Master Thesis
Electronic Music

Mobile Application Development for Musical Creation

The realization of a Music Art Gallery with *Ionic* and *Web Audio API*

Alessandro Fiordelmondo
mat. N° 12666

supervised by
Alberto Novello

Academic Year 2019/2020

Contents

Introduction

This thesis describes the realization of *Virtual Sound Sculptures*: a virtual art gallery in the form of a mobile application. Mobile applications, and more generally the Internet, are ubiquitous tools that we use every day in our life. They have shaped our way of life, communication strategies, and ultimately they have altered the way we use and perceive art. Our smartphones and computers are the technologies with which we enter art worlds and more often they are also the media that we use to create. Recent artworks are based, or heavily rely, on the usage of these technologies. *Virtual Sound Sculptures*, the work described in this thesis is one of them. It offers an example of Internet and mobile application to create an interactive sound gallery of sonic sculptures. To achieve this result we used a mixture of programming tools such as *Ionic* and *Web Audio API*. The combination of these *JavaScript*-based technologies allows to transform the mobile phone into an interactive sound interface: to make music, listen to it, and create new forms of listening. *Virtual Sound Sculptures* aims to realize the same experience of a real gallery, but without any visual reference. Only musical sculptures are shown, mounted in the virtual space. And the virtual space is inscribed in the entire globe through the Earth coordinate system.

The first chapter of this thesis introduces the main research topics and questions that attracted me to work on this subject. It introduces reflections about the flowing of the time in the music and in particular the concept of *Vertical Music*, which is the common thread throughout all my efforts as a composer. Then, the thesis deals with the definition and forms of Internet Art. I will show the difficulties to delineate such an art practice and how my artwork can be inscribed inside this artistic context.

The rest of the thesis focuses on *Virtual Sound Sculptures*. At first all the blocks which compose the artwork and the used tools are presented. Then the thesis describes step after step the realization of such a mobile application. For each part, the theoretical and the practical implementation are explained, showing where these technologies are successful and where they fail. This part of the text provides many lines of original code with proper clarifications. To better understand the implementation of every block of code a basic *JavaScript* knowledge is required.

1 Toward the artwork

This work starts with a challenge. In 2016, during a lecture about *Music Aesthetics*, led by Roberto Grisly in a room at the Conservatory of Perugia, we spoke about time, the very first quality by which music can be described, composed and heard. Among all books mentioned around the topic, one of them was particularly highlighted during the lecture: *The Time of Music* by Jonathan D. Kramer. The book analyzes music through the psychological and philosophical human sense of time. In this book, music is analyzed through the psychological and philosophical human conception of time. The lecture continued treating linearity and non-linearity of time in music: from the Baroque linearity, through multipath linearity in the Beethoven's string quartets, to the Twenty-century music, with the loss of the goal-directed linearity in the second School of Wien, the discontinuity and the *moment music* in Stravinsky and then in Stockhausen. At the very end of the lecture Grisly introduced the concept of stasis in music, the non-linearity perfect realization, that in the book is named *Vertical Music*. I was struck by this idea of music without time. If, with the traditional music composition, one organizes the listener's time perception in horizontal succession, then in *Vertical Music*, where no horizontal organization exists but only simultaneous layers of sound, the listeners are the composers. The listeners create the temporal structure of *Vertical Music* through the evolution of their own perception over time. It is the inverse of the classical procedure, where a sort of interaction acts between the music and the listener, a "consciousness interaction"[Fio17] as Bergson would put it[1][Ber89, Ber22]. If our states of consciousness are the roots of our temporal perception, and if our own time is the horizontal development of *Vertical Music*, then it is our consciousness to interact with this kind of music. As I began to find all the discussion around *Vertical Music* topic more and more captivating, started arising in me the curiosity to realize an ideal musical piece. After four years, many things have changed and my enthusiasm regarding the topic has partially reduced. Even if I still think that this concept of music remains really interesting, nevertheless, it involves a quasi-complete passive activity of composition because of its very definition. The realization of such music might seem easier than it really is, leading to other musical issues beyond the composition itself. I owe a lot to the idea of *Vertical Music*: during these years, this challenge and this "simple" concept allowed me to discover new artistic worlds. For this reason, I have chosen it again as master thesis' work, embark again towards new technologic and aesthetic challenges. This work marks my third effort in the context of *Vertical Music* and I will talk about the episodes which brought me here only after introducing the guidelines that rule this vertical world.

[1]Henri Bergson's thought was particularly focused on time perception: one perceives a duration in relation to single consciousness states; the time perception is given by the overlapping of a sequence of impressions, internally acquired and deciphered by our consciousness.

1.1 Vertical Music

A piece of *Vertical Music* is temporally undifferentiated in its entirety. It lacks phrases, progression, goal direction, and every temporal feature. The result is a single present stretched out into an enormous duration, a potentially infinite "now" that nonetheless feels like an instant. This music exists between simultaneous layers of sound rather than between successive gesture. A vertical piece, then, does not exhibit large-scale closure either a beginning. It does not build to a climax, does not purposefully set up internal expectations, does not build or release tension, and does not end but simply ceases. No event depends on any other events rather the *Vertical Music* is just one large event[Kra88]. In *The Time of Music*, Kramer suggests some historical pieces of such music, including Iannis Xenakis' *Bohor I* (1962), Larry Austin's *Caritas* (1969), Terry Riley's *Rainbow in Curved Air* (1969), and more, often linked to the American's scene of minimal music. Although all these pieces can be inscribed inside a music form without development, they do not grant the *Vertical Music* form because a process is always contemplated. *Vertical Music* doesn't allow any process, even an indirect one. The perfect example of *Vertical Music*, outside the music field, is the sculpture. This analogy, which is the base of this work, is given in the book *The Time in Music*:

> *"Listening to a vertical musical composition can be like looking at a piece of sculpture. When we view the sculpture, we determine for ourselves the pacing of our experience: We are free to walk around the piece, view it from many angles, concentrate on some details, see other details in relationship to each other, step back and view the whole, contemplate the relationship between the piece and the space in which we see it, close our eyes and remember, leave the room when we wish, and return for further viewings. No one would claim that we have looked at less than all of the sculpture though we may have missed some of its subtleties!, despite individual selectivity in the viewing process. For each of us, the temporal sequence of viewing postures has been unique. The time spent with the sculpture is structured time, but the structure is placed there by us, as influenced by the piece, its environment, other spectators, and our own moods and tastes. Vertical music, similarly, simply is. We can listen to it or ignore it. If we hear only part of the performance we have still heard the whole piece, because we know that it will never change. We are free to concentrate on details or on the whole. As with sculpture, the piece has no internal temporal differentiation to obstruct our perceiving it as we wish."*[Kra88, p. 57]

Therefore, still using Kramer's words:

> *"Vertical music is that in which nonlinearity predominates over linearity, that which differs most from traditional Western music. Vertical music tries not to impose itself on the listener, nor to ma-*

> *nipulate to use a popular buzzword from the 1960s, when verticality in music was particularly strong! an audience. The context of vertical music allows a listener to make contact with his or her own subjective temporality. It is music of subjectivity and individuality.'*[Kra88, p. 57]

Baside the American Minimalists, the very initial concept of *Vertical Music* can be found in their progenitor, Erick Satie, and their successor, the *Ambient Music* [2]. Still, for the same reasons as for the Minimalists, these works contain only an initial idea of music verticality. The more coherent and complete example of *Vertical Music* can be found in the form of music installations, music sculptures, and many "objectified" pieces of music[3]. Often, *Vertical Music* is realized with a visual reference in which the visual medium's peculiarities supports the sound medium.

My first attempt in this context is a music album called *Musica Verticale* (*Vertical Music*) that I completed in 2016. It consists of four pieces or *moments*. Each moment is characterized by only one event with a duration of around ten minutes, and it is built in a vertical form, e.g. layers of simultaneous sounds. They are complex frozen timbres: a multilayer of different percussion coda (*Moment I*), a multilayer of voices (*Moment II*), a multilayer of brass (*Moment III*), and a multilayer of string instruments (*Moment IV*). Nothing happens except the beginning and closure of every moment. Because of these two essential events, my challenge was not won: these are unremovable constraints for a container which has finite duration such as an album. Thinking on how to remove these limits led me to a brand new world: the Internet and Internet Art. There is no way to remove these events in an album, or live performance, only the Internet can. The Internet opened for me a new horizon toward new musical forms and possibilities. Removing the beginning and the closure of a piece is just one of them, and it is not a trivial thing.
This step threw me into the Internet developer world: from an aesthetic and philosophical matter to a purely technical solution. For the first time I had to study several programming languages: from the pure *HTML* and *CSS* through the basics of *JavaScript* for the frontend and *PHP* for the backend, to finally understanding how to create music that exists only through a browser on the Internet. *Web Audio API*, the *JavaScript* tool to make and process music on

[2]Probably, one of the best examples of *Vertical Music* is *Vexations* (1893) by Erick Satie. In *The Time of Music*, Kramer describes an anecdote with this piece, regarding the experience of *Vertical Music*. While the author never refers to the *Ambient Music* in the book. However, even if the term is very general, many musical pieces of this genre strongly recall the *Vertical Music* idea. Brian Eno's *Ambient Music* (such as the album *Music for Airports*) and the *Muzak music* (concept of Erick Satie coining) are the most important historical examples of ambient music.

[3]Music based on a physical object (or more in general on the visual art). The wind and water organs are perfect examples of musical sculpture. One of the most famous is *The Singing Ringing Tree* in UK, designed by Tonkin Liu group. But even the classical musical installation can be seen as a musical sculpture. In all these cases the musical artwork experience is focused around one or more sound objects.

the web, was a major discovery. As a consequence, in 2017 I realized *Moment V*, a web installation, the fifth of the album's four movements, and the project of my bachelor thesis. The web site was a sort of virtual space hosting a music sculpture, a multilayer of organ's sounds. The active role of the human visitor was just to listen, having no interactivity with sound. My aim was to replicate the same existence condition of sculptures inside a museum in a virtual container. Therefore, like in the museum, the music sculpture was persistent. The users could enter the web site and listen to a sculpture that never stopped. If several users simultaneously entered the web site from different locations, they would listen to the same sound: exactly like entering a museum with other people and observing the same sculpture, with the same light, with the same surrounding sound. The sound of the sculpture was conceived as a real-time process that sets the sound in motion without producing any development[4]. There was no visual reference on the website, a completely black page with only some information about the artwork.
The challenge of a *Vertical Music* without temporal boundary seemed overcome, but the feeling of "reality" as in a virtual gallery was lacking. I started considering extending the virtual space in *Moment V*, to allow the user to move inside, having a sort of mild interaction, like the one with a sculpture in a museum: no *consciousness interaction* but *space interaction*. This is the main concept of the work presented in this thesis. The virtual museum becomes the new container of several *Vertical Music* sculptures, a more coherent kind of music album.

2 The artistic context and Internet Art

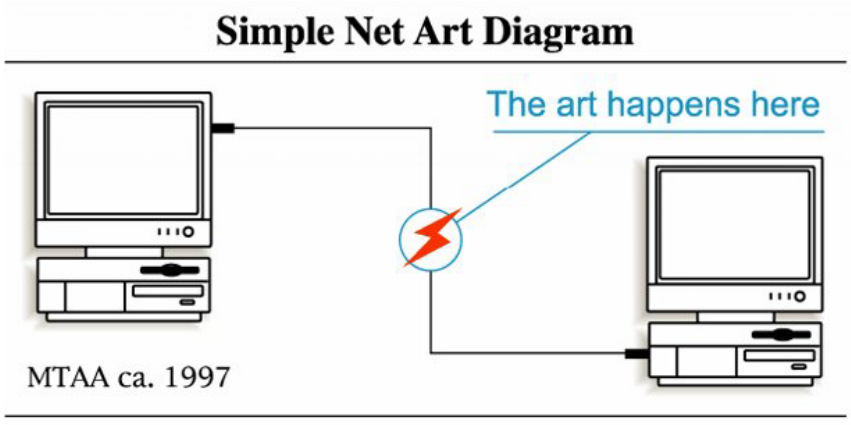


Figure 2.1: the *Simple Net Art Diagram* (SNAD) created by the artist duo MTAA in 1997

In order to understand the research progress contained in this thesis, it is important to properly introduce the reader to the artistic context in which the presented artwork is inscribed. Even if a partial idea of the music inspiration

[4]This important sound process will be used in this work again. It will be better described in Sound Generator.

of this thesis is already given in the above introduction, I have not described the prolific artistic context that developed in the late 90' and 2000s' from the very medium at the origin of my work: the Internet. Music on the Internet changes its form, its limits and the way to listen to it. It seems reasonable to assume that if a composer generates music using a web browser it is because she/he seeks from the Internet something impossible to achieve with the non virtual music medium. What sense would it make to create a classical composition on the Internet? For this work, the Internet was the only option to create a sculpture without beginning and closure. Here, the Internet is an extension of musical possibilities. Since the artworks presented in this text are generally based on the Internet, a question has to be clarified: can these artworks be called Internet Art? To answer this question we need to describe the background of this art practice.

It is difficult to give a proper definition of Internet Art. This art practice has developed in less than 30 years. A summarizing graph of all its history is displayed in the web site *Net Art Anthology*[5] by Rhizome. Many artworks, tendencies, and artistic movements have come to life during the last three decades making it arduous to outline a common thread. Even in *Net Art Anthologie*, it has been defined as a difficult task to sketch an Internet Art canon. Internet Art is an art practice made with the Internet but not only. If I were to give a personal definition, I would refer to the *Fluxus* movement. *Fluxus* debuted in 1963. The founder was George Maciunas which he wrote in the manifesto:

> *"Purge the world of bourgeois sickness, 'intellectual,' professional & commercialized culture, PURGE the world of dead art, imitation, artificial art, abstract art, illusionistic art, mathematical art, — PURGE THE WORLD OF 'EUROPANISM'!"*
>
> ...
>
> *"Promote living art, anti-art... NON ART REALITY to be fully grasped by all peoples, not only critics, dilettantes and professionals."*

Central idea of the *Fluxus* is the total integration of art and life. There are no specific fields, practices and, in general, ways to realize such art. Fluxus projects are wide-ranging and often multidisciplinary, humorous, and based on everyday, inexpensive materials and experiences, including everything from breathing to answering the telephone. Performances have been a significant part of the movement[Ked]. An exemplar performance is *Cut Piece* (1964) by Yoko Ono. In the performance, the audience had to cut pieces from the Yoko Ono's dress. In *I Went* and *I Met* by On Kawara a series of postcards was sent to his friends detailing aspects of his life. An example is the message 'I AM STILL ALIVE' sent to his parents. Important in the field of mail art is Ray Johnsons (even though it has never been an official member of the *Fluxus*).

[5]https://anthology.rhizome.org

In the project *New York School of Correspondence* he sent photocopies of his own original drawings, which often included prompts like 'please add hair to Cher' or 'draw a bunny'.
Because of its connectivity of individuals, the Internet appeared to be the perfect tool to virtually extend *Fluxus*' interactive happenings. One might say that Internet Art is nothing but the natural evolution of the *Fluxus* movement. The main Internet artworks are those created in the early years of the Internet, between 1994 and 1998. There are six fundamental artists: the Slovenian Vuk Ćosić, the English Health Bunting, the two Russians Alexei Shulgin and Olia Lialina and the duo JODI from the Netherlands and Belgium. Exemplar works are *King Cross Phone-In* (1994) by Health Bunting. He orchestrated a scheme that involved many people calling public phones in and in the surrounding area of London King's Cross railway station. On his website *Cybercafe.org*, founded in 1992, Bunting posted the phone numbers to all of the public phones and encouraged his followers to do one of the following: call in a pattern, call at a certain time, call and speak to a stranger, or show up and pick up the telephone. Bunting used his website as an informative source to let his readers know how to partake in his project[6]. Alexei Shulgin and Olia Lialina created several web pages with artwork with which the users could interact. These artworks are strictly based on the interaction artist/user. Olia Lialina was particularly focused on the narration possibilities in a web interaction system while Alexei Shulgin was interested in the most graphical web interaction. After 1998, Internet Art exponentially expanded. Many artworks maintained the focus on web interaction while others developed on several branches: YouTube videos and films, blogs and even performances outside the web. Nevertheless, they are often linked to performance, community and interaction but they lose the centrality of the Internet medium. Just an example is *Refresh* (2007) by Kristin Lucas. It is a real-life performance. Lucas filed to legally change her name from 'Kristin Sue Lucas' to 'Kristin Sue Lucas' as a way to "refresh" herself. Basically the same as refreshing a web page. She really appeared in front of a judge in California to petition her case. In this performance the Internet is an inspiration but not the medium.
How can we then define Internet Art? One of the most significant approaches which simplify Internet Art is the *Simple Net Art Diagram* (1997) in *figure 2.1.* Internet Art "happens" and therefore can be thought of as an action or as a performance; Internet Art happens "here", in-between two terminals, in-between the artists and the audiences where there is no hierarchical definition. Summarizing, Internet Art can be defined as interactive art and the artworks are the product of this interaction[MTA15, DM03]. Another important point of Internet Art is the post-medium condition. This concept is analyzed in many articles. In *Can Art History Digest Net Art?* (2009), Julian Stallabrass says:

> *"[...] the internet is not a medium, as painting is, but rather an encompass simulation of al reproducible media."*[Sta09, p. 169]

[6]http://irational.org/cybercafe/xrel.html

It says that the Internet can only exist inside the web and we can make use of it only inside of it.

> *"Net art's material utterly anything having to do with the internet."*[Sta09, p. 169]

In summary, the Internet is the medium with which one can simulate every other medium, the Internet is both space and the instrument with which one makes use of its art and finally, the Internet is also the goal of the Internet Art itself. However, the Internet is not strictly referring to the Internet. Here the difficulty lies. The Internet, as well as the lightning in the *Simple Net Art Diagram*, are the goals of Internet Art. Internet Art is then the communication, the interactivity, the sharing between the artists and the audiences. The artwork in the Internet Art as well as in the *Fluxus* movement is the meeting between these two human groups, where the boundaries are also blurred.
There is not a defined musical practice inscribed in Internet Art. During its 30 years, Internet Art has never been so much focused on Music or Sound. Visual Art and Game Art were the main disciplines. It depends on the interactivity. In Game Art, this aspect is obvious and essential. In Visual Art, interactivity is a means to expand the practice in a temporal dimension. Music does not need this augmentation, since music is by definition a temporal art, and interactivity might impoverish instead of expanding the artwork. The composer determines the rules of interaction, and no longer the artwork, which would be created by probability linked collaborative choices. The listeners no longer listen to the artwork but they play with it. However, the Internet can extend the action area of music. It can provide a further dimension to this art: space. Space has always been considered an important element in music. On the Internet, this dimension can be amplified, increasing the experience to listen to music. Artworks based on the use of maps are exemplary. Among them, the most known project is *radio aporee ::: maps* [7], a huge field-recordings sound map. It is a global sound map dedicated to field recording, phonography and the art of listening. It contains recordings from numerous urban, rural and natural environments uploaded by many users. It is a huge sound archive and within the map, the users can explore the sound of the environment from all over the world. From the same platform there is also *radio aporee ::: miniatures for mobiles* [8], an augmented reality project in public space. Here many artworks have been realized in several forms, from the sound art to the story-telling, artistic documentation, and hybrid experimental formats. It is possible to listen to these artworks while walking, through our telephone and GPS localization. The sounds continuously change and fade along your path, depending on the piece, and the listener's position, speed and direction.
The work presented in this thesis cannot be completely categorized to the artistic field of Internet Art. Its goal is not communication, sharing and interactivity even if I deeply use the potentialities of Internet and mobile applications

[7] *radio aporee ::: maps* link: https://aporee.org/maps/
[8] *radio aporee ::: miniatures for mobiles* link: https://aporee.org/mfm/

to break the temporal boundaries of a classical musical piece; the interactivity is used to synthesize a spatialized experience. However the artwork is a determined work and the goal is the mere musical experience. The audience does not intervene except in the listening process.

3 *Virtual Sound Sculptures* - A Music Art Gallery in a virtual space

I chose to call this work *Virtual Sound Sculptures*, to clearly describe for the user its principle idea. It is a music art gallery in a virtual space containing several artworks. Every artwork will consist of a musical piece in the form of *Vertical Music*, it develops in time as a sculpture: that is no development, but like every sculpture, and in general every artwork in a gallery, its development is created by the observers. They decide a path of points to observe, composing their own time experience of a static object. In this work, space is a virtual space wrapping the artworks, defining the boundaries of the time experience. This came from a desire to realize *de-objectified* musical sculptures: references are meant to help listening and not observation. This would not happen in the real world as sound always needs a source, which is most of the time visible.
This work has the form of a mobile application through which the user accesses the virtual space inscribed in the Earth coordinates system and theoretically, it could embrace the entire globe. Every artwork of the music gallery is limited in space and restricted to a specific region. Its position is defined by several coordinates points (three to a maximum of ten). The audience in the art gallery, e.g. the user of the mobile application, can listen to the artworks only within those specific boundaries. Similarly to the experience of a physical art gallery, the observers can move around, get away from or close to each artwork. They can listen to different layers of the musical sculpture, moving between the coordinate points in which the sculpture is defined. A group of artworks occupies a larger space, defining a trajectory through a city, a park, and so on, like an art gallery defining a path of artworks inside the hosting structure.
The realization of such a mobile application is based on two essential elements: the audience (mobile app users) and the sound sculpture, while the core of the mobile application is the system of interaction between them. The audience explores the virtual sound space through a pair of headphones. Thanks to a position tracking system (Position Tracking System), every user is described by a couple of coordinates and a direction in the virtual space. Each artwork consists of several sounds, layers of a musical sculpture distributed in the virtual space (with as many as the coordinates point by which is described). Each sound sculpture is created by a specific audio synthesis algorithm that could be defined as a sample granulator with large sound grains (Sound Generator). It was considered the perfect tool for the ideal *Vertical Music* form. Indeed with the usage of such an audio synthesis the sound can be frozen. The application of large grains allows to put in motion the frozen sound, keeping the naturalness of the original sample but avoiding any horizontal development.
To provide a realistic experience of sound in space, we created a 3D audio

system (3D Audio System). We assigned each sound sculpture with an omni-directional sound pattern. The sonic interaction between user and sculpture is defined as a function of the distance between the users' and the sound sculpture coordinates. The direction of the users is used to further define their point of view, so that they can move around the artwork and throughout it, with a complete 360° experience.

4 Overview of Technologies

Ionic & Capacitor

Ionic is an open-source Software Development Kit for hybrid mobile applications. The strength of *Ionic* is that only one and the same codebase is necessary to develop mobile applications (*iOS* app and *Android*), web site, progressive web applications, and even desktop applications. The codebase is basically that one for a web site, consisting of *HTML*[9], *JavaScript*[10] and *CSS*[11]. With these few languages, it is possible to build a big amount of different cross-platform applications.
Ionic is a set of web components. A web component is a technology supported by modern browsers and it is basically a custom *HTML* element with, behind the scene, specific functionalities carried out by *JavaScript* and *CSS*. A slide menu, that a user selects to see and click a list of instances in a typical application, is one example of a web component. *Ionic* provides a set of such elements that allow building native applications, provides a set of such elements that allow building native applications for *Android* or *iOS*. With *Ionic*, it is possible to use all the smartphones' sensors, such as the camera, accelerometer, gyroscope, magnetometer, light sensor, proximity sensor, the GPS, and the telephone service. This is particularly interesting in a preformative context.
Capacitor is another product of the Ionic Company primarily built to be coupled with *Ionic*. *Capacitor* allows the transition from the web application to a native mobile application, it is a mobile application that runs the web site inside itself, or a web browser that exclusively embeds only the designed website. The result is a mobile application indistinguishable from a purely native app. And, even if this wrapping process gives a less performative application than a native mobile one, it still delivers a functioning app for the average user. In addition to the possibility to write only one code for cross-platform applications (instead of learning and writing *Java* for *Android*, *C#* or *Swift* for *iOS* and *JavaScript* for the web), we are able to use all the features (API,

[9] *Hypertext Markup Language* (*HTML*) is the standard markup language for documents designed to be displayed in a web browser.

[10] *JavaScript* is a high-level and prototype-based object-orientation programming language. *JavaScript* is one of the core technologies of the World Wide Web. It enables interactive webpages and is an essential part of web applications. The vast majority of websites use it for client-side page behavior, and all major web browsers have a dedicated JavaScript engine to execute it.

[11] *Cascading Style Sheets* (*CSS*) is a style sheet language used for describing the presentation of a document written in a markup language like *HTML*.

Plugins, libraries and so on) of *JavaScript*, with mild tweaks.

TypeScript & Angular

In this context, I have used *TypeScript*[12] instead of pure*JavaScript*[13]. *TypeScript* is a syntactical superset of *JavaScript*. Practically, it introduces optional static typing on *JavaScript*: the variable can be classified as number, string, array, object or constructor and it can not be changed in the code. As well as this feature, *TypeScript* introduces the modules and interface usage, too. Although it brings more language restrictions, the static typing produces a more readable code both for the programmers and the computer. *JavaScript* programs can be read in *TypeScript* and vice versa, allowing no restrictions for old plugins and libraries written in *JavaScript*. For this thesis work I used *Angular*, a *JavaScript* framework that allows building reactive *JavaScript*-driven web applications. Angular allows building an application inside one and the same *HTML* page which is constantly re-rendered, giving to the user the impression to navigate in different pages. This so-called "single-page application" structure improves application performance by reducing the interactions with the server.

Web Audio API

JavaScript provides a specific API (*Application Programming Interface*) used exactly for sound synthesis and manipulation. Its name is *Web Audio API*. This API is a series of exposed code fragments that can be used to create audio tasks in a web browser. It is a collection of objects written in lower-level-languages that interact with the low-level functionalities of the browser that can be invoked through *JavaScript* code. *Web Audio API* introduces the possibilities of processing the audio on the web since 2013. After `<bgsound>` in 1996, which allowed to play background music just inside Internet Explorer, and `<audio>`, in 2008, that extended the audio playback for all browsers, *Web Audio API* was the biggest advancement for audio applications on a browser. A set of *audio node*s linked together defines the route of the audio stream, which goes from the sound source (such as the microphone input, buffer source or oscillators), through a processing path (such as filters and effects), to an audio output (mostly the speakers). Oscillators, buffer sources, delays, panners, filters, compressors, convolvers and few others are the entire set of *audio node*s provided by the audio context. With this limited set of *audio node*s, it seems particularly hard to build a detailed and specific audio process. However, *Web Audio API* provides an *audio node*, called `AudioWorklet`, that allows generating custom DSP effects from code. `AudioWorklet` yields however a significant worsening in the computational performance and in the audio quality. Even if *Web Audio API* is still below the level of current audio software, in quality and processing possibilities, its own musical potentiality is essentially linked with that of Web and Mobile Applications. Future development might surely

[12]https://www.typescriptlang.org/
[13]https://www.javascript.com/

overcome such limitations[Tur17, Smu13].

5 Position Tracking System

Being interactivity, thus user position, a fundamental aspect of my system, I needed a Position Tracking System, with the ability of tracking the movement of the user within one meter in the Earth coordinate system. In this way, the interaction between the users and the artworks would become smooth enough to avoid wide leaps from a position to another, thus creating a realistic and convincing sound localisation experience.
The first system selected was the *Global Position System* (GPS). It seemed a good candidate because it provides the coordinate on the Earth and the direction of the device in a unified way. In short, the GPS is a satellite-based radio navigation system that provides the geolocalization data in real-time where there is an unobstructed line of sight to four or more GPS satellites. Currently, there are between 24 to 32 active satellites around the Earth globe[oTUA08]. However, this system has some indicative problems which would compromise the mobile application. Here, the real problem is the time steps between a received data to the next. Because of different factors, especially depending on the connection between the receivers and the satellites, the system pushes data without any constant rate. Every obstacle which interferes with at least one of the four required signals causes an interruption on the data transmission. Obviously, this can happen for at least a few seconds, after which the connection restores (if one is not surrounded by big obstacles, like mountains or buildings block). So, in a normal situation, this would account for a time of data capture from a minimum average of two seconds to a maximum average of 20 seconds. This is good enough to track a car position but is not precise enough for this thesis. In addition, the GPS adds other problems that introduce error in the absolute coordinates estimation, such as the inaccuracy of the satellites' atomic clock, the multipath effect caused by the propagation of the signals across the ionosphere and troposphere, and the DOP (Dilution of Precision) effect, that depends on the position of the satellites and the receiver at the moment in which the computation is carried out[DGS10]. Finally, the direction calculated by the GPS is very unreliable: it is computed by the vector formed between two couples of coordinates. So without movement there is no certainty of the estimated direction.

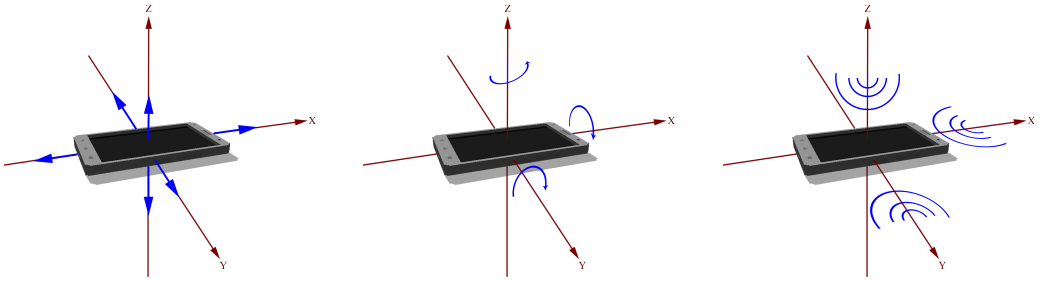
5.1 *Sensor Fusion*

The literature suggests *Sensor Fusion* as a solution to the GPS inaccuracy: "combining two or more data sources in a way that generates a better understanding of the system" (Brian Douglas). That means creating a more consistent, accurate and especially dependable system. The data sources can be provided by the sensors but even by mathematical models: is possible to partially prevent some behavior in the physic world and use it to correct the data provided by the sensors[Dou]. In this application, the input of the *Sensor*

Fusion system is the GPS, accelerometer, gyroscope, and magnetometer data together. The output, the goal of the application, is a fast and continuous position tracking and a smooth movement of the user's direction In this ensemble of sensor data, the only one which returns an absolute position is the GPS[14]. So, excluding the errors in the coordinate estimation (listed above), we will take this system as the absolute reference. Our main goal is to fix its data gaps and provide a better tracking of the direction.

Device Orientation

To obtain the absolute orientation, the accelerometer, gyroscope and magnetometer sensors have to be merged in a unified system. This kind of system is so-called *Attitude & Heading Reference System* (AHRS) which has to provide three values: *pitch*(Θ), *roll*(Φ) and *yaw*(Ψ) angles. The system will return the absolute direction of the device. To solve this problem there are many possible mathematical solutions. When computational power is not an issue, often the *Madgwick Algorithm* is used. For this application purpose, a more basic system was enough. To understand the orientation estimation it is important to briefly explain how every single sensor works and where it fails. The accelerometer



(a) Accelerometer **(b)** Gyroscope **(c)** Magnetometer

Figure 5.1: Device sensors in 3D plane

measures the acceleration in three axes (*figure 5.1a*). It can be thought of as a mass leaned on three springs. And each spring represents one axis. Every spring's compression and expansion caused by the device movement, represent the value of the accelerometer in that specific axis. Therefore, in a static position, the accelerometer won't measure null values but the constant gravity force ($9.81^{m}/_{s} = 1G$). Depending on the system, the accelerometer can be required with these raw values, often with G values (that means to divide every axis by the gravity force) and sometimes without the static force (that

[14]During the explanation of the *Fusion Sensor* system, I will often use the terms "absolute" and "relative", regarding the sensors' measurement. With the first one, I mean one measurement in relation to the Earth coordinates system (that is the ultimate system for the Position Tracking), whilst, with the other one, a measurement without any relation to a bigger system, devoid of any relations in general apart from that one with the mobile device itself.

means to constantly subtract the orientation of the device to get only the device motions). The gyroscope measures the angular velocity in the same three axes (*figure 5.1b*): the device rotation rate on each axis defined in degrees per second ($^{deg}/_{s}$). And finally, the magnetometer measures the magnetic field, always about the same three axes (*figure 5.1c*). The value for each axis is defined in micro Tesla (μT).
The accelerometer returns a noisy measurement from each one of the three axes, caused by the gravity force. To eliminate this noise is quite easy, using an average filter or other low pass filters. However, this introduces a delay, reducing, drastically, the velocity of the system. On the other hand, the gyroscope does not have this kind of problem since it doesn't measure gravity, so its values are less noisy and it has a fast response. However the gyroscope is quite useless, considering that it measures only the rotation velocity (as it will be shown soon, to get the position by the velocity it brings other errors). Indeed, almost ever, it is used together with the accelerometer or magnetometer. The magnetometer is maybe the most noisy sensor since nowadays pollution of magnetic fields and the one of the smartphone itself have a big impact on the measurement result.
We started from the gyroscope data considering that it measures the velocity of the orientation changes in the three different axes. Indeed, if the integration of each angular velocity would be calculated, the relative device orientation in relation to the 3-dimensional plane should be obtained. That means, it should return the *pitch*, the *roll* and the *yaw* angles. However, despite its accuracy, a noisy signal, even minimal, returns an increasing drift when integrated causing an always increasing error in the orientation estimation. This problem is really common in sensor usage, as it will be seen below with another example. The gyroscope is not a good starting point but the accelerometer is. Even if it doesn't directly return information about the orientation, it is possible to use the accelerometer to get the device's actual position in relation to the gravity force (and so, in relation to the ground). Therefore it is possible to derive at least two of the three AHRS angles without drift errors.
The device position in relation to the ground is obtained by calculating the angles between the vector force and the 3-dimensional plane. In the 3-dimensional plane the xy-plane is parallel to the ground and the z vector is perpendicular to it. The vector force (r) is calculated by the *Pythagorean Theorem* in which each of the three values calculated by the accelerometer represents the vector projection in each axis (r_x, r_y, r_z):

$$r = \sqrt{{r_x}^2 + {r_y}^2 + {r_z}^2} \tag{5.1}$$

The angles (a_x, a_y, a_z) formed between the angle force (r) and the three axes is obtained to calculate the arccosine of each projected vector (r_x, r_y, r_z) in

unity value (between -1 and 1):

$$\begin{aligned} a_x &= \arccos\left(\frac{r_x}{r}\right) \\ a_y &= \arccos\left(\frac{r_y}{r}\right) \\ a_z &= \arccos\left(\frac{r_z}{r}\right) \end{aligned} \tag{5.2}$$

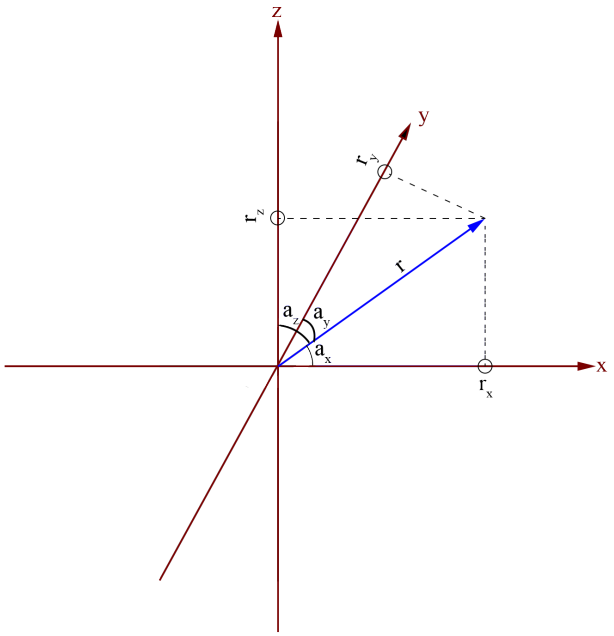
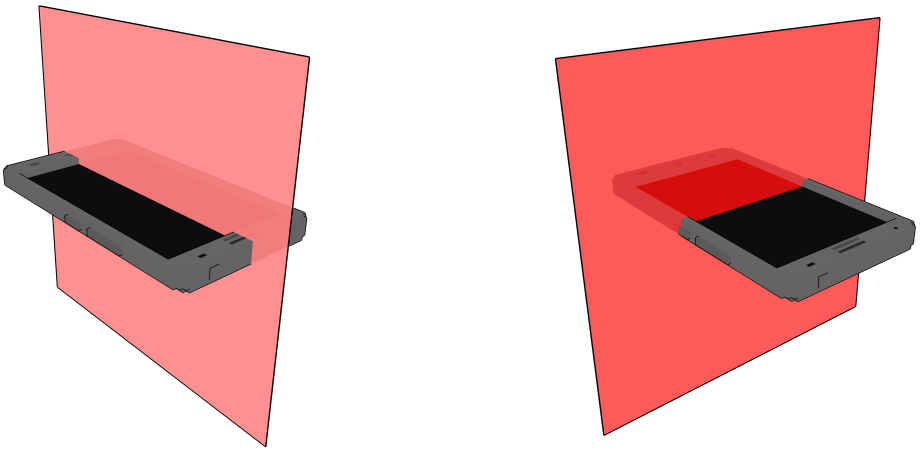


Figure 5.2: Accelerometer position in relation to the 3-dimensional axes

With this method, two problems arise. If one would consider these calculations in a 3D simulation there would be unexpected behaviors. But, removing the z-axis is enough to observe a more clear behavior. Indeed, the angle a_x measures the rotation around the y-axis, that is the rotation of the xz-plane (*figure 5.3a*), while the angle a_y that one around the x-axis, that is the rotation of the yz-plane(*figure 5.3b*). But the a_z angle does not measure the rotation of the xy-plane (as one could imagine) but the up and down position of the device, so the entire 3-dimensional plane rotation. In other words, one can move the a_x angle without moving the a_y one and vice versa, but it is not possible to move the a_z angle without moving the a_x and a_y vectors and vice versa.
The second problem is that the arccosine of a vector doesn't measure an entire angle. Once the device has been revolved by 180° in the xy plane and if the user keeps rotating it will turn back toward the 90° instead to proceed toward the 270°. The solution lies at the a_z angle: depending on the sign of this angle, it says if the device is rotating between the first and the second quadrant (between 0° to 180°) or between the third or fourth quadrant (between 180° to 360°). At this point, a partial device orientation is achieved. Indeed, the rotation of the xz- and yz-plane represent respectively the *pitch*(Θ) and the



(a) xz-plane **(b)** yz-plane

Figure 5.3: Rotation planes derived from the accelerometer

roll(Φ) angles. Because of the noisy signal of the accelerometer, a step of filtering is required. And this leads to a delay in the data (in the 3D simulation, one should notice a significant slowdown of the system). To fix this problem the gyroscope has to be taken into account. The merging of the accelerometer with the gyroscope can be used to get the instant rotation of the planes, integrating the angular velocity as follows:

$$\alpha_n = \alpha_{n-1} + \omega \Delta T \tag{5.3}$$

Where α_n is one of the orientation angles (Θ, Φ or Ψ in degrees) in the instant (n), ω is the angular velocity ($^{deg}/_{s}$) and ΔT is the time interval (s). Consequently, it can be periodically updated by accelerometer data with which the drift error is cleared. However, while the *pitch* angle and the *roll* one can be updated by the accelerometer, the *yaw* angle (the rotation of the xy-plane) continues to increase the drift error. This is because, as has been seen, the accelerometer doesn't calculate this angle. Therefore, we have to merge the system accelerometer-gyroscope with the magnetometer to obtain the absolute orientation. The absolute direction of the device (the north) is the *yaw* angle calculated as the inverse tangent between the magnetism measured in the x-(m_x) and y-axis (m_y) of the magnetometer sensor.

$$H = \arctan 2(m_y, m_x) \tag{5.4}$$

However, this is not the same north of the Earth coordinates system. The *True North*, or *Geodetic North*, represents the rotating axis of the Earth and is where lines of longitude (meridians) converge. The *Magnetic North* (shown by the magnetometer) lies $500km$ away from the *True North*. The difference between the compass result and the *True North*, called the magnetic inclination, change in every part of the globe. To remove this error we have to add the *magnetic inclination* to the compass result. The inclination can be cal-

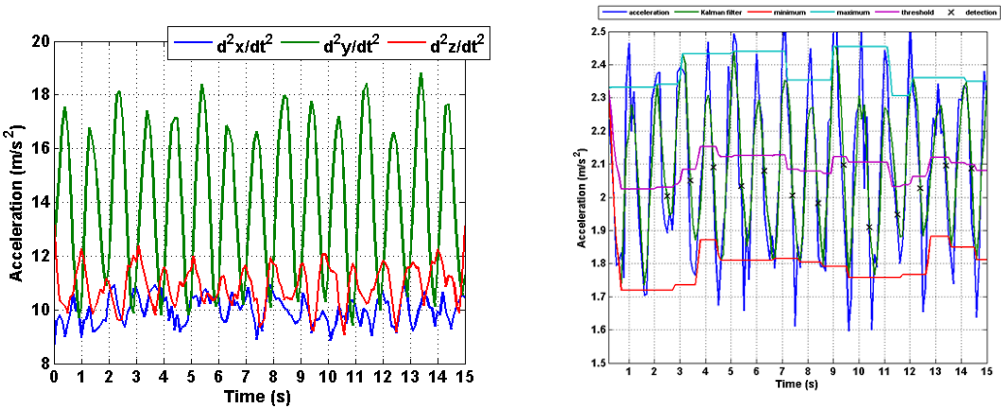
culated in different manners, starting by the user's coordinate position. We finally filter the magnetometer's noisy signal using the accelerometer values to periodically update the gyroscope yaw calculation.

GPS data gaps and *Pedometer*

Once the absolute direction is obtained, we can interpolate the data to fill the gaps between GPS' coordinates. The accelerometer would seem the best solution to accomplish this step. Indeed, having the orientation would be enough to get the acceleration in that direction, integrate it to obtain the velocity, and then integrate it another time to finally produce the space covered. However the double integration of noisy data produces a considerable drift.
To solve this issue we then used a method called *Pedometer*. A *Pedometer* is often used in "health management" and "gym" applications, with the count of uses up calories, the weight and much more. The main functionality is to count the walking steps but it also provides the distance, speed and so on. The core of the *Pedometer* is the step counter. It is achieved by measuring the ongoing accelerations and decelerations to which a smartphone is subjected during the walking action. This movement is recorded by the accelerometer. In this case, we can dispense the device orientation and work only with the normal of the acceleration vector (r in the *equation 5.1*). Plotting this we can exactly see what one would expect (*figure 5.4a*). The recorded data is a series of peaks (accelerations) and valleys (decelerations) in the graph. A pseudo-period of this signal can be thought of as a step. As proposed by Sébastien Ménigot in *Pedometer in HTML5* a *Kalman filter* is added to improve the robustness of detection by applying a denoising process.

$$\begin{aligned} G_t &= P_{t|t-1}H_t^T(H_tP_{t|t-1}H_t^TR_t)^{-1} \quad \text{Optimal Kalman gain} \\ \hat{a}_{t|t} &= \hat{a}_{t|t-1} + G_ta_t \quad \text{Updated acceleration estimated} \\ P_{t|t} &= P_{t|t-1} - K_tH_tP_{t|t-1} \quad \text{Updated covariance estimated} \end{aligned} \tag{5.5}$$

where $P_{t|t-1}$ is the error covariance matrix, $\hat{a}_{t|t}$ the signal estimated at time t, H_t the matrix of the estimation model and R_t the covariance of the noise measured. Even if this equation is described for the acceleration matrix, it can be applied also in the single acceleration vector to reduce the computational cost. The step detection is done on this denoised signal.
Every step is acquired when the signal cuts a threshold. This threshold is constantly updated, and it corresponds to the average of the extreme acceleration values on a window of two seconds. In addition to that other conditions are added to avoid false predictions such as a minimum estimation time ($200ms$) and an inferior and superior limit. These two last conditions tell that a step is counted only if the signal reaches these two values before cutting the threshold. Furthermore, these two limits are updated in relation to the main threshold (*figure 5.4b*)[Mén].



(a) (b)

Figure 5.4: Accelerometer in the *Pedometer*

With the merging of the *Pedometer* and the device orientation, the advancement towards an absolute direction (that means in relation to the *True North*) is obtained step after step. Merging this with the GPS, we can inscribe every step into the Earth coordinates system through the equations:

$$\varphi_2 = \arcsin\left(\sin\varphi_1 \cdot \cos\delta + \cos\varphi_1 \cdot \sin\delta\cos\theta\right) \tag{5.6}$$

$$\lambda_2 = \lambda_1 + \arctan 2(\sin\theta \cdot \sin\delta \cdot \cos\varphi_1, \cos\delta - \sin\varphi_1 \cdot \sin\varphi_2) \tag{5.7}$$

where φ_1 is the starting latitude, λ_1 is the starting longitude, θ is the heading, δ is the angular distance. The angular distance is calculated as:

$$\delta = \frac{d}{R} \tag{5.8}$$

where d is the distance (the step size) and R is the Earth radius ($6371km$)[Ven20]. So, the latitude (φ_2) and the longitude (λ_2) are calculated starting from the GPS data, if it is available, otherwise from the previous step. However, the first absolute value which enters in this process must be a latitude and longitude given by the GPS. Here, the only uncertain variable is the step size. With a value between 50 and 70 cm, and with a maximum delay of 30 seconds for the GPS data receiver, the error is contained between one meter and a maximum of three meters. And this is an acceptable value for the application requirements.

Implementation

Capacitor provides a plugin that interfaces with the motion sensors of smartphones. It is called *Motion*. Another set of tools, named *Sensors*, makes it possible to manage the entire set of smartphones' sensors. However, in both options, a clear tool for magnetometer usage is missing. Furthermore, with *Sensors* only one sensor at once can be used, and this is particularly strange.

Only with intricate solutions, this problem can be circumnavigated. For this reason, a less complex and more efficient option is the usage of an external plugin. The plugin is *eMagnetometer* that is provided by Cordova[15]. and can be easily used inside the *Capacitor*'s environment. *eMagnetometer* plugin provides the raw data about the magnetometer in the three axes. The *Motion* plugin inside *Capacitor* can provide the accelerometer with or without gravity and the rotation rate (the gyroscope). An example is given by the following code. This is a simple *Angular* service that provides the data of the device motions for the whole application.

```
1  import { Injectable, NgZone, OnInit } from '@angular/core';
2  import { Plugins, MotionOrientationEventResult, MotionEventResult } from
       '@capacitor/core';
3
4  const{ Motion } = Plugins;
5
6  @Injectable({
7    providedIn: 'root'
8  })
9
10 export class MotionData implement OnInit{
11   constructor(private zone: NgZone) {}
12   //Accelerometer with Gravity
13   public accelRaw = {x:NaN, y:NaN, z:NaN};
14   //Accelerometer without Gravity
15   public accelG = {x:NaN, y:NaN, z:NaN};
16   //Gyroscope
17   public gyro = {alpha:NaN, beta:NaN, gamma:NaN};
18
19   private watchAccelerometer(){
20     return Motion.addListener('accel', (data: MotionEventResult) => {
21   this.zone.run(() => {
22     // read the accel with G
23           this.accelRaw.x = parseFloat(data.accelerationIncludingGravity.
                 x.toFixed(4));
24           this.accelRaw.y = parseFloat(data.accelerationIncludingGravity.
                 y.toFixed(4));
25           this.accelRaw.z = parseFloat(data.accelerationIncludingGravity.
                 z.toFixed(4));
26     // read the accel without G
27           this.accelG.x = parseFloat(data.acceleration.x.toFixed(4));
28           this.accelG.y = parseFloat(data.acceleration.y.toFixed(4));
29           this.accelG.z = parseFloat(data.acceleration.z.toFixed(4));
30     // read the gyroscope
31           this.gyro.alpha = parseFloat(data.rotationRate.alpha.
                 toFialphaed(4));
32           this.gbetaro.beta = parseFloat(data.rotationRate.beta.
                 toFialphaed(4));
33           this.gbetaro.gamma = parseFloat(data.rotationRate.gamma.
                 toFialphaed(4));
34       });
35     });
36   }
37   private start(){
38     this.watchAccelerometer()
39   }
40   ngOnInit(){
41     start()
42   }
43 }
```

Listing 5.1: The *Motion* plugin usage

[15]https://cordova.apache.org/

This code provides three objects with the accelerometer and the gyroscope data. These can be "injected" on all pages of the application thanks to the service system of the *Angular* framework[16]. So it is possible to simultaneously use the same data for an interactive map, for the audio processing and still for other tasks. A sketch of usage on any page is the following:

```
import { MotionData } from 'the/path/of/the/service'

export class Page{
  constructor(private motion: MotionData) {
    console.log(this.motion.accelRaw.x)
  }
}
```

Listing 5.2: Recalling the service in an other page

For the usage of the *eMagnetometer* plugin, it is needed a synchronization between the plugin and *Capacitor*. And it is done by only typing the command `ionic cap sync` on the terminal, inside the workflow path of the application. Furthermore, the plugin has to be imported as a provider (that means, in summary, to import as a global service) and is done inside the application's module declaration. After the necessary steps are done it is possible to add the magnetometer in the previous code. Importing the *eMagnetometer* as an external service in the importing section:

```
import { Magnetometer, MagnetometerReading } from '@ionic-native/
    magnetometer/ngx';
```

Listing 5.3: Import the *eMagnetometer*

adding the plugins in the constructor, always as a service:

```
constructor(
  ...
  private magnetometer: Magnetometer
){}
```

Listing 5.4: the *eMagnetometer* in the service constructor

And then adding a public object to share with other pages and the function with which the magnetometer is read:

```

watchMagnetometer(){
  return this.magnetometerWatch = this.magnetometer.watchReadings()
    .subscribe( (data: MagnetometerReading ) => {
  this.magne.x = data.x;
        this.magne.y = data.y;
        this.magne.z = data.z;
    });
}
```

Listing 5.5: *eMagnetometer* reading

[16] `ngZone` is another service provided natively by *Angular*. The most common use of this service is to optimize performance when starting a work consisting of one or more asynchronous tasks that don't require UI updates or error handling to be handled by *Angular*. `OnInit` is another *Angular* service that is called when a page is loaded.

Finally, this function has to be called inside the start function:

```
private start(){
  ...
  this.watchMagnetometer()
}
```

Listing 5.6: *eMagnetometer* in the `start()` function

At this point, the GPS is the only missing tool. However, *Capacitor* provides the *Geolocation* plugin with which it is possible to accede to the smartphone's GPS localization. The usage of this plugin is simple like the previous ones:

```
const{ Motion } = Plugins;
const{ Geolocation } = Plugins;
...
public gps = {lat:NaN, lon:NaN} //the gps object
// read GPS function
private watchGPS(){
  return Geolocation.watchPosition({enableHighAccuracy:true}, (data, err)
       => {
    this.gps.lat = data.coords.latitude;
    this.gps.lon = data.coords.longitude;
  });
}
//call it in the start function
private start(){
  ...
  this.watchGPS()
}
```

Listing 5.7: the *Geolocation* plugin

Now, with the necessary tools, their merging has to be computed. To obtain an AHRS system, the accelerometer, the gyroscope and the magnetometer must be joined together. The *Motion* plugin already provides the merging between accelerometer and gyroscope which works well enough.

```
public orient = {alpha:NaN, beta:NaN, gamma:NaN};

watchOrientation(){
  return Motion.addListener('orientation', (data:
      MotionOrientationEventResult) => {
    this.zone.run(() => {
    this.orient.alpha = parseFloat(data.alpha.toFixed(4));
    this.orient.beta = parseFloat(data.beta.toFixed(4));
    this.orient.gamma = parseFloat(data.gamma.toFixed(4));
    });
  });
}
```

Listing 5.8: the *Motion* plugin's orientation

This is a no-absolute orientation. Therefore the absolute heading has to be calculated with the magnetometer.

```
//the heading\
public heading:number = NaN;
//from rad to deg
private deg:number = 180 / Math.PI;

compass(){
  this.heading = -1 * (90 - (Math.atan2( this.magne.y, this.magne.x ) *
      this.deg));
  if(this.heading < 0){
    this.heading = 360 + this.heading;
```

```
10    }
11    this.heading = ( this.heading + this.declination ) %360;
12  }
```

Listing 5.9: the `compass()` dunction

Considering the declination already calculated[17] the the xy-plane orientation is obtained, as explained above. Then the heading of the *Motion* orientation can be periodically corrected through the following function.

```
hstep:number = 0;
refreshCompass:number = 100;
previousOrientX:number = NaN;
getHeading(){
  if (this.hstep == 0){
    this.prevOrientX = this.orient.x;
    this.compass();
    this.hstep++;
  } else {
    this.heading = ( this.heading + ( this.orient.x - this.prevOrientX )
        ) % 360;
    this.prevOrientX = this.orient.x;
      if (this.hstep == this.refreshCompass){
  this.hstep = 0;
      } else {
     this.hstep++
      }
    }
}
```

Listing 5.10: The merging of the compass with the no-aabsolute orientation

And this function has to be called at regular intervals in a *JavaScript* interval function:

```
interval = setInterval(() => {
    this.getHeading();
}, intervalTime)
```

Listing 5.11: Orientation correction in an interval function

Based on the `intervalTime` variable (in milliseconds) and the `refreshCompass` one (which allows refreshing the compass every time the `hstep` variable reaches its value) the heading will be corrected in regular intervals equals to a quantity of millisecond obtained by the multiplication between these variables.This correction can generate radical leaps due to the errors in the orientation calculated by the Motion plugin which are unmanageable and may significantly detach from the real heading calculated by the compass. And also the errors produced by the magnetometer are important. To avoid these problems, an averaging filter can be added to both layers, with the proper and right tweaks. Having this basilar AHRS system, the *Pedometer* and the GPS have to be merged to complete the motion tracking system.
As regards the *Pedometer*, some smartphones have an internal system already capable of counting steps. Apple smartphones especially have such a system. And it can be simply used through the *Pedometer* plugin provided by *Cordova*. However, many smartphones don't have this system. For this reason, a *Pedometer* has to be coded to be used with only the accelerometer. In this

[17]It is done by an *HTTP request* through the link https://www.ngdc.noaa.gov. Specifying in the link the latitude and the longitude, the request returns the declination value.

work, I have rearranged that one proposed by Sébastien Ménigot in *Pedometer in HTML5*, which I have already mentioned. From this algorithm I have extracted only the step detector, leaving out all the other functionalities.
According to the *equation 5.5*, the *Pedometer*'s *Kallman filter* is coded as follows:

```
function Kalman() {
  this.G  = 1; // filter gain
  this.Rw = 1; // noise power desirable
  this.Rv = 10; // noise power estimated
  this.A = 1;
  this.C = 1;
  this.B = 0;
  this.u = 0;
  this.P = NaN;
  this.x = NaN; // estimated signal without noise
  this.y = NaN; //measured

  this.onFilteringKalman = function(ech){
    //signal: signal measured
    this.y = ech;

    if (isNaN(this.x)) {
      this.x = 1/this.C * this.y;
      this.P = 1/this.C * this.Rv * 1/this.C;
    }
    else {
    // Kalman Filter: Prediction and covariance P
    this.x = this.A*this.x + this.B*this.u;
    this.P = this.A * this.P * this.A + this.Rw;
    // Gain
    this.G = this.P*this.C*1/(this.C*this.P*this.C+this.Rv);
    // Correction
    this.x = this.x + this.G*(this.y-this.C*this.x);
    this.P = this.P - this.G*this.C*this.P;
    };
    return this.x;
  };
  this.setRv = function(Rv){
    //signal: signal measure
    this.Rv = Rv;
  };
}
```

Listing 5.12: the Kalman filter

While the entire system of the *Pedometer* is the following one:

```
function Pedometer() {
  this.acc_norm = new Array(); // amplitude of the acceleration
  this.var_acc   = 0.; // variance of the acceleration on the window L
  this.min_acc   = 1./0.;  // minimum of the acceleration on the window L
  this.max_acc   = -1./0.; // maximum of the acceleration on the window L
  this.threshold = -1./0.; // threshold to detect a step
  this.sensibility = 1./30.;  // sensibility to detect a step
  this.countStep = 0;           // number of steps
  this.stepArr   = new Array(); // steps in 2 seconds

  this.stepSize  = 50; // step size of the pedestrian (cm)
  this.distance = 0;
  this.filtre = new Kalman();

  this.setCountStep = function(count) {
    this.countStep = count;
  };

  this.setStepSize = function(stepSize) {
    this.stepSize = stepSize;
```

```
21    };
22
23    this.setSensibility = function(sensibility) {
24      this.sensibility = sensibility;
25    };
26    // initialization of arrays
27    this.createTable = function(lWindow) {
28      this.acc_norm = new Array(lWindow);
29      this.stepArr = new Array(lWindow);
30    };
31
32    this.update = function() {
33      this.acc_norm.shift();
34    };
35    // compute norm of the acceleration vector
36    this.computeNorm = function(x,y,z) {
37      var norm = Math.sqrt(Math.pow(x,2)+Math.pow(y,2)+Math.pow(z,2));
38      var norm_filt = this.filtre.onFilteringKalman(norm);
39
40      return norm_filt/9.80665;
41     };
42      // seek variance
43    this.varAcc = function(acc) {
44      var moy  = 0.;//mean
45      var moy2 = 0.;//square mean
46      for (var k = 0; k < acc.length-1; k++) {
47        moy += acc[k];
48        moy2 += Math.pow(acc[k],2);
49      };
50      this.var_acc = (Math.pow(moy,2) - moy2)/acc.length;
51      if (this.var_acc - 0.5 > 0.) {
52        this.var_acc -= 0.5;
53      };
54      if (isNaN(this.var_acc) == 0) {
55        this.filtre.setRv(this.var_acc);
56        this.setSensibility(2.*Math.sqrt(this.var_acc)/Math.pow(9.80665,2))
              ;
57      }
58      else {
59        this.setSensibility(1./30.);
60      };
61    };
62    // seek minimum
63    this.minAcc = function(acc) {
64      var mini = 1./0.;
65      for (var k = 0; k < acc.length; k++) {
66        if (acc[k] < mini){
67    mini = acc[k];
68        };
69      };
70      return mini;
71    };
72
73    // seek maximum
74    this.maxAcc = function(acc) {
75      var maxi = -1./0.;
76      for (var k = 0; k < acc.length; k++) {
77        if (acc[k] > maxi){
78    maxi = acc[k];
79        };
80      };
81      return maxi;
82    };
83
84    // compute the threshold
85    this.setThreshold = function(min, max) {
86      this.threshold = (min+max)/2;
87    };
88
89    // detect a step
```

```
90    this.onStep = function(acc) {
91      this.varAcc(acc);
92      this.min_acc = this.minAcc(acc);
93      this.max_acc = this.maxAcc(acc);
94
95      this.setThreshold(this.min_acc, this.max_acc);
96
97      var diff = this.max_acc - this.min_acc;
98      // the acceleration has to go over the sensibility
99      var isSensibility   = (Math.abs(diff) >= this.sensibility)
100     // if the acceleration goes over the threshold and the previous was
            below this threshold
101     var isOverThreshold = ((acc[acc.length-1] >= this.threshold) && (acc[
            acc.length-2] < this.threshold));
102     var isValidStep     = (this.stepArr[this.stepArr.length-1] == 0);
103     if (isSensibility && isOverThreshold && isValidStep) {
104       this.countStep++;
105       this.stepArr.push(1);
106       this.stepArr.shift();
107     } else {
108       this.stepArr.push(0);
109       this.stepArr.shift();
110     };
111   };
112   this.setDistance = function() {
113     this.distance = this.countStep * this.stepSize;//cm
114   };
115 }
```

Listing 5.13: the *Pedometer*

This is a common *JavaScript* library, but as already mentioned, it can be simply used in an *Ionic* environment and through *TypeScript* language, without any limitation. In an *Angular* project like this one, it can be added in the *Angular* workspace configuration (*angular.json*) under the voice `''scripts''`, along with other libraries or mere scripts.

```
1  "scripts": [
2    "./jslib/pedometer.js",
3    ...
4  ]
```

Listing 5.14: declaration of the library in the *Angular.json* file

And then it has to be declared the constructor of the library (in this case `Pedometer`) on the page where it has to be used.

```
1  declare const Pedometer;
```

Listing 5.15: declaration on the page

Creating a new `Pedometer` object inside the page with the proper set up (declaring the `sensibility` and the `stepSize`), the library is ready to use:

```
1  private pedo = new Pedometer();
2  private sensibility:number = 1/20;
3  private stepSize:number = 70; //cm
4
5  this.pedo.sensibility = this.sensibility;
6  this.pedo.sensibility = this.stepSize;
7
8  public setp:number = NaN;
```

Listing 5.16: `Pedometer` initialization

To use the *Pedometer*, the norm of the accelerometer has to be calculated at regular intervals. Because of that, another *JavaScript* interval function (or the same of the code 5.11) has to be called. And after creating a table through the `createTable()` method (that is the initialization) inside the interval function the *Pedometer* acts, as shown in the following code.

```
1  private sr = 100; //Sample rate HZ
2  private pms = 1000 / this.sr; //period ms
3  private ps = 1 / this.sr // period in s
4
5  private start(){
6    ...
7    this.pedo.createTable(Math.round(2 / this.ps));
8    interval = setInterval(() => {
9      let norm = this.pedo.computeNorm(this.accelRaw.x, this.accelRaw.y,
           this.accelRaw.z);
10     this.pedo.acc_norm.push(norm);
11     this.pedo.update();
12     this.pedo.onStep(this.pedo.acc_norm);
13     if(this.steps < this.pedo.countStep){
14       // here update GPS point
15       this.steps = this.pedo.countStep
16     }
17   }, this.pms)
18 }
```

Listing 5.17: `Pedometer` interval

After the condition `if`, at *line 13*, the actual GPS point will be updated. The coordinates will be acquired at this exact point, except when the GPS is available. Furthermore, the GPS has to acquire the coordinate at least one time before the initialization of the *Pedometer*. This will allow to calculate the step coordinates starting from an absolute point. The step coordinates computation is accomplished by the following function, which is the merging between the *Pedometer* and the device orientation:

```
1  private rad = Math.PI / 180; //deg to rad
2  private R = 6371 * 1000 //Earth radious in m
3
4  updatePoint(){
5    let lat1 = this.gps.lat * this.rad;
6    let lon1 = this.gps.lon * this.rad;
7    let h = this.heading * this.rad;
8    let theta = this.stepSize * 0.01 / this.R; //angular dist
9    let lat2 = Math.asin(Math.sin(lat1) * Math.cos(theta) + Math.cos(lat1)
         * Math.sin(theta) * Math.cos(h))
10   let lon2 = lon1 + Math.atan2(Math.sin(h)*Math.sin(theta)*Math.cos(lat1)
         , Math.cos(theta)-Math.sin(lat1)*Math.sin(lat2));
11   this.gps.lat = lat2 * this.deg;
12   this.gps.lon = lon2 * this.deg;
13 }
```

Listing 5.18: `updatePoint()` function

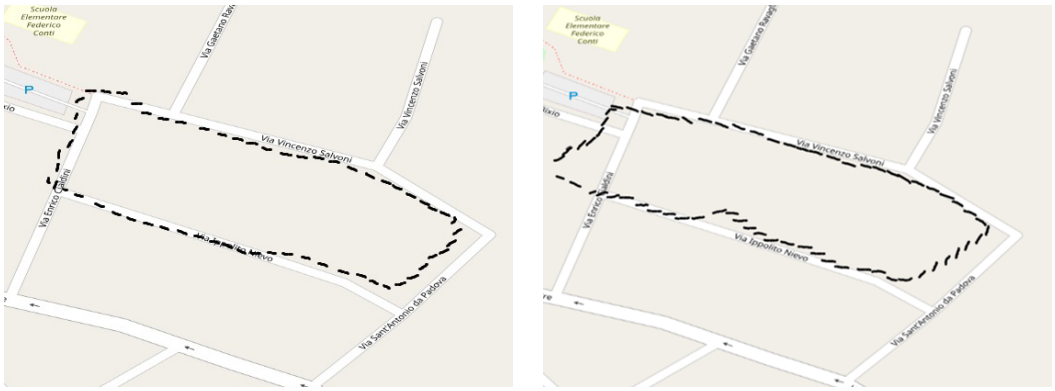
That is nothing but the computation of the *equations 5.6*, *5.7* and *5.8* mentioned above.
In summary, the `start()` function, which describes the entire workflow of the motion tracking system, can be recapped as follows:

```
1  start(){
2    // start sensor reading
3    this.watchOrientation();
4    this.watchAccelerometer();
```

```
5     this.watchMagnetometer();
6     this.watchGPS();
7     // initialize the pedometer
8     this.pedo.createTable(Math.round(2 / this.ps));
9     // start the process
10    interval = setInterval(() => {
11      // call the heading function
12      this.getHeading();
13      /*
14      * chek if the gps coordinates have been acquired
15      * at least one time before starting with the pedometer
16      */
17      if (this.gps.lat != null && this.gps.lon != null) {
18        // Compute the norm (the vector force)
19        let norm = this.pedo.computeNorm(
20    this.accelRaw.x,
21    this.accelRaw.y,
22    this.accelRaw.z);
23        // Update the pedometer with the norme
24        this.pedo.acc_norm.push(norm);
25        this.pedo.update();
26        this.pedo.onStep(this.pedo.acc_norm);
27        /* check if a new step is detected
28         * if there is a new step
29         * update the gps point
30         */
31        if(this.steps < this.pedo.countStep){
32    this.updatePoint()
33    this.steps = this.pedo.countStep
34        }
35      }
36    }, this.pms)
37  }
```

Listing 5.19: the `start()` function

Such an algorithm is quite dependable. However, even if the GPS gaps are covered, this system may have still residual ones, much smaller. These gaps may occur every time the coordinates are acquired by the GPS. They exactly occur between the path computed by the *Pedometer* and the new GPS coordinates. The uncertainty of the *Pedometer*'s step size added to the GPS's error on the absolute position estimation is the cause of these gaps. With a good GPS connection, which means a low average time interval between the acquired coordinates (around five seconds), only neglectable errors are produced. With a less performant GPS, but always under 30 seconds on the interval between the coordinates, the errors remain around 1 meter. In the worst cases, they can reach three meters. With a delay bigger than 30 seconds the errors are unpredictable. The application is provided with a timeout which is nothing but a built-in method of the *Geolocation* plugin. This method allows calling an error function when the GPS does not receive any data for a time higher than an established one. For the application, once the GPS is frozen for more than 30 seconds, the user will receive an alert and the application will get in a sleep state until when the GPS is available again.
Some records of this algorithm performance can be seen in the following figures. In the area in which these records have taken place: Jesi - a middle sized city in the center of Italy, the interval time of the GPS data receiving is around 15 seconds.



sensibility = 1/10
stepSize = 50 //cm

sensibility = 1/8
stepSize = 70 //cm

Figure 5.5: Position Tracking test. The black lines are a group of points which represent every step

6 3D Audio System

The position tracking system is in close relation with the 3D audio system as we've seen. For this application, I did not build a complete 3D audio system. In fact, it would make little sense to provide the system of an elevation plane. That means allowing the differentiation between vertical planes, between above and below. As well as the difficulties to build a reliable vertical sound spatialization in a 3D audio system.
There are many ways to realize a 3D audio system and each one has its strengths and its weaknesses. In this work three different binaural algorithms have been taken into account: the *Duplex Algorithm*, the *Quasi-Specific Head Related Transfer Function Algorithm* and the *General HRTF Algorithm*. To select the most attendible for the user, a human listening test has been done.

6.1 Duplex Algorithm

The realization of the *Duplex Algorithm*, based on the *Duplex Theory*, which describes only the horizontal plane, is based on the wavefront differences in the two ears, e.g. the difference of intensity and time between the same sound event reaching the left and the right ear. These two differences are the *Interaural Intensity Difference* (IID) and the *Interaural Time Difference* (ITD). This is the base of every binaural system because it describes the main functionalities of the human's localization system[Beg94]. The *Duplex Theory* describes an ideal and unreal scenario: the model, by which these differences are calculated, is a perfect sphere rather than a human head. With the sphere approximation, important details, such as the filtering happening because of the human ear shape (the auricle, the ear canal and the eardrum), the shoulders are lost. We consider the sound on the left side of the sphere with an angle between 0° and -180° and on the right side of the sphere with an angle between 0° and 180° (*figure* **??**).

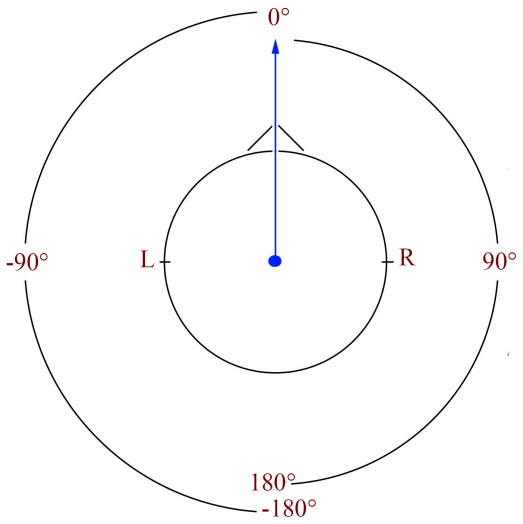


Figure 6.1: Spheric head model

With the ITD, the arrival times of the same sound in the two ears are calculated. The *Angle of Incidence* function with which the *long-path wave* is calculated. The *long-path wave* is the path that the sound covers to reach the more distant ear.

$$D = r \cdot \theta + \sin(\theta) \tag{6.1}$$

Where D is the *long-path wave*, r is the head radius (in meter) and the θ is the azimuth angle between the sound and the spheric head (in radiants). Therefore it is possible to obtain the ITD by the following equation:

$$\Delta t = \frac{D}{c} \tag{6.2}$$

Where Δt is the time (in seconds) and c is the sound velocity in the air ($\simeq 343 m/s$)

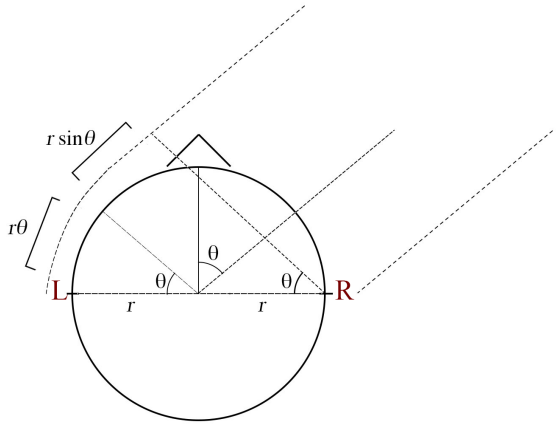


Figure 6.2: Angle of Incidence

On the other hand, to calculate the IID, the sphere shall be imagined as a low-pass filter. This filter would act only for sounds with a wavelength smaller than the diameter of the sphere. The sounds with the same or bigger wavelength will be refracted around it.
One of the best-known models is that one described in *A structural Model for Binaural Sound Synthesis*(1998) by C. Phillip Brown and Richard O. Duda. This model is realized with a 1st order linear filter:

$$H_{HS}(\omega,\theta) = \frac{1 + j\frac{\alpha\omega}{1\omega_0}}{1 + j\frac{\omega}{1\omega_0}}, \quad 0 \leq \alpha(\omega) \leq 2 \tag{6.3}$$

Where the frequency ω_0 depend on the head diameter:

$$\omega_0 = \frac{c}{d} \tag{6.4}$$

While the coefficient α is in relation to the θ angle which defines the localization of the zero in the filter. Therefore, there will be an increment of $6dB$ if $\alpha = 2$ whilst a cut off if $\alpha < 1$.

$$\alpha(\theta) = \left(1 + \frac{\alpha_{min}}{2}\right) + \left(1 - \frac{\alpha_{min}}{2}\right)\cos\left(\frac{\theta}{\theta_{min}}180^\circ\right) \tag{6.5}$$

To obtain a good approximation of the IID, the authors, Brown and Duda, suggest the following value for the previous equation: $\alpha_{min} = 0.1, \quad \theta_{min} = 150$[BD98].The filter just proposed it acts more like a particular shelving filter than a common low-pass filter.
However, this model does not take into account the elevation position of the sound and it does especially not differ between front and rear positions. A model proposed in *Stereophonic Sound Recording* by Christian Hugonnet and Pierre Walder tries to define a standard behavior in the elevation plane. They determine three areas, in relation to the sphere, (front, up and rear) for which it is given an increment of specific spectral regions (respectively to the three areas, the increments are around $3kHz$, $8kHz$, and $1kHz$)[HW97]. They don't specify how to apply these increments, but the effect can be tasted with the usage of three peak filters, one for each area. The results are too weak for the front-rear differentiations.
To optimize this algorithm I have introduced a small block to it, with the aim to differentiate between front and rear positions without scientific verification. This block is going to join with the *Duplex Theory*, adding a third ear to the model. The third ear is placed at 180° from the sphere's point of view, exactly in the rear. So, this ear has to be considered completely deaf when the sound is in front of the sphere; it will become sensible once the sound will pass ±90° (that is the right or left ear); more the sound is closed to the *180* (the absolute rear position) and more the ear will be sensible. The third ear perception of sound is to be considered only for the refracted frequencies. That means only for those frequencies with a wavelength bigger than the head diameter. The smaller is the wavelength the smaller is the intensity acquired. Therefore, the

third ear implements a basic 1st-order low-pass filter such the following one, described by the transfer function:

$$H(z) = \frac{1}{2}\left(1 + \frac{z^{-1} + f_c}{1 + f_c z^{-1}}\right) \tag{6.6}$$

Where the f_c can be understood as the coefficient related to the frequency cut, and therefore to the head diameter:

$$f_c = \frac{\tan(\omega_0) - 1}{\tan(\omega_0) + 1} \tag{6.7}$$

And the ω_0 as explained in *equation 6.4*.
In this way a sort of *Rear Intensity Difference* (RID) is calculated. Furthermore, when the sound is in the rear position, the third ear will acquire it before the left and right ears do. Introducing a further delay in the system that can be named *Front Time Difference* (FTD). And this is computed one more time with the *angle of Incidence* function. Where the θ is 90° when the sound is completely in the rear position (±180°), and it goes down zeros when the sound approaches one of the real ear (±90°). And that means that only when $\theta < -90$ or $\theta > 90$ the angle in the FTD is calculated as $\theta_{FTD} = |\theta| - 90$. In other words, this block is nothing but an additional implementation of the *Duplex Theory* with an additional and imaginary ear.
Therefore, the resulting signal of the third ear (RID) will be mixed with the output of the standard *Duplex Theory* and the estimated delays (FTD) will be added to both of the two ITDs. The resulting system can be described by the diagram in *figure 6.3*.

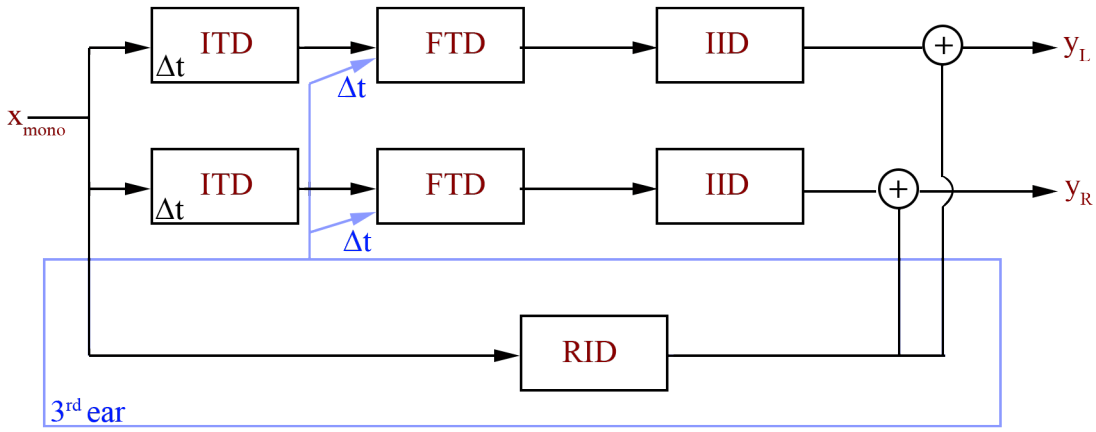


Figure 6.3: Diagram: *Duplex Theory* + 3rd Ear

Implementation

In general, the implementation of such a system is quite straightforward when using the lower-level coding languages. In the case of higher-level languages, such as *Web Audio API*, the implementation becomes more intricate. As has already been seen, the problem in *Web Audio API* is the scarce list of nodes and the difficulties to implement custom ones. With the `AudioWorklet` node such a filter is realizable, but the low sound quality and the computational work produced by this node makes this possibility unacceptable. The best solution is to approximate the filter behavior with an already provided node. For this reason, I realized the IID with a shelving filter, the `'highshelf'` filter mode provided by the `BiquadFilterNode`.

```
1  fc = 343 / head_diameter;
2
3  iidLeft = audioContext.createBiquadFilter();
4  iidRight = audioContext.createBiquadFilter();
5  iidLeft.type = 'highshelf';
6  iidRight.type = 'highshelf';
7  iidLeft.frequency.value = fc;
8  iidRight.frequency.value = fc;
```

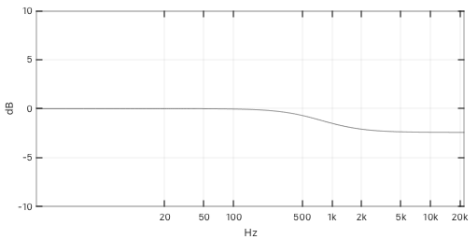
Listing 6.1: IID with a Shelving Filter

Thus, the IID is realized just handling the filter's gain on the base of the sound position in the horizontal plane (the azimuth angle). As shown in the following code:

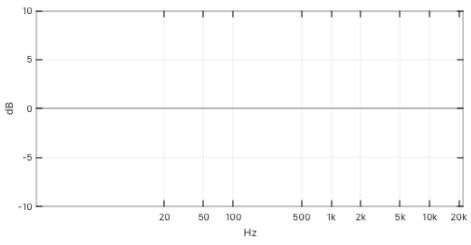
```
1  iid(azimuth:number){
2          //azimuth 0 <right> 180
3    //azimuth 0 <left> -180
4    if(azimuth>90){azimuth = 180 - azimuth}
5    else if (azimuth<-90){azimuth= -180 - azimuth}
6    iidL = ((azimuth/90) * 6);
7    iidR = ((azimuth/90) * -6);
8    iidLeft.gain.setTargetAtTime(iidL, audioContext.currentTime, 0.05)
9    iidRight.gain.setTargetAtTime(iidR, audioContext.cntx.currentTime,
         0.05)
10 }
```

Listing 6.2: IID with a Shelving Filter

The behavior of this implementation differs from the *Brow-Duda filter*, as shown in the figures below. Especially for the front sounds and positions on the opposite side of the ear for which the IID is calculated. Nevertheless, it mimics quite well its functionalities in broad terms.



Duplex filter: 0°



Sehlving filter: 0°

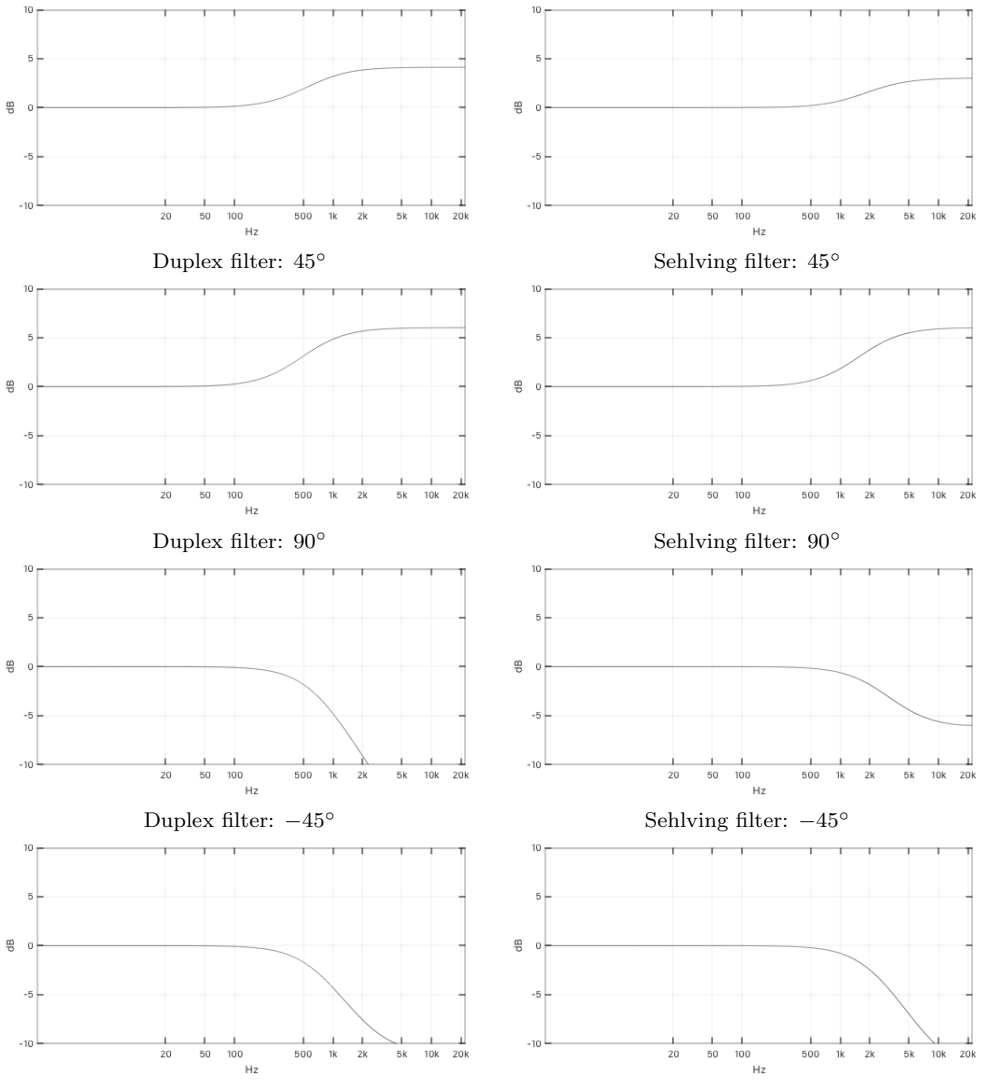


Figure 6.4: Differences between the Shelving and Duplex filter

The ITD and the FTD can be calculated as a single delay node for each ear and it can be easily executed by the `DelayNode`. Instead, the RID can be executed by the `'lowpass'` filter mode provided one more time by the `BiquadFilterNode`. The code below shows the implementation of these three blocks through two main functions,`itd()` and `front()`. It is nothing but just the simple implementation of what has been explained in the first part of this chapter.

```
1  itdLeft = audioContext.createDelay();
2  itdRight = audioContext.createDelay();
3
4  ridRear = cntx.createBiquadFilter();
5  ridRear.type = "lowpass";
6  ridRear.frequency.value = fc; //fc = 343 / head_diameter
7  ridRear.Q.value = 2;
8
9  //the main control of rear sounds
10 //in -> RID -> gainRear -> out
11 gainRear = cntx.createGain()
12
```

```
13 //the main control of front sounds
14 //in -> ITD -> IID -> gainFront -> out
15 gainFront = cntx.createGain()
16
17 itd(azimuth:number, frontDel:number){
18         //azimuth 0 <right> 180
19   //azimuth 0 <left> -180
20   ////frontDel = front delay in s
21   theta = (azimuth/180)*Math.PI;
22   if(theta > 0){
23     theta = Math.abs(Math.PI/2 - Math.abs(Math.abs (theta) - Math.PI/2 ))
24     itdL = 0
25     itdR = (( head_radious * theta + Math.sin(theta))/ 343 );
26   } else {
27     theta = Math.abs(Math.PI /2 - Math.abs( theta + Math.PI /2 ));
28     itdL = (( head_radious * theta + Math.sin(theta))/ 343 );
29     itdR = 0
30   }
31   itdLeft.delayTime.setTargetAtTime(itdL + frontDel, audioContext.
         currentTime, 0.05)
32   itdRight.delayTime.setTargetAtTime(itdR + frontDel, auidoContext.cntx.
         currentTime, 0.05)
33 }
34
35 front(azimuth:number):number {
36         //azimuth 0 <right> 180
37   //azimuth 0 <left> -180
38   az = Math.abs(azimuth)
39   if (az > 90){
40     rear = (az-90)/90;
41     front = 1 - rear*0.8
42     theta = (az/180)*Math.PI - Math.PI/2;
43     frontDel = (( head_radious * theta + Math.sin(theta))/ 343 )
44     gainRear.gain.setTargetAtTime(rear*rearFator, audioContext.
           currentTime, 0.05)
45     gainFront.gain.setTargetAtTime(front*frontFactor, audioContext.
           currentTime, 0.05)
46     return frontDel;
47   } else {
48     gainRear.gain.setTargetAtTime(0, audioContext.currentTime, constant)
49     gainFront.gain.setTargetAtTime(1,audioContext.currentTime, constant)
50     return 0;
51   }
52 }
```

Listing 6.3: Implementation of the ITD, FTD and RID blocks

As it can be noticed, only two blocks have been added. They are the main controls for rear and front sounds. The `rearGain` node simply controls the gain of the RID filter. While the `frontGain` one controls the gain of the double chain of IID and ITD. Especially this part has not been explained above. With these two gain blocks, the intensity of differentiation between front and rear sounds can be handled, highlighting or smoothing the rear sound effect. The results produced by these blocks will be better discussed in the 3D audio systems test chapter.
To conclude the *Duplex Algorithm* implementation, the combination of each block has to be done by the connection between every node.

```
1 merger = audioContext.createChannelMerger();
2 // input -> any input node
3 // output -> any destination node
4
5 //LEFT EAR
6 input
```

```
7   .connect(itdLeft)
8   .connect(iidLeft)
9   .connect(merger, 0, 0)
10 //RIGHT EAR
11 input
12   .connect(itdRight)
13   .connect(iidRight)
14   .connect(merger, 0, 1)
15 // REAR
16 input
17   .connect(ridRear)
18   .connect(gainRear)
19
20 gainRear.connect(output)
21 merger.connect(gainFront).connect(output)
```

Listing 6.4: *Duplex Algorithm*: node connection

Therefore the sound movement function will be as follow:

```
1  move(azimuth:number){
2         //azimuth 0 <right> 180
3   //azimuth 0 <left> -180
4    frontDel = front(azimuth);
5    itd(azimuth, frontDel);
6    iid(azimuth);
7  }
```

Listing 6.5: *Duplex Algorithm*: movement function

6.2 Quasi-Specific HRTF Algorithm

A *Head-Related Transfer Function* (HRTF) is a recording of the acoustic response of the space inside the human ear canal to an impulsive (percussive and short) stimulus. The HRTF is acquired with specific microphones inside the human ear. Each of these is recorded through an impulse (or most commonly with a sweep) in several positions in the horizontal and vertical planes, at a fixed distance around a real head. For this thesis, I used ready-made ones. In comparison with the *Duplex Algorithm*, the usage of the HRTF has the advantages of the binaural information's richness and the simplicity of the usage. In contrast with the *Duplex Algorithm*, in which a sphere is applied as an approximation model, here, with the HRTF, a real head is used. Every impulse (that is a stereophonic impulse) is characterized by its own ITD and IID. Both of these are particularly detailed. Especially the IID is extremely elaborated in comparison with that one of the *Duplex Algorithm*. This is because the entire body effects on sound (the ear conformation, the shoulders, the real shape of the head, and so on) are captured in the HRTF. Therefore the binaural information is particularly rich. In addition, the usage is simple. To obtain the simulation of a general sound sample in a specific position relative to the listener one needs only to apply the convolution of the sound sample with the recorded HRTF impulses at that position. The system can then interpolate between several HRTFs to represent the movement of the sound sources in space. However, the system has its own weakness: the HRTF individuality and the computational cost. The HRTF impulses change from person to person depending on the subject who recorded them and then to

her/his specific head and body conformation. Furthermore, the more detailed the system (that means more impulses are used), the more the computational cost increases[Beg94].
A "spacific" HRTF system would require the usage of the specific user's own HRTF set. The application of a *quasi-specific HRTF Algorithm* means the usage of a series of almost 50 ready-made HRTF sets with different body conformations. The series is the open-source HRTF measurements made at IRCAM and AKG in the context of the *Listen* project[18]. In addition to giving a complete series of impulses for the horizontal and vertical planes, they also provide well-documented information about every subject's body conformations. Therefore, in the *Quasi-specific HRTF Algorithm*, the user chooses one of the 50 HRTFs set by comparing its body conformation with the information provided by the library.

Implementation

The implementation of such an algorithm, a "quasi-specific" one, does not differ from that one "specific", except for the HRTF set used. The core of the algorithm is the convolution of these impulses and the simple passage from one to another. Practically, one can use such an algorithm with its own HRTFs set, simply loading it in the system.

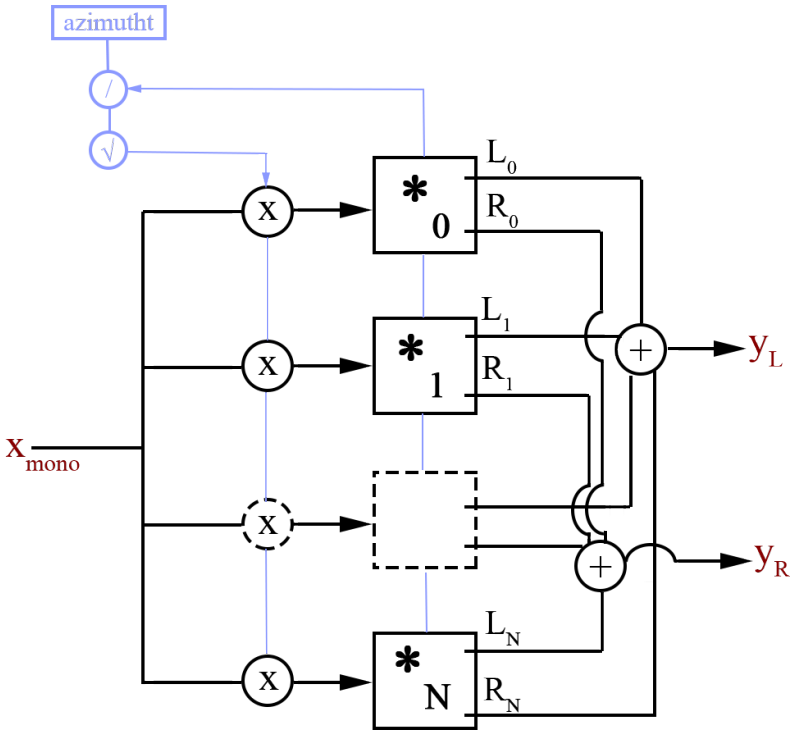


Figure 6.5: The *Quasi-Specific HRTF Algorith* diagram

[18]The entire database and the whole documentation can be found in the following link http://recherche.ircam.fr/equipes/salles/listen/.

Some implementations of binaural systems with the HRTF impulses in *Web Audio API* are already designed, and one can simply find them on the web. An example is the algorithm proposed by Thibaut Carpentier in *Binaural Synthesis with the Web Audio API*[Car15]. This implementation is distributed as a library and, although it is really simple to use with its own HRTF set, I have found it not very efficient. The library requires a significant increment of the computation work thus the generation of several clicks. The problem arises from the complex implementation of the convolution operations. In this work, I have used my own implementation with which I have attempted to simplify the audio stream process to prioritize real-time interactivity over spatial precision: instead of dividing the stream in frames, I have divided it into convolution chains. An overview can be seen in the diagram in *figure 6.5*. The implementation consists of fixed convolver blocks ($*_N$), as many as the number N of HRTF impulses. And, on the base of the position of the sound (azimuth), relative to that one of the HRTFs, it goes or not goes in specific convolver blocks. And this is allowed by the application of gains ($\times$, the multiplication blocks). Each gain is calculated by the relation between the position described by the single HRTF and the actual position of the sound. Furthermore, each gain is controlled by an amplitude-panning multiplication ($\sqrt{}$) with which the passage from one convolver chain to another is smoothed. And this can be seen with more details in the code. Having a ready loaded array of HRTF buffer impulses, the general structure of such implementation can be coded as follow:

```
//the ready loaded impulses array
hrtfArray:AudioBuffer[];
hrtfLen = hrtfArray.length;

//initialization of nodes
gain:GainNode[] = [];
hrtfsChannels:ConvolverNode[] = [];

for(let i = 0; i < this.hrtfLen; i++){
  gain[i] = audioContext.createGain()
        gain[i].gain.value = 0;
        hrtfsChannels[i] = audioContext.createConvolver()
  hrtfsChannels[i].buffer = buffer;
        input
    .connect(gain[i])
    .connect(hrtfsChannels[i])
    .connect(output)
}
```

Listing 6.6: *Quasi-Specific HRTF Algorithm*: initialization structure

The next part of code shows the movement function of the algorithm:

```
controlArray:number[] = [1, 0];

setChannelGain(ch:number, val:number){
  gain[ch].gain.setTargetAtTime(val, audioContext.currentTime, 0.05)
}

move(azimuth:number){
  //azimuth: 0 = front
  //azimuth: 90 = left
  //aziumth: 180 = behind
  //azimuth: 270 = right
  pos = azimuth / 360 * hrtfLen;
```

```
13    a = Math.floor(pos);
14    b = Math.floor(pos + 1);
15    A = a % hrtfLen;
16    B = b % hrtfLen;
17    gainA = Math.sqrt(Math.abs(a - pos));
18    gainB = Math.sqrt(Math.abs(b - pos));
19    setChannelGain(A, gainA)
20    setChannelGain(B, gainB)
21    if (A!== controlArray[0] && B!== controlArray[0]){
22      setChannelGain(controlArray[0], 0)
23    }
24    if (A!==this.controlArray[1] && B!=this.controlArray[1]){
25      setChannelGain(controlArray[1], 0)
26    }
27    controlArray = [A, B];
28  }
```

Listing 6.7: *Quasi-Specific HRTF Algorithm*: `move()` function

An important point of this code is `arrayControl` which allows the usage of two convolver chains at once (the reason to calculate two positions `A` and `B` in the code). That allows interpolating the transition between two HRTF impulses, covering the space between the fixed positions. And this is acted by a simple amplitude-panning method between two HRTF (the `A` and the `B`).
In summary, this system is very efficient because of its simplicity. There is no fragmentation and overlap of the audio stream but only a system of cross-fade toward a series of chains. Furthermore, all the processes of node-generation and node-connection are calculated only once, in the initialization of the system. The implementation does not produce any clicks, the computational load is reduced and the spatialization is convincing.

6.3 General HRTF Algorithm

The last type of 3-dimensional audio implementation used in this work is the *General HRTF Algorithm*. Theoretically, such an algorithm works in the same way as the previous one. The only difference is the use of only one HRTFs set. And it can be reached through two different methods: the *HRTF average* or the *KEMAR* head. These two methods return almost the same result. In the first case, the solution is to compute the average of a series, more or less wide, of HRTFs sets. The other solution is to create an artificial head (or better a bust) with the conformations equal to the average obtained by the body conformation of various subjects. The impulses are recorded with this artificial head (the *KEMAR* is the most famous bust). These methods' result would be one set of HRTFs approximately suitable for more subjects. Both of these approaches are not very reliable. The fact that they are produced by an average, inevitably leads toward a smoothing process. This means, losing all the smaller features and especially the singularities which are the real straight point in the HRTFs usage. I felt the need to mention this option in this work due to the importance of giving a general view of the main 3-dimensional audio implementation standards.
An algorithm based on the first case would be already provided by *Web Audio API*. The node `PannerNode` supports an algorithm called HRTF. And it applies the convolution with only one non-individualized HRTFs set. It is not

clear the core of the algorithm because the *Web Audio API* specification crucially lacks documentation on this topic. However, as it has been mentioned by Thibaut Carpentier in the already mentioned *Binaural synthesis with the Web Audio API* it would seem that the HRTFs set used is derived by the IRCAM *Listen* HRTF database. It is not clear however if there is an *HRTF average* and what kind of average has been made.

Implementation

The implementation of a 3-dimensional audio system via `PannerNode`, which is already provided by *Web Audio API* is straightforward. The generation of the `PannerNode` and the specification of its mode (`HRTF`) are the only required steps to implement:

```
1 panner = audioContext.createPanner()
2 panner.panningModel = "HRTF";
3 input.connect(panner).connect(output);
```

Listing 6.8: `PannerNode` initialization

It is possible to specify other details, such as the algorithm with which the distance is calculated, several parameters to set up the distance algorithm, the rear and front diffusion cone for the source, and so on. However, in this example code, like the previous and the next ones, I put only the essential things. The `PannerNode` requires another horizontal plane set up. Here, the angles between 90° and 180° are the front side of the listener, while the rear positions are described by the angles between 180° to 360°. Therefore, the left side will be from 90° to 0° to 270°, whilst the right side will be from 0° to 180° to 270°. The movement can be coded as follow:

```
1 position3D(azimuth:number){
2   x = Math.cos(azimuth/180) * Math.PI
3   z = Math.sin(azimuth/180) * Math.PI
4   panner.positionX.setValueAtTime(x, audioContext.currentTime);
5   panner.positionZ.setValueAtTime(z, audioContext.currentTime);
6 }
```

Listing 6.9: `PannerNode - move()` function

Where `positionX` represents the axes from the right ear (−1) to the left ear (1) whilst the `positionZ` represents the axes from the rear position (−1) to the front ones (1)[19].

6.4 3D audio systems test

The selection between the three spatialization algorithms has been done through a human listening test. The choice of the participants has been completely random, without any distinction between people of the field (such as musicians, sound engineers and so on) and common people, ages and gender. The test was in the form of a web site that the participants could log into. The 3-dimensional

[19]The `positionY` represents the vertical axes.

audio systems used are the same algorithms that have been described above. The test was split into three parts:

1. Essential Data Submission: the test was completely anonymous, the subject had not to enter his name, his e-mail, and nothing else about his personal data. However, four fundamental information was required: his gender, his head circumference, his shoulder circumference, and his hairstyle.

2. Localization test: the main part will be better described below. Basically, for each 3D audio system, the subject was asked several times to find and then indicate the sound in the horizontal plane.

3. Quality test: for each 3D audio system, the subject was asked to give a personal opinion about the quality of the system.

The first part of the test was crucial because of the *Quasi-Specific HRTF Algorithm* and partially, also because of the *Duplex Algorithm*. The *Listen*'s HRTF sets provide a lot of information about the body conformation of the sourced subject (around 50 parameters for each subject). However, it was not thinkable to ask a subject to get all these data. Therefore, it has been picked only four parameters that are more incisive. Furthermore, in the *Listen*'s HRTF set, some of the information is left blank in some subject's card (but the parameters chosen for the test are always available).
With these data, the subject had compiled a personal profile that was compared with the profiles of the *Listen*'s HRTF database. An automated system chose an HRTF set on the basis of similarities between the subject of the test and that one of the database. The circumference has been also used to the calculation of the spheric model diameter, for the *Duplex Algorithm*.

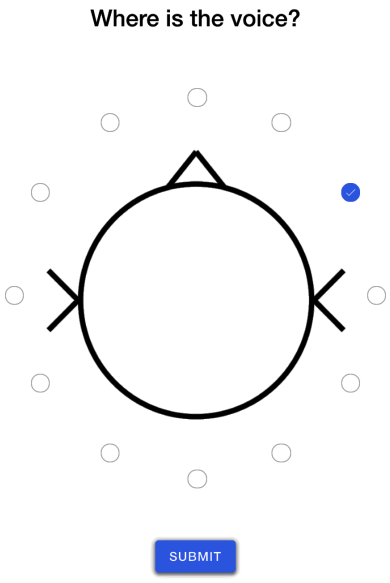


Figure 6.6: The second part of the test, with the twelve dots around the figure.

The second part of the test has been the main part. For each spatialization system, the participant is asked to localize the sound in the horizontal plane between twelve positions around the listener. The sound is then moved and stopped in a new position. This procedure repeats for seven trials. The interface is represented in *figure 6.6*. In the test, it has not been taken into account the differences between the right and the left side because of the symmetry of the human head. That means that the IID and the ITD calculated on one side are the same on the other side, with the only difference that the interaural differences are calculated for the opposite ear. And for this reason, the perception of them should be equal (or better symmetric). When one hears a sound at 60° (on the right side of his head) he has a certain perception and localization of the sound on the base of the interaural differences. Its left ear acquires the sound with a certain IID and ITD in relation to the right ear. If then, he hears a sound at −60° (on the left side) he should have quasi-perfect symmetric perception and localization of the sound compared to the sound at *60*. Its right ear acquires the sound with the same interaural differences acquired before by the left ear. Considering this, the test has been done to detect the localization in the vertical level in relation to the figure and so from the front to the rear positions. Therefore, every subject had to choose each time the sound randomly stopped in one position between 0° and ±180° (with a step of 30° between each fixed position) covering all the seven possibilities (0°, ±30°, ±60°, ±90°, ±120°, ±150°, 180°).
Finally, in the last part of the test, the subject is asked to give a personal opinion about the quality of each system. This part has been done by two playbacks of an audio file: one time dry, without any audio process, and one time processed by the binaural system. My goal was to detect the differences in the spectral quality of the sound - how much each system modifies the spectral of an audio file. However, a topic like this offers a high subjectivity level, producing useless objective data. Therefore this part has not been taken into account in the final analysis. Only the second part has been considered.

Data Analysis

The data analysis was done by calculating the average of the errors made in the selection of each position. The error represents the distance between the effective position of the sound and the selected position by the subject. The error is calculated as a percentage, where 0 is when the effective position and the selected one are equal, whilst 100 is when they have a maximum distance of 180° from each other. The results are summarized by the plot in *figure 6.7*. The first thing that has to be noticed is the large level of errors in the differentiation between the front and the rear position, especially for that algorithm that uses HRTFs. And this goes beyond the possibilities of every binaural system[20]. Also in reality this is a complex task, but in this case it is simply solved by the possibility of the individual to rotate the head. However, in the *Duplex Algorithm*, it would seem that this problem is reduced. This is because

[20]With the usage of a "specific" *HRTF Algorithm* this error would radically reduce.

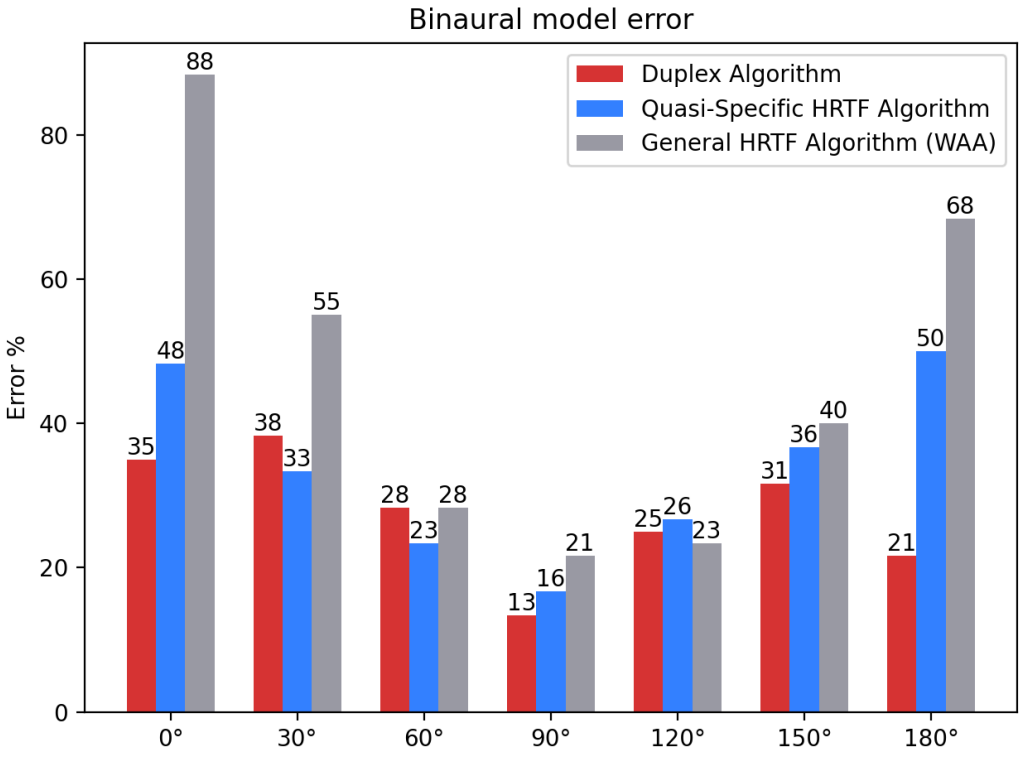
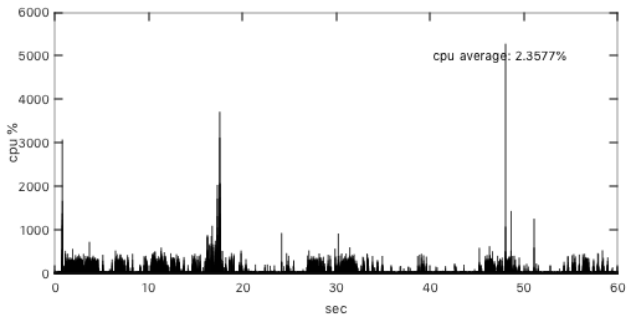
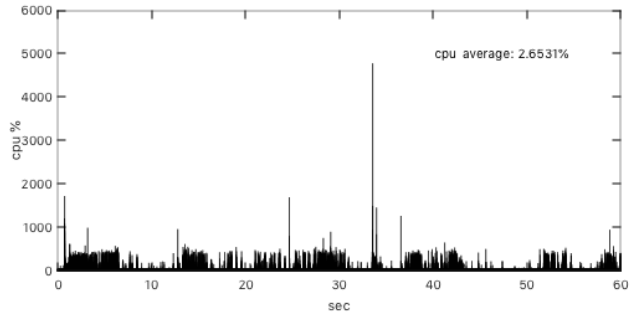


Figure 6.7: The test results

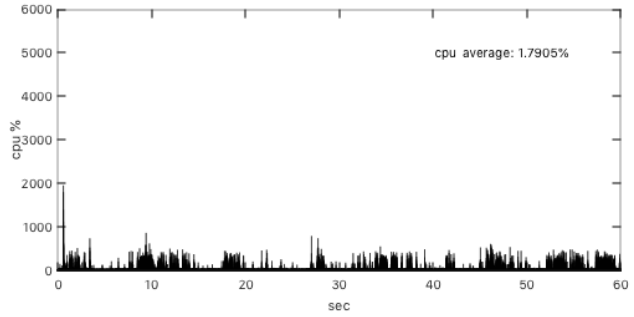
of the third ear with which it is possible to emphasize the differentiation between the front and the rear position. But this brings the side-effect of the loss of naturality of the system. The better is the differentiations the greater is the intensity of the third ear's filter in relation to the frontal sound.
It is important to interpret these results in connection with the computational load of each system. To make an analysis of the CPU required for each algorithm, several records of the processing unit usage have been done. The records have been taken during the test compilation. The results are the following:



Duplex Algorithm CPU record



Quasi-Specific HRTF Algorithm CPU record



General HRTF Algorithm CPU record

Figure 6.8: CPU usage for the 3D audio algorithms

The lightest process has been recorded for the *General HRTF Algorithm*. This is obvious, because, as has already been said, the algorithm is made through the `PannerNode`. Since this node is an internal *Web Audio API* node, built exactly for this purpose (the binaural spatialization), it works at the lower level, that means at the browser level, and this leads to a better computational work. However, it has to be noted that the other two algorithms, the *Quasi-Specific HRTF* and the *Duplex algorithm*, do not determine a heavy processing load, remaining under an average of 3% CPU activity. In conclusion, considering the results provided by the test and the CPU analysis, the *Duplex Algorithm* has been chosen as the best solution to be implemented in the application.

7 Sound Generator

To erase every horizontal development in the sound one should take a brief microacoustic event, with the duration near the threshold of auditory perception, and constantly repeat it. A duration between one-thousandth of a second and one-tenth of a second does not allow any development of sound. And this is also the fundamental quantum of *Granular Synthesis*, the grain. By combining thousands of grains over time, it is possible to create an animated sonic atmosphere (Curtis Road)[Roa01]. But with the same technique it is possible to create a completely frozen sound atmosphere (the *Freeze Effect*), too.
Inside a grain so brief, many interesting features are lost. Let's take for example the sound of the violin. Considering only the simplest gesture (or technique) such as the movement of the bow, the vibrato, the tremolo or the harmonics produced in the bridge of the instrument. Through these gestures, if the

other parameters remain unchanged (such as the note in particular, and the general dynamic), the sound produced is not contemplated with development but nevertheless in movement. The typical grain can not contain these kinds of movements. Therefore to increase the number of sound features in a frozen sound it should be increased the duration of the grain. But in this case, there is not a precise grain duration which depends on the sound itself and on the subjective idea of development.
One can distinguish two families of granulators on the basis of the sound nature of their grains. When the grains are generated by a synthesis procedure (that are wavelet synthesis, sinusoids, impulses, etc) the granulator can be defined as *Synth Granulator*, while can be defined as *Sample Granulator* when the grains are extracted from an audio sample. The technique to increase the duration of the grain in the freeze effect makes little sense in the *Synth Granulator*. On the other hand, in the *Sample Granulator*, it can create different effects. In addition to freezing a sound without losing some of its fundamental features, it is also possible to create a freezing sound overlapping different features of itself in different temporal moments. Let's take for example the audio recording of a gong beat. It is possible to transform it into a complex and frozen stream by superposing different fragments (grains) of its coda (this solution is often used in my sculptures' composition). With this technique, the resulting sound is a frozen sound not lacking in "naturalness". That means a sound that does not change in time, does not create phrases, does not give expectations but nevertheless it is moving. It is "natural" because it morphs within an acoustical range of possibilities which are not perceived as temporal development but as an act of existence (like the breathing of an immobile animal). In this way every layer of a single sculpture is built. Therefore every sculpture is made of a group of several frozen sounds, organized in a specific space. From every point of view (or listening), on the base of the listener position, every frozen sound will be highlighted or minimized giving an ever-changing perspective of the sculpture (exactly like the ever-changing perspectives of a traditional sculpture).

Implementation

The algorithm used in this work comes from the granulator synthesis proposed by Carmine Emanuele Cella[21]. Here, the algorithm is a reduction from the original one, and it is really simple even because Cella's implementation is one of the best proposals in the ratio quality-simplicity in my opinion[22]. The reduction is given by the cutting of several parameters. The original algorithm is implemented in *Max/MSP*. Here, without taking into account any analogies with this software I will directly explain it in *Web Audio API*, in a general form. Giving an audio sample X with a length T in milliseconds, a simple

[21]I had the opportunity to attend some of its lectures around Italy where he has often explained this smart and simple solution to implement a granulator.

[22]The original algorithm is implemented in *Max/MSP*. The simplicity comes from the so-called 'objects economy'. That means the implementation and re-use of less than ten objects (without considering the mathematical operators).

grain can be described with the following diagram:

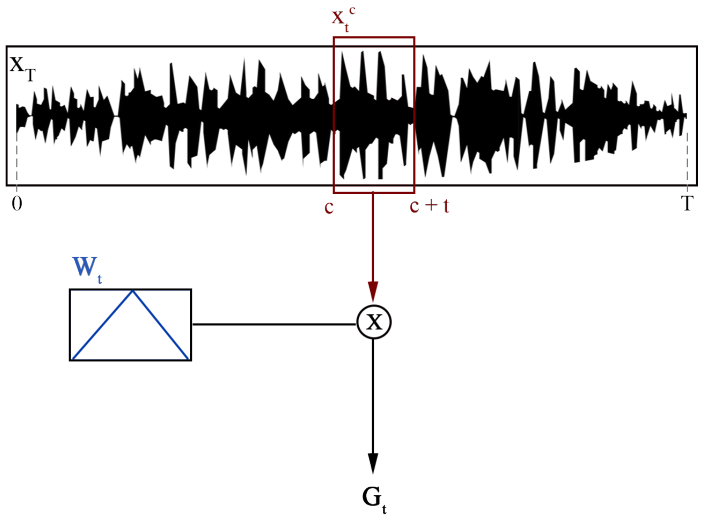


Figure 7.1: Diagram: Grain, the granulator core

and then with the following equation:

$$G_t = W_t x_t^c \quad \text{where} \quad \begin{cases} t < T \\ 0 < c < T - t \end{cases} \tag{7.1}$$

Here, x is one frame of the original sample X, taken at the c point of the sample with a length t in seconds; W is the envelope of the grain with the same length t. The envelope, that is a window function (such as Hamming, Hanning, Blackman, and so on) is used to avoid clicks at the start and end of the grain. The c, the cue point of the grain, has to be taken with a value between 0 and $T - t$. And t has to be a value lower than T.
In this algorithm, working only with these two parameters plus the window type brings a lot of interesting results, without even considering the other important ones. However, in addition to completely leaving out the other parameters, the use of c and t will be significantly restricted. The c will be chosen randomly. The t will be chosen randomly, too, but around a fixed value (usually between one and two seconds). In *Web Audio API*, considering the buffer of the sample loaded and the buffer of the window created, such a grain will be simply played as follow:

```
1 //create the sample node
2 sample = AudioContext.createBufferSource();
3 //create the a gain node to do the window envelope
4 wind = AudioContext.createGain()
5 //fill the smaple node with the buffer of source file
6 sound.buffer = buffer;
7 //T = the length of the sample
```

```
8  T = sample.buffer.duration;
9  //t = the length of the grain
10 t = getFrameTime(fixed_t, rnd_percent_t);
11 //c = generate a random cue between
12 cue = Math.random()
13 c = cue * (dur - t);
14
15 sound.start(AudioContext.currentTime, c, t);
16 gain.gain.setValueCurveAtTime(wbuffer, AudioContext.currentTime, t)
```

Listing 7.1: basic grain code

where the t is generated by the function `getFrameTime`, giving the fixed t (`fixed_t`) and a percentual random value (`rnd_percent_t`):

```
1 getFrameTime(fixed_t:number, rnd_percent_t:number){
2     rng = fixed_t * rnd_percent_t/alpha;
3     T = t + Math.random()*(rng-(-rng))+(-rng);
4     return T;
5 }
```

Listing 7.2: the `getFrameTime` function

And `alpha` is a fixed variable with which it is possible to indicate how much the random value affects the `fixed_t`. The bigger is the value the smaller is the effect of the random value. In this case, it is set at 1,5.
Obviously, one grain is followed by another. Therefore after `t` seconds, the whole procedure of the code in the *listing 7.1* will have to be repeated. And then, after the repetition, it will have to be repeated another time. Such a loop, in *Web Audio API*, it can be implemented by transforming the previous code in a function and call it within itself every time the window is finished to play:

```
1  play_grain() => {
2    sample = AudioContext.createBufferSource();
3    wind = AudioContext.createGain()
4    sample.buffer = buffer;
5    T = sample.buffer.duration;
6    t = getFrameTime(fixed_t, rnd_percent_t);
7    cue = Math.random()
8    c = cue * (dur - t);
9
10   sampl.start(AudioContext.currentTime, c, t);
11   wind.gain.setValueCurveAtTime(wbuffer, AudioContext.currentTime, t)
12
13   //callback when the sound is stopped
14   sample.onended = () => {
15     sample.disconnect()
16     play_grain()
17   }
18 }
```

Listing 7.3: basic grain function with repetition

The method `onended` of the `BufferSourceNode` is called every time the started sound stops to play. In this case after `t` seconds. Instead, the `disconnect` method is the only way to delete an *audio node* in the `AudioContext`. Another parameter that can be added to enrich the resulting sound is the density of the grains repetition. And this can be obtained by only adding a delay

between one grain and the next one.

```
1   sample.onended = () => {
2     sample.disconnect()
3     setTimeout(() => {
4       play_grain()
5     }, delay)
6   }
```

Listing 7.4: `setTimeout` function - add a delay between every grains

Here the `delay` variable has to be calculated like the `t`, with a random factor. However, in this way, it has not achieved a proper granulator. With this function, it will obtain only a resampled version of the original sample, with or without gaps between the frames. Indeed, the fundamental step is the overlapping one. That means to have different voices that simultaneously make this resampling function. The fact to have randomic time both in the frames and in the delays between the grains, in the polyphonic and overlapping system, is the cause of an ever-changing intensity. In addition, the gaps are covered by the other voices. This polyphonic system is the base of the movement without development. To achieve this in the *Web Audio API*, the entire system of the single grain (or better, the voice of grains) has to be wrapped inside a class.

```
1   export class gran{
2     constructor(
3       audioCentex:AudioContext,
4       sampleBuffer: AudioBuffer,
5       windBuffer: Float32Array
6     ){
7       this.wind = this.audioCentex.createGain();
8     }
9
10    private state:boolean;
11    //Audio Nodes
12    private wind: GainNode;
13    private sample: AudioBufferSourceNode;
14    //Parameters
15
16    //grain time
17    public fixed_t: number;
18    public rnd_percent_t: number;
19    //delay time
20    public fixed_d: number;
21    public rnd_percent_d: number;
22
23    //play_grain method
24    play() => {
25      this.sample = AudioContext.createBufferSource();
26      this.wind = AudioContext.createGain()
27      this.sample.buffer = buffer;
28      T = this.sample.buffer.duration;
29      t = this.getFrameTime(this.fixed_t, this.rnd_percent_t);
30      cue = Math.random()
31      c = cue * (dur - t);
32      d = this.getFrameTime(this.fixed_d, this.rnd_percent_d);
33
34      this.sampl.start(AudioContext.currentTime, c, t);
35      this.wind.gain.setValueCurveAtTime(wbuffer, AudioContext.currentTime,
            t)
36
37
38      this.sample.onended = () => {
```

```
39        this.sample.disconnect()
40        if (this.state){
41          setTimeout(() => {
42            this.play()
43          }, delay)
44        }
45      }
46    }
47    getFrameTime(fixed_t:number, rnd_percent_t:number){
48          rng = fixed_t * rnd_percent_t/alpha;
49          T = t + Math.random()*(rng-(-rng))+(-rng);
50        return T;
51    }
52      public start(){
53            this.state = true;
54            this.play()
55        }
56        public stop(){
57            this.state = false;
58        }
59  }
```

Listing 7.5: the grain `class`

23

Now, it is possible to recall this class many times as the voices required by the granulator. And every voice will be independent. Clearly, the more voices, the more computational load is required but also the better resulting sound quality. A number of eight voices is the best choice regarding the ratio between the quality of the sound produced and the computational load. Another important step is the choice of the sample that has to be granulated. There is not a particular and fixed principle with which these samples have to be chosen. Not every sound produces satisfactory outcomes. The selection has to be done after testing several samples. Probably, a factor to always avoid is the presence of attack transients in the samples. It will bring to create ever-changing rhythmic patterns, more or less regular but always with a sort of a development idea.

8 The Sculpture Class

The Sound Generator and the 3D audio system join together to form a bigger system, an important block for the entire application. Their marriage leads to the creation of the *Sculpture* class. Practically, this class is the container of all the audio processes. Each sculpture, in this virtual gallery, is built and plays according and thanks to the processes described in this class. Therefore, for every sculpture, a new *Sculpture* class is called. An overview of this can be summarized by the diagram in *figure 8.1*. As has been shown in the previous chapter, each granulator (G_v) is formed by several voices (V voices) overlapped between them. And each voice performs the task of fragmenting and resampling an audio sample (as_S where S is the number of samples), which is the granulation task. The output of each granulator is a mono output. Here, a

[23] As it can be noticed the sample node has to be always regenerated and then destroyed after playing. One can ask: why isn't the same node reused? This is one of the strange behaviors of *Web Audio API*: After the sound has been stopped, an `AudioBufferSource` node can not be reused.

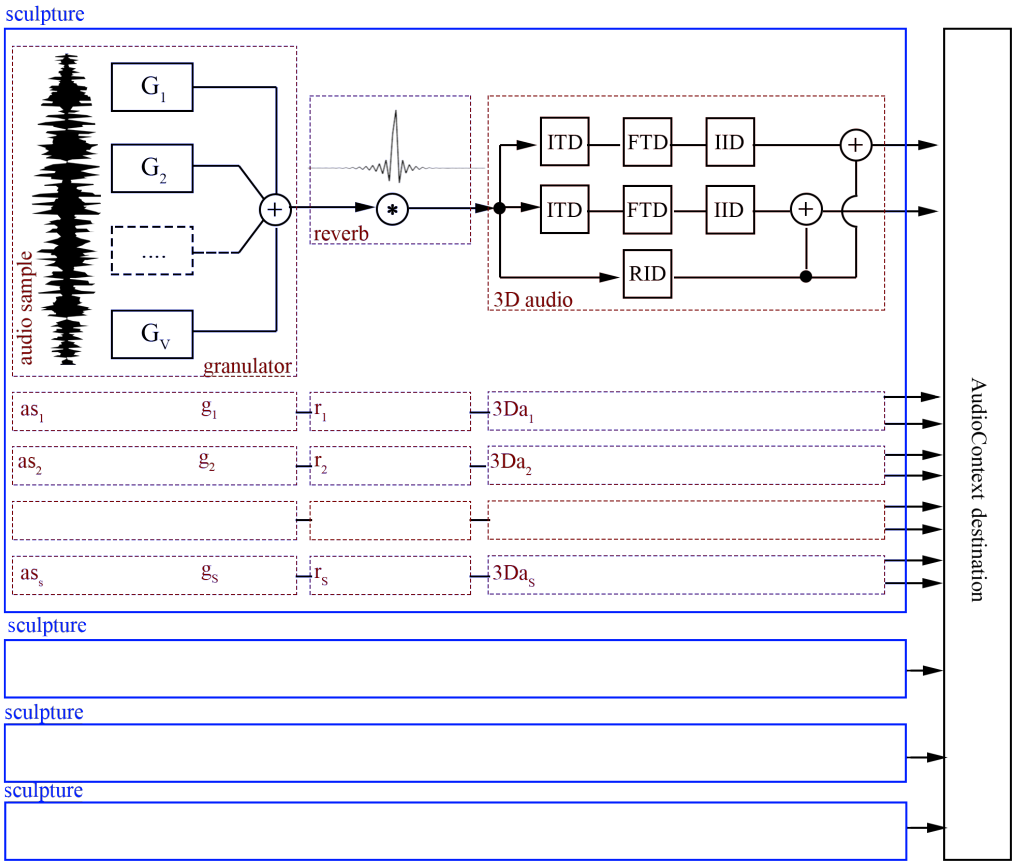


Figure 8.1: the *Sculpture* class diagram

step that has not been explained before is the reverberation (r_S). It may seem strange and inappropriate the reverberation usage before the sound spatialization. Indeed it is so because this reverb is used to particularize the sound rather as a pure reverb. And this means to use the impulse of the reverb more like an equalization with which is possible to wrap the grains together, a general sound timbre to each granulated audio sample. Without going to modify in a depth way the audio sample's sound peculiarities, the task of the reverb is to give a general and unified sound. This is completely a subjective and aesthetic choice. The reverberation is accomplished by a `convolverNode`$*$ with a mono impulse response. The output is sent to the input of the 3-dimensional system ($3Da_S$). And here, on the base of a sculpture's established positions and of the orientation of the device the input is processed by the *Duplex Algorithm*. Furthermore, each point of the sculpture is processed by a further algorithm. This accomplishes the distance between the listener and the sculpture's point. It is just a gain which value is given by the following equation:

$$g_d = 1 - \frac{min[d, d_M]}{d_M} \tag{8.1}$$

Where d is the distance calculated through the difference between the listener coordinates and those ones of the sculpture's points, d_M is the maximum distance and g_d is the resulting gain. Over d_M the point can not be heard anymore. Therefore, the sculptures keep playing even outside the areas which they define, but with a volume lower and lower in inverse proportion to the distance with the listener.
At the very end, the stereo output will be sent toward the `AudioContext`'s output, the `destination` node. The entire code of the Sculpture class can be found in the Appendix.

9 Algorithms' communication system

From a general point of view, the core of the application is performed by these three main algorithms, the Position Tracking system and the 3D audio system that join together with the Sound Generator algorithm. Important is their constant communication. Not much else has been implemented, except for an interactive map, a minimal user graphics interface system, and a backend communication system. These are only partially treated in this text because of their less importance regarding the main context of this thesis. But, it is important to mention how the three main functionalities are connected into the mainframe of the application. The Position Tracking system is the pure heart of the entire system, everything in the application is based on it. And for this reason, its entire code is in the form of an *Angular* service that can be injected in every application section. The interactive map in the application uses this service; the graphic user interface (in which the map is included) is linked to this service; the backend service strictly communicates with this one; the *Sculpture* class is in a closer connection with this service. This is the main root of the application and for the sake of completeness, I have included the entire code with further explanation comments in the Appendix.
The creation of each sculpture with its audio parameters, its audio samples and its placement in determined coordinates is a process carried out by an intricate communication between the backend service (the *Angular* service reserved for the communication with a server), the motion tracking service, and the *Sculpture* class. The path of this process is summarized by the following diagram:

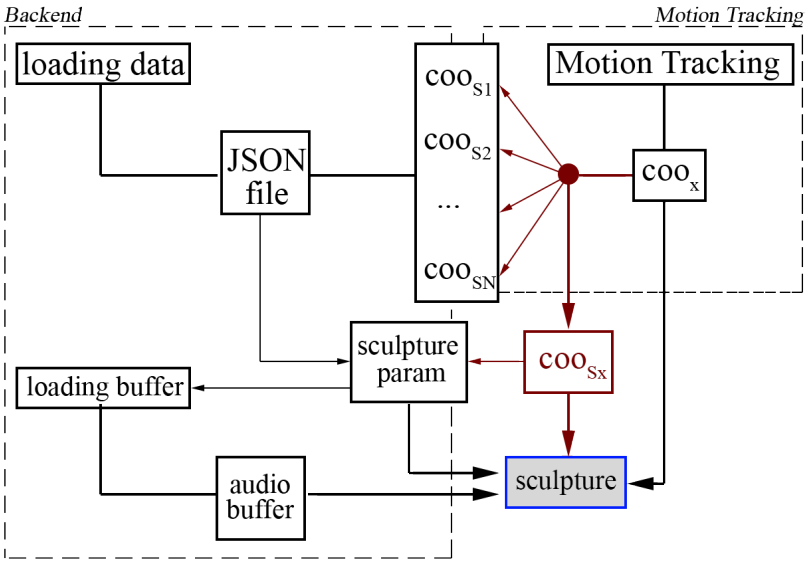


Figure 9.1: The communication between the *Backend* service, the *Motion Tracking* service and the *Sculpture* class

Practically, the backend service loads the sculpture data before the application is initialized. It is the very first task that the application carries out. The data are contained inside a *JSON* file in which all information about each sculpture is given. For each sculpture the *JSON* file is structured as follows:

```
1  {
2    "name": string,
3    "color": #esa-num,
4    "soundSpec": {
5      "vox":[num],
6      "param": [{
7          "gain": num,
8          "time": num,
9          "timeRND": num,
10         "dencity": num,
11         "window": windowType
12       },
13    ]},
14     "coords": [
15   {"lat": num, "lon": mun},
16     ]
17 },
```

Listing 9.1: sculptures data in the *JSON* file

The *JSON* file contains the name of the sculpture (the color is used in the map component), the parameter and the coordinates are given. And both the audio parameters and the coordinates are referred to each granulator inside each sculpture. Therefore, the `"vox"` at *line 5*, the `"param"` at *line 6*, and the `"coords"` at *line 14* are all arrays and every instance of the array is referred to only one granulator. So, coming back to the diagram in *figure 9.1*, the motion tracking service acquires the data loaded by the backend service, but only

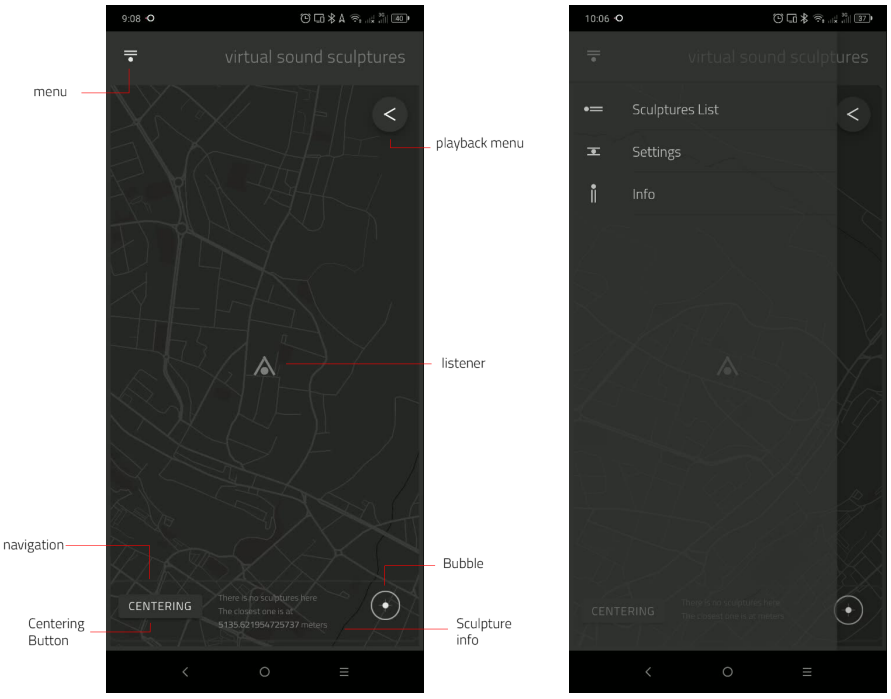
the coordinates data rather than the entire set of information[24]. During the motion tracking, the relation between the actual position and the loaded set of coordinates is constantly checked (in the red dot of the diagram). If the actual position is close to at least one couple of coordinates of one sculpture (then with a distance lower than the d_M value), a new sculpture will be generated with parameters related to the data which contain those coordinates. To load the audio sample the *Sculpture* class refers to the backend service again. Here, the service provides a function with which the buffer loading is carried out on the base of the selected sculpture's name. And then each buffer for each sculpture's granulator is sent to its voices. Finally, the sculpture interaction with the user is performed by a process inside this class. Here, the motion tracking data and the coordinates defined for each granulator are compared in real-time. The results are the azimuth and the distance of the user in relation to every sculpture's granulator. Once the distance of the listener from the coordinates of every granulator of one sculpture is bigger than the d_M value, that sculpture will be eliminated.
At first sight, it may seem a complex system. But in an *Angular* environment, all these things are more clear than in a block diagram. And all this because of the powerful technology of the *Angular* services and the easy communication between pages.

10 User Interface

As can be seen in the *figure 10.1a*, the resulting work is a simple application with a minimal graphic interface. The screenshot represents the case without any sculptures around the listener. The entire application consists of an interactive map, one navigation section, one main menu and one for the playback. With the map, the listener position and that one of the sculptures (that can be seen in *figure 10.3*) is given. The map is a typical interactive one, constantly centered with the listener position but, with also the possibility to move its center and zoom in and out. The navigation section provides a simple centering button (like that one in the *Google Maps*[25]) with which it is possible to re-center the view of the map in the listener's coordinates. The sculpture info in this section tells the distance of the nearest sculpture. But, if one is already listening to at least one sculpture, this part will show a list of sculptures' names. Also in this section, there is a bubble indicator with which one can "center the bubble" or, in other words, put the mobile phone parallel to the ground. And this helps to get a better and more accurate heading. Then, there is a basic playback menu. If it is clicked it will be expanded, showing a play and pause buttons (as shown in the figure below). By clicking on these buttons one can start or pause the sculpture's playback. Finally, there is the main menu that is a side menu. One can reach this by clicking on the upper-

[24]This is performed outside both the motion tracking and backend service. it is carried out on another page in which these two services are linked. But I want to leave out this point to avoid other complexities.

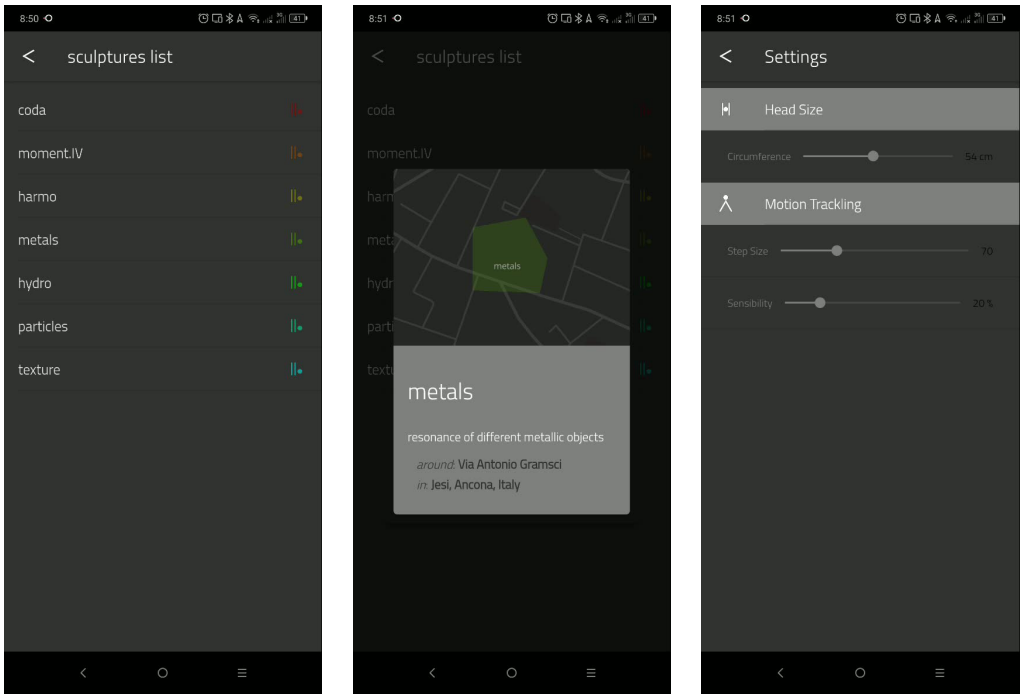
[25]https://www.google.com/maps/



(a) Home page of the application **(b)** Side menu

Figure 10.1: Mobile Application's Home Page

left button (*menu* button in *figure 10.1a*) or by simply swiping right on the screen. The menu is shown in *figure 10.1b*. From this, one can reach a page in which every sculpture is listed (*figure 10.2a*), a page in which one can set essential parameters to improve the application performance (*figure 10.2c*) and a page of general information about the application. In the sculpture list page,



(a) Sculpture list page **(b)** Sculpture card **(c)** Settings page

Figure 10.2: The other page of the application

in addition to having a list of all the sculptures, one can click to a single item of the list to get a sculpture's introduction card (*figure 10.2b*). Here, only the name, a brief description, and the area of the sculpture's localization (the city, ragion, state and the street around which the sculpture is placed) are given.
In the settings page, one can configure three important functionalities: 1) The head circumference regards to the 3D audio system. Allowing one to choose the head size in the application, it will partially improve the spatialization of the sound which will be in relation to the user's head; 2) The step size and 3) the sensibility regarding the *Pedometer*. This allows the choice of these parameters, which are strictly correlated to the user stature, will improve the motion tracking system and the gaps shown in chapter Position Tracking System. However, these settings are not necessarily required. It would be too cumbersome to force the user to measure the head and the step size, to reach the perfect performance of the app. Therefore, these parameters are already set to medium values. This makes sure that one can use the application without any changes, getting no major errors.

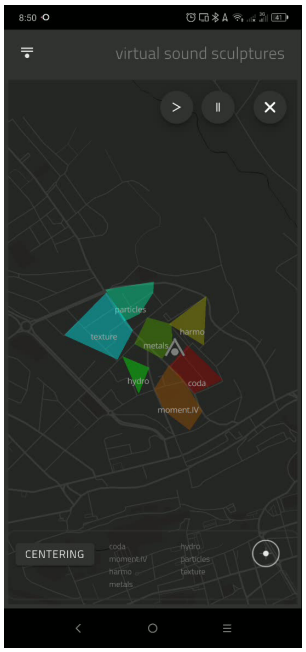


Figure 10.3: The application's Home page when there are sculpture around the user

The screenshot above (*figure 10.3*) shows the case when the listener is surrounded by many sculptures. Every highlighted area shows the boundaries of one sculpture. It is nothing but a convex hull, the perimeter produced by the outer sound points (coordinates) of the sculptures. And this describes the localization of each sculpture.

Conclusion

The presented thesis describes the realization of a music mobile application called *Virtual Sound Sculptures*. The application is a virtual gallery in which several sound sculptures are exhibited. It is a virtual museum inscribed within the Earth coordinate system, and in theory, the entire globe. The museum presents sonic artworks without any visual reference. The text examines different aspects of the realization of such an application. It is described mainly from a technical perspective to demonstrate the usage of specific technologies for musical creation in application development. The text is focused on the usage of *Ionic* and *Web Audio API*. The creation of different software strategies has been described. In particular the position tracking algorithm, the 3-dimensional audio algorithm and the sound granulator (Sound Generator algorithm). Through these algorithms, the positive and negative peculiarities of these technologies have been examined.

This work is strongly based on synthesizing the museum experience. The two biggest and most important algorithms, the position tracking and the 3D audio system, are developed to provide the user with a better experience of the artworks. They are the main additions with respect to my previous work *Moment V*. *Ionic* is a great instrument for mobile application development. Requiring only *JavaScript* and *HTML*, it is possible to build applications that look like the native one, reaching all functionalities of a mobile device. However, this wrapping process described in chapter four (Overview of Technologies), returns a less performant application than the native one. In this case the performance is remarkable and rarely a problem. The coder must use *Web Audio API* with attention: a process like *Sound Generator* can bring an increment in the computational cost and overload the mobile CPU. Around 16 *audio node*s are continuously created and removed every second. Therefore, I put extreme attention to the placement of the sculptures, to avoid as much as possible sound overlap. *Ionic* is designed to build basic mobile applications; it is not meant to create virtual spaces, navigation systems, and so on. With the right considerations, its efficiency can be very high also for complex cases like the artwork here presented.

The listening of such sculptures through the application may seem particularly strange. Often beta testers of mobile and Internet applications are asked to interact with the software for the developers to draw final conclusions, notice uncommon behaviours and to fine-tune the algorithm itself. This is a usual approach in Internet Art. The interaction with artwork, the simple "do something" and observe your interaction, is an important factor in nowadays art in general, even outside the Internet. I don't find this kind of participation so interesting: Participation becomes the focus and the artwork loses its importance. The interaction in *Virtual Sound Sculptures* is a synthesis of the interaction in a museum. The users do not change the sculptures, but only their position in respect to the artworks. Here there is interactive participation like that one in the museum, but the artwork itself is not interactive. This is one of the reasons why *Virtual Sound Sculptures* can not be completely inscribed within the definition of Internet Art.

This work is particularly conceptual. Indeed its main subject is an extreme state of the music: the stasis - the essence of the *Vertical Music*. It is focused on the potential of time in music and the ways with which music can distort our perception of time. As shown in chapter one (Toward the artwork), this work can be related to Erick Satie, minimalism and Ambient music, those movements or pieces in which the horizontal development of music is questioned. The sculpture image and gallery space introduce another artistic element which qualifies this work more as an art installation than a standard music piece: it is a virtual installation that lives on the Internet. But again, even if the Internet is an essential element we can not establish *Virtual Sound Sculptures* as Internet Art. This issue lies in the Internet Art definition. We should extend this definition to embrace many other artworks which use the Internet as an essential element. Only in this new definition we could define my artwork as Internet Art.

Virtual Sound Sculptures is my last contribution to my research into *Vertical Music*. A journey of four years, which brought me from the publication of the *Musica Verticale* album to the discovery of Internet possibilities in art and finally to the realization of a mobile application. I personally think that this experience has been fundamental. Step after step it led me toward the discovery of new worlds. Worlds that are mostly on the technological field. Nevertheless, the revelation of new technologies has always been the answer to aesthetics and creative needs. This is an important approach in the artistic learning process. Now I feel I can leave this musical topic - *Vertical Music*. I feel the necessity to face new musical research questions. For example concerning the performance's gestures and virtual reality. Not by chance, those are partially introduced in the work of this thesis. *Virtual Sound Sculptures* will continue to be developed. My goal is to fill many spaces around the world with musical sculptures, creating sound sculpture exhibitions in some cities and then parks, urban centers and so on. I'd like to extend the possibilities of the sound generator, and finally I'd like to create a platform in which other artists can create their own sound sculpture exhibitions.

This experience brought me to the discovery of two new fields of interest. A 360° virtual space can strongly enhance the experience of a finite artwork. On the other hand, the realization of the mobile application made me discover the potential of mobile devices. Mobile telephones seem the perfect tools for artistic performance: containing several complex sensors in a small, rather inexpensive and ubiquitous device makes it very interesting for performance and installations. This is especially true considering that one can easily control all data using simple high level applications such as *Ionic* and *Capacitor*. Mobiles can be effectively employed as motion sensors for a dance performance, light sensor for installations, a visualizer for real-time scoring, a simple controller for live electronics.

References

[BD98] C. Phillip Brown and Richard O. Duda. A structural model for binaural sound synthesis. *IEEE Transactions on Speech and Audio Processing*, 6(5):476 – 488, September 1998.

[Beg94] Durand R. Begault. *3-D Sound for virtual Reality and Multimedia*. Ames Research Center, MoffettField, California, 1994.

[Ber22] Henri Bergson. *Durata e simulataneità*. Raffaello Cortina Editore, Milano, via Rossini 4, first edition 2004 edition, 1922.

[Ber89] Henri Bergson. *Saggio sui dati immediati della coscienza*. Raffaello Cortina Editore, Milano, via Rossini 4, first edition 2002 edition, 1989.

[Car15] Thibaut Carpentier. *Binaural synthesis with the Web Audio API. HAL - archives-overtes.fr*, December 2015.

[cor]

[DGS10] Eugenio Denti, Roberto Galatolo, and Francesco Schettini. *An AHRS based on a Kalman Filter for the integration of inertial, Magnetometric and GPS data*. Technical report, University of Pisa, Department of Aerospace Engineering, Pisa, 2010.

[DM03] M. Deseriis and G. Marano. *Net.Art - L'arte della connessione*. Shake Edizioni Underground, Truccazzano (MI), 2003.

[Dou] Brian Duglas Douglas. *Understanding Sensor Fusion and Tracking*.

[Fio17] Alessandro Fiordelmondo. *Moment V & la concezione vertical del tempo secondo Jonathan D. Kramer*. Bachelor Thesis - Conservatory 'Francesco Morlacchi' of Perugia, 2016-2017.

[Fio20] Alessandro Fiordelmondo. *About the Music in the Internet Art*. Jenuary 2020.

[HW97] Christian Hugonnet and Pierre Walder. *Stereophonic Sound Recording. Theory and Practice*. John Wiley & Sons, Chichester, New York, 1997.

[Ked] Karen Kedmey. What is fluxus.

[Kra88] Jonathan D. Kramer. *The Time of Music*. Schirmer Books, 866 Third Avenue, New York, 1988.

[Mén] Sébastien Ménigot. *Pedometer in HTML5 for Firefox OS and Firefox for Android*.

[MTA15] MTAA. *The Absolutely, Positively Last and Final Interview on the Subject of MTAA's "Simple Net Art Diagram." Ever. (AKA The Never-Ending Interview).* Net Art Anthology, 2015.

[oTUA08] Department of Transportation (USA) and Federal Aviation Administration. *Global Positioning System Wide Area Agumentation System (WAAS) Performance Standard*, October 2008.

[Roa01] Curtis Road. *microsound*. The MIT Press, Massachusetts, 2001.

[Scha] Maximilian Schwarzmüller. *Flutter vs React Native vs NativeScript vs Ionic vs PWAs*.

[Schb] Maximilian Schwarzmüller. *Ionic - Build iOS, Android & Web Apps with Ionic & Angular*.

[Smu13] Boris Smus. *Web Audio API*. O'Reilly Media, 1005 Gravenstein Highway North, Sebastopol, first edition edition, March 2013.

[Sta09] J. Stallabrass. *Can Art History Digest Net Art?* In D.Daniels and G. Reisinger, editors, *Netpioneers 1.0 - Contextualising Early Net-based Art*, pages 165–179. Strenberg Press, Berlin, 2009.

[Tur17] William Turner. *Javascript for Sound Artists*. CRC Press Tylor & Francis Group, 6000 Broken Sound Parkway, New York, 2017.

[Ven20] Chris Veness. *Calculate distance, bearing and more between Latitude/Longitude points*, 2020.

Appendix

In this section, the main part of the entire code is provided. The entire code of *Virtual Sound Sculptures* is given in the following GitHub link: https://github.com/alessandrofiordelmondo/virtual-sound-sculptures.
The GitHub repository offers the updated codes, they may be different from those in the text. The application is continually updated.

The **motion tracking service** treated in chapter 5. It is the central system of the application. It provides the position and the heading of the users and their relation with the sculptures.

```
import { Injectable, NgZone } from '@angular/core';
import { Plugins, MotionOrientationEventResult, MotionEventResult } from
    '@capacitor/core';
// import the service for the backend communication
import { BackendService } from './backend.service';
// import the service for the interactive map
import { InteractiveMapService } from './interactive-map.service';
import { Magnetometer, MagnetometerReading } from '@ionic-native/
    magnetometer/ngx';
import { CoordHead, DistBear } from './interfaces'

const{ Motion } = Plugins;
const{ Geolocation } = Plugins;
declare const Pedometer;

@Injectable({
  providedIn: 'root'
})

export class MotionTrackingService{
  private deg = 180 / Math.PI;
  private rad = Math.PI / 180;
  private R = 6371 * 1000 //Earth radious in m
  constructor(
    private zone: NgZone,
    private backend: BackendService,
    private magnetometer: Magnetometer,
    private interactiveMap: InteractiveMapService
  ) {}
  private init:boolean = false;
  sr = 100; //Sample rate HZ
  pms = 1000 / this.sr; //period ms
  ps = 1 / this.sr // period in s
  private interval:any;
  //PEDOMETER variables
  private pedo = new Pedometer();
  private steps:number = 0;
  sensibility:number = 1/20;
  stepSize:number = 70;
  //SENSORS
  private accel = {x:NaN, y:NaN, z:NaN};
  public orient = {x:0, y:0, z:0};
  private magne = {x:NaN, y:NaN, z:NaN};
  private magnetometerWatch:any;
  //HEADING
  heading:number = 0;
  private prevOrientX:number = NaN;
  private refreshCompass:number = 100;
  private hstep:number = 0;
  private declination:number = 0;
  //GPS Listener coordinate
  listenerCoord:CoordHead = {lat: null, lon: null, head: null};
  //Listener to Source relation
  public L2S: DistBear[] = []
```

```
53
54    // RELATIVE ORIENTATION
55    watchOrientation(){
56      return Motion.addListener('orientation', (data:
            MotionOrientationEventResult) => {
57        this.zone.run(() => {
58          this.orient.x = 360 - parseFloat(data.alpha.toFixed(4));
59          this.orient.y = parseFloat(data.beta.toFixed(4));
60          this.orient.z = parseFloat(data.gamma.toFixed(4));
61        });
62      });
63    }
64    // ACCELEROMETER
65    watchAccelerometer(){
66      return Motion.addListener('accel', (data: MotionEventResult) => {
67        this.zone.run(() => {
68          this.accel.x = parseFloat(data.accelerationIncludingGravity.x.
                toFixed(4));
69          this.accel.y = parseFloat(data.accelerationIncludingGravity.y.
                toFixed(4));
70          this.accel.z = parseFloat(data.accelerationIncludingGravity.z.
                toFixed(4));
71        });
72      });
73    }
74    // MAGNETOMETER
75    watchMagnetometer(){
76      return this.magnetometerWatch = this.magnetometer.watchReadings()
77        .subscribe( (data: MagnetometerReading ) => {
78          this.magne.x = data.x;
79          this.magne.y = data.y;
80          this.magne.z = data.z;
81        });
82     }
83     // GPS COORDINATES
84     watchGPS(){
85       return Geolocation.watchPosition({enableHighAccuracy:true}, (data,
             err) => {
86         this.listenerCoord.lat = data.coords.latitude;
87         this.listenerCoord.lon = data.coords.longitude;
88       });
89     }
90     initializeGPS(){
91       return Geolocation.getCurrentPosition().then(data => {
92         return [data.coords.latitude, data.coords.longitude]
93       })
94     }
95     // COMPASS HEADING - true north north(0)east(90)south(180)west(270)
96     compass(){
97       this.heading = -1 * (90 - (Math.atan2( this.magne.y, this.magne.x )
             * this.deg));
98       if(this.heading < 0){
99         this.heading = 360 + this.heading;
100      }
101      this.heading = ( this.heading + this.declination ) %360;
102      this.listenerCoord.head = this.heading;
103    }
104    // COMPASS + ORIENTATION
105    getHeading(){
106      if (this.hstep == 0){
107        this.prevOrientX = this.orient.x;
108        this.compass();
109        this.hstep++;
110      } else {
111        this.heading = ( this.heading + ( this.orient.x - this.prevOrientX
               ) ) % 360;
112        this.prevOrientX = this.orient.x;
113        if (this.hstep == this.refreshCompass){
114         this.hstep = 0;
115       } else {
```

```
116           this.hstep++
117         }
118       }
119       this.listenerCoord.head = this.heading;
120     }
121     // get step points in coordinate form
122     updatePoint(){
123       let lat1 = this.listenerCoord.lat * this.rad;
124       let lon1 = this.listenerCoord.lon * this.rad;
125       let h = this.listenerCoord.head * this.rad;
126       let theta = this.stepSize * 0.01 / this.R; //angular dist
127       let lat2 = Math.asin(Math.sin(lat1) * Math.cos(theta) + Math.cos(lat1
              ) * Math.sin(theta) * Math.cos(h));
128       let lon2 = lon1 + Math.atan2(Math.sin(h)*Math.sin(theta)*Math.cos(
              lat1), Math.cos(theta)-Math.sin(lat1)*Math.sin(lat2));
129       this.listenerCoord.lat = lat2 * this.deg;
130       this.listenerCoord.lon = lon2 * this.deg;
131     }
132     getDeclination(lat, lon){
133       this.backend.fetchDeclination(lat, lon).subscribe( data => {
134         this.declination = data.result[0].declination;
135       })
136     }
137     setSensibility(){
138       this.pedo.sensibility = this.sensibility;
139     }
140     setStepSize(){
141       this.pedo.sensibility = this.stepSize;
142     }
143     start(){
144       this.getDeclination(this.listenerCoord.lat, this.listenerCoord.lon)
145       let orientationWatch = this.watchOrientation();
146       let accelerometerWatch = this.watchAccelerometer();
147       this.magnetometerWatch = this.watchMagnetometer();
148       let GPSWatch = this.watchGPS();
149       this.pedo.createTable(Math.round(2 / this.ps));
150       this.setSensibility();
151       this.setStepSize();
152       //loop
153       this.interval = setInterval(() => {
154         this.getHeading()
155         if (this.listenerCoord.lat != null &&
156           this.listenerCoord.lon != null &&
157           this.listenerCoord.head != null &&
158           Number.isFinite(this.listenerCoord.lat) &&
159           Number.isFinite(this.listenerCoord.lon) &&
160           Number.isFinite(this.listenerCoord.head)) {
161           this.interactiveMap.updateMap(this.listenerCoord.head, this.
                  listenerCoord.lat, this.listenerCoord.lon)
162           let norm = this.pedo.computeNorm(this.accel.x, this.accel.y, this
                  .accel.z);
163           this.pedo.acc_norm.push(norm);
164           this.pedo.update();
165           this.pedo.onStep(this.pedo.acc_norm);
166           if(this.steps < this.pedo.countStep){
167             this.updatePoint()
168             this.steps = this.pedo.countStep
169           }
170         }
171       }, this.pms)
172   }
173   stop(){
174     clearInterval(this.interval)
175     Motion.removeAllListeners()
176     this.magnetometerWatch.unsubscribe()
177   }
178 }
```

Listing 10.1: Motion Tracking Service

The **Sculpture** class treated in chapter 8. It is the merging of the 3D Audio system (binaural) and the Audio Generator (granulator). Is the principal class that is dedicated to the sound.

```
1  // import the backend service
2  import { BackendService } from '../backend.service';
3  // import the audio generator system
4  import { GranulatorNode } from './audio-node/granulator/granulator-node'
5  import { SculptureInterface } from '../interfaces'
6
7  export class Sculpture{
8      constructor(
9          public cntx: AudioContext,
10         public scupture: SculptureInterface,
11         private backend: BackendService,
12     ){}
13
14     sounds: GranulatorNode[] = [];
15     reverb: ConvolverNode;
16     gain: GainNode;
17     public isInit:boolean = false;
18     public isLoaded:boolean = false;
19     public isStarted:boolean = false;
20
21     init(){
22         this.isInit = true;
23         this.reverb = this.cntx.createConvolver();
24         this.gain = this.cntx.createGain();
25         this.gain.gain.value = 0;
26         this.reverb.connect(this.gain).connect(this.cntx.destination);
27         this.backend.getUrl('gs://gallery-audio-database.appspot.com/
               reverb/Freeze 4-Audio.wav').then((url:string) => {
28             this.backend.loadAudioData(url).subscribe((rBuf) => {
29                 this.cntx.decodeAudioData(rBuf, (revBuffer:AudioBuffer)
                       => {
30                     this.reverb.buffer = revBuffer;
31
32                     this.backend.getStorageList(this.scupture.name).then
                           ((arrayUrl:string[]) => {
33                         let idx=0;
34                         arrayUrl.forEach(res => {
35                             this.backend.getUrl(res).then((url:string) =>
                                   {
36                                 let i = this.getIndex(url, this.scupture.
                                       name);
37                                 this.backend.loadAudioData(url).subscribe
                                       ((aBuf) => {
38                                     this.cntx.decodeAudioData(aBuf, (
                                           audioBuffer:AudioBuffer) => {
39                                         this.sounds[i] = new
                                               GranulatorNode(
40                                             this.cntx,
41                                             this.scupture.soundSpec.vox[i
                                                   ],
42                                             this.scupture.soundSpec.param
                                                   [i],
43                                             this.scupture.coords[i],
44                                             audioBuffer
45                                         )
46                                         this.sounds[i].connect(this.
                                               reverb)
47
48                                         if (idx < arrayUrl.length - 1){
49                                             idx += 1;
50                                         } else if (idx == arrayUrl.length
                                               - 1){
51                                             this.isLoaded = true;
52                                         }
53                                     })
```

```
                                })
                            })
                        })
                    })
                })
            })
        })
    }

    start(){
        this.isStarted = true;
        setTimeout(() => {
            this.sounds.forEach((s:GranulatorNode) => {
                s.start()
            })
            this.gain.gain.linearRampToValueAtTime(1, this.cntx.
                currentTime+5)
        }, 1000)
    }

    stop(){
        for (let i in this.sounds){
            this.sounds[i].stop()
        }
        this.reverb.disconnect()
        this.isInit = false;
        this.isLoaded = false;
        this.isStarted = false;
    }

    private getIndex(url:string, name:string){
        /*
        url = http url
        name = sculpture name
        */
        const expression = new RegExp(name+'-\\d\\.mp3');
        const filename = url.match(expression)[0];
        return parseInt(filename[filename.indexOf('-')+1]) -1;
    }
}
```

Listing 10.2: Sculpture class

The **Granulator** class, the Sound Generator treated in chapter 7. This class embraces the binaural node (the 3D audio system). This last class has not been included directly in the *Sculpture* class. The inclusion in the *Granulator* class is done because of an improvement in the performance.

```
// import the grain class
import { Gran } from './gran'
import { ToGranulatorParam, Coord } from '../../../interfaces'
// import the binaural class
import { BinauralNode } from '../binaural/binaural-node';

export class GranulatorNode{
    constructor(
        public audioContext: AudioContext,
        public vox:number,
        public param: ToGranulatorParam,
        public coord: Coord,
        public buffer:AudioBuffer,
    ){
        this.gain = this.audioContext.createGain();
        this.bin = new BinauralNode(this.audioContext);

        this.gain.connect(this.bin.input)

        for (let i=0; i < this.vox; i++){
            this.g[i] = new Gran(
                this.audioContext,
                this.param,
                buffer
            )
            this.g[i].connect(this.gain);
        }
        this.gain.gain.value = this.param.gain;
    }

    public gain: GainNode;
    public g: Gran[] = [];
    public bin: BinauralNode;

    public connect(destination){
        this.bin.connect(destination)
    }

    public start(){
        for (let i in this.g){
            this.g[i].start()
        }
    }
    public stop(){
        for (let i in this.g){
            this.g[i].stop()
        }
        this.bin.disconnect()
        this.gain.disconnect()
    }
}
```

Listing 10.3: Granulator

The **gran** class included in the *Granulator*. This class represents every voice of the *Granulator* and it is responsible for the generation of every grain.

```
import { ToGranulatorParam } from '../../../interfaces'

declare var window, getFrameTime, getFrameDelay

export class Gran{
    constructor(
        public audioContext:AudioContext,
        public param: ToGranulatorParam,
        public buffer: any,
    ){
        this.timeOffSet = this.audioContext.currentTime
        this.gain = this.audioContext.createGain();
    }

    private state:boolean = false;
    private timeOffSet:number;
    private gain: GainNode;
    private sound: AudioBufferSourceNode;

    connect(destination){
        this.gain.connect(destination)
    }
    private play(){
        const cue = Math.random()
        this.sound = this.audioContext.createBufferSource();
        this.sound.buffer = this.buffer;
        this.sound.connect(this.gain)
        const dur = this.sound.buffer.duration;
        const starPoint = cue * dur;
        let cuePoint = ((this.audioContext.currentTime - this.timeOffSet)
             + starPoint) % dur;
        let fTime = getFrameTime(this.param.time, this.param.timeRND);

        if (fTime == null || fTime == NaN || fTime < 0){
            fTime = this.param.time/1000;
        }
        let fCue = cuePoint - fTime;
        if (fCue < 0){
            fCue = 0;
        }
        let fDelay = getFrameDelay(fTime, this.param.dencity)
        this.sound.start(this.audioContext.currentTime, fCue, fTime +
            0.1);
        this.gain.gain.setValueCurveAtTime(window, this.audioContext.
            currentTime, fTime)
        this.sound.onended = () => {
            this.sound.disconnect()
            if(this.state){
                setTimeout(() => {
                    this.play()
                }, fDelay)
            }
        }
    }
    public start(){
        this.state = true;
        this.play()
    }
    public stop(){
        this.state = false;
    }
}
```

Listing 10.4: Gran

The **Binaural** class, the 3D Audio system explained in chapter 6. It is the application of the *Duplex theory* with the 3rd ear.

```
declare var head;
export class BinauralNode{
    constructor(
        private cntx: AudioContext
    ){
        this.input = this.cntx.createGain();
        this.output = this.cntx.createGain();
        this.output.gain.value = 0;
        this.gainRear = this.cntx.createGain()
        this.gainFront = this.cntx.createGain()
        this.merger = this.cntx.createChannelMerger();

        this.iidLeft = this.cntx.createBiquadFilter();
        this.iidRight = this.cntx.createBiquadFilter();
        this.ridRear = this.cntx.createBiquadFilter();
        this.iidLeft.type = "highshelf";
        this.iidRight.type = "highshelf";
        this.ridRear.type = "lowpass";
        this.iidLeft.frequency.value = this.fc;
        this.iidRight.frequency.value = this.fc;
        this.ridRear.frequency.value = this.fc;
        this.ridRear.Q.value = 2;

        this.itdLeft = this.cntx.createDelay();
        this.itdRight = this.cntx.createDelay();
        //LEFT EAR
        this.input
            .connect(this.itdLeft)
            .connect(this.iidLeft)
            .connect(this.merger, 0, 0)
        //RIGHT EAR
        this.input
            .connect(this.itdRight)
            .connect(this.iidRight)
            .connect(this.merger, 0, 1)
        // REAR
        this.input
            .connect(this.ridRear)
            .connect(this.gainRear)
        this.gainRear.connect(this.output)

        this.merger
            .connect(this.gainFront)
            .connect(this.output)
    }
    private iidLeft: BiquadFilterNode;
    private iidRight: BiquadFilterNode;
    private ridRear: BiquadFilterNode;
    private itdLeft: DelayNode;
    private itdRight: DelayNode;
    private merger: ChannelMergerNode;
    private gainRear: GainNode;
    private gainFront: GainNode;
    //INPUT
    public input: GainNode;
    // OUTPUT
    private output: GainNode;
    //BINAURAL VARIABLES
    private head = head/100;                //diameter in m
    private h = this.head/2;            //radious in m
    private c = 343;                    //sound velocity m/s
    private fc = this.c / this.head;    //frequency cut
    //DISTANCE VARIABLES
    public rollOff:number = 1;
    public refDistance:number = 0;
    public maxDistance:number = 200000;

```

```
68      private constant:number = 0.05;
69
70      connect(destination:AudioNode){
71          this.output.connect(destination);
72      }
73
74      disconnect(){
75          this.input.disconnect()
76          this.itdLeft.disconnect()
77          this.iidLeft.disconnect()
78          this.itdRight.disconnect()
79          this.iidRight.disconnect()
80          this.merger.disconnect()
81          this.ridRear.disconnect()
82          this.gainRear.disconnect()
83          this.gainFront.disconnect()
84          this.output.disconnect()
85      }
86
87      move(azimuth:number, distance:number){
88          //azimuth 0 - 360 degrees
89          //distance mm
90          if (azimuth != NaN && azimuth >= 0 ){
91              let g = this.linearDistance(distance);
92              this.output.gain.setTargetAtTime(g, this.cntx.currentTime,
                    this.constant);
93              if(azimuth>180){ azimuth= -(180 - (azimuth - 180))}
94              let frontDel = this.front(azimuth)
95              this.itd(azimuth, frontDel)
96              this.iid(azimuth)
97          } else {
98              return
99          }
100     }
101
102     linearDistance(mm:number){
103         //linear distance implementation
104         return 1 - this.rollOff*((Math.max(Math.min(mm, this.maxDistance)
                , this.refDistance) -this.refDistance) / (this.maxDistance -
                this.refDistance))
105     }
106
107     itd(azimuth:number, frontDel:number){
108         //azimuth 0/180(right) 0/-180(left)
109         //frontDel = front delay in s
110         let itdL, itdR;
111         let theta = (azimuth/180)*Math.PI;
112         if(theta > 0){
113             theta = Math.abs(Math.PI/2 - Math.abs(Math.abs (theta) - Math
                    .PI/2 ))
114             itdL = 0
115             itdR = (( this.h * theta + Math.sin(theta))/ this.c );
116         } else {
117             theta = Math.abs(Math.PI /2 - Math.abs( theta + Math.PI /2 ))
                    ;
118             itdL = (( this.h * theta + Math.sin(theta))/ this.c );
119             itdR = 0
120         }
121         this.itdLeft.delayTime.setTargetAtTime(itdL + frontDel, this.cntx
                .currentTime, this.constant)
122         this.itdRight.delayTime.setTargetAtTime(itdR + frontDel, this.
                cntx.currentTime, this.constant)
123     }
124     iid(azimuth:number){
125         //azimuth 0/180(right) 0/-180(left)
126         let iidL, iidR;
127         if(azimuth>90){azimuth = 180 - azimuth}
128         else if (azimuth<-90){azimuth= -180 - azimuth}
129         iidL = ((azimuth/90) * 9);
130         iidR = ((azimuth/90) * -9);
```

```
131             this.iidLeft.gain.setTargetAtTime(iidL, this.cntx.currentTime,
                    this.constant)
132             this.iidRight.gain.setTargetAtTime(iidR, this.cntx.currentTime,
                    this.constant)
133         }
134         front(azimuth:number){
135             //azimuth 0/180(right) 0/-180(left)
136             let az = Math.abs(azimuth)
137             if (az > 90){
138                 let rear = (az-90)/90;
139                 let front = 1 - rear*0.8
140                 let theta = (az/180)*Math.PI - Math.PI/2;
141                 let frontDel = (( this.h * theta + Math.sin(theta))/ this.c )
142                 this.gainRear.gain.setTargetAtTime(rear*0.8, this.cntx.
                        currentTime,this.constant)
143                 this.gainFront.gain.setTargetAtTime(front, this.cntx.
                        currentTime,this.constant)
144                 return frontDel;
145             } else {
146                 this.gainRear.gain.setTargetAtTime(0,this.cntx.currentTime,
                        this.constant)
147                 this.gainFront.gain.setTargetAtTime(1,this.cntx.currentTime,
                        this.constant)
148                 return 0;
149             }
150         }
151
152 }
```

Listing 10.5: Binaural