Hardware and software toolchain for audio and haptics-based Virtual Reality

Aayush Shrestha
Dalhousie University, ay483379@dal.ca

1 INTRODUCTION

One of the key principles of Human-Computer Interaction is Iterative Design where usability concerns are incorporated from the start to make informed design decisions throughout the design process. Unfortunately, in practice, fixes for usability concerns perceived as affecting small populations (such as people with disabilities) are often tackled at the end of the software design lifecycle, resulting in sub-par user experiences [1]. Virtual Reality (VR) although mostly associated with visual phenomena and primarily described as a mediated abstraction of reality perceived visually, is often touted as a democratizing way to access new worlds and experiences [2]. Accessibility, however, thus far has not been a primary consideration during the development of contemporary commercial VR systems for a large diaspora of disabled users.

Building on the principle of accessibility-centric VR systems, this project report documents the work covered as a directed study over the length of a full academic term. The report outlines a thorough phase-wise incremental implementation of software and hardware tools to create a soundscape in VR through audio synthesis and haptics rendering. The imperative aim of this report is to reflect on the implementational achievements and lay a foundation for future work in the field of using VR as an assistive technology for visually impaired users, while also addressing the accessibility implications.

The manuscript begins with a literature review on existing soundscape paradigms, application techniques in VR, its prospects as an assistive technology, and accessibility-centric concerns. This review facilitated the prototype implementation developed in four progressive and sequential phases encompassing primary objectives, namely recording, modeling, and recreating tool-mediated texture contact vibrations and sounds. It also covers methodologies of audio synthesis using 'granular synthesis' and haptics rendering through the process of 'Pulse Width Modulation (PWM)' and their integration with the Unity game engine to create an outdoor navigational simulation. Moreover, the report incorporates the lessons learned throughout the directed studies and distills significant alterations to be implemented as future work.

2 LITERATURE REVIEW

VR technology has traditionally been associated with visual experiences and for most, it simulates spatial information through stereoscopic rendering presented through a Head Mounted Display (HMD) [3]. However, other sensory feedback mechanisms such as audio and haptics that constitute a more immersive VR experience have not been thoroughly leveraged. Thus, if the dominance of visuals in VR can be extricated through sensory substitution [4], it can be used as a capable assistive technology for visually impaired users. Current research focuses on achieving this through the creation of a soundscape in the virtual world [4] as well as through telepresence [5] which proposes an avenue for multimodal interactions with VR representations. Accessibility in VR for non-sighted users is a usability concern that traditional VR systems have been trying to

patch in order to establish a more inclusive experience. However, considerations at a later stage of the system design life cycle have proven to produce sub-par fixes to these usability concerns. Thus, the design process should inherently promote key inclusions to make the VR experience more accessible for non-sighted users as it helps even sighted users during situational disabilities [1], analogous to how a rising tide raises all boats [4]. Fostering empathy for a disability through the use of VR is an effective means of communicating a perspective of another individual's situation; however, challenges arising from misrepresentations of blindness simulations, lack of personal experience, and presence of misconceptions have led to fostering causal empathy and not long-term perspective shifts.

2.1 Generating Design Space in VR for Visually Impaired Users

The visual sensory bandwidth exceeds the bandwidth of haptics and hearing by far and therefore enables faster and more interactive types of locomotion [3]; however, related work in VR for users with visual impairments and/or low vision have mostly examined navigation techniques in indoor spaces with less focus on video spatial renderings and more on spatialized audio and texture mimicry through vibrotactile feedback mechanisms. Niklas Elmqvist in his article [4] terms this approach as sensory substitution which in the case of non-sighted users implies visualization only through the two practical options - touch or sound. Moreover, the article talks about shifting the idea of visual representation to spatial representations through the creation of a soundscape where the user exhibits navigational expertise through a mental map using echolocation and kinesthetic feedback from a cane tip and ambient environmental sounds in a simulated environment [4].

Microsoft Research's Canetroller [6] is an example that demonstrated how rendering virtual objects haptically, including simulating materials' properties and textures, could enable users who were completely blind to successfully navigate and understand virtual scenes when paired with a novel haptic controller that mimicked the interaction of a white cane [1]. A successor to this study [7] employed a multiple degrees-of-freedom cane enabling users to adapt the controller to their preferred techniques and grip along with a three-axis brake mechanism to emulate large shaped virtual objects. Additionally, as three-dimensional rendering, the main channel of information used in current virtual reality is unavailable to people who are blind, new haptic rendering technologies capable of detecting and displaying surface roughness in Telepresence [5] can also be explored as a means to convey virtual 3D objects to people who cannot see [1].

Jain et al. [8] in their research on sound accessibility in VR for Deaf and hard of hearing (DHH) users suggest a morphological search-based design space of input devices and information visualization. The design space includes both syntactic and semantic dimensions for sounds, visuals, and haptics. However, since the review concentrates on visually impaired users, vocabularies encompassing sound and haptic have been explained as viz.

2.1.1 Sound Vocabulary

The taxonomy of sound vocabulary covers two dimensions: source and intent. The "source" is the origin of the sound (e.g., from a character or an object) which is further categorized as localized and nonlocalized depending upon the sound emanating from an actual object/character in the VR world or from those playing in the background. The "intent" on the other hand is the impact of a sound on the users' experience that increases realism and/or conveys critical information.

2.1.2 Haptics Vocabulary

The taxonomy of the haptics vocabulary is articulated as having three feedback dimensions depending upon the delivery, location, and qualitative elements. The first dimension is the haptic form factor which describes the device on which the haptic feedback is delivered. It encompasses

Table 1: Sound Source Categories and Descriptions

Sound Source	Category	Description
Speech	Localized	Spatially positioned speech of a character
	Non-localized	Ambient speech of a narrator
Objects	Inanimate	Sounds from non-living objects
	Animate	Non-speech sounds from living beings
Interaction	-	Sounds from the interaction between multiple objects
Ambient	Point	Spatialized ambient sounds that belong to the VR world
	Surrounding	Non-spatialized ambient sounds
System	Notification	Sounds of critical alerts of specific events
	Music	Background music

a range of delivery mechanisms ranging from traditional VR controllers to non-conventional and emerging haptics-based commodity devices. The second dimension is the location of the body on which the haptic feedback is delivered called the haptic feedback location which includes prominent localized areas of palms, arms, torso, head-mounted, and legs. Finally, the haptics elements constitute three primary quality factors of a realistic haptic experience, namely (i) Intensity (amplitude/strength), (ii) Timbre (sharpness/pitch), and (iii) Rhythm (beats/interval between feedbacks).

2.2 Accessibility conundrums is VR

One of the key principles of Human-Computer Interaction is Iterative Design where usability concerns are incorporated from the start to make informed design decisions throughout the design process. Unfortunately, in practice, fixes for usability concerns perceived as affecting small populations (such as people with disabilities) are often tackled on at the end of the software design lifecycle, resulting in sub-par user experiences [1]. Accessibility has thus far not been a primary consideration during the development of mainstream VR systems. Mammot et al. introduce the concept of situational disability [1] where a normal user can become disabled at certain instances (e.g. When the hands are occupied holding groceries and unable to do any other task) thus emphasizing the concept of considering accessibility as a core part of a VR system's design process. Although contemporary commercial VR systems seem deficit in incorporating accessible mediums for a large diaspora of disabled users, in this literature review, the accessibility concerns are bluntly aimed towards the visually impaired population.

2.2.1 Visual Dependency

VR experiences heavily rely on visual stimuli, making them inherently challenging for individuals with visual impairments. The immersive nature of VR relies on visual graphics, spatial awareness, and visual cues, which can create significant barriers for those who are blind or have low vision. Lack of visual information limits their ability to perceive and interact with the virtual environment. VR environments can be disorienting, especially without visual cues for navigation

Table 2: Jain et al. [8] Categories of Sound Intents

Sound Intent for	**Description**
Conveying critical information	All sounds that are critical for progression in an app
Increasing realism	Ambient or object sounds that increase immersion
Rhythm or movement	Sounds that correlate to or enhance a particular user/object/actional movement
Generating an affective state	Emotional sounds with varying level intonations
Aesthetics or decoration	Non-critical sounds that increase the beauty
Non-critical interaction	Interaction sounds that are not critical to game progression

and orientation. Visually impaired individuals may struggle to understand the layout, identify objects, or navigate within the virtual space. Developing effective auditory or haptic navigation aids and spatial awareness techniques is essential to provide a sense of direction and orientation [9].

2.2.2 Lack of non-visual feedback

VR typically lacks alternative modalities or non-visual feedback to compensate for the absence of visual cues. While some VR applications may include audio cues, haptic feedback, or speech synthesis, their implementation is often limited, inconsistent, or insufficiently utilized. Providing meaningful and comprehensive non-visual feedback becomes crucial to conveying information and enabling participation for visually impaired users [10][3].

2.2.3 Key Areas of Accessibility Concerns in VR

Existing literature points out overlapping themes of concern for accessibility in VR for visually impaired users that revolve around standards of content, interaction modality, and device usability. The papers used in this review that highlight these areas are congruently grouped up in the following Table 3.

- *Content Accessibility*

  One of the key challenges in traditional VR systems is to appropriate representations as per users' abilities. For many assistive VR systems targeting non-sighted users, the audio information conveyed by most VR content is not enough to fill in for the visual cues that seeing users experience. This is mainly due to the lack of definition of metadata necessary to allow multimodal representations [1]. Attempts to establish a standard representation of these data include transforming visual representations into spatial representations through the process of sensory representations [4] along with introducing metadata that encodes objects' haptic properties, such as their materials and textures, for creating realistic haptic renderings [1].

- *Interaction Accessibility*

  VR has the potential to create a "level playing field," providing spaces in which all users may be equal in their capabilities. Due to this very reason, it is necessary that VR systems and

Table 3: Literature Sharing Congruent Areas of Accessibility Concerns in VR

Paper	Content Accessibility	Interaction Accessibility	Device Accessibility
Mamott et al. [1]	✓	✓	✓
Jain et al. [8]	✓		
Philips et al. [2]	✓	✓	✓
Elmqvist et al. [4]	✓	✓	
Williams et al. [9]			✓
Zhao et al. [6]	✓		
Kreimeier et al. [3]		✓	

their representations account for end-user diversity within their design to be comfortable for and usable by a large audience. Thus, for the interaction in VR to be inclusive, two main areas of focus comprise accessible inputs and user interfaces [1]. The input in VR systems is typically through hand-held and one-size-fits-all controllers that assume users have an articulate range of motion. Unlike interaction with stationary or mobile devices, this implicit dictation of users' input actions offers limited to no support for direct input apart from 3D controller motions in mid-air. Thus, there is an opportunity to design VR interfaces to support direct 3D input as well as input through alternative modalities, such as voice and gaze [1].

- *Device Accessibility*

  VR hardware typically makes many assumptions about users' abilities which can lead to accessibility problems [1] more specifically when the challenges are due to motor skills [2]. The physical interfaces of VR systems, such as controllers and head-mounted displays, may not be designed with accessibility in mind and often rely on visual feedback or complex button configurations, making them challenging to use for individuals with visual impairments. Incremental changes in the design of this hardware that aim to reduce the number of individual components and introduce more flexible and ergonomic types of equipment prioritizing non-visual interaction methods are necessary to ensure equal access to VR experiences [9][1].

2.3 VR for fostering empathy

VR is often revered as an effective means of promoting empathy as the system enables the simulation of an actual experience for people to better understand the perils and perspectives of another individual's situation. For simulations that focus on users experiencing a disability, the primary agenda is to foster some kind of empathetic concern and desire to help accommodate people belonging to the group. However, current literature suggests that such kind of disability simulation comes short in one way or another, misleading realities of a disability which can contribute to paternalistic discrimination [11]. Moreover, empathy that is fostered through these simulations is found to arouse compassionate feelings but not necessarily encourage users to imagine other peoples' perspectives [12].

In terms of disability simulation concerning vision, no ample research has been conducted that highlights the significant implications of post-simulation empathy. However, the present literature

on studies concerning VR simulating blindness simulation presents the following challenges.

2.3.1 Lack of Personal Experience

VR experiences aimed at fostering empathy for blind individuals require sighted users to understand and embody the perspective of blindness. However, sighted individuals naturally lack personal experience and understanding of the daily challenges faced by blind people. This knowledge gap can make it difficult to develop accurate and meaningful VR simulations that effectively convey the actual blind experience. Studies [12][7] have shown that following the simulations, empathetic concern (warmth) toward disabled people increased with certain boundary conditions to this effect; however, attitudes about interacting did not improve. Moreover, in some conditions, VR was no more effective at increasing empathy than less technologically advanced empathy interventions such as reading about others and imagining their experiences [12].

2.3.2 Misrepresenting Visual Impairment

Creating a realistic representation of visual impairment in VR poses a challenge. Blindness is not merely the absence of vision; it encompasses a range of visual impairments with different degrees and types of vision loss. Designing visual effects or simulations that accurately depict various types of blindness is a complex task that requires careful consideration that can also mislead people about blindness because it highlights the initial trauma of becoming blind rather than the realities of being blind [11].

2.3.3 Stereotypes and Misconceptions

When designing VR experiences to foster empathy, it is crucial to avoid reinforcing stereotypes or perpetuating misconceptions about blindness [9]. Misrepresenting blind individuals as helpless or dependent can undermine the goal of promoting empathy and understanding. Redmond et al. [7] found that with a number of participants in disability simulations, most of the participants felt more confused, embarrassed, helpless, and more vulnerable to becoming disabled themselves compared to baseline which in itself contradicts the idea of empathy generation.

3 IMPLEMENTATION

3.1 Phase 1: Data Collection and Processing

In this phase of the implementation, raw data was collected in the form of acceleration and audio recordings of the white cane interaction with different outdoor surface textures, namely sidewalks, asphalt/concrete roads, grass lanes, and gravel pathways. The main reason for choosing these surfaces is based on the ultimate objective of creating a VR game that simulates the outdoor navigation of a primary character in a cityscape using a white cane where these surfaces are primarily encountered.

3.1.1 Data Collection: Acceleration and Audio

For the collection of acceleration data, a custom data collection system (Fig. 1) is created using a pair of digital three-axis accelerometers controlled by a microcontroller and attached to a standard white cane where data is captured on a PC through a serial channel. For the collection of audio recordings, a digital microphone was mounted on a tripod and placed near the contact point of the cane and surface. The accelerometer sensors captured high-frequency accelerations whereas the microphone captured audio generated when the white cane interacts with various surface textures in a double-swiping motion over the span of ˜2 sec. Each surface was manually scanned

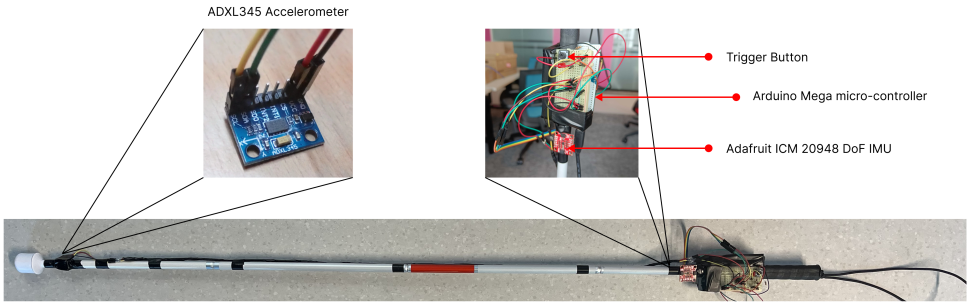

Figure 1: Acceleration data collection device using a standard white cane

(a) (b)

Figure 2: (a) Acceleration data collection of a white cane's interaction with the Sidewalk's texture (b) Zoom H5 Audio recorder used to collect interaction sound

six times by the experimenter (Fig. 2a) while maintaining a constant swipe speed and force resulting in a total of 12 swipes worth of acceleration and audio data set. The double-swiping motion was followed as a standard technique as it is commonly used by visually impaired people during their Orientation and Mobility (OEM) training [13].

High-frequency vibrations were captured using two accelerometer sensors: an ADXL345 three-axis digital MEMS-based accelerometer and a SparkFun ICM-20948 9DoF Inertial Measuring Unit (IMU) affixed to the white cane near the tip and the handle respectively. These chips were selected for their ability to read accelerations along three Cartesian axes, their high bandwidth, and their configurable range of up to $\pm 157, \mathrm{m/s^2}$ $(\pm 16, g)$. An Arduino Mega microcontroller was used to establish a serial interface, connecting both accelerometer sensors to the PC and transferring data. To reduce redundant data collection when the white cane was not in contact with any surface, a trigger button was configured. When pressed, the trigger button initiated the data collection sequence, and it terminated when the button was released.

A custom data collection program *(see Appendix A, Listing 2)* gathers acceleration data from the ADXL and IMU sensors using an Arduino C Library whereas a PC-based application named CoolTerm is used to interface the serial connection that also stores the set of recordings into a txt file after each interaction. Upon startup and until the duration of the trigger button being pressed, the program configures both the accelerometer sensors into $\pm 157\,\mathrm{m/s^2}$ $(\pm 16\,g)$ mode with a sampling rate of 200Hz.

For collecting the audio recordings, a Zoom H5 Digital Multitrack Portable Audio Recorder

(Fig. 2b) was used that uses 2 channel microphone inputs configured to store audio in 24-bit/96kHz Waveform Audio File (WAV) Format.

3.1.2 Data Processing

- Audio Segmentation

  Once the interaction audio of the white cane with different surfaces was recorded, each surface had a set of four files each with a couple of bi-directional sounds. To make sense of the recorded audio, Audacity, a powerful audio editing tool was used to divide the larger audio files into smaller, more manageable segments. Each of these segments focused on a single cane swipe, preserving the unique auditory characteristics of each interaction between the cane and a surface. The goal of segmenting the audio in such detail was to set the stage for the next phase - audio synthesis. By having these well-defined audio snippets, an infinite number of interaction sounds can be produced for a single surface leveraged through granular synthesis, discussed in detail in phase 2 of the implementation.

- DFT321 Algorithm

  Numerous researchers have highlighted that human sensitivity to high-frequency vibrations remains largely unaltered by their direction [14]. Furthermore, devising a high-frequency vibration haptic device with just a single output axis is significantly simpler compared to the complexities of crafting a device with three output axes. These factors prove crucial for condensing recorded vibration data, originally captured across three-dimensional Cartesian directions, into a singular dimension using the DFT321 algorithm [14]. The primary objective of this compression is to generate a vibration signal that is nearly indistinguishable from the original three-dimensional signal and effective in capturing both the spectral energy and temporal information from all three axes, as demonstrated in a sample recording showcased in Fig. 3.

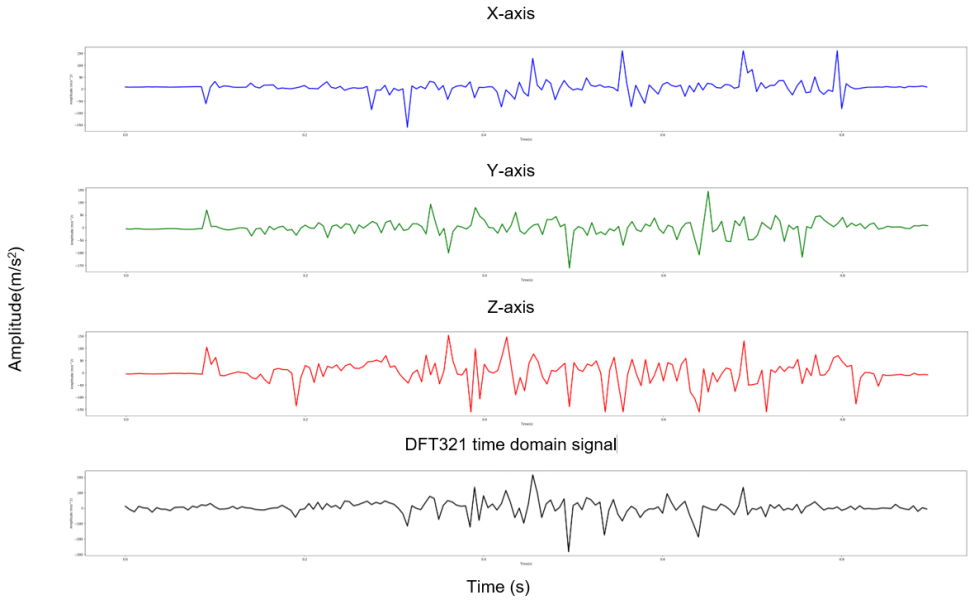

Figure 3: Three-axis time-domain signals are reduced to one-axis using DFT321. The depicted signals were recorded during an interaction of the white cane with a sidewalk

The DFT321 technique employs frequency-domain methodologies to unify the initial three signals into a cohesive signal while upholding the overall spectral power [14]. Initially, the

squared sum and square root of the magnitudes of the smoothed Discrete Fourier Transforms (DFTs) are calculated for each of the original three signals:

$$|\tilde{A}_s(f)| = \sqrt{|\tilde{A}_x(f)|^2 + |\tilde{A}_y(f)|^2 + |\tilde{A}_z(f)|^2} \tag{1}$$

Here $|\tilde{A}_s(f)|$ is the frequency-domain magnitude of the new DFT321 signal and $\tilde{A}_x(f)$,$\tilde{A}_y(f)$, and $\tilde{A}_z(f)$ are the DFTs of each original Cartesian acceleration vectors. The resultant DFT321 signal is established by computing the inverse tangent of the sum of imaginary parts divided by the sum of real parts of the DFTs of the original signals, resulting in an average phase $\phi(f)$:

$$\phi(f) = \tan^{-1}\left(\frac{\mathrm{Im}(\tilde{A}_x(f) + \tilde{A}_y(f) + \tilde{A}_z(f))}{\mathrm{Re}(\tilde{A}_x(f) + \tilde{A}_y(f) + \tilde{A}_z(f))}\right) \tag{2}$$

Subsequently, an inverse DFT operation is carried out on the calculated magnitude and phase to create the new time-domain signal. This algorithm is implemented in Python using the standard Numerical Python (NumPy) and Scientific Python (SciPy) libraries *(see Appendix A, Listing 3)*.

3.2 Phase 2: Audio Synthesis

Conventional methods of producing sound in VR games, often involving the playback of pre-recorded samples, face limitations when it comes to accommodating the various sound variations required for depicting diverse interactions between in-game objects. This challenge necessitates the use of model-based sound synthesis techniques capable of generating numerous unique sound instances without relying heavily on additional sound samples [15]. For the purpose of our game, the sounds emerging from interactions are typically non-musical, often referred to as 'environmental' sounds encompassing actions like scraping, rolling, and bouncing, among others. Hence, preserving the quality of these sounds is crucial for creating a sense of realism and immersion within the gaming experience. Among various available sound synthesis techniques, granular synthesis was chosen for our cause due to its infinite process of creating ample variations of a given sound file while preserving complex sound textures and characteristics [15].

3.2.1 Granular synthesis

As illustrated in Fig. 4, granular synthesis operates by replaying tiny segments of an audio file, known as "grains," while applying an amplitude envelope through a process called "windowing." These grains are then combined in the final output. The intervals between each grain's playback, the length of time for which a grain is audible, the rate at which the original file is read, and the specific position within the source file are all managed separately. As a result, even from a solitary source file, an extensive range of diverse sounds can be produced.

For the process of granular synthesis, Audiokinetic's Wwise synthesizer with Soundseed Grain plug-in was used. This granular synthesis tool has a dedicated filter per grain and 3D spatialization capabilities. All of these parameters can be modulated or randomized, using Real Time Controlled Parameters (RTPCs) and/or the embedded grain modulators, ultimately enabling the generation of an endless parameterized array of novel sounds.

3.2.2 Wwise Granular synth integration with Unity

Before the integration of the Wwise synthesizer and the Unity game engine, the segmented audio files for respective surfaces from the second phase of the implementation were created as sound

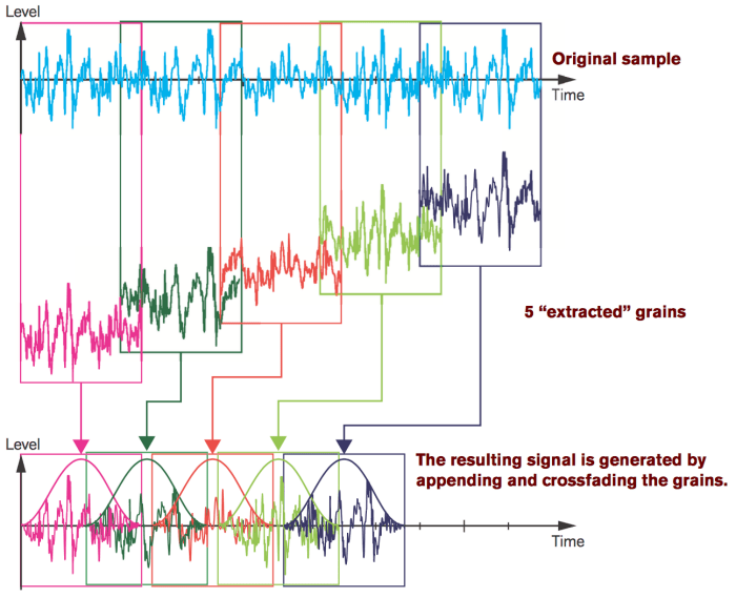

Figure 4: General concept of granular synthesis [16]

banks in the synthesizer. The synthesizer configuration consists of a collection of random containers each with a particular sound bank of the interaction surface that randomizes audio file selection from the banks during run-time. These random containers are controlled by a container switch that governs the initiation and switching between the containers when triggered by a particular event. Any audio file that is randomly selected by the random container is then fed into the granular synthesizer plugin. The pitch and volume are the two attributes of the granular synthesizer plugin that are exposed as RTPCs whose values are provided by Unity. The plugin then outputs the granularly synthesized sound using the connected audio output system. The integration points for the synthesizer and Unity thus become the plugin's RTPCs, each of the random containers, and the container switch.

In the Unity game engine, the two primary constituents that are crucial points of contact for its integration with the Wwise synthesizer are the surface box colliders and the sound player script *(see Appendix A, Listing 5)*. The box colliders are game components that represent invisible spatial boundaries of a particular game object; in this case - the virtual surfaces (road, sidewalks and etc). The box colliders are hence linked with the respective random container and the plugin's RTPCs. Whenever the white cane game object in the game collides with any of these box colliders, a particular event is triggered that activates the container switch in the synthesizer. The switch then decides which container to select depending on the box collider's linkage with the respective random container. Concurrently, the box collider also calls the sound player script that in return sends the pitch and volume values depending upon the white cane game object's interaction speed calculated using a lerp function. This whole process is repeated for each frame of the game until the white cane game object is no longer in contact with any of the box colliders. This whole process is pictorially represented as a block diagram in Fig. 5.

3.3 Phase 3: Haptics Profile Rendering

3.3.1 Rendering Hardware

For rendering the haptics profile, a custom-built hardware assembly making use of vibrotactile transducers (haptuators) is attached to the white cane. The hardware assembly is controlled using an Arduino Nano microcontroller that is serially interfaced to the PC for data transfer

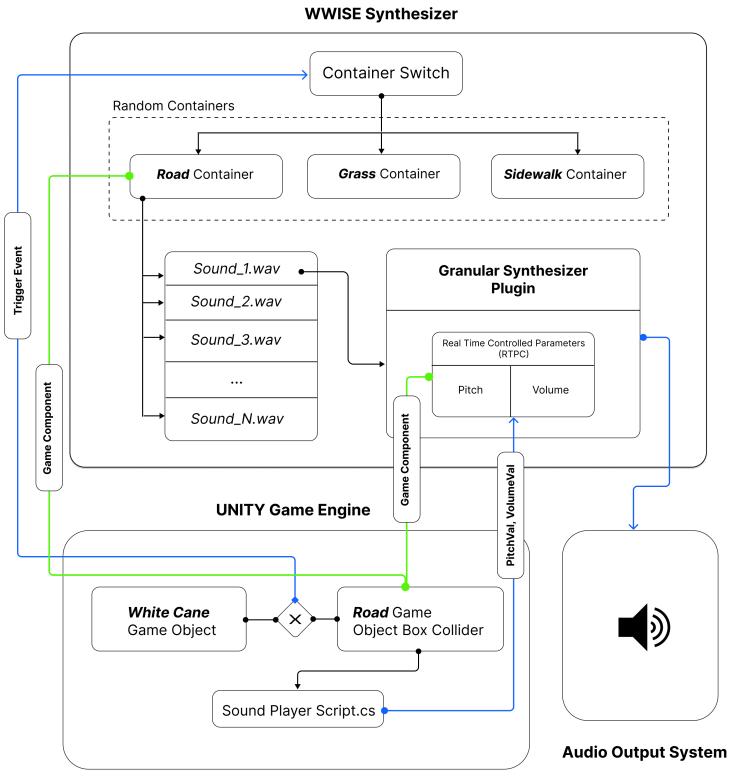

Figure 5: Wwise synthesizer and Unity integration block diagram. The green lines represent the linkage between system components, the blue lines represent inter-system calls whereas the black flow lines represent the intra-system calls.

and the input to the haptuators is through a linear audio amplifier module (model no. LM386). The haptic output is driven by the haptuators (TactileLabs, model no. TL002-14-R) that are firmly attached to the white cane near the tip and the handle using plastic brackets as shown in Fig. 6. The primary reason for using a haptuator over a voice coil actuator is due to its low static friction and commercial availability. Moreover, the specific positioning of the haptuators is to accurately mimic acceleration captured through accelerometer sensors positioned at the same position during the data collection phase as shown in Fig. 1.

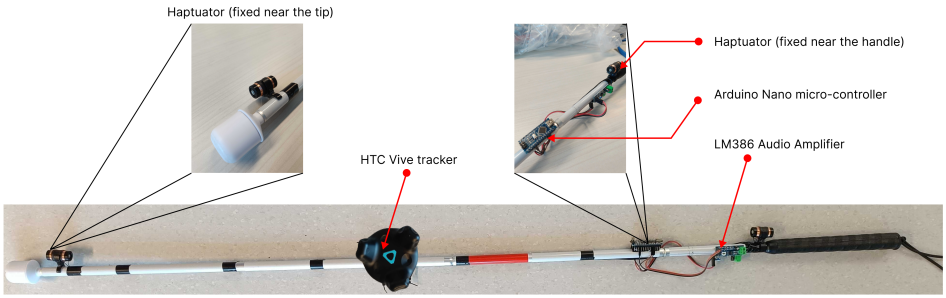

Figure 6: Haptics rendering device using haptuators vibrotactile transducers

A custom Arudino-based C program *(see Appendix A, Listing 1)* is used to connect to the Unity game engine through the serial connection specified at a particular port and a set baud rate. As explained in phase 2, similar to how Unity triggers surface-specific audio to be synthesized on collision of the cane game object with a surface box collider, the same event sends a start haptics flag appended with the surface tag through the serial connection to initiate the haptics

rendering. For example, if the collided box collider is of type - 'road', 'startHaptics_Road' is sent over the serial channel to play the road surface-specific acceleration magnitude signal which was generated and stored during phase 1 of the implementation. Similarly, a stop flag is set to indicate the haptics rendering to be terminated once the cane game object is no longer in contact with any surface box colliders *(see Appendix A, Listing 6).*

3.3.2 Rendering Process

For the current scope of the directed studies, due to time constraints, for each surface, a single DFT321 time domain signal for only one accelerometer sensor (ADXL345 attached near the tip of the cane) was used to render vibrations following the mechanism discussed above. A DFT321 time domain signal is converted into its spectral representation using Fourier transform where only the values of amplitude before the Nyquist cutoff frequency are considered. Thus, since the original acceleration data were sampled at a frequency of 200 Hz, in the frequency domain, 100 Hz is considered the highest attainable frequency as per the Nyquist-Shannon sampling theorem.

Once the spectral domain is calculated, PWM is used to play the magnitude at each frequency through the haptuator for a specified amount of time (˜10 ms). Frequency for a given instant is used to calculate the period of square (pulse width) waveform and a duty cycle of 50% is used resulting in a modulated pulse width with high and low values for an equal amount of time. The Fourier transform is implemented using the standard Python NumPy library whereas the PWM is programmed using an Arduino-based program using standard C libraries *(see Appendix A, Listing 1).*

4 FUTURE WORK

4.1 Improvement in Data Collection Process

The current hardware for acceleration data collection features two different accelerometer sensors which resulted in a bottleneck sampling frequency of only 200Hz. This prohibited higher sampling rates and hindered the overall quality of the collected data. Moreover, when the white cane was used in a bi-direction swiping motion against different surfaces, the metrics such as the speed of the cane and the force exerted weren't considered. The number of samples for each surface and the variation of the surfaces that can attribute to significant discrepancies were also not considered. However, going forward the next version of the data collection prototype will have the same set of sensors and the aforementioned metrics, greater number of samples, and higher sampling rate along with surface variation will be properly addressed.

4.2 Ambient and Spatial Sound Configuration

The current VR game comprises only a limited number of interactive sounds mostly emerging from the white cane interaction with virtual surfaces and a few ambient sounds of ongoing traffic and etc. Since the main objective of the virtual outdoor navigation simulation is to create a soundscape for visually impaired users to navigate using varied sound queues and not visuals, it is essential that sound effects correlating to sound spatialization, source localization, Doppler effect in moving objects, and different ambient sounds be present in the game for an immersive in-game experience. These necessary refinements are planned to be accomplished by using stock audio and even manually recorded audio of possible interactions and using the same Wwise synthesizer along with its extensive list of plug-ins readily available to be integrated with Unity.

4.3 Standardizing Haptic Profile Rendering

The current haptics profiles that are being rendered are Proof of Concepts (POCs) for the Minimal Viable Product (MVP) prototype that lacks specificity and a higher degree of actuality. This is mostly due to the use of specific hand-picked acceleration signals in an unoptimized rendering process resulting in a lack of variance as well as fidelity. The aim, going forth would be to establish a standard mechanism through tried and tested processes that include but are not limited to Linear Predictive Coding (LPC) Signal synthesis or Machine Learning based Signal Processing (MLSP).

4.4 Improved Game-play and Character Navigation

A crucial pre-requisite for the user study of the outdoor navigation VR game would be to create a definite theme for the gameplay rather than one based on open-world navigation. The theme of the gameplay is planned to revolve around a 'scavenger hunt' where users are provided with haptic and auditory queues which they follow to complete a given set of tasks. Moreover, currently, the navigation of the character is controlled using thumb-sticks inhibiting users to use both hands as the other hand is busy holding the cane. Since one of the hands is bound to be occupied with the cane, the use of a thumb stick introduces an added layer of situational disability. To counteract this, the integration of a VR slide mill in the game will not only free up the cognitive load on users to figure out directions but will also provide a natural way to navigate through stationed walking.

5 CONCLUSION

This report briefly surveys the promising role of VR to inclusively accommodate visually impaired users, explores the underlying accessibility concerns in terms of content, interaction, and devices, and highlights key challenges while trying to foster empathy. Moreover, it demonstrates a well-structured implementational approach to creating a tool-mediated VR soundscape for visually impaired users by combining audio synthesis and haptics rendering. The report also documents technical shortcomings during various pre-requisite phases and entails insights into how future work will be carried out in order to remediate them. In summary, this presents a POC for an MVP prototype and establishes the groundwork for a more detailed version of a scavenger hunt VR game focused on outdoor navigational simulation for visually impaired users to explore possible applications comprising of OEM training, accessibility studies, and empathy fostering.

References

[1] M. Mott et al. Accessible by design: An opportunity for virtual reality. In *2019 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 451–454, 2019.

[2] Emily Ackerman. Virtual reality has an accessibility problem. *Scientific American Voices*, September 2021.

[3] Julian Kreimeier, Pascal Karg, and Timo Götzelmann. Blindwalkvr: Formative insights into blind and visually impaired people's vr locomotion using commercially available approaches. In *Proceedings of the 13th ACM International Conference on PErvasive Technologies Related to Assistive Environments (PETRA '20)*, pages Article 29, 1–8, New York, NY, USA, 2020. Association for Computing Machinery.

[4] Niklas Elmqvist. Visualization for the blind. *Interactions*, 30(1):52–56, 2023.

[5] Benedikt Pätzold, Alexander Rochow, Maximilian Schreiber, Rasmus Memmesheimer, Christopher Lenz, Michael Schwarz, and Sven Behnke. Audio-based roughness sensing and tactile feedback for haptic perception in telepresence. *arXiv preprint arXiv:2303.07186*, 2023.

[6] Yuhang Zhao, Cynthia L. Bennett, Hrvoje Benko, Edward Cutrell, Christian Holz, Meredith Ringel Morris, and Mike Sinclair. Enabling people with visual impairments to navigate virtual reality with a haptic and auditory cane simulation. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*, pages Paper 116, 1–14, New York, NY, USA, 2018. Association for Computing Machinery.

[7] Alexa F. Siu, Mike Sinclair, Robert Kovacs, Eyal Ofek, Christian Holz, and Edward Cutrell. Virtual reality without vision: A haptic and auditory white cane to navigate complex virtual worlds. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems (CHI '20)*, pages 1–13, New York, NY, USA, 2020. Association for Computing Machinery.

[8] Dhruv Jain, Sasa Junuzovic, Eyal Ofek, Mike Sinclair, John R. Porter, Chris Yoon, Swetha Machanavajhala, and Meredith Ringel Morris. Towards sound accessibility in virtual reality. In *Proceedings of the 2021 International Conference on Multimodal Interaction (ICMI '21)*, page 12 pages, New York, NY, USA, 2021. ACM.

[9] Michele A. Williams, Caroline Galbraith, Shaun K. Kane, and Amy Hurst. Just let the cane hit it: How the blind and sighted see navigation differently. In *Proceedings of the 16th International ACM SIGACCESS Conference on Computers & Accessibility (ASSETS '14)*, pages 217–224, New York, NY, USA, 2014. Association for Computing Machinery.

[10] Waranuch Jeamwatthanachai, Mike Wald, and Gary Wills. Indoor navigation by blind people: Behaviors and challenges in unfamiliar spaces and buildings. *British Journal of Visual Impairment*, 37(2):140–153, 2019.

[11] Alex M. Silverman. The perils of playing blind: Problems with blindness simulation, and a better way to teach about blindness section. *Journal of Blindness Innovation and Research*, 5(2), 2015.

[12] Anthony J. Martingano, Faviola Hererra, and Sara Konrath. Virtual reality improves emotional but not cognitive empathy: A meta-analysis. *Technology, Mind, and Behavior*, 2(1), 2021.

[13] Gianni Virgili and Gary Rubin. Orientation and mobility training for adults with low vision. *The Cochrane Database of Systematic Reviews*, 2010(5):CD003925, 5 2010.

[14] Joseph M. Romano and Katherine J. Kuchenbecker. Creating realistic virtual textures from contact acceleration data. *IEEE Transactions on Haptics*, 5(2):109–119, 2012.

[15] Jung-Suk Lee, François Thibault, Philippe Depalle, and Gary P. Scavone. Granular analysis/synthesis for simple and robust transformations of complex sounds. In *49th International Conference: Audio for Games*, London, UK, February 2013. Audio Engineering Society, AES.

[16] Dusti Miraglia. Grain: A comprehensive and in-depth first look into propellerhead's new granular synth from reason 10. should it really just be 'taken with a 'grain' of salt'?, 2017.

Appendix A

Listing 1: Haptics Profile Rendering program

```
#include <Wire.h>

const int soundPin_adxl = 8;
const int soundPin_imu = 9;

// Define the frequency range (in Hz) you want to map the accelerometer data to
const int minFrequency = 50;
const int maxFrequency = 500;
// Define the array to store 200 samples of accelerometer data
const int numSamples = 200;
const float maxAcceleration = 16.0;
//haptics flag constants
const String startHapticsFlag = "startHaptics";
const String stopHapticsFlag = "stopHaptics";

//startHaptics_Sidewalk
//startHaptics_Road
const String sidwalkFlag = "Sidewalk";
const String roadFlag = "Road";
const String gravelFlag = "Gravel";
const String grassFlag = "Grass";


float sidewalk_accelerometerData[] = {11.9957814, -9.21458248, -23.5700082,
    11.3311076, 2.10931239, -0.715526674, -25.9875513, 2.9975653, -6.87430486,
    -7.20617744, -16.2885319, 3.98144169, 6.07271322, 6.29952507, -11.7517073,
    12.4924423, 4.59385452, 20.9898635, 16.8637806, 29.9948699, 8.31520837,
    -6.46487079, -5.91893128, -2.00704778, 11.2743066, -4.87936973, 8.71549064,
    1.93152907, -0.00024213223, -8.28404816, -11.2460762, -11.9434974, -4.41131833,
    0.905374799, 2.08265642, 18.2134851, 2.14743718, -14.6391512, -59.4081487,
    -9.7212467, -5.47009807, 29.9572038, -7.60662351, 37.6334092, -8.51491802,
    29.3641215, 5.97559901, 17.5704724, -4.33064044, 46.1924686, 42.5017547,
    15.2176744, 22.1951058, 33.4974066, 45.2259246, 23.0480323, 37.6955117,
    27.7786835, 46.5047569, 34.0076926, 25.0109317, -0.867646158, -36.6897944,
    -116.113146, 15.7095141, -1.31613702, -9.60356609, 35.011701, 76.7573393,
    61.8314583, -73.0270151, 17.1436269, 48.8594951, 39.9559273, 18.8887119,
    12.0483438, 14.3824692, -122.291376, 136.06904, -78.8252051, 80.2496586,
    3.08481888, 26.5704576, -11.9938283, 28.8715189, 114.884402, 38.6832423,
    -61.5113307, 0.505878752, -96.8500346, 23.3638941, 215.915491, 101.176699,
    -60.7742779, 27.4498507, 52.8265268, -18.3745432, 5.17737657, 60.5367632,
    -282.385604, 16.5759108, 34.9335958, -10.5350981, 67.4025087, 54.422634,
    19.6948566, 38.7801071, -174.906868, -13.235666, 55.787524, -38.593609,
    -83.1914024, -21.7093806, 11.7212072, -6.55145649, -60.9866981, -21.9606595,
    -4.7360684, -5.57473964, 7.97172101, -32.6917866, 93.75189, 35.1112426,
    -13.6286771, 17.6733551, 44.3398415, -41.413385, -111.952928, -187.215947,
    13.9578359, 2.1751687, -9.83531682, -13.3178638, 26.7919743, 12.3115783,
    -10.552322, -42.8370955, 14.2738686, 134.878727, -42.5618829, -2.36080394,
    -8.95442195, 7.81003306, -55.6771838, 22.782337, -0.695715284, 22.1655807,
    -13.3778042, -1.56005094, -11.116105, 24.7422776, -33.2086579, 8.47333649,
    22.1948097, 41.4118854, 13.4665196, -12.2505865, -0.497795992, -3.81541039,
    7.65603573, -14.5760736, -4.52887682, 12.2973116, -9.64113763, 14.9421895,
    -5.08907705, -2.26743596, -5.52588992, -4.66530117, 0.995700298, -0.0371005869,
```

```
    -2.42724234, 24.3664112, 3.25746252, -4.78725815, -9.98558795, 14.6514469,
    -20.7625331, 2.85499557, -5.85779184};
float road_accelerometerData[] = {-2.00441164, -11.30262114, -7.04176148, 6.10804368,
    -9.2677073, 0.85253779, -4.05841214, -8.82869467, 5.92913871, -13.17569853,
    -4.00686144, -12.65535475, 1.21160183, 3.19548214, 2.45824047, -9.91845318,
    -9.60866502, -9.50401968, 6.59015286, 3.07346011, -2.97964702, 5.87834398,
    5.74604466, 2.12940048, 1.80735609, -8.76482576, -1.61352609, -8.22156302,
    11.58111061, -2.49780166, 16.62957806, -5.42727276, -2.21121626, 8.85261625,
    -17.30798437, 2.37655641, -14.18867441, 9.28148094, 4.25120469, 19.496621,
    11.88428677, -14.5332248, 3.74044842, 0.70053532, 6.73320703, -4.28217952,
    18.69374606, -0.64765407, 36.0063441, 6.8282555, 2.9195438, 9.76870524,
    10.00421264, 27.17075745, -3.20807223, 6.06000821, 1.05558071, -19.70747681,
    15.28920397, 21.43149463, -4.63225087, 24.73815502, -11.14044741, -9.52868535,
    -2.60133967, 25.49128578, 36.29477487, 13.17706853, 2.25187254, -13.74184919,
    39.53660156, 34.54398927, 3.9914109, 29.55484598, 9.90294442, 30.64573616,
    -17.9357516, 39.74270582, -17.61875958, 76.87537228, -19.52175682, 92.75288547,
    20.6319604, 29.44954511, -17.90990864, -126.2107468, 69.28864535, -23.88311807,
    69.74451545, 1.52272179, 13.33912835, -4.20693199, 4.36512736, -100.57436499,
    56.51063744, 11.2976387, -6.09692474, -68.69224718, -11.71762327, 98.94914647,
    47.82190242, 7.69068253, 67.31366043, -42.13324615, 30.39431905, 83.50846872,
    -28.18277376, 1.41439664, 7.4474355, -71.36617623, -20.35078097, 16.46456657,
    -116.78413394, -19.14835938, 41.38492695, 16.33631196, 19.41288459, 32.8105479,
    -17.59096275, 24.3050553, -54.32740784, 17.39426137, -22.94198288, -77.45198927,
    10.3957197, 11.3806471, -35.88814471, 29.68253222, 74.28004332, 25.11461416,
    -47.24935578, -5.58841319, 5.79223815, 40.31429355, 28.33520565, -3.00138337,
    15.16418121, 14.05431476, 14.46182794, 3.87699584, -42.66396339, 14.19580406,
    -46.85629698, 65.00536698, 9.22765432, 35.26709698, 45.47676465, 44.11916455,
    -25.56262755, 15.38981417, -34.71839717, -15.24306496, -37.40310453, 28.26062385,
    13.68476158, 1.33665073, 5.6604951, -29.94963458, -13.52984517, 13.1189113,
    37.42953683, 28.89611965, -3.39718132, 2.49758966, 0.7496962, -9.6963159,
    -0.15866861, -20.73846659, 0.14011414, 8.82446046, 10.29884761, 10.40730039,
    -10.91096613, 4.97341836, 10.36182148, 1.64654927, 19.83356872, -15.72880949,
    -10.9672644, -5.90643153, 6.2231467, -0.71653556, -14.76591962, -7.46574461,
    -11.66965741, 0.91191166, -1.06115452, 2.17769232, -11.3091628, 14.22459039,
    -2.76072821, -0.29734456, -17.91789769, -14.5164608, -4.15290747, 2.00053358,
    4.83054997, 2.49683269, 0.88702996};
float gravel_accelerometerData[] = {-4.69967399, -5.6509253, -3.15117066, -3.68947705,
    -2.53251943, 0.39088592, -0.8582606, -4.02047801, -2.75695802, -1.74658639,
    -5.78735429, -2.04149525, -0.80428183, -6.99509834, -2.72960072, -1.98105141,
    0.14633161, -2.90760392, -0.08176117, -0.85606646, 0.96421951, -2.59255348,
    2.3974799, -2.71176703, 1.23666434, 0.85266183, -1.69646701, 5.79369349,
    -2.82715609, 6.09151516, -3.17143085, 4.80846217, 5.02108629, 9.62852406,
    3.03013163, 3.47625003, 10.2496775, 5.08918595, 19.84406322, 19.66889358,
    6.97247538, 14.90070175, 9.29373375, 14.5044693, -5.49366209, 12.29210784,
    -0.80112493, 1.83876413, 22.41381846, 15.22050263, -3.59360468, 18.56953471,
    12.57158309, 7.86219442, 21.77272822, 4.67782171, 19.99038555, 22.43061661,
    15.186432, 20.04583557, 8.02877198, 32.59524952, 8.98357555, -13.00872862,
    2.2659087, 23.85562202, -1.82716583, 22.49663124, 15.07834368, 16.57979581,
    5.62476866, 5.36566195, 18.27204343, 6.15406567, 15.60176224, 22.70390371,
    11.1439136, 26.41463445, -4.51703998, 17.50122071, -6.00752234, 7.56839813,
    9.78070694, 7.53891893, 6.1088925, 8.20182817, 10.27889251, 4.93837704,
    6.76962565, 17.49914285, 0.19419603, 6.30638359, 11.9790604, 2.28763886,
    21.33286202, -4.67370568, 16.75653259, 18.21807887, 17.75649148, 13.01500483,
    27.50639277, -8.51781572, 16.52533165, 9.89737965, 15.47223124, 33.79658119,
    -12.43391131, 20.29205438, 30.31549556, -7.19884705, 16.95967639, 12.39561863,
    -2.10958726, 23.44832185, 27.03908321, 17.28726634, 25.8346833, 13.81356481,
```

```
    2.42394266, 24.98817103, 32.87403574, 10.80902486, 13.62788693, 10.56105042,
    7.54538174, 21.87750662, 28.19829364, 12.42098757, 20.19478423, 14.91488342,
    17.0615701, 14.93836634, 12.53328477, 11.12044519, -1.00324925, 9.35055636,
    6.34535786, 3.44333559, 6.81159866, 9.63380845, -8.82848315, -0.87720737,
    7.24101047, -2.82038023, 7.97440495, -1.76706153, -3.28031799, 2.33134314,
    -6.03565883, 4.1597601, 4.33802843, -4.67626328, -0.17868194, -2.01168287,
    -0.04361508, -4.60759596, -2.1291735, 1.88262849, -4.01530902, -3.55945173,
    -1.59895221, -0.93818834, -1.41266978, -0.11224209, 0.27273554, -2.49645384,
    -3.10518316, -2.53704823, -2.92803908, -0.0671321, -3.78517869, -1.47134651};
// float grass_accelerometerData[] = {};

void setup() {
  Serial.begin(115200);
  Wire.begin();
  pinMode(soundPin_adxl, OUTPUT);
  pinMode(soundPin_imu, OUTPUT);
}

void loop() {
  // Iterate through the array and play each magnitude value as a sound
  if(Serial.available()){
    //to read input from the serial input via Unity
    String serialInput = Serial.readStringUntil('\n');
    serialInput.trim();
    String* tokens = splitString(serialInput, '_');
    if(startHapticsFlag.equalsIgnoreCase(tokens[0])){
       float* accelerometerData = selectArray(tokens[1]);
       size_t arrayLen = getArrayLength(tokens[1]);
       for (int i = 0; i < arrayLen ; i++) {
          if (Serial.available() > 0) {
            serialInput = Serial.readStringUntil('\n');
            serialInput.trim();
          }
          if(serialInput.equals(stopHapticsFlag))
            break;
          float normalizedMagnitude = accelerometerData[i] / (maxAcceleration * 2);
          int frequency = map(normalizedMagnitude * 1000, 10, 3200, minFrequency,
              maxFrequency);
          Serial.println(i);
          Serial.println(frequency);
          if (frequency > 0)
            playTone(frequency, 100); // Play for 100 milliseconds
        }
    }
    else {
       Serial.println("start haptics error");
       Serial.println("Flag received - "+serialInput);
    }
  }

}

void playTone(int frequency, int duration) {
  //Calculate the period of the tone by converting the frequency from Hz (cycles per
      second) to the period in microseconds (s)
  //Formula : Period (s) = 1 / Frequency (Hz) * 1,000,000
```

```
  long period = 1000000L / frequency;
  long pulseWidth = period / 2; // High and low durations for square wave
  //Convert the time duration from milliseconds (ms) to microseconds (s)
  for (long i = 0; i < duration*1000L; i += period) {
    Serial.println("playing tone");
    digitalWrite(soundPin_adxl, HIGH); // Sound ON
    digitalWrite(soundPin_imu, HIGH);
    delayMicroseconds(pulseWidth);
    digitalWrite(soundPin_adxl, LOW); // Sound OFF
    digitalWrite(soundPin_imu, LOW);
    delayMicroseconds(pulseWidth);
  }
}

String* splitString(String str, char delimiter) {
  int numTokens = 0; // Count the number of tokens
  for (int i = 0; i < str.length(); i++) {
    if (str.charAt(i) == delimiter) {
      numTokens++;
    }
  }
  numTokens++; // Add one for the last token after the last delimiter

  String* tokens = new String[numTokens]; // Create an array to store the tokens

  char char_array[str.length() + 1]; // Create a character array to store the string
  str.toCharArray(char_array, sizeof(char_array)); // Convert the String to a
      character array

  char *token = strtok(char_array, "_"); // Find the first token separated by "_"
  int tokenIndex = 0;

  while (token != NULL) {
    tokens[tokenIndex] = String(token); // Store the token in the array
    tokenIndex++;
    token = strtok(NULL, "_"); // Find the next token separated by "_"
  }

  return tokens; // Return the array of tokens
}

float* selectArray(String surfaceFlag) {
  if(surfaceFlag.equals(sidwalkFlag))
     return sidewalk_accelerometerData;
  else if(surfaceFlag.equals(roadFlag))
     return sidewalk_accelerometerData;
  else if(surfaceFlag.equals(gravelFlag) || surfaceFlag.equals(grassFlag) )
    return sidewalk_accelerometerData;
  else
    return;
}

size_t getArrayLength(String surfaceFlag){
    if(surfaceFlag.equals(sidwalkFlag))
     return sizeof(sidewalk_accelerometerData)/sizeof(float);
  else if(surfaceFlag.equals(roadFlag))
```

```
    return sizeof(road_accelerometerData)/sizeof(float);
  else if(surfaceFlag.equals(gravelFlag) || surfaceFlag.equals(grassFlag) )
    return sizeof(gravel_accelerometerData)/sizeof(float);
  else
   return;
}
```

Listing 2: Accelerometer sensors data collection program

```
#include "ICM_20948.h"

#include <Adafruit_Sensor.h>

#include <Adafruit_ADXL345_U.h>
#include <math.h>
#include <stdlib.h>
//#define USE_SPI     // Uncomment this to use SPI

#define SERIAL_PORT Serial

#define SPI_PORT SPI // Your desired SPI port.  Used only when "USE_SPI" is defined
#define CS_PIN 2 // Which pin you connect CS to. Used only when "USE_SPI" is defined

#define WIRE_PORT Wire // Your desired Wire port. Used when "USE_SPI" is not defined
// The value of the last bit of the I2C address.
// On the SparkFun 9DoF IMU breakout the default is 1, and when the ADR jumper is
    closed the value becomes 0
#define AD0_VAL 1

//defining ADXL sensor object
Adafruit_ADXL345_Unified adxl_accelerometer = Adafruit_ADXL345_Unified(0x53);
//defining IMU sensor object
#ifdef USE_SPI
ICM_20948_SPI imu_accelerometer; // If using SPI create an ICM_20948_SPI object
#else
ICM_20948_I2C imu_accelerometer; // Otherwise create an ICM_20948_I2C object
#endif
// Configuration for the trigger button and the LED indicator
const int triggerButtonPin = 3; // Pin for the trigger button
const int ledPin = 9; // Pin for the LED
int buttonState = 0; // Current state of the button
int previousButtonState = 0; // Previous state of the button
int samplecount = 1;

void setup() {
  pinMode(triggerButtonPin, INPUT); // Set the trigger button pin as input
  pinMode(ledPin, OUTPUT); // Set the LED pin as output

  //set the BAUD rate
  SERIAL_PORT.begin(115200);
  while (!SERIAL_PORT) {};

  #ifdef USE_SPI
  SPI_PORT.begin();
  #else
  WIRE_PORT.begin();
```

```
  WIRE_PORT.setClock(400000);
  #endif

  init_adxl_sensor();
  init_imu_sensor();
}

void init_adxl_sensor() {
  SERIAL_PORT.println("Initializing ADXL345 sensor");
  if (!adxl_accelerometer.begin()) {
    SERIAL_PORT.println("ERROR: NO ADXL345 detected!");
    while (1);
  } else {
    SERIAL_PORT.println("SUCCESS: ADXL345 detected!");
  }

  adxl_accelerometer.setRange(ADXL345_RANGE_16_G);
  delay(1);
  adxl_accelerometer.setDataRate(ADXL345_DATARATE_3200_HZ);
}

void init_imu_sensor() {
  bool initialized = false;
  while (!initialized) {
    #ifdef USE_SPI
    imu_accelerometer.begin(CS_PIN, SPI_PORT);
    #else
    imu_accelerometer.begin(WIRE_PORT, AD0_VAL);
    #endif

    SERIAL_PORT.println("Initializing IMU sensor");
    if (imu_accelerometer.status != ICM_20948_Stat_Ok) {
      SERIAL_PORT.println("Trying again...");
      delay(500);
    } else {
      initialized = true;
      SERIAL_PORT.println("SUCCESS: IMU sensor detected!");
      imu_accelerometer.setSampleMode(ICM_20948_Internal_Acc,
          ICM_20948_Sample_Mode_Continuous);
      //setting the sample/data rate of the IMU
      ICM_20948_smplrt_t mySmplrt;
      ICM_20948_fss_t myFSS;
      mySmplrt.a = 1; //specify the accel sample rate to maximum, set it to 1 : see
          Table 19 in datasheet DS-000189-ICM-20948-v1.3.pdf
      imu_accelerometer.setSampleRate(ICM_20948_Internal_Acc, mySmplrt);
      delay(1);
      myFSS.a= gpm16; // (ICM_20948_ACCEL_CONFIG_FS_SEL_e) // gpm2, gpm4, gpm8, gpm16
      imu_accelerometer.setFullScale(ICM_20948_Internal_Acc, myFSS);
    }
  }
}

void loop() {
  // Read the state of the trigger button
  buttonState = digitalRead(triggerButtonPin);
  // Check if the button is pressed
```

```
  if (buttonState == HIGH) {
    if (previousButtonState == LOW) {
      SERIAL_PORT.print("Samples");
      SERIAL_PORT.print(", ");
      SERIAL_PORT.print("IMU-X,IMU-Y,IMU-Z");
      SERIAL_PORT.print(",");
      SERIAL_PORT.print("IMU Magnitude");
      SERIAL_PORT.print(", ");
      SERIAL_PORT.print("ADXL-X,ADXL-Y,ADXL-Z");
      SERIAL_PORT.print(",");
      SERIAL_PORT.println("ADXL Magnitude");
    }
    previousButtonState = 1;
    // Button has just been pressed, turn on the LED
    digitalWrite(ledPin, HIGH);
    if (imu_accelerometer.dataReady()) {
      imu_accelerometer.getAGMT(); // The values are only updated when you call
          'getAGMT'
      printAccelerometersReadings( & imu_accelerometer);
    } else {
      SERIAL_PORT.println("Waiting for data");
      delay(500);
    }
  } else if (buttonState == LOW && previousButtonState == HIGH) {
    previousButtonState = 0;
    // Button has just been released, turn off the LED
    digitalWrite(ledPin, LOW);
    SERIAL_PORT.println("=");
    samplecount = 1;
  }
}

void printFormattedFloat(float val, uint8_t leading, uint8_t decimals) {
  float aval = abs(val);
  if (val < 0) {
    SERIAL_PORT.print("-");
  } else {
    SERIAL_PORT.print(" ");
  }
  for (uint8_t indi = 0; indi < leading; indi++) {
    uint32_t tenpow = 0;
    if (indi < (leading - 1)) {
      tenpow = 1;
    }
    for (uint8_t c = 0; c < (leading - 1 - indi); c++) {
      tenpow *= 10;
    }
    if (aval < tenpow) {
      SERIAL_PORT.print("0");
    } else {
      break;
    }
  }
  if (val < 0) {
    SERIAL_PORT.print(-val, decimals);
  } else {
```

```
    SERIAL_PORT.print(val, decimals);
  }
}

void printAccelerometersReadings(ICM_20948_I2C * sensor) {
  sensors_event_t event;
  adxl_accelerometer.getEvent( & event);

  SERIAL_PORT.print(samplecount++);
  SERIAL_PORT.print(", ");
  printScaledIMUAxesReading(sensor); // This function takes into account the scale
      settings from when the measurement was made to calculate the values with units
  printMagnitude(sensor -> accY()/100,sensor -> accZ()/100);
  SERIAL_PORT.print(", ");
  printScaledADXLAxesReading(event);
  printMagnitude(event.acceleration.y,event.acceleration.z);
  SERIAL_PORT.println();
}

// #ifdef USE_SPI
// void printScaledIMUAxesReading(ICM_20948_SPI * sensor) {
//    #else
void printScaledIMUAxesReading(ICM_20948_I2C * sensor) {
  //printFormattedFloat(sensor -> accX()/1000, 5, 2);
  // printFormattedFloat(sensor -> accY(), 5, 2);
  // SERIAL_PORT.print(", ");
  // printFormattedFloat(sensor -> accZ(), 5, 2);
  // SERIAL_PORT.print(", ");
  SERIAL_PORT.print(sensor -> accX()/100);
  SERIAL_PORT.print(", ");
  SERIAL_PORT.print(sensor -> accY()/100);
  SERIAL_PORT.print(", ");
  SERIAL_PORT.print(sensor -> accZ()/100);
  SERIAL_PORT.print(", ");
}

void printScaledADXLAxesReading(sensors_event_t event) {
  SERIAL_PORT.print(event.acceleration.x);
  SERIAL_PORT.print(", ");
  SERIAL_PORT.print(event.acceleration.y);
  SERIAL_PORT.print(", ");
  SERIAL_PORT.print(event.acceleration.z);
  SERIAL_PORT.print(", ");
}

void printMagnitude(float accl_y, float accl_z){
  float magnitude = sqrt(pow(accl_y,2)+pow(accl_z,2));
  SERIAL_PORT.print(magnitude);
}
```

Listing 3: DFT321 Algorithm Implementation

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

```
def smoothed_dft(acceleration):
    N = len(acceleration)
    window = np.hanning(N) # Select the Hanning window function
    windowed_acceleration = acceleration * window
    dft = np.fft.fft(windowed_acceleration)
    return dft

def calculate_frequency_domain_magnitude(acceleration_array):
    frequency_domain_magnitude = 0
    for acceleration in acceleration_array:
        dft_magnitude = np.abs(smoothed_dft(acceleration))
        frequency_domain_magnitude += pow(dft_magnitude,2)
    return pow(frequency_domain_magnitude,1/2)

def calculate_average_phase(acceleration_array):
    sum_imaginary_parts = 0
    sum_real_parts = 0
    for acceleration in acceleration_array:
        dft = smoothed_dft(acceleration)
        sum_real_parts += np.real(dft) # Sum of real parts
        sum_imaginary_parts += np.imag(dft) # Sum of imaginary parts

    average_phase = np.arctan2(sum_imaginary_parts, sum_real_parts) # Calculate average
        phase
    return average_phase

def perform_inverse_dft(magnitude, phase):
    complex_signal = magnitude * np.exp(1j * phase) # Construct complex signal from
        magnitude and phase
    time_domain_signal = np.fft.ifft(complex_signal) # Compute inverse DFT to obtain
        time-domain signal
    return time_domain_signal

def generate_time_domain_signal(acceleration_array):
    frequency_domain_magnitude =
        calculate_frequency_domain_magnitude(acceleration_array)
    average_phase = calculate_average_phase(acceleration_array)
    time_domain_signal = perform_inverse_dft(frequency_domain_magnitude,average_phase)
    return time_domain_signal

chart_cnt=1
excel_file_name = 'Gravel Road_adxl-3200Hz-16g_imu-562Hz-16g.xlsx'
surface_data = pd.read_excel(excel_file_name,sheet_name=None)
for sheet_name in surface_data.items():
    data = sheet_name[1]

    samples = data.get('Samples')
    adxl_mag = data.get('ADXL Magnitude')
    imu_mag = data.get('IMU Magnitude')

    adxl_x = data.get('ADXL-X').values
    adxl_y = data.get('ADXL-Y').values
    adxl_z = data.get('ADXL-Z').values
    adxl_acceleration_array = np.array([adxl_x,adxl_y,adxl_z])
    adxl_time_domain_signal = generate_time_domain_signal(adxl_acceleration_array)
```

```
    imu_x = data.get('IMU-X').values
    imu_y = data.get('IMU-Y').values
    imu_z = data.get('IMU-Z').values
    imu_acceleration_array = np.array([imu_x,imu_y,imu_z])
    imu_time_domain_signal = generate_time_domain_signal(imu_acceleration_array)

    print(np.real(adxl_time_domain_signal))
    print(np.real(imu_time_domain_signal))
    # Create a figure and axis object
    fig, ax = plt.subplots()

    # Plot the line

    ax.plot(adxl_mag,label = 'ADXL Magnitude')
    ax.plot(np.real(adxl_time_domain_signal), label ='ADXL DFT321 acceleration')

    # Add labels and title
    ax.set_xlabel('Time')
    ax.set_ylabel('Amplitude (m/s^2)')
    ax.legend()
    ax.set_title(excel_file_name.split('_')[0]+' acceleration data - '+str(chart_cnt))


    # Create a figure and axis object
    fig1, ax1 = plt.subplots()

    # Plot the line
    ax1.plot(imu_mag,label = 'IMU Magnitude')
    ax1.plot(np.real(imu_time_domain_signal), label ='IMU DFT321 acceleration')

    # Add labels and title
    ax1.set_xlabel('Time')
    ax1.set_ylabel('Amplitude (m/s^2)')
    ax1.legend()
    ax1.set_title(excel_file_name.split('_')[0]+' acceleration data - '+str(chart_cnt))
    chart_cnt+=1

    fig1.tight_layout()
    # Display the plot
    plt.show()
```

Listing 4: Acceleration data plot generation program

```
import matplotlib.pyplot as plt
import pandas as pd
chart_cnt=1
excel_file_name = 'Carpet_adxl-3200Hz-16g_imu-562Hz-16g.xlsx'
surface_data = pd.read_excel(excel_file_name,sheet_name=None)
for sheet_name in surface_data.items():
    data = sheet_name[1]
    # Extract the x and y coordinates from the data
    x = data.get('Samples')
    y1 = data.get('ADXL Magnitude')
    y2 = data.get('IMU Magnitude')

    # Create a figure and axis object
```

```
    fig, ax = plt.subplots()

    # Plot the line
    ax.plot(x, y1, label ='ADXL Magnitude')
    ax.plot(x, y2, label = 'IMU Magnitude')

    # Add labels and title
    ax.set_xlabel('Samples')
    ax.set_ylabel('Y-Z Plane Magnitude (g)')
    ax.legend()
    ax.set_title(excel_file_name.split('_')[0]+' acceleration data - '+str(chart_cnt))
    chart_cnt+=1

    fig.tight_layout()
    # Display the plot
    plt.show()
```

Listing 5: Sweep sound audio player script

```
 using UnityEngine;

 using System.Collections;
 using System.IO.Ports;
using System;

public class SweepSoundPlayer : MonoBehaviour
{
    [SerializeField]
    private AnimationCurve pitchCurve;
    [SerializeField]
    private AnimationCurve volumeCurve;
    [SerializeField]
    private float velocityThresold = 0.5f;
    [SerializeField]
    private float velocityFactor = 0.1f;
    [SerializeField]
    private float dampness = 10f;
    private bool _isPlaying = false;
    private Vector3 _lastPosition;
    private bool startsInteraction = false;
    private uint playingId;
    private GameObject secondaryCollisionGameObj;
    private GameObject registeredCollidingGameObject;
    private Vector3 lastPosition;
    private void Awake()
    {
        _lastPosition = this.transform.position;
    }

    private void Start()
    {
        AkSoundEngine.RegisterGameObj(gameObject);
        _isPlaying = false;
    }

    private void Update()
```

```
{
    //Debug.Log("****Data from accelerometer =" + arduinoPort.ReadLine());
    secondaryCollisionGameObj =
        this.transform.gameObject.GetComponent<CollisionDetectionCustomScript>
    ().
    _secondaryCollsionObject();
    Debug.Log("Secndary Collision object: "+ secondaryCollisionGameObj);

    bool startsColliding =
        CollisionDetectionCustomScript.IsTouching(this.transform.gameObject,
    secondaryCollisionGameObj);

    //Starting the movement
    if(!startsInteraction && startsColliding &&
        !Constants.DefaultPlaneTag.Equals(secondaryCollisionGameObj.tag)){
        startMovement();
    }
    //Ending the movement
    else if(startsInteraction && (!startsColliding || CheckIfStationary() ||
        !registeredCollidingGameObject.Equals(secondaryCollisionGameObj) )){
        endMovement();
    }
    //Updating the movement
    else if(startsInteraction
           && registeredCollidingGameObject.Equals(secondaryCollisionGameObj)
           && !CheckIfStationary()){
        updateMovement();
    }

    lastPosition = this.transform.position;
}

 bool CheckIfStationary() {
    float travelSquared = (this.transform.position - lastPosition).sqrMagnitude;
    return travelSquared < Math.Pow(Constants.stationaryTolerance,2);
}

void startMovement(){
    startsInteraction=true;
    registeredCollidingGameObject = secondaryCollisionGameObj;
    Debug.Log("Movement has been initiated");
}
void updateMovement()
{
    Debug.Log("Movement being updated");
    float velocity = (this.transform.position - _lastPosition).sqrMagnitude;
    _lastPosition = this.transform.position;
    try{
        if (velocity > velocityThresold
        && !_isPlaying)
    {
        playingId = AkSoundEngine.PostEvent("GroundSweepEvent",gameObject);
        Debug.Log("Audio play initiated");
        _isPlaying = true;
    }
```

```
        if (_isPlaying)
        {
            Debug.Log("Movement and audio play ongoing");
            velocity *= velocityFactor;
            float desiredVolume = volumeCurve.Evaluate(velocity);
            float caneinteractionVolume = Mathf.Lerp(100f, desiredVolume, dampness *
                Time.deltaTime);
            AkSoundEngine.SetRTPCValue(Constants.RTPC_CaneInteractionVolume
            ,caneinteractionVolume);
            Debug.Log("RTPC Volume : " + caneinteractionVolume);

            float desiredPitch = pitchCurve.Evaluate(velocity);
            float caneinteractionPitch = Mathf.Lerp(150f, desiredPitch, dampness *
                Time.deltaTime);
            AkSoundEngine.SetRTPCValue(Constants.RTPC_CaneInteractionPitch
            ,caneinteractionPitch);
            Debug.Log("RTPC Pitch : " + caneinteractionPitch);
        }
        }catch(System.Exception e){
            Debug.Log("Exception while playing audio :" + e);
        }

    }

    void endMovement(){
        startsInteraction=false;
        _isPlaying = false;
        AkSoundEngine.StopPlayingID(playingId);
        Debug.Log("Movement has been terminated");
    }
}
```

Listing 6: Haptics player script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO.Ports;
using System.Text;
using System;

public class HapticsProfilePlayer : MonoBehaviour
{

    private bool _isPlaying = false;
    private Vector3 _lastPosition;
    private bool startsInteraction = false;
    private uint playingId;
    private GameObject secondaryCollisionGameObj;
    public static SerialPort arduinoPort = new SerialPort("COM7");
    private bool hapticsStarted = false;
    private Queue<string> sendQueue = new Queue<string>();


    private void Awake()
    {
```

```
    _lastPosition = this.transform.position;
}

private void Start()
{
    AkSoundEngine.RegisterGameObj(gameObject);
    _isPlaying = false;
    initArduinoSerialPort();
    openArduinoPortConnection();
}

private void initArduinoSerialPort()
{
    arduinoPort.BaudRate = 115200;
    arduinoPort.Parity = Parity.None;
    arduinoPort.StopBits = StopBits.One;
    arduinoPort.DataBits = 8;
    arduinoPort.Handshake = Handshake.None;
    arduinoPort.RtsEnable = true;
}

private void sendDataToArduino()
{
    String dataToSend = sendQueue.Dequeue();
    arduinoPort.WriteLine(dataToSend+'\n');
    arduinoPort.BaseStream.Flush();
    Debug.Log("Data sent to Arduino - "+dataToSend);
}

private void OnDestroy() {
    if (arduinoPort != null && arduinoPort.IsOpen)
    {
        arduinoPort.Close();
        Debug.Log("Arduino Port closed");
    }
}

private void Update()
{
    secondaryCollisionGameObj =
        this.transform.gameObject.GetComponent<CollisionDetectionCustomScript>().
    _secondaryCollsionObject();
    bool startsColliding =
        CollisionDetectionCustomScript.IsTouching(this.transform.gameObject,
    secondaryCollisionGameObj);
    try{
    //Starting the movement
    if(!startsInteraction && startsColliding &&
        Constants.WhiteCaneTipTag.Equals(secondaryCollisionGameObj.tag)){
        startMovement();
    }
    //Ending the movement
    else if(startsInteraction && !startsColliding){
        endMovement();
    }
    }catch(System.Exception e){
```

```
            Debug.LogError("Exception while sending data to Arduino -
                "+e.GetBaseException());
        }
    }

    void startMovement(){
        startsInteraction=true;
        sendQueue.Enqueue(Constants.StartHapticsFlag);
        sendDataToArduino();
        Debug.Log("Haptics initiation sent to Arduino");
    }
    void updateMovement()
    {
        //TODO future usage
    }

    void endMovement(){
        startsInteraction=false;
        sendQueue.Enqueue(Constants.StopHapticsFlag);
        sendDataToArduino();
        Debug.Log("Haptics termination sent to Arduino");
    }

    public void openArduinoPortConnection(){
        if(arduinoPort != null){
            if(arduinoPort.IsOpen){
                Debug.Log("Arduino Port is arleady opened");
            }
            else{
                arduinoPort.Open();
                arduinoPort.ReadTimeout = 16;
                Debug.Log("Arduino Port Opened!");
            }
        }
        else{
                Debug.Log("Arduino Port == null");
        }
    }
}
```