrap.github.io/. [Online; accessed 11-March-2021].

[88] Web audio api. https://developer.mozilla.org/en-US/docs/Web/API/Web_Audio_API. [Online; accessed 11-March-2021].

[89] Audio encoder. https://github.com/cstoquer/audio-encoder. [Online; accessed 11-March-2021].

[90] Filesaver.js. https://github.com/Infinidat/file-saver. [Online; accessed 11-March-2021].

[91] Yup. https://github.com/jquense/yup. [Online; accessed 11-March-2021].

[92] Formik. https://formik.org/. [Online; accessed 11-March-2021].

[93] Express framework. https://expressjs.com. [Online; accessed 11-March-2021].

[94] Filipe Portela and Ricardo Queirós. *Introdução ao desenvolvimento moderno para a Web*. FCA, 2018.

[95] Postman. https://www.postman.com/. [Online; accessed March-2020].

[96] Body parser. https://github.com/expressjs/body-parser. [Online; accessed 11-March-2021].

[97] Mongoose. https://mongoosejs.com/. [Online; accessed 11-March-2021].

[98] A-frame documentation. https://aframe.io/docs. [Online; accessed 15-March-2021].

[99] Jakob Nielsen. What Is Usability? *User Experience Re-Mastered*, (September):3–22, 2010.

[100] Ahmed Seffah, Mohammad Donyaee, Rex B. Kline, and Harkirat K. Padda. Usability measurement and metrics: A consolidated model. *Software Quality Journal*, 14(2):159–178, 2006.

[101] Usability test plan template. `https://www.usability.gov/how-to-and-tools/resources/templates/usability-test-plan-template.html`, September 2013. [Online; accessed 28-December-2020].

[102] Aaron Bangor, Philip T. Kortum, and James T. Miller. An empirical evaluation of the system usability scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

[103] The benefits of dark mode. `https://blog.weekdone.com/why-you-should-switch-on-dark-mode/`. [Online; accessed 11-March-2021].

[104] Dark theme. `https://material.io/design/color/dark-theme.html#properties`. [Online; accessed 11-March-2021].

[105] Irum Rauf, Elena Troubitsyna, and Ivan Porres. Systematic mapping study of API usability evaluation methods. *Computer Science Review*, 33:49–68, 2019.

[106] Luis López-Fernández, Boni García, Micael Gallego, and Francisco Gortázar. Designing and evaluating the usability of an API for real-time multimedia services in the Internet. *Multimedia Tools and Applications*, 76(12):14247–14304, 2017.

[107] Marco Piccioni, Carlo A. Furia, and Bertrand Meyer. An empirical study of API usability. *International Symposium on Empirical Software Engineering and Measurement*, pages 5–14, 2013.

This page is intentionally left blank.

Appendices

This page is intentionally left blank.

Appendix A

Requirements

A.1 Wireframes

In this section, we exhibit the wireframes designed during the requirements elicitation process.

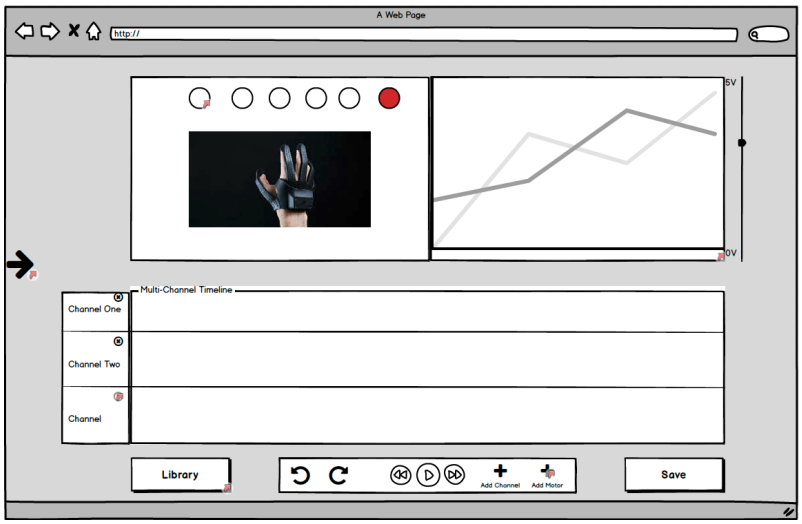


Figure A.1: Start screen

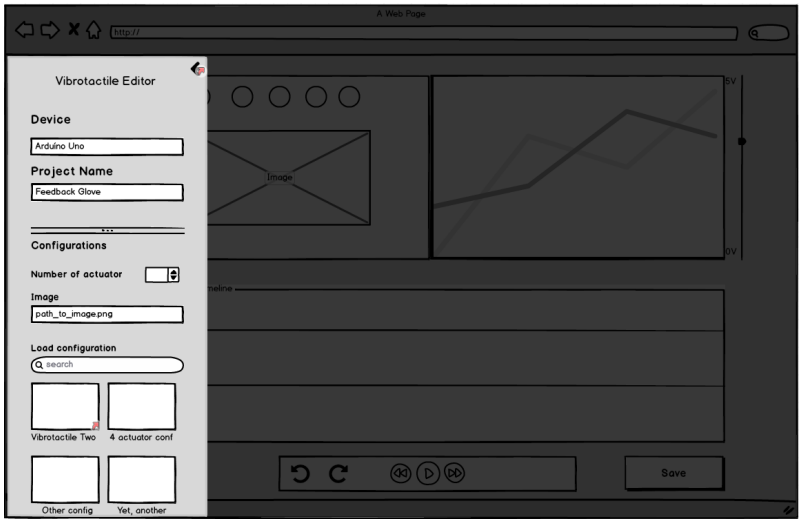


Figure A.2: Configuration screen

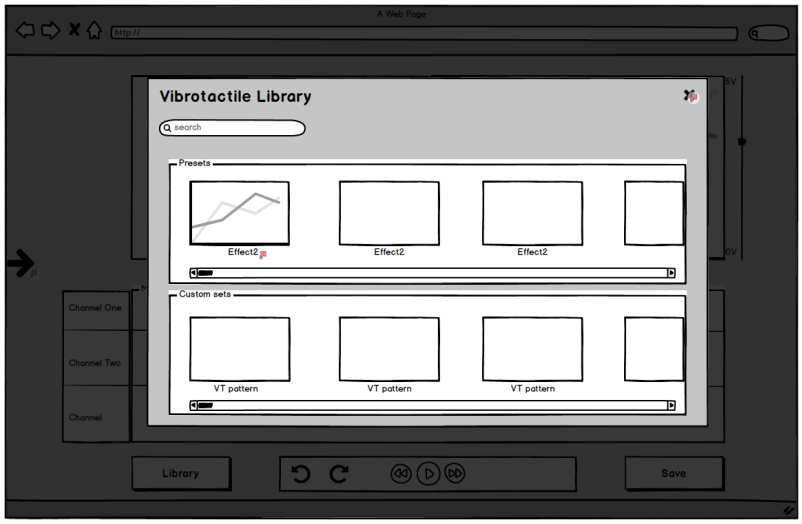


Figure A.3: Library screen

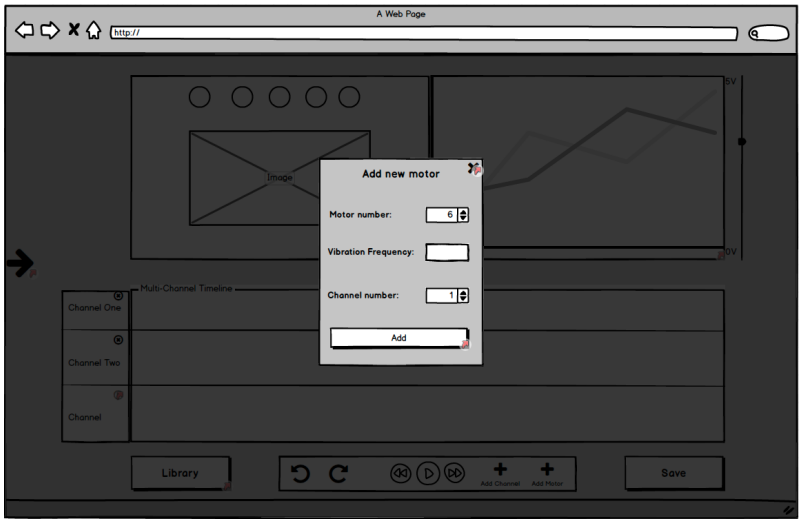


Figure A.4: Add actuator screen

Appendix B

The Vibrotactile Editor System

B.1 A-Frame Component

In this section, we detail all the documentation available about the developed A-Frame component, dubbed as A-Frame Vibrotactile Component.

aframe-vibrotactile-component

npm package not found | license package not found

A-Frame component for vibrotactile feedback

For A-Frame.

vibrotactile

When creating the **vibrotactile** component for A-Frame, we pass as an argument a file exported from [Vibrotactile Editor] (https://github.com/ZeCanelha/VibrotactileEditor. Briefly, this editor allows to create and export a set of vibrotactile sensations, which we refer in this documentation as the vibration file.

Property	Description	Default Value
src	Path to the vibration file	none
event	A-Frame event	none

Basic usage example:

```
<a-scene>
  <a-box vibrotactile="src: vibrations.json; event: mouseenter;"></a-box>
</a-scene>
```

See more examples in the examples folder.

The vibrotactile component offers a set of preset functions that can be used programmatically by the user. For simplicity the following table describes the common parameters between the **sin** and **ramp** functions.

Common parameters

Property	Description	Default Value
samplingRate	Sampling rate in milliseconds	`5`
numberOfActuators	Number of actuators	`6`
actuators	The specific actuators to perform the vibration	`[0,1,2,3,4,5]`
startingTime	Time in milliseconds to start the vibration	`0`
duration	Duration of the vibration in milliseconds	`1000`

vibrotactile.sin(sin, options)

The sin function allows to create a **sinusoidal waveform** vibration.

Property	Description	Default Value
sin	A JavaScrip object containing the sine wave properties	
sin.amplitude	Sine amplitude value between [0,1]	`1`
sin.frequency	Sine frequency value	`5`
sin.phase	Sine phase value	`0`
options	Common parameters	options

Example:

```
<head>
  <title>Vibrotactile Component - Sin example</title>
  <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-vibrotactile-component@1.0.0/dist/aframe-vibrotactile-component.min.js"></script>
  <script>
    AFRAME.registerComponent("vibrotactile-example", {
      init: function () {
        var sin = {
          amplitude: 0.8,
          frequency: 10,
        };
        var options = {
          samplingRate: 10,
          actuators: [0, 1],
          numberOfActuators: 6,
          startingTime: 0,
          duration: 1500,
        };
        var vibrotactile = this.el.components.vibrotactile;
        vibrotactile.sin(sin, options);
      },
    });
  </script>
</head>
<body>
  <a-scene>
    <a-entity vibrotactile vibrotactile-example></a-entity>
  </a-scene>
</body>
```

vibrotactile.ramp(initialIntensity, finalIntensity, options)

The ramp function allows to create a single sawtooth wave, a **ramp waveform** vibration.

Property	Description	Default Value
ramp	A JavaScript object containing the ramp wave properties	
ramp.initialIntensity	Initial vibration intensity value [0,100]	`0`
ramp.finalIntensity	Final vibration intensity value [0,100]	`100`
options	Common parameters	options

The ramp function is similar to the sin. Example:

```
<head>
  <title>Vibrotactile Component - Ramp example</title>
  <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-vibrotactile-component@1.0.0/dist/aframe-vibrotactile-component.min.js"></script>
  <script>
    AFRAME.registerComponent("vibrotactile-example", {
      init: function () {
        var ramp = {
          initialIntensity: 25,
          finalIntensity: 10,
        };
        var options = {
          samplingRate: 5,
          actuators: [3, 4, 5, 6],
          numberOfActuators: 6,
          startingTime: 0,
          duration: 500,
        };
        var vibrotactile = this.el.components.vibrotactile;
        vibrotactile.ramp(ramp, options);
      },
    });
  </script>
</head>
<body>
  <a-scene>
    <a-entity vibrotactile vibrotactile-example></a-entity>
  </a-scene>
</body>
```

vibrotactile.sendVibrations(vibrationFile)

The sendVibrations function allows the programmatic execution of a vibration file passed as a parameter. If no file is specified, the function will execute the file specified at component initialization.

Property	Description	Default Value
vibrationFile	Path to the vibration file exported from the Vibrotactile Editor	Vibration file specified at component initialization

Example:

```
<head>
  <title>Vibrotactile Component - Send vibrations example</title>
  <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-vibrotactile-component@1.0.0/dist/aframe-vibrotactile-component.min.js"></script>
  <script>
    AFRAME.registerComponent("vibrotactile-example", {
      init: function () {
        var waveVibration = "path/to/waveVibrationFile.json";
        var vibrotactile = document.querySelector(#box).components.vibrotactile;

        // Sends the vibrations specified at the src property
        vibrotactile.sendVibrations();
        // Sends the wave vibration file
        vibrotactile.sendVibrations(waveVibration);
      },
    });
  </script>
</head>
<body>
  <a-scene>
    <a-box
      id="box"
      vibrotactile="src: path/to/vibrationFile.json; event:mouseenter"
    ></a-box>
  </a-scene>
</body>
```

vibrotactile.customVibrations(vibrations, samplingRate, numberOfActuators)

The customVibrations function allows the construction of elaborated vibrations, close to the level achieved with the Editor, in a simple programmatic way.

Property	Description	Default Value
vibrations	JavaScript object containing the vibration properties	
vibration.intensity	Vibration intensity	-
vibration.actuators	Array of `Number` indicating the actuators	-
vibration.startingTime	Vibration starting time in milliseconds	-
vibration.duration	Vibration duration in milliseconds	-

Example:

```
<head>
  <title>Vibrotactile Component - Custom vibration example</title>
  <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-vibrotactile-component@1.0.0/dist/aframe-vibrotactile-component.min.js"></script>
  <script>
    AFRAME.registerComponent("vibrotactile-example", {
      init: function () {
        const samplingRate = 5;
        const numberOfActuators = 6;
        var vibrations = [];
        var vibrationObject = {
          actuators: [1, 2],
          intensity: 50,
          startingTime: 0,
          duration: 1000,
        };
        vibrations.push(vibrationObject);

        var vibrotactile = this.el.components.vibrotactile;
        vibrotactile.customVibrations(
          vibrations,
          samplingRate,
          numberOfActuators
        );
      },
    });
  </script>
</head>
<body>
  <a-scene>
    <a-entity vibrotactile vibrotactile-example></a-entity>
  </a-scene>
</body>
```

See more examples.

Installation

Browser Install and use by directly including thebrowser files:

```
<head>
  <title>My A-Frame Scene</title>
  <script src="https://aframe.io/releases/0.9.2/aframe.min.js"></script>
  <script src="https://unpkg.com/aframe-vibrotactile-component@1.0.0/dist/aframe-vibrotactile-component.min.js"></script>
</head>

<body>
  <a-scene>
    <a-entity
      vibrotactile="src: vibrations.json; event: click;"
      geometry="primitive: box;"
    ></a-entity>
  </a-scene>
</body>
```

npm

Install via npm:

```
npm install aframe-vibrotactile-component
```

Then require and use.

```
require("aframe");
require("aframe-vibrotactile-component");
```

This page is intentionally left blank.

Appendix C

Evaluation

C.1 SUS Questionnaire

In this section, we detail the SUS usability questionnaire provided the usability test participants.

Vibrotactile Editor - Questionnaire

*Obrigatório

1. Name *

_____

2. Date *

_____

Exemplo: 7 de janeiro de 2019

3. Gender *

Marcar apenas uma oval.

- ◯ Female
- ◯ Male
- ◯ Prefer not to say
- ◯ Outro: _____

4. Age *

Marcar apenas uma oval.

- ◯ Less than 18
- ◯ Between 18 and 22
- ◯ Between 23 and 30
- ◯ Between 31 and 45
- ◯ More than 45

System Usability Scale

5. I think that I would like to use Vibrotactile Editor frequently. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

6. I found Vibrotactile Editor unnecessarily complex. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

7. I thought Vibrotactile Editor was easy to use. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

8. I think that I would need the support of a technical person to be able to use Vibrotactile Editor *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

9. I found the various functions in Vibrotactile Editor were well integrated. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

10. I thought there was too much inconsistency in Vibrotactile Editor *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

11. I would imagine that most people would learn to use Vibrotactile Editor very quickly. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

12. I found Vibrotactile Editor very cumbersome (awkward) to use *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

13. I felt very confident using Vibrotactile Editor *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

14. I needed to learn a lot of things before I could get going with VibrotactileEditor. *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly disagree	◯	◯	◯	◯	◯	Strongly agree

C.2 Vibrotactile Editor Usability Testing - Tasklist

In this section, we detail the Vibrotactile Editor usability test tasklist.

Tasklist

- Task 1

  Create a new project with the name of your choice, where the device will be an Arduino Nano connected to 6 vibration actuators. Finally, upload the prototype image from the file system, available at Desktop/VTEditorTest/64-teste.jpg.

- Task 2 Add a new pattern to the timeline with the following characteristics:
  Four points, being the first coordinate the time and the second the intensity in percentage:
  - $\approx$ (0,20)
  - $\approx$ (100,50)
  - $\approx$ (200,75)
  - $\approx$ (300,90)
- Task 3 Over the prototype image, create a spatial arrangement of the actuators.
- Task 4 Associate an actuator to the channel containing the pattern created in Task 2.
- Task 5 Save the Project.
- Task 6 Add the "Spikes" pattern from the library to an empty timeline channel.
- Task 7 As in Task 4, associate an actuator to the Channel containing the pattern imported in the previous task.
- Task 8 Add a new channel to the project.
- Task 9 Add a new pattern to the newly created Channel.
  Edit the pattern to match the one in Figure C.1.

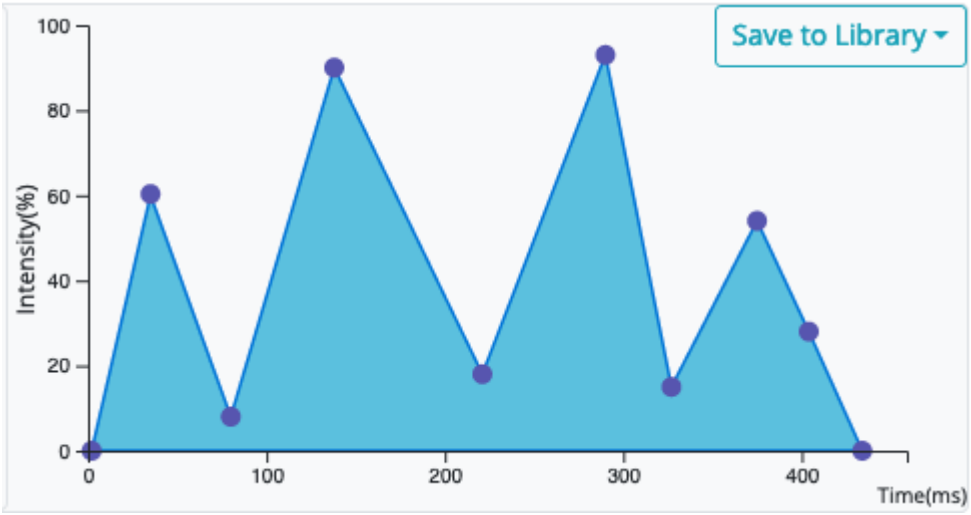


Figure C.1: Pattern Example

- Task 10 Save the edited pattern. Set the following characteristics:
  - Pattern Name: Up and Down
  - Description: Basic up and down pattern
  - Usage: Running
- Task 11 Search for the pattern created in the previous task and add it to the timeline.
- Task 12 Clear the timeline.
- Task 13 Create a timeline with the following characteristics:
  - Channel 1:
    1. Add pattern "Spikes" from the library.
    2. Set the start time at 0ms
    3. Associate actuators 1 and 2.
  - Channel 2:
    1. Create a pattern that matches the one on Figure C.2.
    2. Set the start time at 300ms
    3. Associate actuators 3 and 4.
  - Channel 3:
    1. Add pattern "Pyramid Pattern" from the Library
    2. Set the start time at 500ms
    3. Associate actuators 5 and 6.

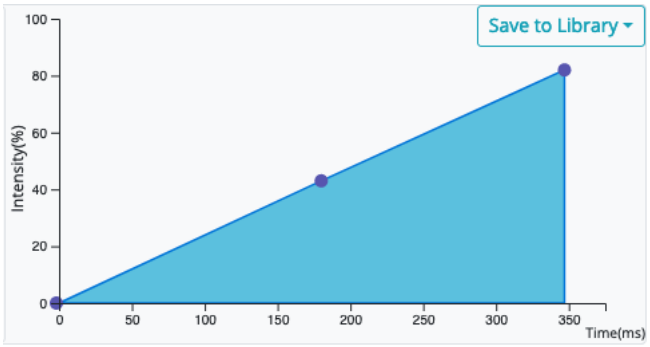


Figure C.2: Pattern Example

Play the timeline.

Obs: Place your arm over the hardware device before playing to feel the vibrotactile sensations created.

- Task 14 Add a 100ms interval between each pattern. Play the timeline.

C.3 A-Frame Tutorial

In this section we describe the A-Frame tutorial developed during the testing phase of the A-Frame Vibrotactile Component API.

A-Frame Tutorial

Introduction

A-Frame is a web framework for building virtual reality experiences using HTML and JavaScript. Therefore, to complete this tutorial, it is expected that the participants have minimal knowledge of these technologies.

Getting Started

A-Frame can be developed from a plain HTML file without having to install anything. To get started, create an `.html` file and include the A-Frame script in the `<head>` tag:

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.1.0/aframe.min.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box position="-1 0.5 -3" rotation="0 45 0" color="#4CC3D9"></a-box>
      <a-sphere position="0 1.25 -5" radius="1.25" color="#EF2D5E"></a-sphere>
      <a-cylinder
        position="1 0.75 -3"
        radius="0.5"
        height="1.5"
        color="#FFC65D"
      ></a-cylinder>
      <a-plane
        position="0 0 -4"
        rotation="-90 0 0"
        width="4"
        height="4"
        color="#7BC8A4"
      ></a-plane>
      <a-sky color="#ECECEC"></a-sky>
    </a-scene>
  </body>
</html>
```

Tutorial

All the a-frame action happens within the `a-scene` entity. In the example above, we have the scene tag with a handful of a-frame entities. These entities represented as HTML elements are called ***primitives***.

Primitives

Entities or primitives can be described as a 3D object which can take different geometric shapes, such as a box, sphere, or a cylinder. To see every primitive that A-Frame provides, refer to the A-Frame documentation, at the bottom of the documentation navigation sidebar.

For this tutorial, we will be focusing on the most basic primitives, detailed in the table below.

Primitive	Description
`<a-box>`	The box primitive creates shapes such as boxes, cube, or walls
`<a-cylinder>`	The cylinder primitive is used to create tubes and curved surfaces
`<a-sphere>`	The sphere primitive creates a spherical or polyhedron shapes
`<a-plane>`	The plane primitive is used to create flat surfaces
`<a-sky>`	The sky primitive adds a background color or 360° image to a scene

Components

If we use the primitives as presented in the table above, the created primitives appear in their default form. To change its attributes, we associate ***components*** with these entities such as **position**, **rotation**, **color**, and **scale**. In A-Frame, components have the same syntax as an HTML attribute:

```
<a-box position="0 0 -4" color="#FF0000"
```

The code above creates a red box in the position given by the (0, 0, -4) coordinates. A-Frame uses a right-handed coordinate system where the negative Z axis extends into the screen.

Position

The position component places entities at certain spots in 3D space. Position takes a coordinate value as three space-delimited numbers.

```
<a-sphere position="0 1 -1"></a-sphere>
```

A-Frame uses a right-handed coordinate system where the negative Z axis extends into the screen.

Rotation

The rotation component defines the orientation of an entity in degrees. It takes x, y, and z, as three space-delimited numbers indicating degrees of rotation.

```
<a-box rotation="45 90 180"></a-box>
```

Scale

The scale component defines a shrinking, stretching, or skewing transformation of an entity. It takes three scaling factors for the X, Y, and Z axes.

```
<a-plane scale="0.5 1 2"></a-plane>
```

Size

The unit of measurement used in A-Frame is the meter. To define the size in A-Frame we have different attributes depending on the object's geometry. The box primitive takes the **width**, **height**, and **depth** attributes to control the size whereas the sphere and circle take the **radius** attribute. A cylinder uses both the **height** and the **radius** attribute:

```
<a-box position="2 0 -5" width="1" height="2" depth="1" color="#FFAA00"></a-box>

<a-sphere position="-2 0 -5" radius="0.75" color="red"></a-sphere>

<a-cylinder
  position="0 0 -5"
  radius="1"
  height="1.5"
  color="#212121"
></a-cylinder>
```

Textures and 3D Models

In A-Frame, besides the color, we can also apply textures to the entities in our scene. The process of applying a texture to an entity is very straightforward and we can apply it to every entity in the scene. Additionally, A-Frame also provides components for loading 3D models, to further enrich the VR scene. A-Frame has an **asset management system** that allows us to place our assets in one place and to preload and cache assets for better performance. We will start here.

Asset Management System

The asset management system allow us to preload our assets, such as models and textures, before rendering the scene. This makes sure that assets aren't missing visually, and this is beneficial for performance to ensure scenes don't try to fetch assets while rendering. Assets include:

- `<a-asset-item>`
- `<audio>`
- `<img>`
- `<video>`

To set a texture to an entity, we need to specify the `src` property. `src` can be a selector to any element in the asset management system. It is customary in A-Frame to use ID selectors for assets: we assign an id to the asset in the asset management system, and then use an ID selector in the primitive we want to apply that asset in. For example, to set a simple texture to an `<a-box>` primitive, we simply load the texture to the asset management system, assign it an id of `my-texture` and specify the source in our primitive as being `#my-texture`.

```
<a-scene>
  <a-assets>
    <img id="my-texture" src="texture.png" />
  </a-assets>

  <a-box material="src: #my-texture"></a-entity>
</a-scene>
```

To load a 3D Model, the process is very similar. A-Frame recommends using glTF model. The `<gltf-model>` component loads a 3D model using a `.gltf` or `.glb` file

```
<a-scene background="color: #ECECEC">
  <a-assets>
    <a-asset-item
      id="cityModel"
      src="https://cdn.aframe.io/test-models/models/glTF-2.0/virtualcity/VC.gltf"
    ></a-asset-item>
  </a-assets>
  <a-entity gltf-model="#cityModel" modify-materials></a-entity>
</a-scene>
```

Animations

The animation component lets us animate and tween values including:

- Component values such as position, visibility, scale, and rotation.
- Component property values such as light, intensity, etc.

As example, below we have a translating box:

```
<a-box
  position="-1 1.6 -5"
  animation="property: position; to: 1 8 -10; dur: 2000; easing: linear; loop: true"
  color="tomato"
></a-box>
```

See more about the animation component in the A-Frame documentation

Entities

So far we have mentioned entities numerous times during this brief introduction to A-Frame, so what is the definition of an entity? In A-Frame entities are placeholders objects to which we plug in components to provide them appearance, behavior, and functionality.

We can attach components to it to make it render something or do something. To give it shape and appearance, we can attach the geometry and material components. The geometry component provides a basic shape fo an entity. The `primitive` property defines the general shape. The material component gives appearance to an entity. We can define properties such as color, opacity, or texture. Entities are inherently attached with the **position**, **rotation**, and **scale** components. Consider the following example:

```
<a-entity
  geometry="primitive: box"
  material="color: red"
  position="2 2 -10"
  scale="2 2 1"
>
</a-entity>
```

Refer to the A-Frame documentation to see further more information.

Creating and register a component

Components of A-Frame's are JavaScript modules that can be mixed, matched, and composed onto entities to build appearance, behavior, and functionality. We can register a new component in JavaScript and use it declaratively from the DOM. Components are configurable, reusable, and shareable. Most code in an A-Frame application should live within components [1].

To use a component, we first must define it before the `<a-scene>` tag, as example:

```
<html>
  <head>
    <!-- Import external component -->
    <script src="foo-component.js"></script>
  </head>
  <body>
    <script>
      // Or inline before the <a-scene>.
      AFRAME.registerComponent("bar", {
        // ...
      });
    </script>

    <a-scene> </a-scene>
  </body>
</html>
```

Let's have a first look to a basic component to get the general idea. This component will log a simple message once when the component's entity is attached using the `.init()` handler. But first, we need to **register** the component. Components are registered with `AFRAME.registerComponent()`. The first argument is the name of the component, which will be used as the HTML attribute name, and for this example it will be `hello-world`. The second is a JavaScript object of methods and properties. In the next example, we have our `init()` handler.

```
AFRAME.registerComponent('hello-world', {
  init: function () {
    console.log('Hello, World!');
  }
});
```

Then we can use our `hello-world` component declaratively as an HTML attribute. Do not forget to include the JavaScript file or declare inline before the `<a-scene>` tag.

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.1.0/aframe.min.js"></script>
    <!-- Import the hello-world component -->
    <script src="hello-world.js"></script>
  </head>
  <body>
    <!-- Attach the component to the scene -->
    <a-scene hello-world> </a-scene>
  </body>
</html>
```

To set a component, rather than via static HTML , is to set it programmatically with `.setAttribute()`. In the example bellow, we set the `hello-world` component on the scene programmatically:

```
document.querySelector('a-scene').setAttribute('hello-world', '');
```

Some components have methods available programmatically, and it is not possible to access them directly through HTML. For example, the Vibrotactile component has three internal methods only accessible through JavaScript. A component's methods can be accessed through the entity from the `.components` object. Consider this example:

```
AFRAME.registerComponent('foo', {
  init: function () {
    this.bar = 'baz';
  },

  qux: function () {
    // ...
  }
});
```

To access the `qux` method:

```
var fooComponent = document.querySelector('[foo]').components.foo;
fooComponent.qux();
```

We can query for elements containing a component with the attribute selector (i.e., [COMPONENT_NAME]), as used in the example above.

Refer to the A-Frame documentation for more information on creating and register a component.

Vibrotactile Component

In A-Frame we can use components created by the community. Here, we will use the vibrotactile component. This component sends vibrotactile feedback to the user (if (s)he is using a special vibrotactile device) when interacting with scene elements that have the component associated. A basic example of its usage:

```
<html>
  <head>
    <script src="https://aframe.io/releases/1.1.0/aframe.min.js"></script>
    <!-- Import the vibrotactile component -->
    <script src="vibrotactile.js"></script>
  </head>
  <body>
    <a-scene>
      <a-box vibrotactile="src: vibrations.json; event: mouseenter;"></a-box>
    </a-scene>
  </body>
</html>
```

The Vibrotactile component takes two arguments:

- A path to a vibration file exported from Vibrotactile Editor
- An A-Frame event which will trigger the vibration.

In the example above, when the mouse intersects with the box primitive, it will trigger the vibrations from the `vibrations.json` file.

Task List

Refer to the examples folder.

Task 1

Make changes in the example `Task1` provided to match the following:

- Change the sky primitive color to `#ADD8E6`
- Apply a x-axis rotation of 45º in the box primitive
- Change the y-axis position of the sphere to 2 meters.
- Create another box primitive and apply a scale transformation of `3 1 1`.

Navigate in the scene using the arrow keys or the "wasd" controls.

Task 2

Make changes in the example `Task2` provided to match the following:

- Apply one texture from the asset management system to the box primitive.
- Apply the "Gas Station" 3D Object Model to the empty entity

Task 3

Make changes in the example `Task3` provided to match the following:

- Apply a translation animation to the `<a-box>` primitive from its initial position to `"2 1.5 -10"`

Task 4

Make changes in the example `Task4` provided to match the following:

Using the `<a-entity>` element provide the following characteristics:

- Give the empty entity the shape and appearance of a yellow (`#FFF200`) cube.
- Position the yellow cube at `"2 1.5 -10"`
- Give to another empty entity the shape and appearance of a red (`#FF0000`) sphere.
- Position the red sphere at `"-2 1.5 -10"` and apply a scale transformation of `2 2 1`.

At this point, we have finished the first section of the A-Frame Tutorial. The main objective was to introduce the basic concepts of this framework. With the knowledge acquired we now want the participant to do two more tasks focused on the use of an external component, the vibrotactile component.

C.4 A-Frame Vibrotactile Component API Usability Testing - Tasklist

In this section we detail the A-Frame Vibrotactile Component API Usability Test tasklist.

A-Frame Component Usability Test

In this usability test, we want the participants to evaluate the usability of the A-Frame vibrotactile component. We are interested in evaluating the level of understandability, learnability, abstraction, and readability of the underlying API of the vibrotactile component.

To do so, we present four tasks where we suggest you create different scenarios in A-Frame. In each of these scenarios, we have different sub-tasks to use the various functionalities of the vibrotactile component.

As you perform these tasks, please leave comments in the code with suggestions on what you think could be improved in the component, even if minor things.

After completing the tasks, please fill out the questionnaire. At this point, we ask you to abstract from the possible complexity of the A-Frame framework and that you focus solely on the use of the vibrotactile component and its different functionalities.

Task List

When performing these, you should have already gone through the A-Frame Tutorial. The tutorial also has the sample code for the following tasks.

1. Make changes in the example Task 5 provided, to match the following:

   Use the Vibrotactile component to create a scene where there is vibration feedback (the vibration is defined by the *vibrations1.json* available in the directory) when the cursor intercepts the red <a-box> in the scene.

   Since we do not have a vibration device to receive vibrotactile feedback, confirm the following output in the browser console:

   *Vibrations from *vibration file* attached with success*

   If any doubts arise, refer to the vibrotactile component documentation.

2. Make changes in the example Task 6 provided, to match the following:

   Use the vibrotactile component to create a scene where an animated sphere with 0.5m of radius is translating from *"-2 0.5 2"* to *"2 0.5 2"* during 500ms. After the animation completes (look in the documentation for the Animation component for animation events) - it should vibrate using the vibrations defined by *vibrations4.json*, available in the folder.

   If any doubts arise, refer to the vibrotactile component documentation.

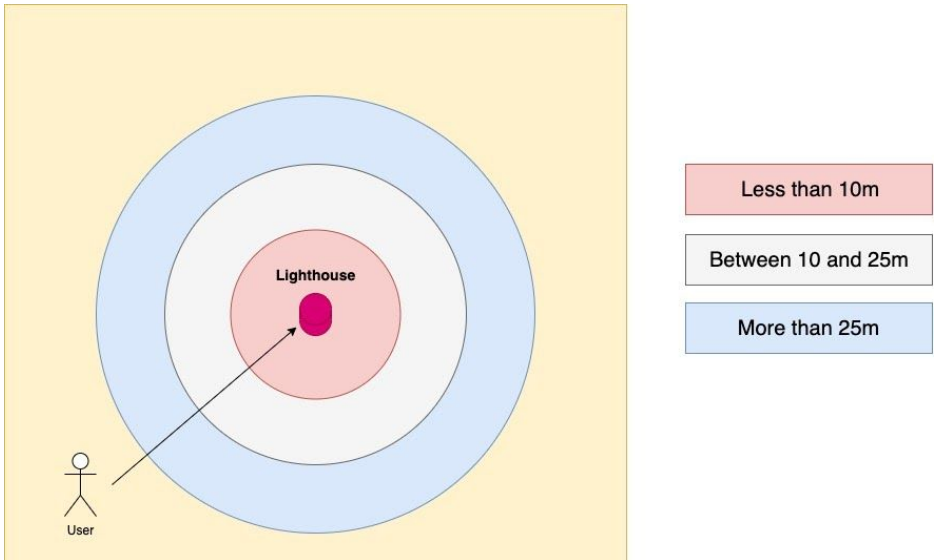
3. Make changes in the example Task 7 provided to create the following scenario:

   Write a simple A-Frame component to register the vibrations to the scene elements such that:
   a. The red box gives vibration feedback when the user clicks it. This vibration consists of a default sin vibration, with 500ms of duration.
   b. The sphere gives vibration feedback when intercepted by the cursor. Here, the vibration feedback should be expressed with a default ramp function with 200ms duration.
   c. Finally, the blue cylinder also gives feedback when clicked. The feedback should be expressed through a ramp vibration, with 10% as the initial intensity, and go up to 80%, with 250ms duration.

   If any doubts arise, refer to the vibrotactile component documentation.

4. Make changes in the example Task 8 provided to create the following scenario:

   Imagine you intend to go to a predefined location. We want you to create a Vibrotactile GPS that receives vibrotactile feedback depending on how far from the location the user is. In this task, we provide a beach scenario with a lighthouse. The objective is to reach the lighthouse, receiving feedback while walking towards it, as illustrated in Figure 1:



- When the user is in the light orange zone, there should be no vibration feedback.
- When the user is in the blue zone, it should receive a vibration with low intensity (25%), expressed in actuators 5 and 6m during 500ms.
- When the user is in the grey zone, it should receive a vibration with higher intensity (50%), expressed in actuators 3,4,5 and 6, during 750ms.
- Finally, when the user is in the red zone, it should receive a sin vibration with max intensity in all actuators, during 100ms.

If any doubts arise, refer to the vibrotactile component documentation.

5. After completing the tasks, we ask you to submit the code (to josemc@student.dei.uc.pt) as well as completing the questionnaire by clicking on the link below.

https://forms.gle/A574ubH4iBzPTGLS9

C.5 A-Frame Vibrotactile Component API Usability Form

Vibrotactile Component API

This questionnaire addresses the different characteristics of the vibrotactile component API.

We reinforce again that the focus is solely on the use of the component's different functionalities, its usability and understanding, while attempting to abstract from any complexity that might be the result of the A-Frame framework.

*Obrigatório

1. Name

_____

2. Age *

_____

3. Tutorial completion date *

_____

Exemplo: 7 de janeiro de 2019

4. Vibrotactile component usability test completion date *

_____

Exemplo: 7 de janeiro de 2019

5. In this usability test I: *

Marque todas que se aplicam.

- [ ] Agree and understand that the information gathered is for research purposes only and that will not be used for any other purposes.
- [ ] Understand that participation in this usability test is voluntary.

API Understandability

6. Do you find that the API types map to the domain concepts in the way you expected? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

7. Do you feel you had to keep track of information not represented by the API to solve the tasks? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

8. Does the code required to solve the tasks match your expectations? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

API Learnability

9. Once you performed the first task, was it easier to perform the remaining tasks / subtasks? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

10. Do you feel you had to learn many dependencies to solve the tasks? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

API Abstraction

11. Do you find the API abstraction level appropriate to the tasks? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

12. Did you need to adapt the API (inheriting from API classes, overriding default behaviors, providing non-API types) to meet your needs? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

13. Do you feel you had to understand the underlying implementation to be able to use the API?Questions *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

API Readability

14. Do you find the API understandable, accessible, and readable ? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree

15. Do you find able to use the API logically ? *

Marcar apenas uma oval.

	1	2	3	4	5	
Strongly Disagree	◯	◯	◯	◯	◯	Strongly Agree