Auralization of Movement

Flora Valentina Latz

Bachelor's thesis

Auralization of Movement

Flora Valentina Latz

29. September 2017

Institute for Data Processing
Technische Universität München

Flora Valentina Latz. *Auralization of Movement.* Bachelor's thesis, Technische Universität München, Munich, Germany, 2019.

Supervised by Prof. Dr.-Ing. K. Diepold and Stefan Röhrl; submitted on 29. September 2017 to the Department of Electrical Engineering and Information Technology of the Technische Universität München.

Institute for Data Processing, Technische Universität München, 80290 München, Germany, `http://www.ldv.ei.tum.de/`.

Abstract

The system proposed in this thesis deals with auralization of movement. A Microsoft Kinect sensor is used to extract motion data from a user by using depth and color information provided by the infrared and Red-Green-Blue Color Space (RGB) camera of the Kinect. Therefore, the user can move freely without having any sensors attached to his body which disturb his kinetic flexibility. The movement data is transferred to harmonious sounding musical output based on defined mapping criteria. The sound output is influenced and based on the user's input and can thus be actively controlled by him. As a result of the harmonious sounding aural output, the user himself is animated in his movement leading to a creativity inspiring circle. In order to implement the entire process, a python algorithm was programmed using the *pykinect*, *SQLite*, *pydub* and *playsound* libraries. The specifics of the system as well as the algorithm itself and its development are the elements for this thesis.

Contents

1. Introduction

1.1. Motivation

One cannot not interact.

(Paul Watzlawick)

Humans are bound to communicate, even trying to do nothing or more precisely trying not to communicate is a specific form of interaction. As computers and machines are attributed a greater and greater part in our daily life, Human Computer Interaction is getting an important aspect which results in a lot of research and evaluation concerning this area. The goal is to make Human Computer Interaction more intuitive, smarter, faster and more comfortable e.g. interacting without being restricted to any input devices resulting in systems like gesture recognition, body tracking and motion detection. As our world is a 3D space, Human Computer Interaction is bound to be able to understand and handle three dimensions in order to interact properly with physical objects.

A considerable component of communication consists of non-verbal communication including kinesics which is the movement of the body [1]. A human body consists of 206 bones which results in over 230 moving parts. Every joint again has up to three degrees of freedom, with respect to the torso these degrees of freedom, however, add up resulting in a total of 17 degrees of freedom for a distant finger of a hand according to Jones et al. [2]. Considering all these different possibilities to move, there is a tremendous amount of possible change in position. Correctly identifying this wide range of movement, however, is a challenge itself. It can be established through sensors placed on specific parts of the body which can track velocity and acceleration or through video recordings. The former has the disadvantage of restricting the user's movement and diminishing his comfort, but the later is limited through camera angle, camera view field and frames per second. This development in tracking sensors enables a new platform for interaction between scientists and artists considering music, dance and drama. Electronic music for instance is the art to define the relationship between sound generation and gesture analysis. This allows for a new field for creativity combining technology and conventional art and also serves as a rising application area for the music entertainment industry.

The proposed system in this thesis also deals with this kind of creativity. The user creates music interactively through his body movement tracked through a Kinect sensor, which observes depth and color information using infrared cameras and an RGB array. This tracking method enables the user to move freely in the physical 3D space without restrictions allowing for more expressiveness and it permits full body motion tracking, more specific skeleton tracking. As a result, the Kinect sensor serves as a real time music controller, extracting movement data and transforming it to musical output. The user should be able to understand his influence on the output to actively control the music via his motion input. Furthermore, the generated sound should always sound harmonic to inspire the user in a positive way. As a result, the system establishes a creativity inspiring circle because the user's creativity is animated through the output of the system which itself is influenced by the user's creativity. The creative act itself is carried out, on the one hand, by the user and, on the other hand, by the one creating the mapping between motion capture and sound.

1.2. Problem Statement

The system can be divided into two parts: First, the motion analysis and, second, the sound generation. The motion analysis extracts the user's movement data from the Kinect and defines features, which are significant for human movement. Questions leading this part of the system are:

- Which data is provided by the Kinect?
- How can the data be stored?
- How can the data be processed for further usage?
- How should the data be used to ensure real time usage?

The sound generation system has the purpose to create the sound output. Questions concerning this part are:

- How is a harmonic acoustic pattern created?
- What are the essential parts of a music piece?
- How can these parts be created and combined?
- Which music elements should be used reaching from individual notes over chords to whole melodies?

The interface between motion analysis and sound generation consists of the mapping between the movement features extracted from the motion analysis and the music data generated by the sound generation system. The basis of this interface are the following questions:

- Which movement features are mapped to which kind of music elements?
- Should the system mimic a music instrument or provide music in a more generative way?
- How can be guaranteed that the user has influence on the musical output?

This system belongs to the area of research for computer and creativity which examines methods how creativity can be implemented in technical systems and algorithms. Furthermore, it is affiliated with the scope of data processing as movement data, on the one hand, and music data, on the other, have to be processed to accomplish the systems objective. The research topic concerning computer and creativity is not only relevant for electrical engineering but also for informatics. Moreover, it is also a matter of interest for the art division concerning dance, music and drama. These subject areas are more and more interweaving with the technical world as technical art approaches like generative adversarial networks for creating art seen at Ahmed Elgammal et. al. [3] get further attention and are considered as forward-looking and trendsetting methods. The proposed system is therefore combining the fields of electrical engineering and informatics in the sense of data processing as well as the fields of music and art concerning the sound generation and creativity process.

1.3. Thesis's Structure

The remainder of this thesis is organized as follows: The second chapter elaborates about the current State of the Art and presents papers and projects which deal with similar problems and tasks. Furthermore, it analyzes the difference with respect to the proposed system and defines this thesis's objective. The third chapter is devoted to the music theory and definition of musical terms needed as theoretical background, whereas the fourth chapter presents the system itself, describes which hardware and software components are used as well as the reasoning for that and explains how the algorithm itself is constructed. Chapter five discusses the results taking into account different evaluation criteria.

2. State of the Art

Concerning related work to express music based on motion capture systems one can differentiate between two major approaches. On the one hand, there are systems based on an instrument imitating e.g. a percussion instrument or a guitar. On the other hand, one can define systems which create music in a more generative way, not comparable to any existing common instrument.

2.1. Instrument Based Sound Output Systems

2.1.1. Theremin

As stated in [4] and [5, pp.139 ff.] the Theremin is an instrument invented 1920 by Leon Theremin, a Russian physicist. It is played without any physical contact but through the movement of one's hands in the near range of two antenna which built the main part of the instrument displayed in Figure 2.1. The proximity to the vertical antenna controls the pitch, whereas the proximity to the horizontal one controls the volume. The principle of the Theremin is based on the overlap of oscillations emitted from two beat-frequency generators. To change the beat frequency one of the frequencies of the two oscillators is altered slightly. As the electromagnetic field is influenced even by little movements, playing the Theremin is considered to be difficult and therefore only few people are able to control pitch and volume in that way. The act of playing is based on continuous aural feedback, which is similar to the proposed system of the thesis.

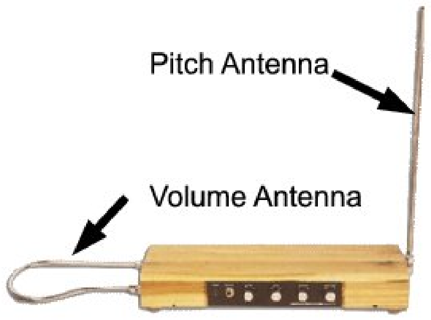

Figure 2.1.: Theremin setup [6]

2.1.2. Virtual Instrument Inspired Sound Output Systems

Furthermore, one can produce music by imitating instruments which results in virtual instruments. This method will be explained at the example of Mu-Hsen Hsu et al. [7]. Furthermore, differences to other approaches shall be elaborated.

Mu-Hsen Hsu et al. in [7] introduced three virtual instruments based on the Kinect: a drum, a guitar and a newly invented instrument named Spider King. For this system's approach, input areas are defined e.g. strings of the guitar or cymbals of the drum. When these input areas are triggered detected through the Kinect, a Musical Instrument Digital Interface (MIDI) event message is generated which causes a sound through an audio library, in this case Virtual Studio Technology [1]. The MIDI message defines pitch, notation and velocity, which is then realized through the Virtual Studio Technology simulating the sound of real instruments.

For the virtual drum, four input areas are defined: Kick, Snare, Cymbal and Hi-Hat as displayed in Figure 2.2. To trigger the proposed areas, right hand, left hand and right knee are captured by the Kinect. "When the coordinate of the triggering point is larger than a specified threshold with respect to the defined regions of the virtual drum sets, program triggers MIDI signals, and then MIDI signals triggers the sound in audio library" [7].

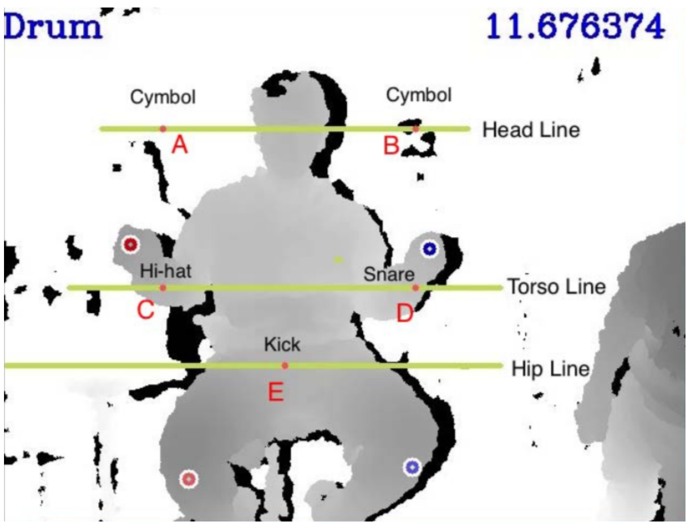

Figure 2.2.: Virtual drum setup with trigger points [7]

For the guitar, however, the input areas as it can be seen in Figure 2.3 consist of a chord selection of six areas set in front of the user's left hand supplemented by a virtual guitar string in front of the user's right hand.

The last instrument, the Spider King, is implemented through a virtual circle

[1]https://www.steinberg.net/de/products/vst.html, Accessed: 11.8.19

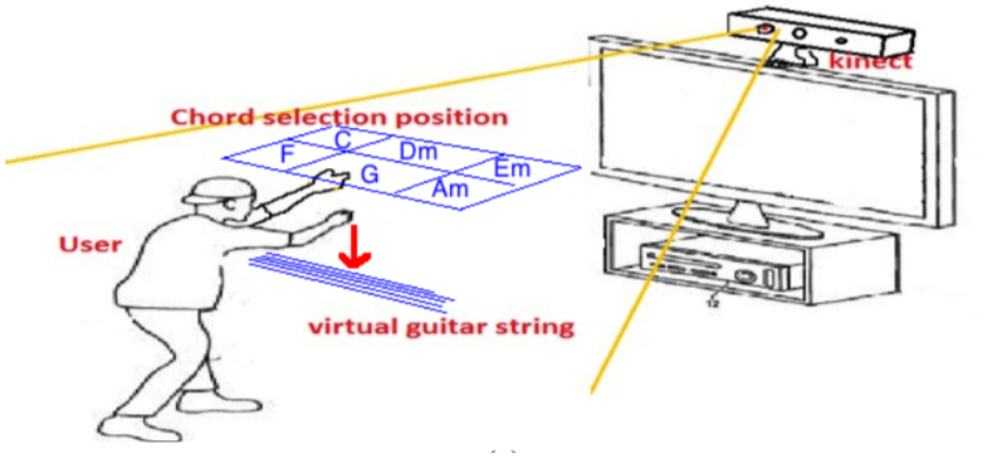

Figure 2.3.: Virtual guitar setup with chord selection position [7]

around the user shown in Figure 2.4. The sound can be triggered through the hand movement, while the degree of the circle is mapped to notes and the distance from the circle's center specifies the low, mid and high tones of the specific note. The volume is altered dependent on the distance of the hand from the Kinect Sensor.

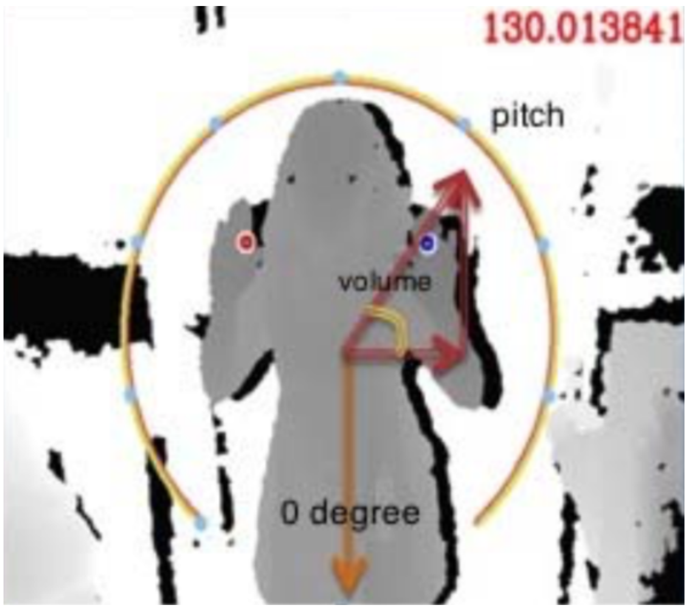

Figure 2.4.: Spider King setup [7]

Furthermore, Enkhtogtokh Togootogtokh et al. [8] invented a virtual cello and a virtual piano. The visualization of the system can be seen in Figures 2.5 and 2.6.

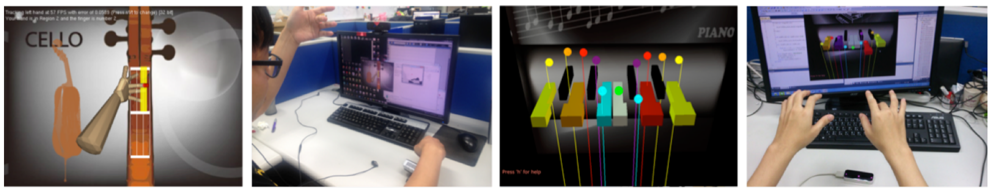

Figure 2.5.: Virtual cello and virtual piano setup [8]

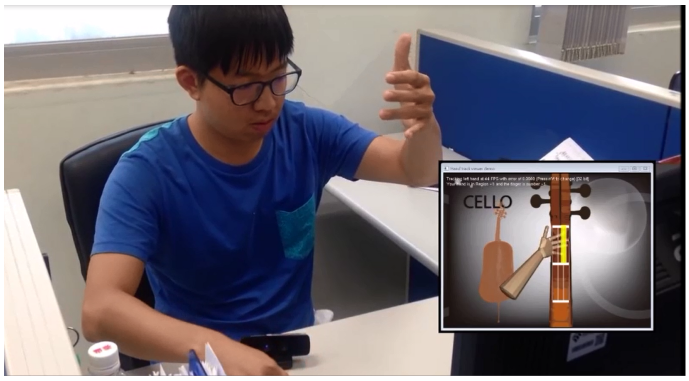

Figure 2.6.: Virtual cello setup [8]

The difference to the system of Mu-Hsen Hsu et al. is that they used Senz3D [2] and Leap Motion [3] instead of the Kinect as they yield the advantage of tracking hand gestures more precisely. In addition, they used a Neural Network or detecting finger gestures for the virtual piano.

In [9, pp.80 ff.] Abassin Sourou Fangbemi and Yanxiang Zhang present a virtual percussion instrument. However, the instrument obtains the disadvantage to use a wireless body-worn inertial measurement unit which decreases the user's freedom in motion.

Shuyao Li et al. realized in [10] a virtual Chinese instrument, named Guzheng. The unusual instrument led to the additional work to prerecord the sound of the Guzheng strings and establish a timbre database.

Athanasia Zlatintsi at al. in [11] concentrates on a virtual air guitar and an upright bass. The characteristics of this system's approach, contrarily to the already proposed ones, are the visual aids invented to help interact with the virtual instruments, e.g. "colored bars accompanied with letters, which show the user which note

[2]https://de.creative.com/p/web-cameras/blasterx-senz3d, Accessed: 11.8.19
[3]https://www.leapmotion.com, Accessed: 11.8.19

is going to be generated if he performs a sound activation gesture" [11] illustrated in Figure 2.7. Furthermore, the air guitar differs from the approach of Mu-Hsen Hsu et al. [7] as the air guitar mode can be altered between two modes, either consisting of single notes or of complete chords.

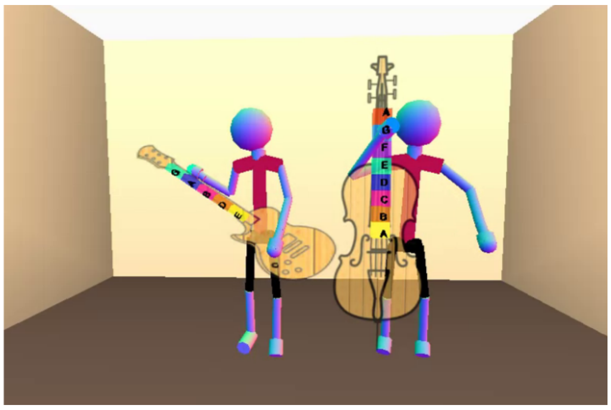

Figure 2.7.: Visual aids of the virtual air guitar and upright bass [11]

Sarika U. Aswar at al. in [12] realized several virtual instruments using a Kinect sensor: a drum, xylophone and a piano. However, the drum is, contrarily to Mu-Hsen Hsu's et al. [7], based on the less reliable approach of color and contour tracking. Instead of motion tracking, the different colored drumsticks are detected and the centroid of the tip is identified as it can be seen in Figure 2.8.

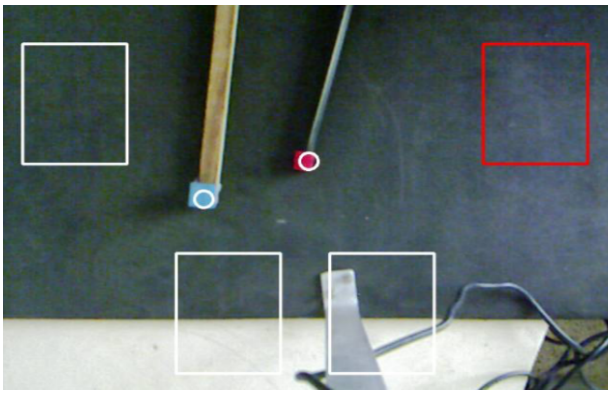

Figure 2.8.: Visualization of the drumstick detection [11]

In conclusion, considering these different virtual instrument approaches, the method of Mu-Hsen Hsu's et al. [7] and Enkhtogtokh Togootogtokh et al. [8] are the best developed ones among them.

2.2. Generative Sound Output Systems

2.2.1. Systems to control Ableton Live

There are systems which enable communication of the Kinect with Ableton Live, a software music sequencer [4]. In [13, ch. 4] Jared St. Jean explains step-by-step how to use the Kinect sensor to regulate specific audio units. The system is based on OSCeleton [5], which is used to send the Open Natural Interaction (OpenNi) skeleton data as Open Sound Control (OSC) messages to the music application.

Other than that mentioned OSCeleton, there also exists another software called Synapse [6]. Similar to OSCeleton it is used to control systems like Ableton Live [7] on the basis of the data extracted by the Kinect Sensor.

2.2.2. Sound Generation through frequency modulation

Tamara Berg et al. [14] introduced a system which maps human motion onto musical notes. To track the human motion, a Kinect is used for skeleton tracking which yields 15 joint coordinates. In order to accomplish a mapping between movement and sounds, several features of every joint contribute to generate a certain note. These features consist of velocity, acceleration and change of location of specific body parts. In order to work with the skeleton coordinates, OpenNi framework [15] and NITE Middleware from PrimeSense [16] is used. The sound generation is realized through frequency modulation, which takes a carrier frequency, a modular frequency and an amplitude as input and gives a complex waveform as output. Each joint differs in its frequency modulation and therefore representing its own unique instrument. “The goal was that each joint should have its own customizable sound source and that the performer and audience should easily be able to discern the sound changing and have a general idea of which sounds are coming from which joints” [14].

2.2.3. Sound Generation through a mapping function

Min-Joon Yoo et al. in [17] have a similar approach using OpenNi and NITE [15, 16] but they divide the skeleton into 5 parts e.g. left arm and right arm. Every segment is assigned to one sound being described through several parameters. These parameters are mapped to the velocity of the joints out of which one body segment consist e.g. left hip, left knee, left ankle and left food yielding the left leg. As a result, the sound is an outcome of position and velocity of the body movement. Instead of

[4]https://www.ableton.com/de/live/, Accessed: 12.8.19
[5]https://github.com/Sensebloom/OSCeleton, Accessed: 12.8.19
[6]https://synapsekinect.tumblr.com/post/6307790318/synapse-for-kinect, Accessed: 22.6.19
[7]https://www.ableton.com/de/live/, Accessed: 12.8.19

working with frequency modulation like Tamara Berg et al. [14] they implemented a mapping function: the speed of the sound is mapped to the velocity of each joint, whereas the pitch of the sound corresponds to the position, e.g. the relative position of the end joints, meaning “if the end-joints [. . .] [are] closer to each other than the normal pose" [17, Eco], the scale gets more tense.

Kevin Clark used the mentioned Synapse software [6] to build a different sound generation system [8]. Contrary to Min-Joon Yoo et al. in [17], he mapped certain gestures to music output, implementing two applications: Point Wellness and Puppet Master. The former maps different skeleton positions like lifting both arms to specific sound effects, which influence the underlying wellness sound as it can be seen in Figures 2.9 and 2.10. Puppet Master enables the user to control for example the drum beat or groove through certain gestures. The system is mainly used for health and wellness applications e.g. for enrichment of children at risk or diagnosed with developmental delays.

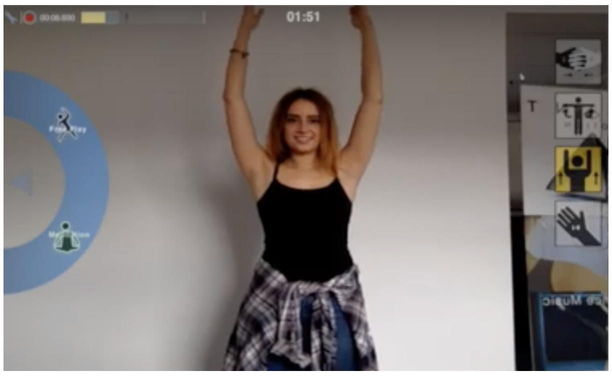

Figure 2.9.: Visualization of the Point Motion’s gestures 1 [9]

2.2.4. Sound Generation through combined methods

Lisa Naugle et al. [18] enhanced the generative system by implementing several mapping possibilities instead of only one like Tamara Berg et al. [14] and Min-Joon Yoo et al. in [17]. On the one hand, the stage can be separated in various parts, each of them mapped to particular sounds played when the dancer is detected in that particular area. In addition, change of direction is used to produce rhythmic sound. On the other hand, preexisting soundtracks can be altered by parameters influenced through the dancer’s movement. This can be implemented using frequency modulation like Tamara Berg et al. in [14] did. As a third mapping option, the pitch

[8]http://www.pointmotioncontrol.com/pilotprogram, Accessed: 24.6.19
https://www.bostonmagazine.com/health/2016/05/09/point-motion/, Accessed: 24.6.19
[9]https://www.youtube.com/watch?v=CYzH4JxKaGQ, Accessed: 12.8.19

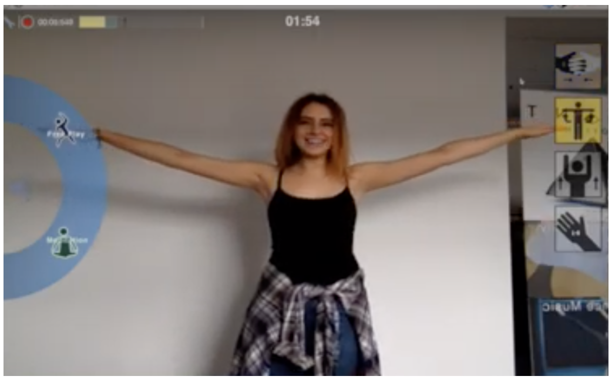

Figure 2.10.: Visualization of the Point Motion's gestures 2 [9]

and/or timbre can directly be mapped to the extracted movement characteristics like Min-Joon Yoo et al. in [17]. Mapping the pitch, however, is stated to sound mechanical. In order to implement the whole system, three libraries are established: One motion capture library for movement characteristic, one sound library and one mapping library for the various mapping possibilities.

Nagual Sounds by Artur Reimers and Mark Moebius [10] achieved the most harmonic sounding output with their system and therefore, the dancer again is inspired by the music inducing a creativity supporting loop.

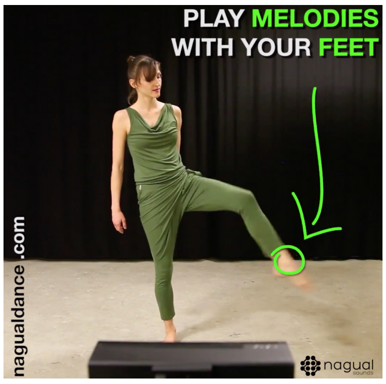

Figure 2.11.: Visualization of feet functionality of Nagual Sounds [11]

To control the music the functionalities of legs and arms are separate as seen in Figures 2.11 and 2.12. The legs control the trumpet and piano, can switch between

[10]https://www.heise.de/tr/artikel/Der-Koerper-macht-die-Musik-2797589.html, Accessed: 22.6.19
https://www.golem.de/news/kinect-traumtaenzer-1402-104581.html, Accessed: 22.6.19
[11]https://youtu.be/4Tgz1AMLyvg, Accessed: 12.8.19

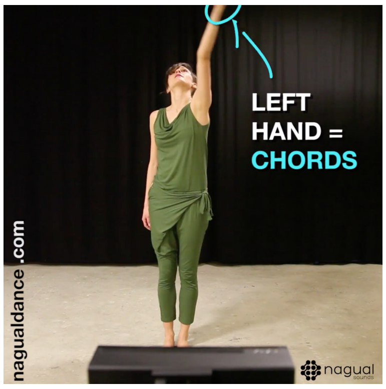

Figure 2.12.: Visualization of hand functionality of Nagual Sounds [11]

several ton modules and create a melody. The left hand controls the drums, whereas the right hand influences the guitar sounds.

Furthermore, the dance-floor itself is divided in four different parts displayed in Figure 2.13, where one can switch between chords and tempo in order to implement quiet as well as ecstatic parts in one song.

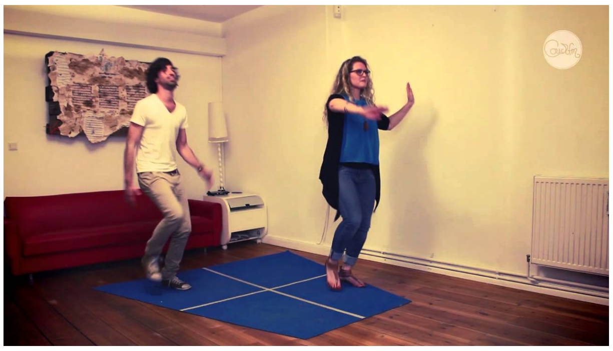

Figure 2.13.: Visualization of Nagual Sounds's dance floor [12]

Additionally, as illustrated in Figure 2.13 the software also supports two user dancing at the same time, where the music is divided into melody instruments and rhythmic instruments. The user gets visual feedback in order to better understand his influence on the music. In addition, the user can change between electronic music genres like house, ambient and dubstep. The basis of the song is predefined in the system, the user, however, has the possibility to interact with the predefined song through his movement. The goal for the future is to enable other developer to create sound packages for the Nagual Sounds software, which will be included as additional packages.

[12]https://www.youtube.com/watch?v=cd34a7v68BI, Accessed: 12.8.19

Concluding, one can say, that Lisa Naugle et al. [18] and Artur Reimers and Mark Moebious with Nagual Sounds [13] invented the best developed wholesome systems, combining features of Tamara Berg et al. [14] and Min-Joon Yoo et al. in [17].

2.3. Thesis's Objective

The objective of the proposed system is not to build a new kind of instrument like the Theremin. The hardware components are limited to the Kinect sensor-based motion detection and should not include antennas, strings or other instrument components. Furthermore, there is no specific instrument to serve as a basis for the sound generation system like it can be seen at Shuyao Li et al. in [10] with the Chinese Guzheng or Athanasia Zlatintsi at al. [11] with the air guitar and upright bass.

As a conclusion, the sound generation system pursues a more generative approach for implementing the musical output like Tamara Berg et al. [14] and Min-Joon Yoo et al. in [17] did. Similar to the mentioned approaches, the motion analyses system of this thesis is based on the entire body instead of only focusing on the hand or hands like Enkhtogtokh Togootogtokh et al. in [8] and other similar methods implementing virtual instruments, which like their real counterparts are played by hand(s). The scope of the proposed system is to use the whole body's motion as input for the musical output. Therefore, the motion tracking system has to be capable to track the entire body. Nonetheless, it does not have the requirement, to be as specific as the motion tracking system where single finger motion is crucial for the musical output like it is needed for the system of Enkhtogtokh Togootogtokh et al. [8]. This thesis does not cover any other tracking devices apart from the Kinect, therefore, space and accuracy of the movement analysis are limited.

Furthermore, the thesis's project is not web based like the approach of Athanasia Zlatintsi at al. in [11] but a desktop application with a locally stored library. It does not rely on other projects like OpenNi and NITE Middleware to communicate with the Kinect as Min-Joon Yoo et al. [17] and Jared St. Jean [13, ch. 4] with their projects do, but uses a python library called pykinect to ensure more flexibility in the way the motion data is stored and used.

The system is not based on the recognition of several predefined gestures but takes the body as a whole entity to produce the sound. There are systems like Enkhtogtokh Togootogtokh et al. in [8] and Athanasia Zlatintsi at al. in [11] scanning the motion data for certain determined hand gestures which then event-based trigger some sound. Furthermore, there are systems e.g. Sarika U. Aswar at al. in [12] and Mu-Hsen Hsu et al. in [7] which use a combination of gesture recognition and location tracking. When a specific gesture is observed by the system, a sound is played, however, the location where the gesture has taken place defines which

[13]https://www.heise.de/tr/artikel/Der-Koerper-macht-die-Musik-2797589.html, Accessed: 22.6.19
https://www.golem.de/news/kinect-traumtaenzer-1402-104581.html, Accessed: 22.6.19

kind of note is played. In contrary, the proposed algorithm is not triggered by any prescribed hand gestures but tries to extract a rhythm from the movement of the entire body. The system of point motion [14] uses whole body gestures like standing straight arms reaching to the side instead of single hand gestures which resembles more the whole-body approach proposed here.

Moreover, some projects implement a continuous approach like Tamara Berg et al. [14] and Min-Joon Yoo et al. in [17] where each joint or the velocity of the joint is mapped to a sound. As a result, the music output is a result of position and velocity of the body's movement. This method, however, does not strive for an overall harmonic sound pattern or a musical output which at itself is enjoyable. This forms the difference between this continuous approach and the outcome implemented by the presented algorithm.

The first part of the proposed system tries to derive a rhythmic pattern from the body's movement, which is similar to one of the methods of Lisa Naugle et al. in [18], where the change of direction is used as an indicator to define a beat through pendular movement. The reminder of the system seeks to implement a harmonic music piece. Visual feedback like it was realized by Tamara Berg et al. [14] or any user interface exceeding the aural feedback from the system is out of the scope of this thesis.

In conclusion the objective of this thesis's system opposed to the already existing state of the art projects can be expressed as follows: In order to create a harmonic sound pattern as an output to generate a creativity inspiring circle, the system implements an approach using the whole body instead of only body parts. In contrary to recognizing certain gestures mapped to specific sounds, a rhythm is detected. The user should be able to actively influence the musical output, which should be generated in real time. Therefore, three criteria can be extracted, further evaluated in Chapter 5: Firstly, a real time approach should be implemented, secondly, the user ought to be able to influence the musical output by his entire body motion and thirdly, the aural output shall sound harmonious and creativity inspiring.

[14]http://www.pointmotioncontrol.com/pilotprogram, Accessed: 24.6.19
https://www.bostonmagazine.com/health/2016/05/09/point-motion/, Accessed: 24.6.19

3. Music Theory

3.1. Musical Terms Definition

Music theory is used to differ between various music pieces and categorize them into different genres where the music elements are similar or the same. *Rhythm* and *pitch* are the most basic elements music is built with [1]. Among these, there are *harmony*, *timbre* and *texture* which define a music piece [19].

In the following the terms needed for this thesis will be explained: Catherine Schmidt-Jones and the website "Deciphering Music Theory" explain a *rhythm* as follows: [19][1] Music is dependent on time, therefore, without time there is no music. *Rhythm* is the placement of pulses in time. A *rhythm* is hence a repeating pattern organizing the music into *bars* and *phrases*. It is created through a repetitive pulse. Visualizing speaking the *rhythm* is "the part of the song your toes tap along to and your head nods to" [2].

The *beat* can have different meanings but in this thesis is referred to as a repetitive rhythmic pattern which maintains a steady pulse. However, there does not have to be a pulse on every *beat*. [19]. One *beat* can have different values e.g. quarter notes or eighth notes. The *beats* are organized in *bars*, which usually contain the same amount of *beats* with a leading stronger *beat* at the beginning [19]. The *time signature* indicates how many *beats* are grouped into one *bar* by the upper number, whereas the lower number displays the *beats* value [3].

The *tempo* is the velocity in which the rhythmic pattern or the *beat* is played. It is measured in Beats per Minute (bpm). The genre is also widely influenced by the *tempo* as e.g. hip-hop has a bpm ranging from 60-100 and techno from 120-140 [4].

The musical output of the proposed system is based on *loops*. A *loop* is a short piece of music which can be repeated seamlessly and therefore be used to build up an entire song. It is especially useful to create a drum *rhythm* as this usually consists of a 1- or 2-bar *rhythm* repeated over time throughout the music piece [5].

According to Catherine Schmidt-Jones [19] *harmony* is established through more

[1] http://decipheringmusictheory.com/?page_id=662, Accessed: 25.7.19

[2] https://www.dummies.com/art-center/music/how-to-establish-rhythm-in-music-theory/, Accessed: 25.7.19

[3] https://music.tutsplus.com/tutorials/the-theory-of-rhythm-in-music–cms-19823, Accessed: 25.7.19

[4] https://learningmusic.ableton.com/de/make-beats/tempo-and-genre.html, Accessed: 25.7.19

[5] https://www.dummies.com/art-center/music/recording-music/creating-musical-loops/, Accessed: 25.7.19

than one *pitch* playing at the same time. It is defined through the relationship of the notes that happen in parallel. “Harmony does not have to be particularly ‘harmonious’” [19], however, in this thesis the goal is to create a harmonious sounding *harmony* consisting of a drum line and a melodic line, which in this case is a guitar line. The melodic line “is just another term for the string of notes that make up the melody” [19].

3.2. Human Perception of Rhythm

As stated by Sam Brinson [20] “a good rhythm is imperfect”. Sound generated by computers without the help of neural networks lack a certain humanoid effect to be as enjoyable as human fabricated music. This fact is also encouraged by Taylor Beck [21] who researched in this area with the intend to find a better way to humanize computer-generated music.

Furthermore, Bruno Repp elaborates in his article [22] that mistakes in musical performances, like a note played wrong, a left out note or a played note where none was written, often go unnoticed. In his study only 38% of the errors were detected by musicians themselves.

These facts built an important factor for this system’s real time evaluation further explained in Chapter 5. As imperfections and mistakes in the rhythm are not disturbing the harmonic perception of a sound pattern but somehow can enhance the musical experience, the system is left with enough time to detect a beat and output it.

4. Implementation

4.1. System's Components

4.1.1. Hardware Elements

Microsoft Kinect

For this thesis a Microsoft Kinect was used as motion capture system. Developed primarily for a gaming console, measurements are "affected by some unavoidable error sources" [23]. However, the accuracy still satisfies the requirements for the proposed purpose. In order to assure more precise skeleton tracking, a Kinect version 2 is used, which outruns the version 1 in precision [24]. The Xbox One model 1520 used in this thesis was released on November 22, 2013 [1].Together with a universal serial bus adapter it is used as a Kinect for Windows v2.

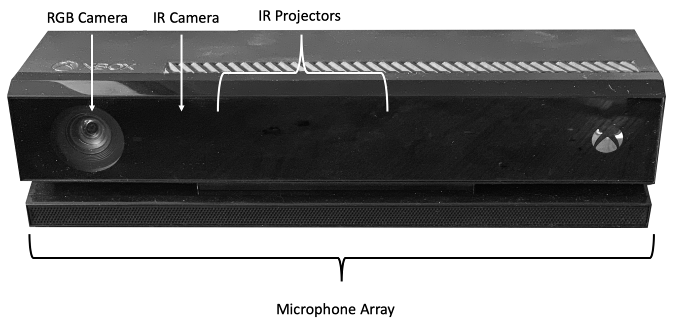

Figure 4.1.: Hardware structure of Kinect version 2

The technical details are as follows [23, 10]: The Kinect version 2 consists of two cameras, a depth sensor and a microphone. The two cameras can be seen in Figure 4.1: an RGB camera on the left and an infrared (IR) camera to the right of the RGB camera. The IR camera is supported by three projective pulse modulated

[1] https://www.spiegel.de/netzwelt/gadgets/microsoft-konsole-xbox-one-erscheint-am-22-november-im-deutschland-a-920421.html, Accessed: 17.7.19

infrared projectors situated to the right of the infrared camera. These illuminate the examined space and rough objects, pervade glass and form speckles which can be read by the IR camera. The RGB camera provides a full high definition color picture with 1920 x 1080 pixels at a maximum of 30 frames per second, whereas the IR camera resolution is 512 x 424 pixels. The depth sensor is based on Range Camera technology provided by PrimeSense. It applies the Time of Flight method to generate the depth image from the information of the IR spectrum. “The depthmaps are 2D images 16 bits encoded in which measurement information is stored for each pixel” [23]. The depth map origins from the same lense as the IR camera and provides the same resolution.

The speech recognition and sound source location are based on four microphones. The sound system supports 3D stereo speech, digital signal processing and filters the background noise at the same time. The set of quaternion microphone arrays is located at the bottom of the Kinect as it can be seen in Figure 4.1.

Functions provided by the Kinect version 2 are dynamic motion capture, image recognition, speech recognition and aural input. The sensor is able to track six skeletons parallel, each supporting 25 skeletal body joints. In addition, the player’s heart rate and facial expression can be observed as well as the weight put on each limb and the velocity of these. Furthermore, the Kinect version 2 also supports thumb tracking and can therefore be used to identify hand gestures. The operative measuring range reaches from 0.5 m to 4.5m and has an angle of view of 70 x 60 degrees. A summary of the technical details can be seen in Table 4.1.

Table 4.1.: Technical Features of Kinect version 2 Sensor [23]

Description	**Value**
Infrared (IR) camera resolution	512×424 pixels
RGB camera resolution	1920 × 1080 pixels
Field of view	70×60 degrees
Framerate	30 frames per second
Operative measuring range	from 0.5 to 4.5 m
Object pixel size (GSD)	between 1.4 mm (@ 0.5 m range) and 12 mm (@ 4.5 m range)

The sensor provides skeleton tracking based on an algorithm developed by the Microsoft Research Institute of Cambridge using the depth image. The technology is based on the Time of Flight method which works as follows: the observed scene is illuminated by the infrared projectors. The camera measures for every image point the time needed for the light to travel from emitter to target and back. Considering that the measured time is proportional to the distance-to-object measurement, for every image point the distance of the observed object can be calculated.

The Kinect version 2 supports point clouds as well as colorized point clouds. The first step to generate a point cloud is to create a pixel matrix with depth values based on the depth map. To transform the two dimensional data into 3D data, the provided mapping function from the Software Developement Kit (SDK) can be used or an implementation using perspective projection can be applied, both methods resulting in a list of X,Y,Z – coordinates. The list can then be represented as a point cloud. In order to obtain a colorized point cloud, the color image has to be used. However, it has to be transformed first as it supports a different resolution. For the transformation of the colorimetric information a mapping function from the SDK is applied returning a pixel matrix with color information for ever pixel of the depth map. In the last step the list of X,Y,Z – coordinates is combined with the color information and displayed as a colorized point cloud. The whole process is illustrated in Figure 4.2.

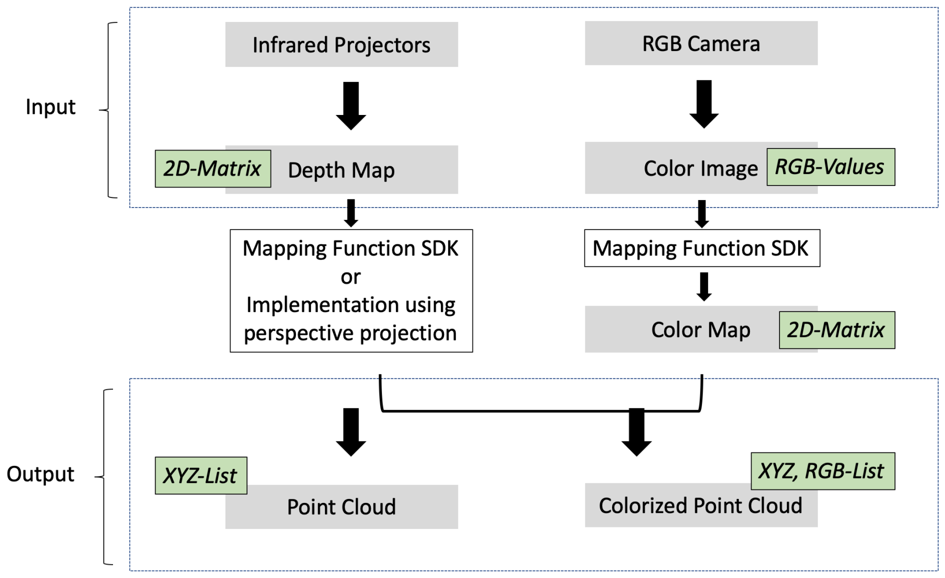

Figure 4.2.: Schematic representation of input and output of Kinect version 2

4.1.2. Software Elements

Kinect Data Processing

For the proposed system the programming language Python was used to implement the algorithm explained further in Section 4.2. Python provides an extensive number of libraries which can be included to limit the complexity of the code. As a result, the code not only gets simpler but also shorter, which assists to keep a better overview over the code's structure. Furthermore, it is simple to use and comparatively easy to learn. In addition, for implementing a virtual environment the open source distribution Anaconda was used as well as PyCharm by JetBrains [2] as integrated development environment.

To interact with the Kinect on programming level, the open source library *pykinect2* was included available on GitHub [3]. For working with this library, Anaconda 32-bit version, including *NumPy*, the *comtypes* library and the Kinect for Windows SDKis needed. Anaconda [4], on the one hand, is a Python and R programming language distribution for data science and machine learning. It facilitates libraries, package and environment management. The SDK, on the other hand, "enables developers to create applications that support gesture and voice recognition, using Kinect sensor technology on computers running Windows 8, Windows 8.1, and Windows Embedded Standard 8"[5]. The library itself can easily be installed with the aid of pipinstall. As written on the project's website the library “Enables writing Kinect applications, games, and experiences using Python” [3].

Alternatively to the *pykinect2* library, one could also use OpenNi and NITE as Tamara Berg et al. in [14]. “OpenNI (Open Natural Interaction) is a multi-language, cross-platform framework that defines APIs [Application Programmin Interface] for writing applications utilizing Natural Interaction” [15, Eco].It enables the communication between visual and aural sensors. It provides e.g. a representation of the full body calculated out of the sensor's input data. The NITE middleware provides the functionality of gesture and skeleton tracking [16]. OpenNi and NITE, however, need a lot of setup beforehand resulting in a lot of time-consuming work, which makes *pykinect2* preferable in terms of effectiveness. Furthermore, there is Libfreenect with OpenKinect [6] which is a “userspace driver for the Microsoft Kinect” and supports RGB, Depth Images, Motors, Accelerometer and Audio. In spite of all that, it does not support any skeleton tracking and is therefore also inapplicable for this system. The last alternative is to use the Microsoft Kinect SDK with C++, which also requires more setup and, in addition, involves a more complex programming language by using C++. As a result, the *pykinect2* library was estimated to be the best choice to implement this thesis's system.

The library provides certain functions which were used for the algorithm's implementation: The *PyKinectRuntime* class yields several methods, on the one hand, to get information about the status, and on the other hand, to get data. Former, for example, indicates if there is a new color frame or body frame detected by the Kinect, later returns the data to the color or body frames itself. Furthermore, there are methods e.g. *body_joint_to_color_space()* which can convert the joint coordinates to color space returning all skeletal joints. 26 joints are supported by the Kinect and also by the *pykinect2* library. The *PyKinectV2* file assigns numbers to joints, as a

[2]https://www.jetbrains.com/pycharm/, Accessed: 1.8.19
[3]https://github.com/Kinect/PyKinect2, Accessed: 1.8.19
[4]https://www.anaconda.com/distribution/, Accessed: 1.8.19
[5]https://www.microsoft.com/en-us/download/details.aspx?id=44561, Accessed: 1.8.19
[6]https://github.com/OpenKinect/libfreenect, Accessed: 1.8.19

result, the number 16, for example, resembles the right hip joint. Every joint provides its location as x- and y-values as well as its orientation.

The process of getting a x- or y-coordinate from a specific joint point is visualized in Figure 4.3. Firstly, a Kinect object has to be instantiated, secondly, the last body frame has to be requested with the *get_last_body_frame()* method, thirdly, the bodies of the requested frame have to be saved, which results in an array with all six available skeletons. The appropriate index for the one tracked skeleton has to be used to store the needed skeleton and then its joints can be extracted. These joints are given to the *body_joints_to_color_space()* method to get all 26 joint coordinates as an array. The last step is to choose the right index for the needed joint and decide if the x- or y-coordinate is needed.

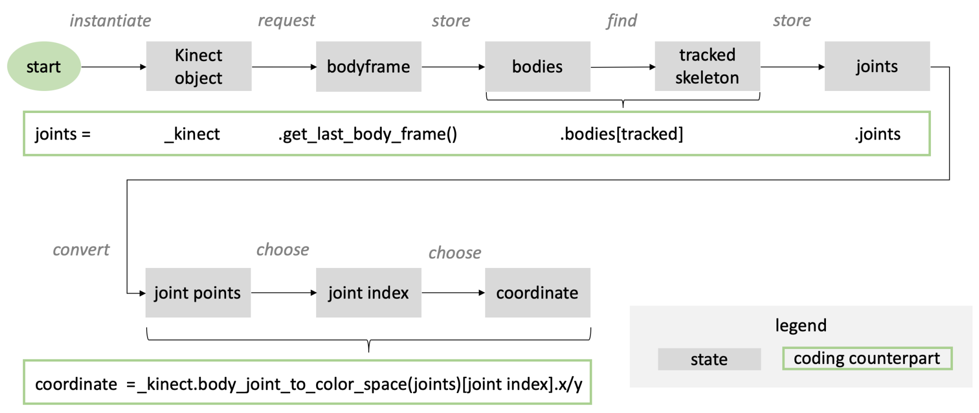

Figure 4.3.: Schematic representation of acquiring coordinates from joint points via pykinect2

Components of the Sound Output System

The sound output system is displayed in Figure 4.4 and Figure 4.5, differentiating between a simple and a more complex one applied by different versions of the algorithm. For version evolution, please refer to Section 4.3.1. The audio output of Version 1 and 2 uses the *winsound*library to output a beep noise displayed in Figure 4.4. The *winsound* library provides a function called *Beep()* which takes a certain duration and frequency as input and then outputs the defined sound through the computer's audio output device.

The sound output system from Version 3 ongoing, visualized in Figure 4.5, is more complex. Sound samples from Looperman [7] are stored in a local file on the computer. Using the SQLite a local database is created. The database consists of the bpm of the sound loops, which serves as primary key, and four paths stored as

[7] https://www.looperman.com, Accessed: 7.8.19

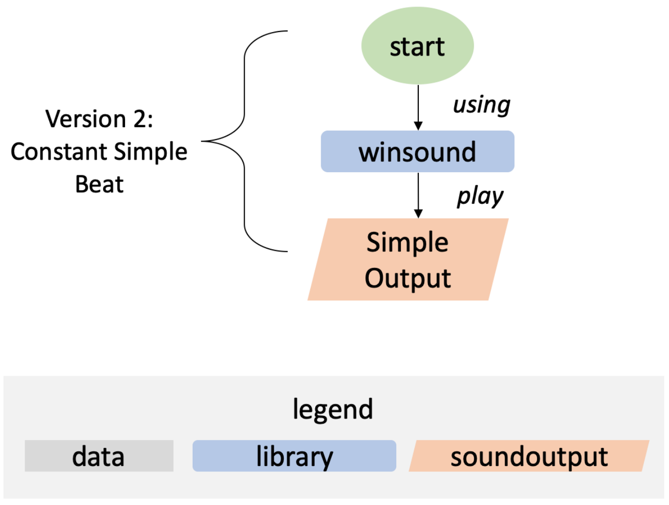

Figure 4.4.: Schematic representation of the simple sound output system

strings. These paths lead to a drum loop file in a Waveform Audio File Format (WAV) matched to the bpm as well as three guitar loops with the identical bpm. Working with the *pydub* library, the individual loops are altered and combined as needed and stored again. To combine two loops, they are first compared by their length. Afterwards, the shorter loop is overlaid by the longer one, which automatically cuts the longer loop to the length of the shorter and therefore results in a combined loop where drum and guitar sound are audible until the end. These combined loops are then played with the aid of the *playsound* library which provides a *playsound()* method. This method takes in a path and outputs the audio file, which is stored at that path, via the sound output device of the computer.

4.2. Structure of the Algorithm

In this chapter it will be elaborated how the underlying algorithm of the system works concentrating on the latest version of the algorithm: Version 6: Constant Drum Loop and Variable Guitar Loop. The next section, however, will discuss the differences to former versions.

The entire process can be divided into two phases as it can be seen in Figure 4.6: the calibration phase and the sound generation phase. The objective of the calibration phase is to calculate the mean value between two consecutive detected beats, whereas in the sound generation phase a harmonic musical output should be generated based on the mean time value provided by the calibration phase.

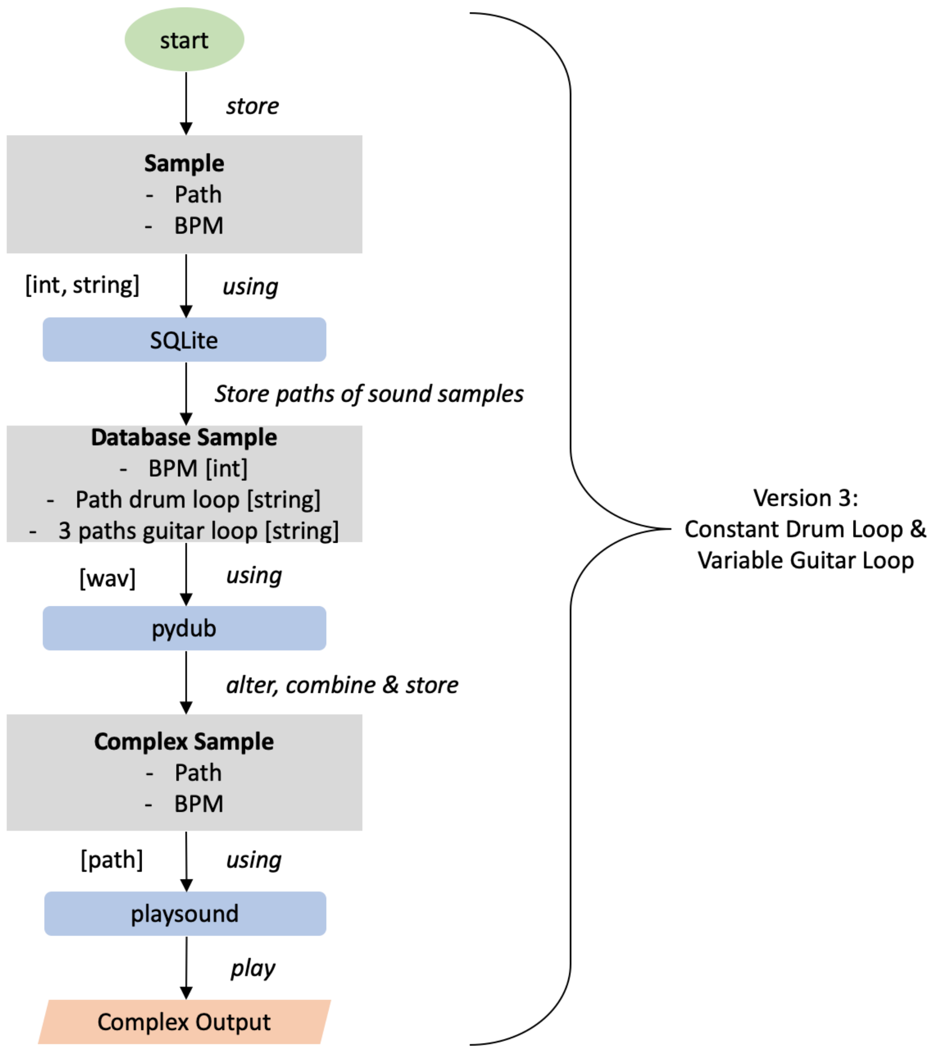

Figure 4.5.: Schematic representation of the complex sound output system

4.2.1. Calibration Phase Structure

The process of the calibration phase is displayed in the upper half of Figure 4.7. In order to detect a beat, the coordinate joints of the hip have to be stored in a usable way. The *find()* method of the *Videostream* class returns the needed joint point coordinate of the hip joint. These are then used as a input for the *calculations.velocity()* method which calculates the velocity of the hip joint in x-direction. Using this calculation, the *beat()* method, which is can be seen on the right hand sight, is able to

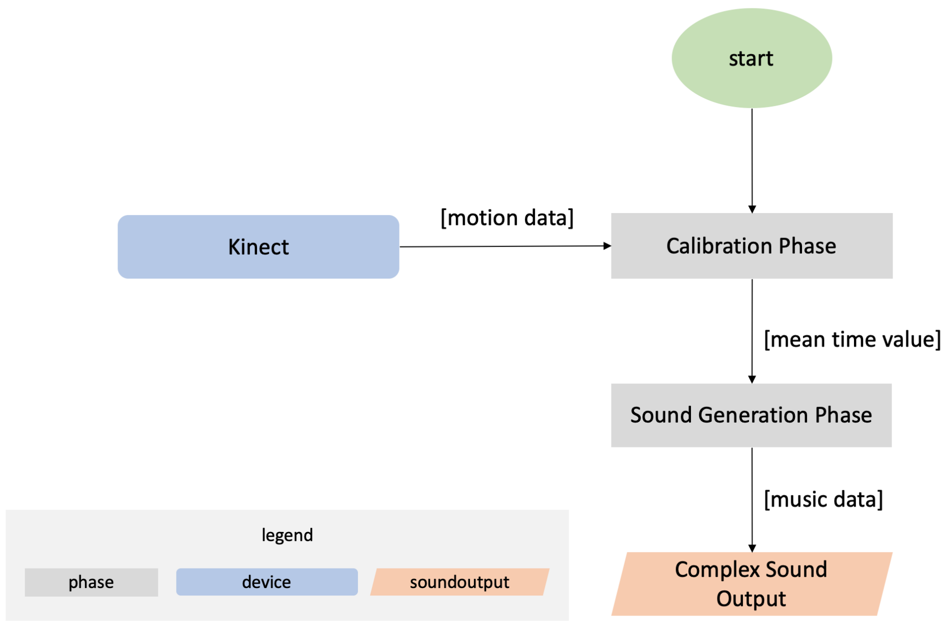

Figure 4.6.: Schematic representation of the algorithm's phases

detect if a beat occurred. Based on the output of this function, the main file outputs a simple beep sound using *winsound*.

Afterwards, as shown on the left hand side, the mean time value between two consecutive beats is calculated and returned by the *ConstDrumBeat* class method *calc_mean_time()*, which results in the end of the calibration phase.

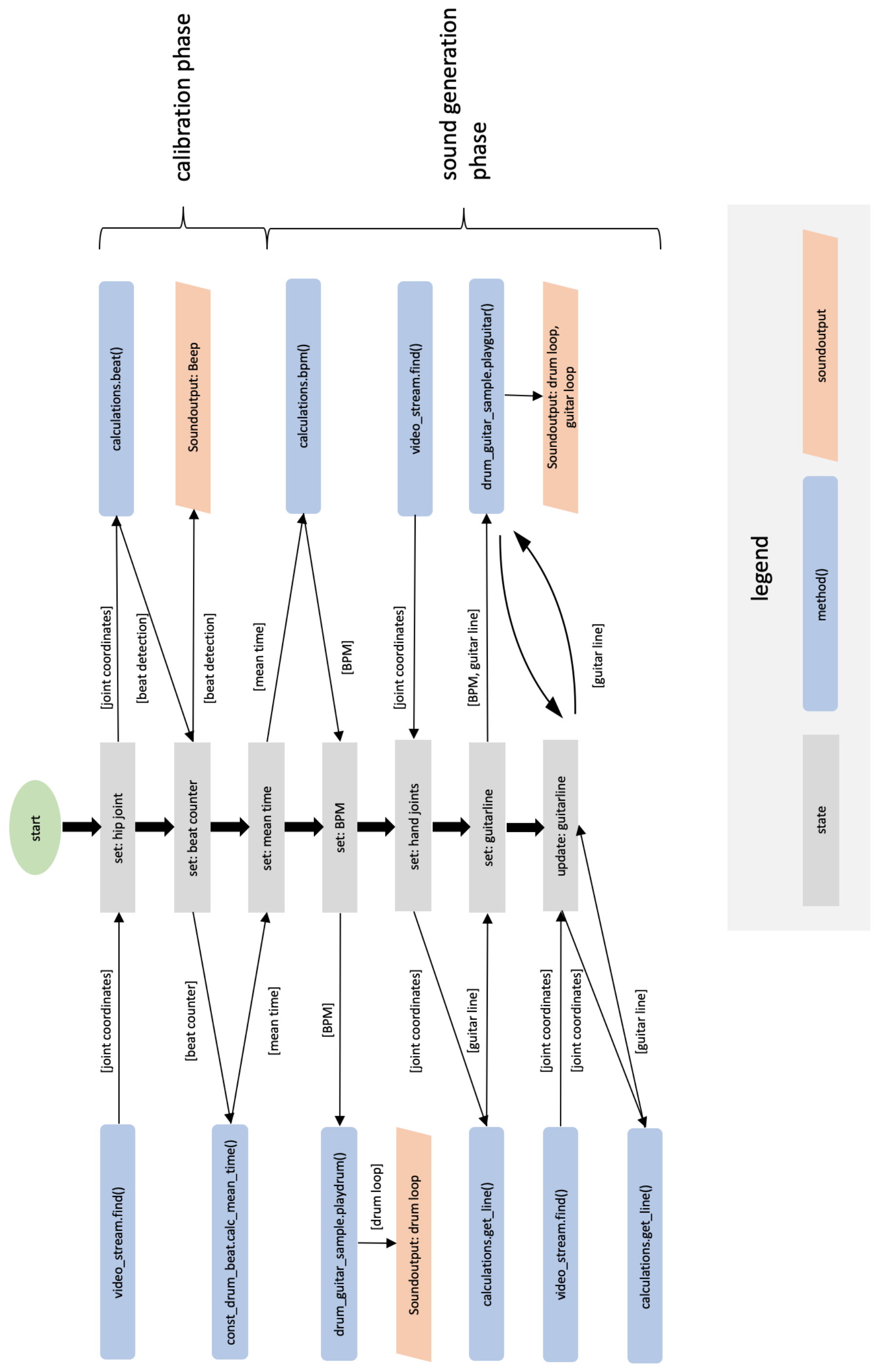

Figure 4.7.: Flow chart representation of the algorithm

4.2.2. Sound Generation Phase Structure

The first step in the sound generation phase is to calculate the bpm out of the mean time value between two consecutive beats as it can be seen in the lower half of Figure 4.7, presenting the sound generation phase. This can then be used to choose the appropriate sound samples which are best fit to the movement speed of the user. The *bpm()* method always returns a bpm value to which an appropriate drum and guitar loop can be found in the database.

As displayed on the left hand side of Figure 4.7 this *bpm* is given as an input variable to the *playdrum()* method of the *DrumGuitarSample* class, which then takes the appropriate path string from the database and plays the drum loop via *playsound*.

The system is able to output three different guitar loops to every chosen drum loop. In order to choose the right guitar loop to combine with the existing drum loop, the hand position of the right hand has to be tracked. Therefore, like in the calibration phase the method *find()* of class *VideoStream* provides the needed joint coordinates, in this case the right hand middle joint. These coordinates are fed into the *get_line()* method belonging to the calculations file which is a simple mapping function. It maps the three guitar lines to upper, middle and lower area of the Kinect's tracking range and returns the guitar line variable.

The guitar line variable is used together with the bpm value to add in a guitar loop. The process is illustrated in the lower right corner of Figure 4.7: The *playguitar()* method of the *DrumGuitarSample* class takes both values as input and outputs with the aid of the sample database the appropriate combination of drum and guitar loop.

After the loop is played, the main file checks again for the hand position using the same cycle as before: getting joint coordinates from *videostream.find()* and updating the suited guitar line from *caclulations.get_line()*, which is then used to play a different drum guitar combination with *playguitar()* if necessary. This loop process is also visualized on the bottom of Figure 4.7. For a extended version on how the algorithm works, please refer to Section A.1 in the appendix.

4.3. Approach

4.3.1. Version Evolution Outline

Figure 4.8 visualizes the different versions through which the algorithm developed.

The first version, an unsteady simple pulse, supports only a basic beat as the only sound played is a beep noise with a defined frequency and duration. The beep sound always occurs when a beat is detected and gives immediate aural feedback to the user.

The second version, a steady simple pulse, introduces the two-phase system. In the calibration phase the time between two consecutive detected beats is measured

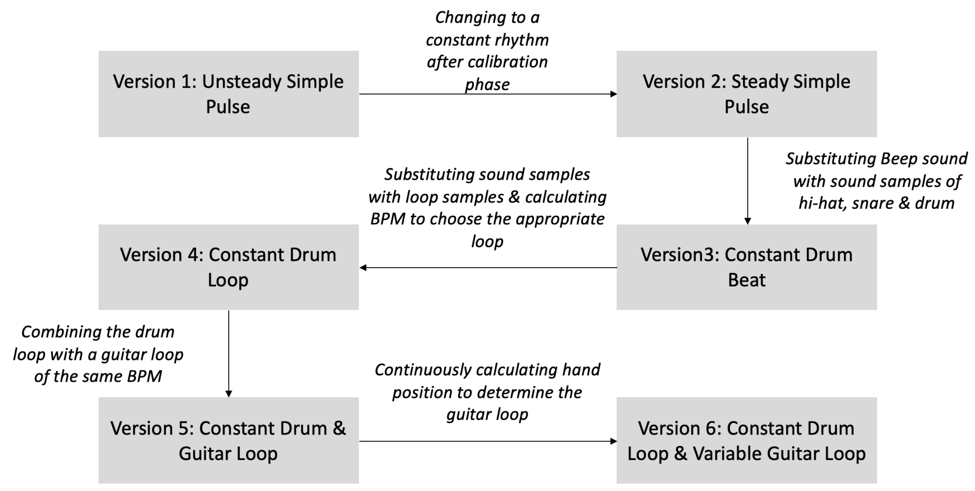
Version 1: Unsteady Simple Pulse
Changing to a constant rhythm after calibration phase
Version 2: Steady Simple Pulse
Substituting Beep sound with sound samples of hi-hat, snare & drum
Version3: Constant Drum Beat
Substituting sound samples with loop samples & calculating BPM to choose the appropriate loop
Version 4: Constant Drum Loop
Combining the drum loop with a guitar loop of the same BPM
Version 5: Constant Drum & Guitar Loop
Continuously calculating hand position to determine the guitar loop
Version 6: Constant Drum Loop & Variable Guitar Loop

Figure 4.8.: Schematic representation of the algorithm's version evolution

and a mean value is calculated representing the mean time between two consecutive beats in the user's dance style. The mean time is used to generate a steady pulse to form a beat usually as explained in Section 3.1. In Version 2 the steady pulse is yet only a beep sound with defined frequency and duration.

Version 3 implements a constant drum beat. To change from a steady pulse to an actual drum beat, the beep sound is exchanged with sound samples of hi-hat, snare and drum. These single WAV files are ordered and combined to build a simple drum beat.

In Version 4, drum loops instead of single note samples are used to create a drum beat. After calibration phase, the mean time value is transferred to a bpm value which indicates the tempo of a music piece, see Section 3.1. With SQLite a database is generated, storing a different drum loop WAV file for various bpm values. After the bpm value is calculated, the appropriate drum loop is chosen and played continuously. As a result, the speed of the user's movement influences the tempo of the music piece because the beat detected in the motion data matches the rhythm in the chosen drum loop as they have identical bpm.

Version 5 adds a guitar loop to the drum loop of Version 4. Therefore, the database also stores a suited guitar loop for every stored bpm value.

To reach the last algorithm version, Version 6 (Constant Drum Loop and Variable Guitar Loop), three different guitar loops are stored in the database for every saved bpm value. The choice of the guitar loop is based again on the user's movement data to give the user more influence on the music.

For more details on how the algorithm evolved, please refer to Section A.2 in the appendix.

4.3.2. Motion Tracking Methods

In order to find the best working motion tracking method for beat detection, different approaches have been tested. The methods can be classified by the body joints which were used to build the model. Three different body joints have been considered as it can be seen in Figure 4.9: Foot joint, hip joint and head joint.

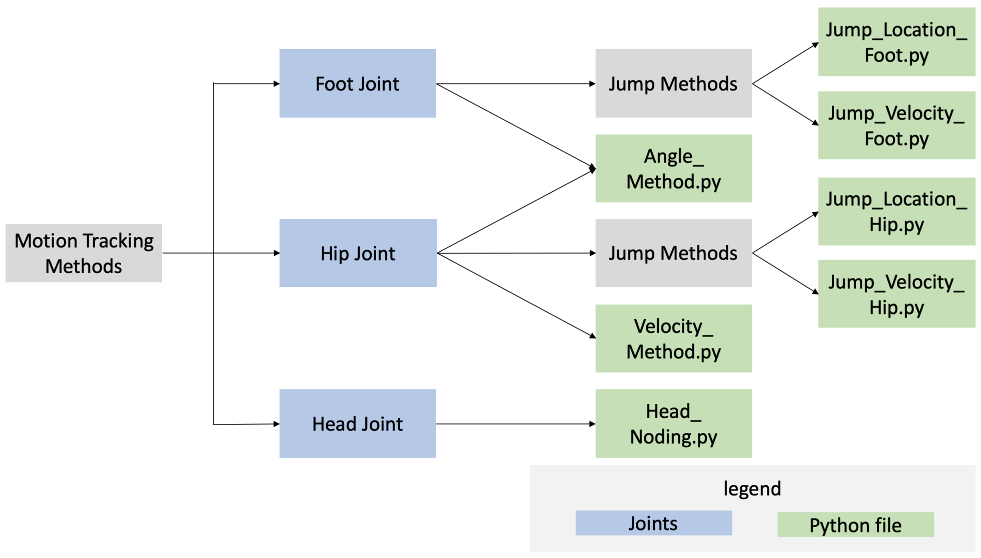

Figure 4.9.: Block diagram of the different motion tracking methods

The method used for the proposed algorithm is the velocity method using the hip joint. The idea of the approach is to detect a beat when the hip is at the most outward location as this resembles how people dance when moving the hips to a rhythm. To implement this, the velocity of the hip joint is tracked and searched for a change in sign which indicates that the hip reached his most outward position. For further information please refer to Section 4.2. This approach evaluated to work the best as it, on the one hand, detected almost every expected beat and, on the other hand, also worked for faster movements.

Another method evolved as some people dance to a beat through jumping every time the pulse occurs. In order to track this “jumpstyle” dancing, several approaches were implemented. On the one hand, the hip joint was used together with the same approach as of the velocity method except for tracking the y-coordinate value instead of the x-coordinate value (Jump_Velocity_Hip.py). In addition, the beat was only detected for one of the sign changes resulting in the fact that the beat is only identified when getting back on the ground or when reaching the maximum of the jump in air depending on the chosen sign change. To improve this approach the if-clause, checking for velocities near zero and excluding beat detection in that case, was deleted as jumping from the floor always starts with a velocity near zero.

A similar approach implemented the same idea, however, using a mean value of the two feet joints instead of the hip joint (Jump_Velocity_Foot.py). This should avoid the problem that the hip can move below its initial value because knees are getting bent when absorbing the jump's shock.

These two methods, however, did not catch all intended beats and furthermore, detected some when nobody was jumping. Because of that unreliable output, it was not conducted further.

On the other hand, another method, concerning the location instead of the velocity, was implemented once using the hip joint (Jump_Location_Hip.py) and once the mean value of the two feet (Jump_Location_Foot.py). For that purpose, a standard value was set in the beginning, indicating where hip or feet are positioned in the initial state. A beat was said to be detected when the difference between current value and standard value, called boundary value, got big enough, reached a maximum and was thus diminishing again. However, the user needed to jump very high to trigger a detected beat. Another problem that occurred with this approach was the dependency on the distance from user to the Kinect as a certain boundary value worked best with only a constant distance from the Kinect. Because of these drawbacks and the ones mentioned above for the method geared to the velocity method, the jump tracking approaches all together got neglected in the further development of the algorithm.

Considering that some people dance only by nodding their head, one approach tried to track the head joint using the velocity method on the y-coordinate (Head_Noding.py). The results, nonetheless, were estimated as poorly, which might be attributed to the fact that the head joint is only one joint, not moving extensive enough to detect a change in velocity accurately.

An approach using one of the hip joints as well as both feet joints is the angle method (Angle_Method.py). It is based on the angle between two vectors, one reaching from the right foot to the left foot and the other from the left foot to the left hip. The needed vectors were calculated through subtraction of the joint vectors which were saved in a matrix ring buffer beforehand. Afterwards, the vectors were normed using the linear algebra functions library from *numpy*. Applying the arccos function, the angle between the two vectors was calculated. In the next step, the change of angle was calculated and also stored in a ring buffer similar to the velocity calculation in the velocity method. Furthermore, the current value was compared to the recent one looking for a change in sign. This change in sign was triggering the beat output sound as it indicated that the biggest deflection of the hip joint had been reached. However, the method was also not detecting all intended beats and was outperformed by the velocity method in terms of accuracy.

4.4. Problems and Solutions

4.4.1. Calibration Phase

The first problem faced in the calibration phase was due to the Kinect's ability to track up to six skeletons parallel. Therefore, when the Kinect observed more than one skeleton, a constant beep sound occurred indicating a permanent beat detection. In order to solve this problem, the *find()* method of the *Videostream* class got adapted to only use the first tracked skeleton for any further calculations, all other tracked skeletons get neglected. As a result, another person stepping in the Kinect's range does not disturb the process. When the Kinect loses track of the observed skeleton, it outputs a "No body found" string and waits for 0.2 seconds before searching again for this skeleton. Another consequence of the problem's solution is that only the same person which was observed when starting the algorithm will be identified as the tracked skeleton when stepping back into the Kinect's range. Therefore, to change the person being tracked by the algorithm, the program has to be rerun.

Another problem occurring at the calibration phase was that beats got detected when moving only rarely or not at all. This was due to the algorithm motion tracking method to detect the velocity sign change which occurs almost permanently when standing still as every person always moves slightly or shifts weight. To solve that problem, another if-clause was added checking if the change in velocity was greater than a certain threshold. If the difference between the velocity before and after sign change is sufficient, a beat gets detected otherwise it gets dismissed.

Furthermore, it should be ensured, that the time interval *t* after which a new skeleton frame is requested from the *Videostream* class is constant. As calculations take different amount of time, depending on the fact whether a beat was detected, the sleep time of the algorithm has to be adapted. To find a solution, first, the calculation time was observed reaching from almost 0 seconds when no beat was detected to around 0.3 seconds for a beat detection and output. In the next step, a sleep-time function was created belonging to the *Calculation* class. It takes two time-stamps as input, one before requesting the next skeleton frame and one after beat calculation and beat output in case of detection. Taking the predefined time interval *t* invoked with the implementation of the *Calculation* class, the method outputs the sleep-time needed by subtracting the time difference from the time interval *t*. By this means, it is guaranteed to have a constant time interval *t* between two successive used time frames. However, in case of a chosen time interval which is shorter than the time needed for calculations, a constant time interval cannot be assured.

4.4.2. Sound Generation Phase

The first problem occurred in the sound generation phase between Version 2, Steady Simple Pulse, and Version 3, Constant Drum Beat. For the description of the different versions please refer to Section4.3.1. In the beginning the idea for Version 3 was to exchange the Beep noise of the unsteady beat with WAV files of drum samples every time a beat gets detected. This idea, however, stated two problems: On the one hand, the length of the WAV files exceeded the duration of the beep noise which was set to 300 milliseconds causing problems when beats got detected fast after each other, on the other hand, an unsteady drum pattern which was the sound output result of the idea did not match the systems objective of a harmonious sounding output as a unsteady drum pattern is not a enjoyable aural experience. To improve the former problem, the WAV files were shortened to the best possible extend. As a result, faster beat rhythms could be established, yet the WAV files still constituted an upper limit to the speed of the beat rhythm, which was eventually solved through the use of complete drum loops in Version 4.

To solve the later problem of the unsteady drum pattern, Version 3 evolved, dividing the process in a two stage system: The calibration phase with a unsteady beep noise pulse and the sound generation phase which used a steady drum pattern without changes in velocity, based on the mean time of the unsteady beep noise pulses beforehand. The result was a harmonious sounding drum rhythm after calibration phase which therefore met the objective's requirements.

The second problem developed when trying to implement a guitar line suiting the drum line. Research yielded the information that sound samples of guitar lines consisted rather of complete loops instead of single note WAV files as used so far. However, using complete guitar loops faced following problems: on the one hand, a guitar loop had to be found which matched the outputted steady drum rhythm, stating the question, what kind of matching criteria were appropriate. On the other hand, the already existing drum rhythm consisted out of single WAV files so far. As the one audio output device could not play two sounds in parallel, the drum rhythm's WAV files and the guitar loop WAV files respectively, the drum rhythms would first have to be transferred to a single WAV file and subsequently combined with the guitar loop file. As a result, the algorithm would have to create WAV files by itself.

The alternative was to use single note or chord sound samples for the guitar line as well. Problems faced by this alternative were: Firstly, for building up a new guitar line, extensive musical expertise and guitar knowledge would have been needed. Secondly, the single note or chord sound samples could not create a fluently sounding guitar line. This was caused through the fact that a string is usually still vibrating and creating audio waves when the next string is played, but with single note files the sound "played" before has no influence on the next WAV file which then creates a choppy sound. Thirdly, the various single note WAV files would have to be combined beforehand, therefore, stating a calculation and programming effort which would increase exponentially when another line e.g. a bass line would be added.

Considering advantages and disadvantages of both alternative, the first alternative, using loops, was further pursued. However, instead of creating a new WAV file out of the drum rhythm, the single WAV files were exchange with a complete drum loop as well based on the mean time of the calibration phase. This yielded Version 4, a constant drum loop. For further details please refer to Section 4.3.1.

To match the drum and the guitar loop in an appropriate manner, they have to have the same bpm, for technical music term definitions see Section 3.1. Therefore, the bpm states the solution as a matching criterion to create a harmonious sounding output. In order to output the drum and the guitar line on a single audio output device, the two WAV files have to be combined to a single WAV file beforehand. However, the calculation effort was decreased tremendously as only two files had to be overlaid. This approach resulted in Version 5, a constant drum and guitar loop.

Another problem occurred because of the used *pydub* library. To combine two WAV files, they are overlaid and afterwards exported. When exporting a file with the same name twice in one program run, windows throws an error, because the file gets opened twice without being closed in between. The first implementation idea to always use the same combined file and store the right combination of drum and guitar loop in this file before playing it via *playsound*, did throw the described error. To avoid this error, the three combinations of the drum loop with the three different guitar loops are generated and stored in the first iteration of playing and thenceforward not altered anymore but merely chosen depending on the hand position. For further explanations please refer to Section 4.2.

The last problem faced in the sound generation phase concerned the crossover between the single loops. As a loop sample only consists of several seconds, it eventually has to be looped. When using a perfect music loop, the end should match the beginning perfectly. However, the used loops sometimes do not merge perfectly, so there occurs a hear-able disruption when looping. To improve that aural experience, one loop is appended at itself with a cross fade of 30 milliseconds. This ensures a smoother transition and is yet a short enough cross fade period to avoid suppressing the first beat in the next loop which usually is the most accentuated and therefore crucial for the aural perception.

5. Evaluation

This chapter discusses the results of the proposed system taking into account three predefined evaluation criteria. Firstly, the real time implementation, secondly, the influence the user has on the system, and thirdly, the harmonious sounding output criterion.

The system has a real time requirement as the motion data of the user should be transformed to music output as fast as possible. This requirement ensures that the user can more easily understand his influence on the aural output which states the second evaluation criterion. The algorithm is coded with the help of python which uses an interpreter instead of a compiler and is therefore limited in speed. However, it is still sufficiently fast for this application. When calculating if a beat was detected, maximum calculation time adds up to 0.3 seconds. A time interval of 0.3 seconds between two consecutive frames was tested to be sufficient to track beats accurately. Choosing a even smaller time interval of less than 0.3 seconds did not result in more exactness concerning beat detection. As a result, calculation is done fast enough to ensure real time application given the findings of Section 3.2.

The beep noise occurring in the calibration phase, for further details please refer to Section 4.1.2 and Section 4.2, serves as almost immediate aural feedback for the user and also supports the real time criterion because the beat does not have to occur exactly on point as stated in Section 3.2. After calibration phase, the bpm is calculated and the appropriate drum loop is outputted without any concerning latency disturbing the user's experience which again satisfies the real time approach. Similar to the drum loop, the guitar loop is added without hear-able latencies. The drum and guitar loop are humanoid products and therefore, contain already some rhythmic errors which makes them sound more enjoyable, for further information see Section 3.2.

However, concerning the real time influence of the user on the guitar line through his right-hand position some concessions had to be made. The hand position is not checked permanently but only after one guitar line loop has ended as it would not sound harmonious changing mid-line. Therefore, there is a tension between the real time criterion and the harmonious sounding output criterion where an appropriate balance has to be found. As a solution, after the entire loop has been played the next guitar line is chosen and outputted in real time according to the hand position. For the user's experience, however, this might lead to the observation that he has no immediate influence on the guitar line by moving his hand as it only influences the guitar line at certain points in time. As a conclusion, one can say that the real time criterion has been achieved, however, in the later part of the algorithm could be improved.

The second criterion deals with the influence the user has on the actual output and evaluates to which extend he can be aware of his influence. In the acclimatization phase which is the beginning of the calibration phase, defined in length through the variable *acc*, the user can generate a beep noise through is hip movement. The acclimatization time is needed to give the user the possibility to understand his influence on the beat generation which he can observe by the immediate aural feedback. Through his speed of motion, he can actively determine the bpm for the drum loop and therefore the tempo of the complete sound output. However, once the tempo is set, it stays steady as changing the tempo concurrent to the user's movement speed would interfere with the harmonious sounding output criterion. When the rhythm is set, the user can still actively influence the melody by moving his right hand up and down. However, there is no immediate aural feedback because the system checks the hand position only in certain time intervals as described above.

Therefore, the user indeed has influence on the musical output, even real time influence on a certain extent. His awareness of his influence is determined through his own observations and realizations throughout the acclimatization time and by the fact if he understands how to generate a detected beat. The user could also be informed how the algorithm works to enable a better understanding of his influence. This would decrease the time it takes for the user to figure out that his hip movement is mapped to the beat detection and later on that his hand position determines the guitar line. This would lead to a more natural usage of the system instead of a try and error approach of the user. The system is missing a visual feedback implementation which could further enhance the usability. In spite of the beneficial contribution of a visual feedback, this implementation was out of the scope of this thesis objective. Concluding one can say that the influence criterion is achieved with this system, however, it could be further enhanced through a more real time approach in the sound output phase or through a visual feedback system.

The last criterion evaluates the systems goal to output a harmonious sounding musical output which therefore can inspire the user positively in his movement. Thus, a creativity inspiring circle is generated where computer system and human enhance each other. The sound output of the calibration phase is just a simple beep noise of a certain duration and frequency which by itself cannot be described as music. However, the calibration phase has no claim to inspire by itself but to set the basics for the sound output phase which therefore is the creativity inspiring phase. The drum loop generated in the first step of the sound output phase is combined of different percussion instrument parts like hi-hat or cymbal and thus sounds not as monotonous as the beep noise. As a loop is already a music piece itself as stated in Section 3.1, the criterion is already achieved to a certain extend. Furthermore, as there are drum loops for different tempos the likelihood of originating the same drum loop twice in different program runs is rather small and results in a new aural experience for the user every time the system is rerun. To counteract the monotony which develops as the loop is continuously replayed, the guitar line is added. As there are also three guitar lines for every bpm stored in the database, it supports the concept to create a unique aural experience every time the system is rerun. The guitar lines are matched according to their bpm which also results in a harmonious sounding output and generates a music piece with a guitar melody underlined by percussion sounds. To prevent the musical output of a single system's run to get to repetitious as guitar lines are also looped, the output switches between the three lines depending on the user's input and therefore creates three completely different musical pieces. In summary, the harmonious sounding output criterion is accomplished in the output phase through a variety of drum loops and thrice as many guitar loops matched appropriately.

In Table 5.1 is presented to which extend the three criteria are realized by the different algorithm versions explained in Section 4.3.1. The amount to which the criterion was achieved is estimated by a number reaching from one to five with five stating that the criterion was fully accomplished and one resembling that the criterion was fulfilled to a unsatisfying extend.

Table 5.1.: Accomplishment of the Evaluation Criteria in the Version Evolution

	Real Time	**User's Influence**	**Harmonic Output**
Version 1	5	5	1
Version 2	4	3	2
Version 3	4	3	3
Version 4	4	3	4
Version 5	4	3	5
Version 6	5	5	5

As one can see, the real time criterion and the user's influence were already achieved in Version 1. However, as there exists tension between this two criteria and the harmonic output criterion, they declined again as the harmonic output criterion got improved with every version evolution up to Version 5. Despite the worsening in Version 2 to 5, Version 6 achieves to balance out the three criteria and hence all three criteria of the system's objective have been achieved to as satisfying extend.

Acronyms

bpm	Beats per Minute
IR	infrared
Max/MSP	Max/Max Signal Processing Software
MIDI	Musical Instrument Digital Interface
OpenNi	Open Natural Interaction
OSC	Open Sound Control
RGB	Red-Green-Blue Color Space
SDK	Software Developement Kit
WAV	Waveform Audio File Format

Bibliography

[1] A. Pennycook, “Actions speak louder than words: Paralanguage, communication, and education,” *TESOL Quarterly*, vol. 19, 06 1985.

[2] K. Jones and K. Barker, *Human Movement Explained*, ser. Physiotherapy practice explained. Butterworth-Heinemann, 1996. [Online]. Available: https://books.google.de/books?id=TGoPvgAACAAJ

[3] A. Elgammal, B. Liu, M. Elhoseiny, and M. Mazzone, “Can: Creative adversarial networks, generating "art" by learning about styles and deviating from style norms,” in *Inproceedings of the eighth International Conference on Computational Creativity (ICCC)*, 06 2017.

[4] K. D. Skeldon, L. M. Reid, V. McInally, B. Dougan, and C. Fulton, “Physics of the theremin,” *American Journal of Physics - AMER J PHYS*, vol. 66, pp. 945–955, 11 1998.

[5] K. Bijsterveld and J. v. Dijck, *Sound Souvenirs - Audio Technologies, Memory and Cultural Practices*, 01st ed. Amsterdam: Amsterdam University Press, 2009.

[6] Y. Wu, P. Kuvinichkul, P. Y.K. Cheung, and Y. Demiris, “Towards anthropomorphic robot thereminist,” in *2010 IEEE International Conference on Robotics and Biomimetics, ROBIO 2010*, 01 2011, pp. 235 – 240.

[7] M. Hsu, W. G. C. W. Kumara, T. K. Shih, and Z. Cheng, “Spider king: Virtual musical instruments based on microsoft kinect,” in *2013 International Joint Conference on Awareness Science and Technology Ubi-Media Computing (iCAST 2013 UMEDIA 2013)*, Nov 2013, pp. 707–713.

[8] E. Togootogtokh, T. K. Shih, W. G. C. W. Kumara, S.-J. Wu, S.-W. Sun, and H.-H. Chang, “3d finger tracking and recognition image processing for real-time music playing with depth sensors,” *Multimedia Tools and Applications*, vol. 77, pp. 9233–9248, 05 2017.

[9] L. T. D. Paolis and P. Bourdot, *Augmented Reality, Virtual Reality, and Computer Graphics - 5th International Conference, AVR 2018, Otranto, Italy, June 24–27, 2018, Proceedings*, 1st ed. Berlin, Heidelberg: Springer, 2018.

[10] S. Li, K. Xu, and H. Zhang, “Research on virtual guzheng based on kinect,” in *AIP Conference Proceedings*, vol. 1967, 05 2018.

[11] A. Zlatintsi, P. P. Filntisis, C. Garoufis, A. Tsiami, K. Kritsis, M. Kaliakatsos-Papakostas, A. Gkiokas, V. Katsouros, and P. Maragos, “A web-based real-time kinect application for gestural interaction with virtual musical instruments,” in *Sound in Immersion and Emotion*, 09 2018, pp. 1–6.

[12] S. U. Aswar, A. B. Deshmukh, and S. B. Rathod, “Creating musical instrument using kinect sensor,” in *Proceedings of WRFER-IEEEFORUM International Conference*, 05 2017, pp. 50–54.

[13] J. Jean, *Kinect Hacks: Tips & Tools for Motion and Pattern Detection*, ser. Hacks series. O’Reilly Media, 2012. [Online]. Available: https://www.oreilly.com/library/view/kinect-hacks/9781449332181/ch04.html

[14] T. Berg, D. Chattopadhyay, M. Schedel, and T. Vallier, “Interactive music: Human motion initiated music generation using skeletal tracking by kinect,” in *Conference: Society for Electro-Acoustic Music in the United States (SEAMUS)*, 02 2012.

[15] *OpenNI User Guide*, OpenNI organization, July 2011. [Online]. Available: https://github.com/OpenNI/OpenNI/blob/master/Documentation/OpenNI_UserGuide.pdf

[16] *Prime SensorTM NITE 1.3 Framework Programmer’s Guide*, PrimeSense Inc., 2010. [Online]. Available: https://andrebaltazar.files.wordpress.com/2011/02/nite-controls-1-3-programmers-guide.pdf

[17] M.-J. Yoo, J.-W. Beak, and I.-K. Lee, “Creating musical expression using kinect,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 05 2011, pp. 324–325.

[18] L. Naugle, F. Bevilacqua, and I. Valverde, “Virtual dance and music environment using motion capture,” in *In Proceeding of IEEE Multimedia Technology And Applications Conference*, 2001.

[19] C. Schmidt-Jones and R. Jones, *Understanding Basic Music Theory*. OpenStax CNX, 2013. [Online]. Available: https://cnx.org/contents/KtdLe6cv@3.74:_GmJ4ENa@7/Understanding-Basic-Music-Theory-Course-Introduction

[20] S. Brinson, “3 little facts about rhythm,” 11 2014, accessed: 12.8.19. [Online]. Available: https://medium.com/world-of-music/3-surprising-facts-about-rhythm-49a9710ef38a

[21] T. Beck, “When the beat goes off: Errors in rhythm follow pattern, physicists find,” 7 2012, accessed: 12.8.19. [Online]. Available: https://phys.org/news/2012-07-errors-rhythm-pattern-physicists.html

[22] B. H. Repp, “The art of inaccuracy: Why pianists errors are difficult to hear,” *Music Perception: An Interdisciplinary Journal*, vol. 14, no. 2, pp. 161–183, 1996. [Online]. Available: https://mp.ucpress.edu/content/14/2/161

[23] E. Lachat, H. Macher, T. Landes, and P. Grussenmeyer, “Assessment and calibration of a rgb-d camera (kinect v2 sensor) towards a potential use for close-range 3d modeling,” *Remote Sensing*, vol. 7, pp. 13 070–13 097, 10 2015.

[24] O. Wasenmueller and D. Stricker, “Comparison of kinect v1 and v2 depth images in terms of accuracy and precision,” in *Conference: Asian Conference on Computer Vision Workshop (ACCV workshop)*, 11 2016.

A. Appendix

A.1. Extended Algorithm Explanation

A.1.1. Calibration Phase Structure

The process of the calibration phase is displayed in the upper half of Figure 4.7. In order to detect a beat, the coordinate joints of the hip have to be stored in a usable way. To do so, the class *Videostream* is instantiated. It's method *find()* takes as input a *joint_index*, in this case 16 for the right hip joint, and a coordinate, x- or y-coordinate or more specific a Boolean *True* or *False* respectively, and returns the needed joint point coordinate.

At the first execution time, the method also has to iterate through the six possibly tracked skeletons and determine the one, which is currently tracked and thereupon set the value of the *tracked* variable. This ensures that the system cannot be disturbed by another person stepping the tracking range of the Kinect. The *Videostream* class itself is initialized with a source *src* which indicates if the Kinect should be used for motion data capturing or a mock-up version instead.

The mock-up version was implemented to enable working with the algorithm without the Kinect. It consists of a pre-recorded dance sequence, where the hip joint x-coordinate is saved every 0.1 seconds in an array which in case of need is deserialized with the *pickle* module. In order to provide the coordinates of the appropriate frame in time, the input variable *t*, indicating the time difference between two consecutive requested frames, is divided by 0.1. This yields the amount by which the mockup-array's index has to be incremented. By this means, the *Videostream* class can always provide a coordinate independent of the Kinect as a data capturing source.

The captured joint coordinates are then saved in a ring buffer and the *Calculation* class produces a mean value out of three consecutive x-coordinate values to ensure a smoother output and avoid errors through jitter. Furthermore, the *calculations.velocity()* method calculates the velocity of the hip joint in x-direction. Using these calculations the *beat()* method is then able to detect if a beat occurred. To identify a beat, the velocity ring buffer is searched for a change of sign. If one is observed, it is additionally checked if the difference between the consecutive velocity values is higher than a certain predefined threshold value to sort out beats produced through trembling or very small movements. Furthermore, the first movement starting from a velocity equaling null is also excluded. In conclusion, a beat should be detected at the peak when the hip is changing direction. The beat-detecting method returns a Boolean value to the main file which, based on that value, outputs a simple beep sound using *winsound* as it can be seen in Figure 4.7.

The mean time value between two consecutive beats is calculated by the *ConstDrumBeat* class method *calc_mean_time()*. After enough beats are passed, indicated by the acclimatization variable *acc*, the method stores the time at every detected beat. Using this stored time stamps, it calculates the mean value between five consecutive beats and returns it, which results in the end of the calibration phase.

A.1.2. Sound Generation Phase Structure

The first step in the sound generation phase is to calculate the bpm out of the mean time value between two consecutive beats as it can be seen in the lower half of Figure 4.7, presenting the sound generation phase. This can then be used to choose the appropriate sound samples which are best fit to the movement speed of the user.

The *Calculation* class has a method called *bpm()*, which takes the mean time value as an input and returns the bpm value by dividing 60 through the mean time value. However, before returning the bpm value, it verifies if the calculated value exists in the database by comparing the value to a bpm array provided by the *get_bpm_array()* function of the *sample_database* file. If the value exists in the database, the bpm value is simply returned. If it does not exist, the method evokes another method called *find_nearest()* which takes the bpm value and array and returns the nearest bpm value to the calculated one which exists in the array and therefore in the database. As a result, the *bpm()* method always returns a bpm value to which an appropriate drum and guitar loop can be found in the database.

As displayed on the left hand side of Figure 4.7 this *bpm* is given as an input variable to the *playdrum()* method of the *DrumGuitarSample* class, which then takes the appropriate path string from the database and plays the drum loop via *playsound*.

The system is able to output three different guitar loops to every chosen drum loop. In order to choose the right guitar loop to combine with the existing drum loop, the hand position of the right hand has to be tracked. Therefore, like in the calibration phase the *VideoStream* class method *find()* provides the needed joint coordinates, in this case number 11, which belongs to the right hand middle joint. These coordinates are fed into the *get_line()* method belonging to the calculations file which is a simple mapping function. It divides the Kinect tracking range vertically in three spaces, resulting in a lower, a middle and an upper area. Depending on where the right-hand joint is located, a number is assigned to the guitar line variable, the upper area resulting in a one, the middle in a two and the lower in a three. As a result, the guitar line can be chosen by the movement of the right hand.

The guitar line variable is used together with the bpm value to add in a guitar loop. The process is illustrated in the lower right corner of Figure 4.7: The *playguitar()*

method of the *DrumGuitarSample* class takes both values as input. Furthermore, it takes the appropriate path strings from the sample database and stores them. Using the *overlay()* function of the *pydub* library, it combines both loops checking beforehand which one is the longer one to cut it to the right length.

As the other drum and guitar loop combinations might also be needed later on, all three possible combinations are already created and stored separately at the beginning of the method. Additionally, a flag variable called *have_to_calculate* is set to *False* afterwards to prevent calculating the combined files again. This avoids an error which is thrown by windows when opening up a file again which is already opened. This happens because the *pydub* library does not close the files after exporting the WAV data. In the end, however, only the drum guitar loop combination corresponding to the tracked guitar line is outputted by the *playguitar()* method.

After the loop is played, the main file checks again for the hand position using the same cycle as before: getting joint coordinates from *videostream.find()* and obtaining the suited guitar line from *caclulations.get_line()*, which is then used to play a different drum guitar combination with *playguitar()* if necessary. This loop process is also visualized on the bottom of Figure 4.7.

A.2. Extended Version Evolution Description

The first version, an unsteady simple pulse, supports only a basic beat as the only sound played is a beep noise with a defined frequency and duration. The beep sound always occurs when a beat is detected and gives immediate aural feedback to the user. The algorithm consists of only one phase and does not differ between calibration and sound generation phase.

The second version, a steady simple pulse, however, introduces the two-phase system. As a beat usually consists of a steady pulse as explained in Section 3.1, the system has to establish a constant pulse. To base the time intervals of the steady pulse on the input motion data from the user, there is a calibration phase needed. In the calibration phase the time between two consecutive detected beats is measured and a mean value is calculated representing the mean time between two consecutive beats in the user's dance style. The algorithmic approach to that is elaborated in Section 4.2. After that calibration phase, the mean time is used to generate a steady pulse. In Version 2 the steady pulse is yet only a beep sound with defined frequency and duration.

Version 3 implements a constant drum beat. To change from a steady pulse to an actual drum beat, the beep sound is exchanged with sound samples of hi-hat, snare and drum. These single WAV files are ordered and combined to build a simple drum beat. The drum beat consists of eight notes in the length of one bar, which is repeated over and over again. For the musical terms definition please refer to Section 3.1. The hi-hat plays eighth notes. Therefore, it is playing on every detected beat like the beep noise in Version 2. The snare joins the hi-hat on every third and seventh note of every bar, whereas the kick-drum is played on every first, second, fifth and sixth note of every bar.

As it is difficult to output two WAV files at the same time with one sound output device, the sound sample files first have to be combined. To implement the simple drum beat following WAV files were created: a single hi-hat WAV file, a combination of hi-hat and snare and a combination of hi-hat and kick-drum. In the *calc_drum_beat* method of the *ConstDrumBeat* class a loop was build which outputs the appropriate WAV file depending on a *beat_indicator* variable displaying the beat number of the bar. After one bar, thus eight beat numbers, the *beat_indicator* is set to zero again to continue the drum beat from the beginning. Therefore, a continuously looping simple drum beat is established out of single WAV files.

In Version 4, drum loops instead of single note samples are used to create a drum beat. After calibration phase, the mean time value is transferred to a bpm value which indicates the tempo of a music piece, see Section 3.1. With SQLite a database is generated, storing a different drum loop WAV file for various bpm values. After the bpm value is calculated, the appropriate drum loop is chosen and played continuously. As a result, the speed of the user's movement influences the tempo of the music piece because the beat detected in the motion data matches the rhythm in the chosen drum loop as they have identical bpm.

Version 5 adds a guitar loop to the drum loop of Version 4. Therefore, the database also stores a suited guitar loop for every stored bpm value. When outputting the combination of drum and guitar loop, first a combined WAV file has to be created with *pydub*. Afterwards, this WAV file can be played in a loop to create a music piece fitting the user's motion speed.

To reach the last algorithm version, Version 6 (Constant Drum Loop & Variable Guitar Loop), three different guitar loops are stored in the database for every saved bpm value. The choice of the guitar loop is based again on the user's movement data to give the user more influence on the music. As further explained in Section 4.2, the height of the user's right hand is used as an input defining which of the three guitar loops is chosen. Through moving the hand up and down while dancing the guitar loop can be altered after one loop ends, giving immediate aural feedback to the user. The six versions and alterations among them are visualized in Figure 4.8.