Universidad Carlos III de Madrid
www.uc3m.es

Ingeniería de sistemas audiovisuales

2015-2016

Trabajo Fin de Grado - Bachelor Thesis

"Implementation of a Real-Time Dance Ability for Mini Maggie"

Guillermo Elías Alonso

Supervisor: Esther Salichs

Director: Javi F de Gorostiza

Leganés, 5th July

Acknowledgements

I would like to thank my family for all the support they have given to me during my studies in the university; it is thanks to them that I have always felt the desire to try harder.

I would like to thank my fellow students for being great companions during all the university period. Besides, it is thanks to their great skills that I have become more competitive.

And last but not least, I would also like to thank my friends, especially the closest ones, for they have always been there to support me on everything and to maintain this great personal bond even from a long distance.

Contents

Abstract

The increasing rise of robotics and the growing interest in some fields like the human-robot interaction has triggered the birth a new generation of social robots that develop and expand their abilities. Much late research has focused on the dance ability, what has caused it to experience a very fast evolution. Nonetheless, real-time dance ability still remains immature in many areas such as online beat tracking and dynamic creation of choreographies.

The purpose of this thesis is to teach the robot Mini Maggie how to dance real-time synchronously with the rhythm of music from the microphone. The number of joints of our robot Mini Maggie is low and, therefore, our main objective is not to execute very complex dances since our range of action is small. However, Mini Maggie should react with a low enough delay since we want a real-time system. It should resynchronise as well if the song changes or there is a sudden tempo change in the same song.

To achieve that, Mini Maggie has two subsystems: a beat tracking subsystem, which tell us the time instants of detected beats and a dance subsystem, which makes Mini dance at those time instants. In the beat tracking system, first, the input microphone signal is processed in order to extract the onset strength at each time instant, which is directly related to the beat probability at that time instant. Then, the onset strength signal will be delivered to two blocks. The music period estimator block will extract the periodicities of the onset strength signal by computing the 4-cycled autocorrelation, a type of autocorrelation in which we do not only compute the similarity of a signal by a displacement of one single period but also of its first 4 multiples. Finally, the beat tracker takes the onset strength signal and the estimated periods real-time and decides at which time instants there should be a beat. The dance subsystem will then execute different dance steps according to several pre-stored choreographies thanks to Mini Maggie's *dynamixel* module, which is in charge of more low-level management of each joint.

With this system we have taught Mini Maggie to dance for a general set of music genres with enough reliability. Reliability of this system generally remains stable among different music styles but if there is a clear lack of

minimal stability in rhythm, as it happens in very expressive and subjectively interpreted classical music, our system is not able to track its beats. Mini Maggie's dancing was adjusted so that it was appealing even though there was a very limited range of possible movements, due to the lack of degrees of freedom.

Figure index

1. Introduction

1.1 Motivation

While some people are conjecturing about the future conquest of the world by excessively intelligent and powerful evil robots, so far they have been a great help to our society. Thanks to their speed and precision and to the increasingly sophisticated technology in the field of robotics, factories have boost their output with ultra-speed chain production, some dull tasks do not need to be made by humans anymore, doctors can do better interventions with high-precision aid and we have even had help in the kitchen or with the cleaning at home.

But robots have not always had purely functional purposes. Sometimes they are not even meant to have a specific function but they been brought to the world simply to pleasure and entertain people, to respond interactively to our actions and gestures or even feelings, to be our friends and have a great time with them, to perform spectacles alone to the public or along with other humans or even to entertain hospitalised kids or the elders in residences or at home as well.

Social robots have had a dramatic impact on current society. Some robots like Furby have marked the past of various generations, have entertained many children around the world and have been a great companion for the childhood of many people. Some people have even become fans of some robots, especially in Asia. Already in 1952 the release of the manga series *Tetsuwan Atomu*, also known as Astro Boy in most occidental countries, or other popular Japanese series like *Doraemon,* had triggered a high praise and fondness for robots, which were regarded as close friends by most children or likewise for more adult people.

One of the reasons for which we have developed such big admiration for robots is that we get easily impressed by their actions, since robotics have developed quite rapidly in the last years and 20 years ago robots were not capable of doing most tasks the same way they are able to do nowadays. We

are seeing different baby robot generations grow; we watch in the media how they learn new abilities and how researchers bring them up.

Something they have learned quite recently is the ability to dance. In the past, robots could just continuously make inarticulate artificial movements along with the music. Nowadays, as shown in many performances such as those with the QRIO robot made by Sony, some robots have developed the ability to move in a similar manner compared to us and make us feel they move their hips with the same passion some people do; or in other performances, such as recently in the 2016 Eurovision contest, robots even compete with humans to show they have better dance abilities than us.

Nevertheless, dancing is not only a matter of having many human-like joints and being able to perform many flexible and different moves and change between them very fluently. If our robot does not execute a pre-programmed choreography, it has to be capable of thinking of new dance steps as music is being played and to execute each dance step along at the rate of the music. May this seem natural and uncomplicated for most of us, this has not been fully accomplished in robots yet.

Not only performing different dances for different sorts of music or even for different parts of the same piece of music is an utterly complex task for a robot, but just synchronising with it and moving along with it online, as music is heard, is difficult as well. If it was already quite complicated for robots to dance according to the rhythm of a single piece of music, it is even much more confusing to them to dance along with several consecutive pieces of music, to detect a new song has begun and that it requires another pace or even other kind of dance steps.

Because of the subjective nature of all these tasks and all the metaphysical questions that they carry about the ultimate reach of imitating human emotions, this field of robotics has attired much interest for new research and new experimentation. The Social Robots Group in the RoboticsLab of the University Carlos III of Madrid, with which this thesis has been written, currently explores many of these topics such as motivations and emotional control of robots, visual and auditory human-robot interaction, multimodal robot-human interaction and the capacity of robots to have a dialog

with people. We are working in making robots friendlier and heartier, share our feelings and thoughts with them, in making people think they are not just metal and plastic, making people happier with new company.

However, all this poses questions about the sincerity of the response of robots to our gestures and feelings, can they really feel what they are programmed to say? Even if robots can make us laugh and have indeed an emotional effect on us, their deterministic nature impedes us considering we have mutual feelings. Nevertheless, with computers we can execute pseudorandom processes that humans cannot differentiate from real random processes and very often they are even better than us at doing so. Would you rely more on a human or a computer to have a random number from one to six? Are we humans actually a deterministic chemical system as well, but complex enough that if we abstract our behaviour we seem subjective and quasi random?

1.2 Objectives

In this bachelor thesis we are going to add a new social ability to one of our robots at the RoboticsLab of the University Carlos III of Madrid. In our case, we are going to make the robot Mini Maggie learn how to dance synchronously with music she hears real-time.

The main objective of this project is to build an online system in which we hear the music real-time from the microphone, analyse it and detect its beats in order to synchronise with it. When the robot is synchronised with the piece of music, it will have to move at the detected beat instants so that it can execute dance steps along at the rate of the music.

The main difficulty of this project lies on our desire to create an online system, so the robot should react to music real-time, to stop if it does not detect music anymore and to move at another rate if the music changes. Another important consideration being a real-time system, the robot must not act with a long delay but it has to process the input signal almost instantaneously from the point of view of human perception.

To accomplish this, the robot will have to fulfil three general requirements:

- **Beat tracking**:
    - Mini Maggie will analyse the music and extract its periodicities and sound peaks in order to find the **frequency** (extracting the so-called tempo in the music theory field) and **phase** of the corresponding beat signal of the input music signal.
    - Mini Maggie will hear the music and detect when music starts and ends so that the beat signal is always zero when music is off.
    - Mini Maggie will need to update the detected frequency and phase online since the incoming song could have parts at different tempi and the microphone signal could have consecutive songs as well.
    - She will have to dance synchronously with a general set of music, without making unnecessary assumptions about incoming music for the system to track beats of pieces of most music styles.
    - Beats will have to be detected with an unperceived delay and sudden tempo changes will have to update detected frequency without a long delay.
- **Dance generation:**
    - Choreographies should fit to the broadest set of input music.
    - We want Mini to execute dance steps according to her decisions and not hardcode a human-made choreography that fully matches a specific song that will be purposely fed into the microphone.
- **Integration and global control:**
    - Beat tracking and dance generation will have to operate together so that Mini Maggie executes a dance step at the time instants a beat was detected.
    - The whole system will become idle or activated if an external system requires that so that the dancing ability can be integrated in Mini Maggie's global system.

2. Previous research

2.1. Origin of dancing robots

Aulus Gellius wrote in his book *Attic Nights* that an ancient Greek philosopher and mathematician named Archytas, born 428 BC, is reputed to have made what is said to be the first robot of humankind, a wooden machine in the form of a pigeon, which was able to fly thanks to some air mechanism. [36] Nevertheless, although some people call this flying gadget a robot, even if it had indeed been created, it would not be a robot in the modern sense.

An ancient invention that resembles nearer what we nowadays understand as robot was designed by Leonardo da Vinci in 1495. [37] This mechanical knight was able to stand, sit, raise its visor and move independently each arm, the neck and the jaw. It is disputed that his design was really put into practice and that Leonardo da Vinci once constructed this robot but his accurate description made it possible for Mark Rosheim to reconstruct it in 2002 and prove that it was fully functioning.

However, the first autonomous robots that were driven by electricity were made by William Grey Walter in the 1950s. [38] They were called tortoises because of the shell that covered them and it's similar size and slow speed. They were capable of autonomously detecting obstacles and avoiding them thanks to a rotating photoelectric cell.

Even if William Grey Walter tortoises were electrically driven, they still had not the nature of modern robots since they were analogical. With the rise of digital technology, and specifically computers, there was an increasingly fast development in the field of robotics. An example of this development can be seen in the Honda E and P robot series, made for research in bipedal locomotion. [39] The first one of this series, the E0 robot, made in 1986, could just walk and took 5 seconds to move just one step. With the next robot generations of the E series, Honda increasingly gained speed for the robots, 5 years later they could reach a speed of 4.7 km/h and in 1993 the robot could already climb stairs of step over obstacles. Since then, Honda tried to make fully humanoid robots with its experimental P series reaching to the ASIMO

model in 2000 since today, which can perform many different complex actions like dancing, recognising objects or gestures and interacting with humans.

Since then, robot technology has become mature enough to not only perform non-cognitive actions such as simple moves. Nowadays, there is a new wave of research in more complex actions involving more logical processing or subjective-like thinking and memories such as interacting with humans and that has made possible the rise of social robots, which can be used for entertaining purposes and can even create a bond between the user and the robot.

This interaction between humans and robots has permitted robots, for example, to dance with humans, as many demonstrations have showed, such as at Digital Content Expo 2010 in Tokyo, where Japanese humanoid robot HRP-4C (nicknamed Miim) dances along with people in a performance to the public. But interaction does not only happen between robots and humans. In 2003 Sony created the QRIO robots, which could already dance together as showed in some performances. In January 2015 at Tokyo's Marunouchi building there was a demonstration where there were even 100 do-it-yourself robots dancing together.

Social robots are still in research phase and they have not been really integrated in the market yet. However, research has been very fast lately and some nations like Japan have showed an increasing interest for robots, what has accelerated dramatically their development. [40] Maybe in the next years go-go dancers will gradually be replaced by dancing robots in Japan.

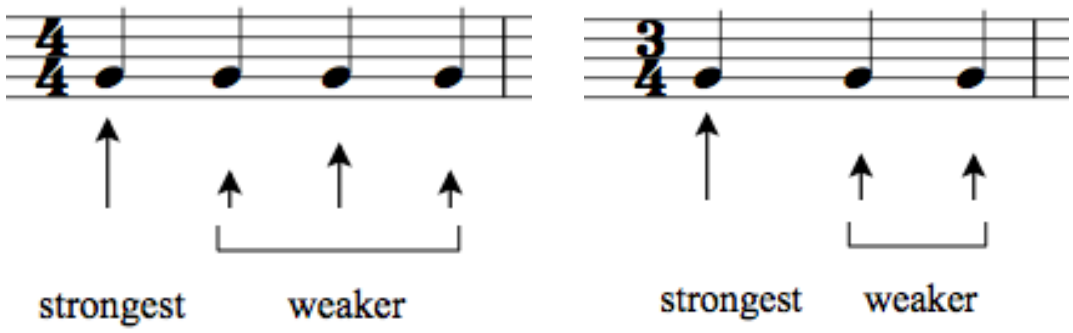
2.2. Beat tracking systems

2.2.1. Beat, subbeat or not a beat

Most pieces of music we are usually used to hear do not only have one type of beat. There is usually a hierarchy of beats in which some of them are louder, softer or structurally more important than others, what implies our robot has to filter somehow with which beats it wants to synchronise.

To define this hierarchy in a sheet, music is fit within bars, which indicate the position of the fundamental beats, and a time signature that specifies the

hierarchy of subbeats. The time signature can change in time within a single piece of music but in most of them it remains constant. We can divide time signatures into two main groups: those that divide beats into two subbeats (generally the most common), and those that divide them into three subbeats (for example, in waltzes). Splitting beats into more subbeats is in most cases just a combination of these two main groups (for example, quadruple time signatures can be thought of being twice duple time signatures) and in other cases, like splitting it into five subbeats, very rare.



(a) (b)

Figure 1: (a) Representation of the most common beat structure. (b) Representation of another common beat structure, especially in waltzes.

All this means that we do not hear all beats as easily and our perception depends on the music context as well. Besides, if we want to find a single BPM rate of a piece of music, that implies there are more than just one possible rate and we should decide which one fits better. If you are hearing a waltz and try to tap along at its rate, would tap with the fundamental beats or with each three subbeats?

To address this issue, Paul M. Brossier, Matthew Davies and Martin F. McKinney did an experiment in which they asked 40 participants to tap along in time with several 30-second excerpts of pieces of music of different genres. [1] Each piece had a unique and stable time signature (80% corresponding to the binary time signature group). The results were that the subjects tended to tap at the beat rate nearest to approximately 120 BPM independently of the fundamental rate or subrates. That means that if a waltz were too fast we would

tend to tap along in time with the fundamental beats but if it were slower we would tend to do it with each of the three subbeats.

This generates double uncertainty to real-time beat tracking systems, where the system at some instant could track the beats at the fundamental rate but later it changes to another subrate, what is generally undesirable. Besides, music sometimes has rhythmic phenomena such as syncopation in which more intense beats do not match the fundamental beats, especially in genres like jazz. In other cases, rythm could be complex enough that a time signature could not be abstracted so simply even by humans or the beat hierarchy is much complex than in the presented cases.

2.2.2. Peak detection algorithms

When a beat happens in a piece of music, usually that means the intensity of sound is largely superior at that time instant with respect the time history; that is, a peak occurs. Therefore, a simple way to track the beats could be just to search local energy peaks as microphone data comes in. Nonetheless, this is not very reliable, as we shall see.

For this algorithm, we would just have to calculate the energy at every time instant as the quadratic sum of input microphone data (for example, each 7 ms or each 1024 samples) and compare with the mean energy history. But if the signal has a very large variance, that may mean we find a beat too often since it is easier for a time instant to hold a higher energy in comparison to the local audio context. As a solution, to decide that at a time instant there was indeed a beat, we could impose the local peak to be steeper if previous input had a large variance. [2]

Even if this analyses quite superficially the input signal without much detail, this could be enough for applications where we want to track the beats of highly percussive music like electronic and hip-hop. Beside, this simple algorithm has a very low computational cost and that would make it feasible in almost any low-end hardware.

Nevertheless, this beat detection algorithm is rather unreliable for most other applications. If a piece of music contains a very powerful voice or any

other non-percussive instrument like saxophone or violin, it could mask even a very percussive accompaniment. This could be partially fixed by executing this algorithm independently at different frequency bands and that could be enough if the masked percussive parts are always in a different frequency band than the masking non-percussive parts like voice. But for music genres such as classical music and rock music, which do not have a high percussive character and beats are generally softer, this algorithm may not detect many beats or detect them quite arbitrarily. Besides, if a piece of music has a very complex rhythmic pattern, even if a robot dances moving at correct beats, we could have the perception that it does not move along at in time with the music. Moreover, most domestic hardware like personal computers have enough processing power to deal with a more detailed analysis of the audio input.

Peak detection algorithms are widely used and have many other applications apart from audio analysis so they have been thoroughly researched in other domains such as medicine [3], especially for electrocardiograms [4]; image processing [5]; speech recognition [6]; or research on general data flow, which could have every kind of applications such as traffic control and economic analysis [7]. Nevertheless, the subjective nature of music and the difficulty of synchronising at its rhythm by just simply searching for input peaks, has made research in this domain look for other kinds of algorithms that fit better for audio and specifically music and take into account human psychoacoustics.

2.2.3. Onset detection and feature extraction

A realisation of peak detection specifically for audio is called onset detection, since what gives us information about the rhythm of a piece of music is not exactly its raw signal peaks but note onsets, that is, the beginning of musical notes. Considering an audio signal does not have clearly defined note onsets at specific time instants, as in a sheet of music, we often speak about onset strength, i.e. the degree of onset or transient, which is related to the beats.

Particularising beat tracking algorithms for music signals makes them much more powerful since the onset strength at a particular time instant does not only depend on the sound intensity but also on other features. In this way, we can extract more relevant information or use high-level musical knowledge to make our algorithm more robust. In the next sections, we shall explore different features of the audio signal that some authors have used for onset detection.

2.2.3.1. Spectrum-related features

As we said, when a beat occurs in a piece of music, not only a sudden change of the sound intensity happens but also other features of the signal change sharply, e.g. the frequency spectrum of the signal.

Kristoffer Jensen and Tue Haste Andersen (2003) made a comparison of how the use of different features affected onset detection. [8] Among them, they used pure sound amplitude and other features based on the frequency domain. Thanks to their comparison using the same input audio signal, we can see that sound intensity does not necessarily offer the best results.

Since transients are sharp changes in the time domain, the importance of variations in the high frequencies becomes apparent for onset detection. Thus, these authors use features such as the spectral centroid and high-frequency content for their analysis. The spectral centroid of an audio signal is a measure of the frequency centre of gravity of the spectrum and is related to timbre and specifically the subjective perception of sound brightness. [9] Brighter sounds have a higher spectral centroid than sombre sounds. It is calculated as follows: [10]

$$SC(t) = \frac{\sum_{n=1}^{N} a(t,n) \cdot f_c(n)}{\sum_{n=1}^{N} a(t,n)}$$

Where *a(t, n)* is the amplitude of the n-th frequency band at time instant *t* and *fc* is its centre frequency.

The high-frequency content of a signal is a weighted sum of the amplitude of each band, where the high-frequency bands are given more importance with a bigger weight:

$$HFC(t) = \sum_{n=1}^{N} n^k \cdot a(t,n)$$

Where n^k is the weighting factor and different values of k produce different distributions of weight. Paul Masri proposes $k = 1$ for his audio transient analysis work [11] whereas Kristoffer Jensen and Tue Haste Andersen use $k = 2$, [8] thus giving more importance to high frequencies.

An interesting aspect of HFC is that it does not only give information about the high-frequency energy but also about the global sound intensity (if the factor k is not too high). For this reason, this is a good way to evaluate at the same time both the variations of amplitude in the time domain and the spectrum of the audio signal.

2.2.3.2. Adding human perception

Human beings do not perceive sound pressure in a linear way but rather in a logarithmic scale. This implies that, for example, if two violins are performing together the same piece of music, we do not perceive sound to have double intensity that just one violin.

For this reason, some authors prefer to use a logarithmic-scaled representation of sound amplitude for their onset detection algorithms. Anssi Klapuri made a comparison of the logged and the non-logged model with different transient piano sounds and concluded that the logged-model finds more accurately the time instant where the onset occurs and that it filters more efficiently other local amplitude maxima that correspond to the same onset. [12]

Our hearing does not only sense sound pressure logarithmically but also frequency. That means our ear separates sound into wider bands in the higher frequencies, thus having more resolution at lower frequencies. There are many models of how our hearing filters sound into such frequency bands.

The Bark scale [13] and the mel scale [14] map the frequency domain into a logarithmic-scaled one. These models map the lower frequencies rather linearly whereas higher frequencies are mapped more logarithmically. Some authors like Tristan Jehan [15] use the Bark scale whereas Daniel P.W. Ellis [16] uses 40 mel bands for his onset detection algorithm apart from the above mentioned logged-scaled representation of intensity as well.

Additionally, there are other psychoacoustical details that we could take into account to adapt the onset detection to human perception, such as the masking of consecutive sounds in the time and frequency domain, which mp3 uses for compression purposes. Another widely aspect of hearing used for onset detection is the relation between loudness perception and frequency, [17] as expressed in the equal-loudness contours. [18] This implies that we do not hear sound with the same amplitude and different frequency equally loud. This relation of loudness with respect to frequency even changes with the sound level. Besides, these curves are just standardised and in reality vary between each person, what exemplifies the complexity of human perception and its limited reach for audio analysis.

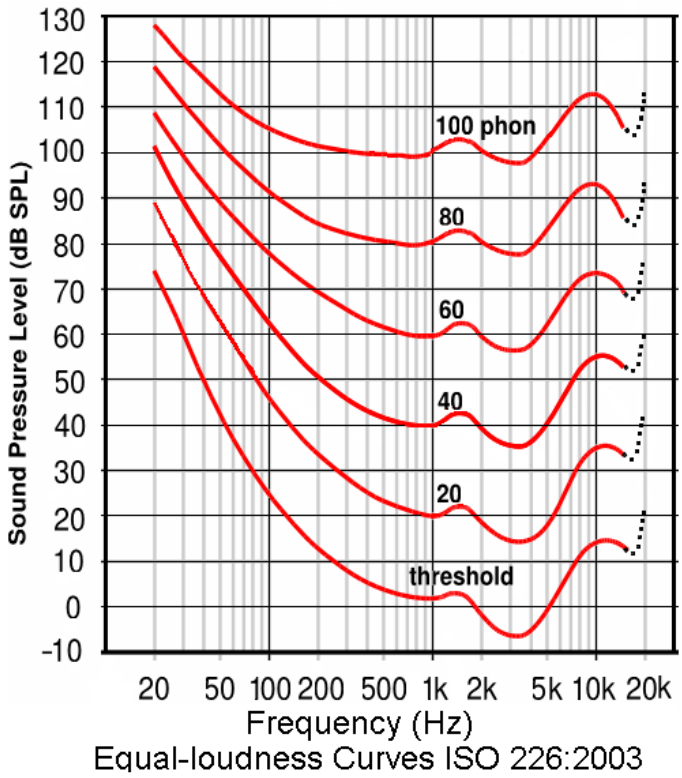


Figure 2: Representation of the equal-loudness curves (ISO, 2003).

2.2.3.3. Using high-level musical knowledge

We can particularise even further our audio signal and treat it as music so that we can take into account musical features that help us find the position of beats. Masataka Goto and Yoichi Muraoka took this approach and made two different real-time beat tracking systems that were robust enough to detect the beat structure of real-world signals. [19][20]

Most western popular music has a set of drums, which mark quite clearly the rhythm of a piece of music. One of these systems from these authors takes advantage of this by extracting the bass drum from the piece of music, which would most probably mark the most prominent beats, and the snare drum, which would give a hint about the inner sequence of strong and weak beats. [19]

Later, they tried to make another system that would also work for drumless music. In this case, they used the fact that most western music has chord changes at the beginning of bars, the same place where the stronger beats happen. When a chord is maintained, we can observe some stability in the spectrum of the audio signal whereas it changes significantly when the chord is switched. [20]

2.2.3.4. Alternatives approaches in feature extraction

All the above-explained techniques have many things in common that we usually take for granted, like doing a frequency analysis with the FFT. Here we present a set of alternatives that some authors have used for feature extraction in audio signals.

Performing the FFT of a signal periodically in time, what is generally called the short-time Fourier transform (STFT), presents some problems related to its fixed resolution depending on its window size, which specifies what amount of the signal is taken to calculate the FFT. If the window is too large, we have a poor time resolution at transients and if it is too small, the STFT has

poor frequency resolution at low frequencies. [21] This implies there is a trade-off between time and frequency resolution.

To tackle that, there have been two main approaches. On the one hand, Alexey Lukin proposes to vary the window size to imitate the time-frequency resolution of humans and adapt to local signal features like transients. [21] On the other hand, another transform, the discrete wavelet transform (DWT), does not provide a fixed resolution like the STFT but high time resolution and low frequency resolution for high frequencies and high frequency resolution and low time resolution for low frequencies, what is more similar to human hearing and can also be used for beat detection. [22]

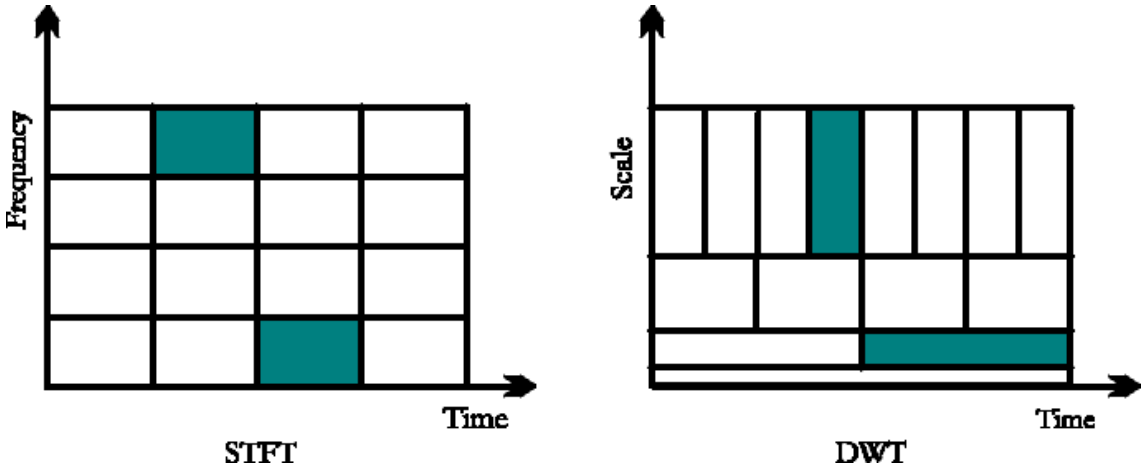


Figure 3: Time-frequency resolution of STFT and DWTW (wavelet.org).

Additionally, in onset detection we are generally interested in how features vary in time. As a consequence, most onset detection algorithms are based on calculating the first-order difference between the value of the feature at the current instant and that at the previous one. But this is not the only approach for finding how they change in time.

Masri and Bateman use the ratio between both instants, normalised with the signal energy for their HFC analysis explained above: [11]

$$\frac{HFC(r)}{HFC(r-1)} \cdot \frac{HFC(r)}{E(r)}$$

Where HFC(r) is the high frequency content at current frame, HFC(r-1) at previous one and E(r) its energy.

Another approach is to take not only the previous consecutive instant to compute the difference but M previous instants: [17]

$$C(r) - \frac{\sum_{\tau=1}^{M} C(r-\tau)}{M}$$

Where C(r) is the value of the feature at current instant. This results in a smoothing of the onset detection signal compared with the sharpness of using the first-order difference.

2.2.3.5. Which one is the best feature?

We have seen that there are many different features that we can extract from a signal in order to track the beats of a piece of music. Therefore, a question may arise: which is the feature that offers the best results for a beat tracking system? Nevertheless, there is not a definite answer.

Particularising the type of music from which we want to infer the beat positions can help us make more assumptions that we can use to track beats, as Masataka Goto and Yoichi Muraoka did, as we exposed above. As a consequence, they may have better results for this set of pieces of music but this system is useless for other pieces. Thus, particularising implies a trade-off between having better results but a narrower range of use.

Some authors have tried to compare many of these different features. Among these works, we can find that of Nick Collins [17], Kristoffer Jensen and Tue Haste Andersen [8] and Matthew E. P. Davies, Norberto Degara and Mark D. Plumbley [22]. Nevertheless, it is very difficult to make cross-work comparisons since there is not a standardised set of pieces of music to track beats and, therefore, each work may have different results according to the test data they used.

Another critical element in comparing different methods is the lack of a single ground truth, as we explained in section 2.2.1. This is due to the subjective nature of music and the many different interpretations that music can have for different listener, not only because of our psychoacoustics but also because our memories and culture can alter the way we perceive music.

2.2.4. Music period detection

For a general set of music, just computing the onset strength signal with respect to some features and extracting the beats according to some threshold, does not provide a robust beat tracking system. Sometimes, features like the spectral centroid or chord change do not always vary along in time with the beats.

Additionally, if the music is turned off, humans are still able to abstract would-be beats and keep tapping or dancing along in time with the previous music, since there exists a repetitive pattern. If we extract how often the beats are repeated, i.e. the BPM (beats per minute), we can use this information to make our system more robust. There are two main methods to do that: using the autocorrelation or using comb filters.

2.2.4.1 Autocorrelation

Autocorrelation is used in many fields of study, e.g. statistics and signal processing. Nonetheless, different fields of study do not have the same definition for it, so we will talk about autocorrelation as in signal processing, the domain we are interested in.

For a finite-length discrete signal *s[n]*, the kind of signal that we can process in computers, the autocorrelation is defined as follows:

$$A(\tau) = \sum_{\tau=0}^{\tau=N} s[n] \cdot s[n-\tau]$$

Where τ represents a delay and *s[n-* τ*]* is signal *s[n]* delayed by τ.

In other words, the autocorrelation is a measure of the similarity of a signal with itself delayed by different amounts. If our signal has some periodic content with period T, the autocorrelation will be the biggest at τ = T. Consequently, we can extract the BPM of a piece of music by finding at which delay the autocorrelation has the largest value. This information can then be used in our beat tracking system to find the speed in which beats are repeated. [16]

If we want to make an algorithm to compute the autocorrelation function using this definition exactly, it does not scale linearly since it has a computational cost $O(n^2)$, so other algorithms are used such as the MKC algorithm. [23] Nonetheless, we could also use an alternative definition of the autocorrelation: [24] [25]

$$A(\tau) = iFFT\{FFT(s[n])FFT(s[n])^*\}$$

However, if we just want the autocorrelation of a limited set of delays, scalability is not as important. For example, in the case of finding the BPM of a piece of music, we could suppose the speed could just range from 40 to 250 BPM, therefore, it is not necessary to compute the autocorrelation with every possible delay.

Ultimately, if finding the period is not enough for our application and we want how this number corresponds in BPM, we have to compute the inverse of the period, since BPM is a frequency measure, and multiply the result by 60, considering BPM is a measure of the beats in one minute and not in one second:

$$BPM = \frac{60}{T}$$

Where T is the period we found with the autocorrelation function.

2.2.4.2. Comb filters

A comb filter is a system that adds a scaled and delayed version of its own output to the input signal: [26]

$$y(t) = x(t) + \alpha \cdot y(t - \tau)$$

Where τ is the delay and α is the gain. This lag causes constructive and destructive interferences that make the system filter out or amplify all multiples of a base frequency.

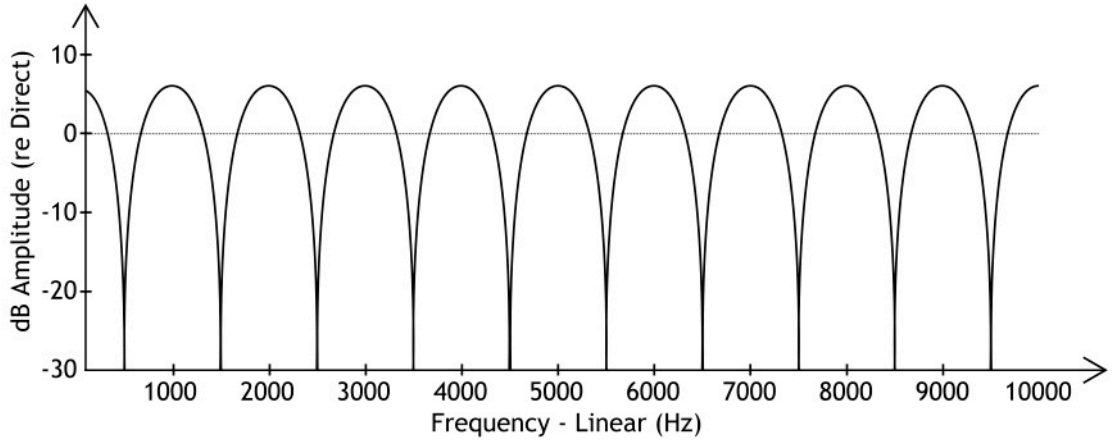


Figure 4: Frequency response of a specific comb filter (recordingology.com).

Another property of comb filters is that if they are fed with a periodic signal of period equal the delay of the comb filter ($T = \tau$), resonance occurs and the output is bigger. [27] We can use this property such that we let our signal pass through a set of parallel comb filters with different delays τ and see which comb filter has the largest output, which will be the period of our piece of music. If needed, we can then convert the period into BPM, as explained before.

Not only to have a high precision but also for the algorithm to work properly, we need a fair amount of comb filters with different delays, since they resonate at one particular frequency, not a range of frequencies. However, using many comb filters is computationally expensive. Eric D. Scheirer had good results with a bank of 150 comb filters ranging frequencies logarithmically from 60 BPM to 240 BPM. [27] Hanchel Cheng and Sevy Harris propose, however, to make computations in the frequency domain with the FFT to reduce computational cost. [28]

2.2.4.3. Comparison

The use of comb filters and autocorrelation have some things in common. Analytically, the operations they compute are similar, comparing to a delayed version of the signal. However, there are important differences as well.

The main advantage of comb filters is that they do not only resonate at multiples or fractions of the delay τ (2τ, 3τ, $1/2\tau$, $1/3\tau$), but also at simple rational relationships such as $3/2\tau$, $3/4\tau$, etc. [26] From a psychoacoustic point

of view, this approximates better the ground truth, since a tempo of 60 BPM in a binary piece of music contains a subtempo of 120 BPM as well, and different humans may tap at different of these tempi (see section 2.2.1).

With autocorrelation methods, we just can extract the tempo of the music, but this is not enough to track the beats since we also need the phase. However, with comb filters, with can simultaneously extract the beat frequency and the phase since it estimates the output at each phase angle of each lag whereas the autocorrelation integrates it. [27]

Additionally, to find the autocorrelation at $\tau = T$, the autocorrelation only compares the signal with a version delayed by a single period and not by a bigger number of periods whereas the comb filter compares it infinitely far in time, yet with less weight as it gets further. [27]

Nonetheless, the autocorrelation has the advantage that it is more efficient in memory usage due to the fact that we have to use one comb filter for each single lag, as we explained above, while the autocorrelation involves all different lags with it and they are generally simpler to implement.

2.3. Real-time dancing robots

In previous sections we talked about general beat tracking systems that could be applied not only to robots but to other applications as well. When a robot knows when it has to move it has to decide which dance step it will perform. Besides, beat tracking systems for robots have some particularities. In this section we are going to comment some of the many issues that have dancing robots in order to synchronise to music and perform dances according to it.

2.3.1. Ego noise

One of the problems that have beat tracking systems specifically on robots is that signal quality is damaged with the sound of the motors while the robot is dancing, what is called *ego noise*, the noise that the robot produces.

This lowers the signal-to-noise ratio, thus inferring in the reliability of the system.

To solve this problem, a simple solution could be just to increase the volume of the music a lot so that the signal-to-noise ratio remains high. However, recently more sophisticated methods have been developed, which include a pre-processing stage to the beat tracking system where ego noise is eliminated from the incoming audio signal. The first approaches arose in 2008 with Mizumoto [41] and Murata, [42] which focused mainly on rejecting the voice of the robot, one of the sources of ego noise.

The first beat tracking system that took into account ego-motion was developed in 2012 by Oliveira, Ince, Nakamura and Nakadai, which improved the robustness of the beat tracking system by 15 points compared to that without ego-motion estimation. [43]

To estimate ego-motion noise, they propose to pre-record a set of audio data of noises caused by different joints and movement speeds to model each possible kind of noise. Then, at each audio frame, joint state is acquired and speed is estimated and the robot looks into the audio database for the nearest neighbour in terms of joint position and speed and next uses the corresponding audio extract to subtract it from the incoming audio signal. Finally, they use a general beat tracking algorithm with the audio signal with ego-motion noise reduction.

2.3.2. Automated choreography

While robot manufactures, showing in public performances how well their robots can dance, usually prefer meticulously pre-thought choreographies by specialists, since they are usually more impressive and totally synched with the music. However, in recent years there has been an increasing interest in researching how are robots able to create their own dances with the smallest human intervention.

A more specific issue inside this problematic is how the robot has to make a transition from a joint state to another joint state, i.e. how it has to change from a posture to another smoothly and in a natural way. This issue has

been explored by Gunwoo Kim et al. for their research in virtual dancing characters in computer animation. [44]

First of all, they prepare a database of dances that the robot has learned by motion capturing of the performances of a dancer. These dances include specific joint angles at beat instants and their transitions; therefore, if the robot wants to make a dance based of parts of the dances of the database, it has to clearly identify transitions. However, that could be too slow since the database can be massive, due to the high number of degrees of freedom and joints, which means a high dimension. To tackle this, they use PCA (Principal Component Analysis), what basically reduces dimension while trying to minimise the loss of relevant information, and detect transitions from that.

Bipedal robots have an additional complexity while moving since they need to remain stable in order not to fall. For this reason, the Japanese AIST has implemented an interface for their robot HRP-4C, nicknamed Miim, that hides to the user all the necessary management for the transitions between positions while taking into account stability management at the same time. [45] If the user tells Miim to do movements that are not possible for stability reasons, this software also has the capability of letting the robot execute similar movements, which do not put the robot in danger.

Other authors like Guangyu Xia et al. have focused in making self-created choreographies as more human as possible by extracting real-time emotions from music and adding some randomness. [46] They represent emotions as a two-dimensional vector using Thayer's two types of emotional qualities. Then, with the use of some training data, they label each possible position the robot can perform with one emotion vector. To make dancing more random, they propose the use of Markov chain, which models dancing by representing postures by states while transitions between states have a specific probability.

3. System description

3.1. Used technologies

3.1.1. ChucK

ChucK is a strongly timed programming language for real-time audio processing and synthesis created by Ge Wang. It is open-source and is supported by Linux, MacOs X and Windows. [30] Its creation was motivated by the lack of the notion of time in most programming languages and the excessive abstraction of time in high level computer music languages. [29]

The main objective while creating this language was to address this issue and offer to the developers a way to control exactly when each task is executed. As design goals for the language, Ge Wang set the flexibility as a priority to let programmers express their ideas without difficulties into code and allow fast prototyping for rapid testing. [29] For this purpose, the author tried to make the language as readable as possible. Although the author recognises the importance of the performance, it is not set as a top priority for the language itself and it is handed over to the developers to provide them with the maximum control.

ChucK supports a simple sample-synchronous, non-preemptive concurrent programming model in which many shreds (ChucKian threads) can be synchronised in time. It supports the use of unit generators that let the programmer use the loudspeaker and microphone and create different output signals, filters, basic signal processing and instruments and synthesisers. There are also unit analysers, which make possible time-frequency domain transformations and feature extraction for features such as the spectral centroid and spectral flux. It supports MIDI and the OSC protocol as well.

ChucK is a C-like language but its main feature is the ChucK operator =>, which permits assignment of variables and the definition of streams in a left-to-right manner with the combination of various ChucK operators. [31] Besides, it has an object system, which parallels the conventions of C++ and Java. [30]

ChucK is easy to install but it was already installed and integrated in the robot Mini Maggie since it was used for other previously added abilities such as speech recognition, noise filtering and sound generation. Besides, it provides a powerful set of classical tools of signal analysis, what made ChucK the ideal language for the first part of this project. Nevertheless, the lack of documentation made it difficult to learn how to use all its possibilities and features.

3.1.2. OSC Protocol

OSC (Open Sound Control) is a simple protocol for real-time sound based communication among computers, synthesisers and other multimedia devices. [32]

It offers high resolution time tags, data-typed communication, pattern matching language to specify multiple recipients of a single message, message "bundles" which act as a single block and a query system to dynamically find the capabilities of an OSC server. [32]

The main transmission of OSC is done via packets, in which the receiver is the server and the emitter is the client. OSC packets can be messages or bundles. OSC messages begin by an address pattern started by '/', i.e. "/oscevent" or "/synth2/channel1", that allow messages to be easily categorised and customised. It is followed by a type tag string started by ',' where the different arguments that contain the message are defined. In the last place, we find the value of the arguments in the same order they were defined.

The OSC protocol is used in applications such as sensor-based electronic music instruments, multiple-user shared musical control, virtual reality, networked LAN music performance or web interfaces.

3.1.3. ROS

The Robot Operating System is a set of libraries and tools which are aimed for robot development. Its goal is not to be a framework with the most

features but to support code reuse in robotics [33] and encourage collaborative robotic software development by providing a highly modular framework. [34]

The ROS project arose from the work at Stanford University of the STanford AI Robot (STAIR) and the Personal Robots (PR) program around 2007. [34] Then, Willow Garage, a hardware and software developer for robot applications, [35] along with many other institutions and many different types of robots took part in the extension of the core ROS concepts leading to an open-source and very distributed system that makes it possible for many people to work easily in the same project and addresses the need of a collaborative framework in the robotics research community.

ROS provides an inter-process communications infrastructure and robotic-specific libraries and tools such as the Robot Geometry Library, which aids the developer in controlling the relative location of the different parts of the robot, and Robot Description Language, to describe robots in a machine-readable way. It supports many command-line tools for debugging, plotting, and visualising in a simple way such as rviz, a general purpose and three-dimensional visualization of exchanged messages, and rqt, which allows the developer to plot variables, visualise a live ROS system and manually manage messages for debugging.

ROS uses client libraries to ease the job of the programmer by transforming ROS concepts into code. To main stable libraries are for applications in c++, python and LISP. Nevertheless, there is a large collection of other experimental libraries that can be used for other programming languages such as Java, Go, R, Lua and Ruby.

Each independent ROS process is called *node*, which generally performs a specific task. Various nodes performing different independent but connected tasks form a ROS system. In order for nodes to find each other, exchange messages or invoke services, they need the *ROS Master*, which, as a consequence, has to be invoked before invoking any node. State of the nodes can be saved in the *Parameter Server*, which is part of the master. Inter-nodal communication in ROS works by subscribing or publishing to *topics*. If a node is subscribed to the topic “/robot2/light_sens_1”, it will receive all the messages that are sent to this topic and if it publishes to the topic “/hr2/arm_joint”, it can

send messages to this topic. This system allows at the same the intercommunication of nodes and their independency by decoupling the emission and reception of messages. However, if a request-reply model is more appropriate for our application, we can still use *services*.

The collaborative nature of ROS has made it become one of the most influential robotic frameworks among the research community. It has brought much attention thanks to its uncomplicated nature and its many successful works using ROS like the PR2 of Willow Garage and it has allowed many professors in the robotic academic world let their students apply in a simple way the theoretical concepts learned in class.

3.2. Mini Maggie overview

Mini is a social robot that is oriented to the elders with cognitive deterioration. It has been created by the Social Robots Group of RoboticsLab in the university Carlos III of Madrid. It has a similar aspect to the robot Maggie, who also belongs to this work group.

Mini is a desktop robot in order to facilitate its transportation and eliminate the problems of needing to go to charge the batteries. However, she owns an internal battery as well so that she is able to perform a controlled shut down and so that people can move it manually from one location to another without a necessary restart. For central control, it has an i5 processor.

Mini has five degrees of freedom, which are controlled by dynamixel servos: one for each arm, two for the head and one for her base.

As opposed to the robot Maggie, Mini has a soft doll-like cover to improve the appearance for the interaction of the users. Besides, it has been given a lot of expressivity by adding different light devices:

- Two 128x128 uOLED screens that simulate the eyes of the robot, where different gifs are displayed to emulate different emotions (anger, sadness, happiness, tiredness…) and blinking of eyelids as well.
- Two RGB leds to simulate cheecks.

- One RGB led for the heart, which changes in color according to the state of the own robot and simulates Mini's heartbeat by turning on and off.
- VU metre that simulates the movement of the mouth, which operates according to the intensity of Mini's speech.

Besides, it has three tact sensors, which are located in the shoulders and the stomach and are controlled by an Arduino Mega board. It has other devices as well in order to perceive the environment, such as Kinect to detect if someone approaches Mini Maggie and a microphone, which is installed in the middle part of the robot.

Mini has a table, which shows different multimedia content (music, videos and photos), and other functionalities such as entering the Internet or making video calls by Skype as well.

In the software level, Mini works over the ROS framework. Its control is based in a state machine, programmed with Smach, whereas transitions are controlled by the dialog system IWAKI. In order to control the inputs and outputs of this dialog system, there are two packets called multimodal fussion and multimodal fission.

Initially, Mini starts in the SLEEPING state. When someone touches her stomach, she wakes up, greets the user and goes to the WAITING state. This is the central state and from this one we can go to all the different states according to the ability that we want the robot to perform.

Interaction with Mini can be performed through the tact sensors and/or speech with the microphone. Both speech recognition and the synthesiser that is used to generate Mini's voice are performed with Loquendo.

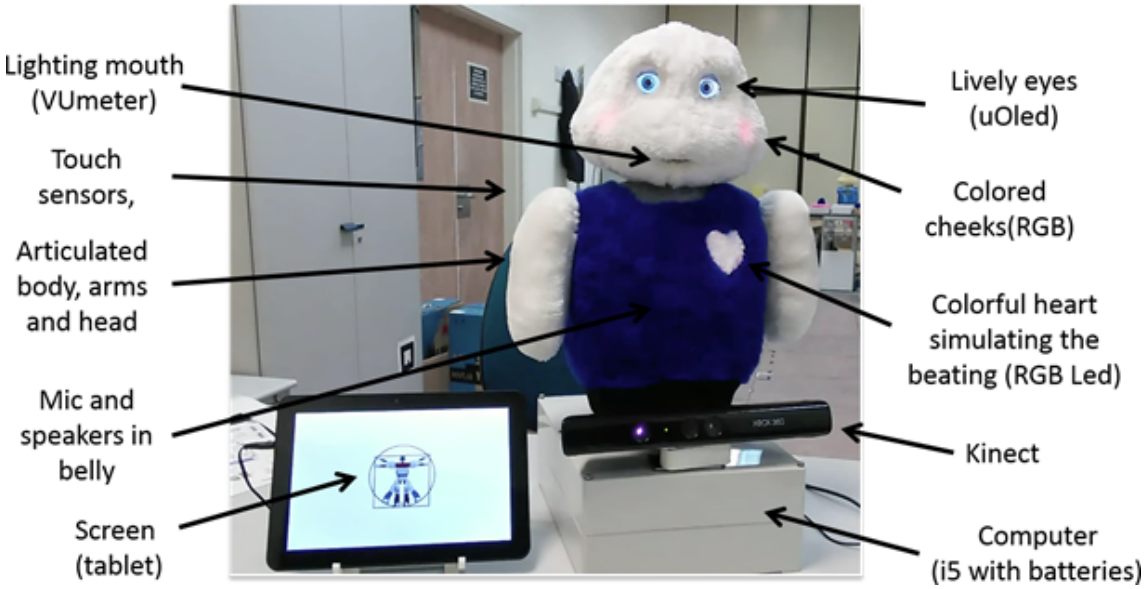


Figure 5: Robot Mini Maggie.

3.3. Implemented system overview

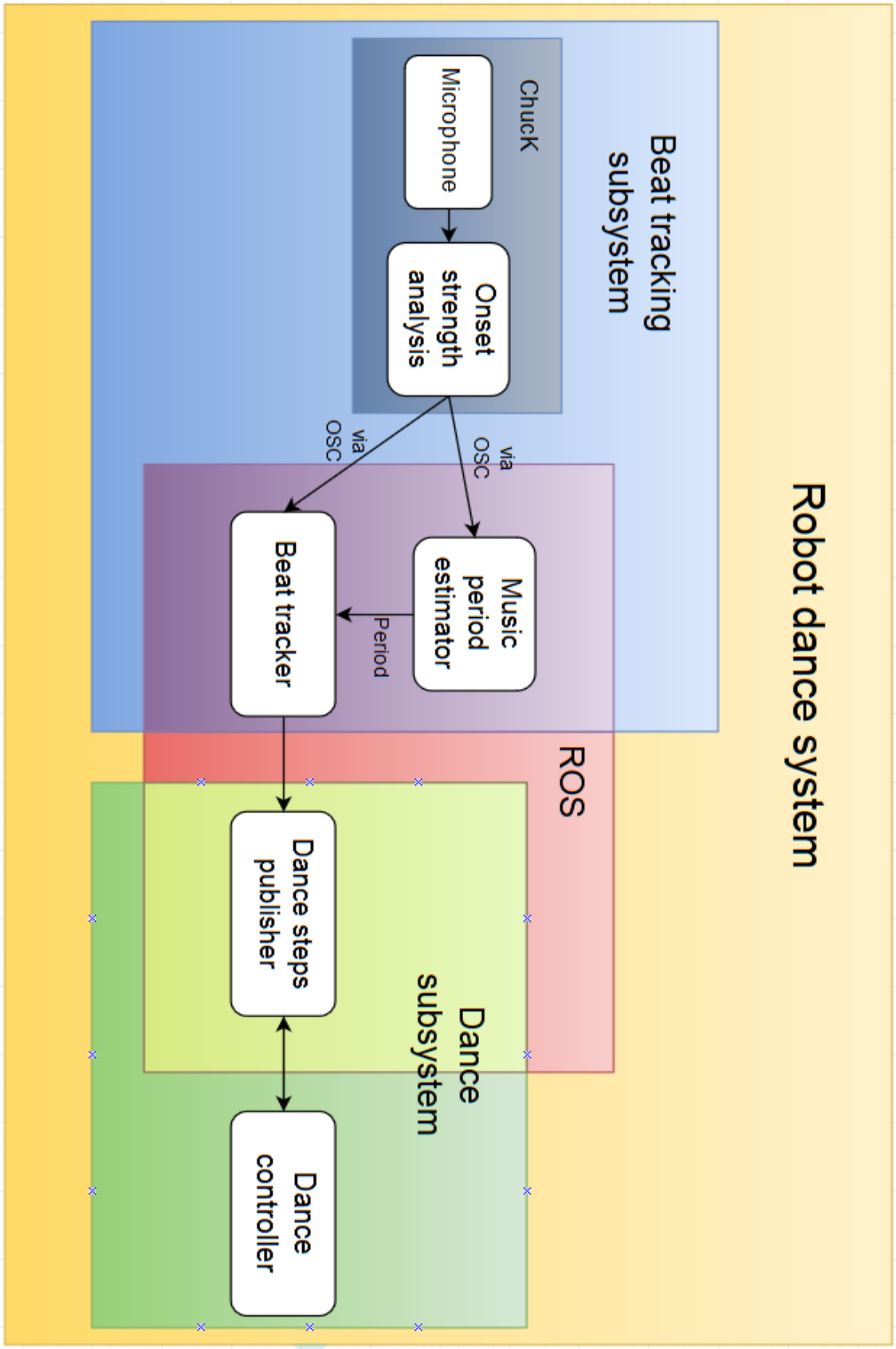


Figure 6: Global overview of the implemented dance system for Mini Maggie.

The dance system for mini Maggie is comprised of two subsystems: the beat tracking subsystem and the dance subsystem. The beat tracking subsystem tells the robot when to move and the dance subsystem decides what move it will perform.

On the one hand, the beat tracking subsystem begins by making an onset strength analysis of the microphone input signal with the ChucK programming language. From this analysis, an onset strength signal is outputted and is then delivered via the OSC protocol to the music period estimator and the beat tracker, both written in different C++ files. The music period estimator extracts the BPM using the autocorrelation of the onset strength signal and it hands it over to the beat tracker, which with this information in addition to the onset strength signal tracks the beats.

On the other hand, the dance controller guards a set of possible dance steps saved in a text file. It is in charge of reading the dance steps from this file, deciding the appropriate dance step and telling the dance step publisher about it. The dance step publisher will then publish the specific movements to perform to the robot joints according to the dance step brought by the dance controller when the beat tracker commands to move.

Robots do not only have one single ability, but they can execute many different tasks in very different domains, such as singing, playing different games, helping people, as well as dancing. That means this dancing system will ultimately be in another supersystem that manages all the abilities and controls when each ability should be performed. For this reason, an external topic called “dance_command” has been created to start and finish this dance ability for a higher-level system to manage it among other abilities. If a “finish” string value is sent to this topic, the beat tracker will shut down, thus making all the dance system become idle since it relies on the beat tracking subsystem to move. When a “start” string is received, the beat tracker will operate again and, therefore, the dancing ability as well.

3.4. Beat tracking subsystem

This system is in charge of tracking the beats of a piece of music. We can view this system as a block where an input signal is processed and outputs another signal. In this case, it has the microphone signal with respect to time as an input and the output is a set of ones or impulse train (or any similar way of describing a binary output) with respect to time, which have to correspond to the beats of the heard music.

This system can be represented as follows:

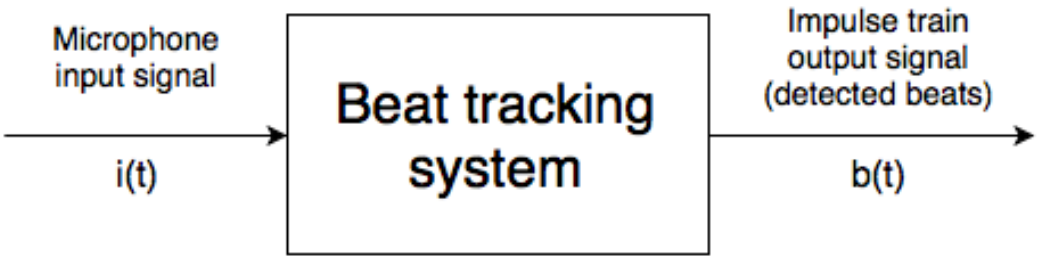


Figure 7: Beat tracking system as a block with an input and output signal.

Where the output signal *b(t)* represents the detected beats and mathematically is:

$$b(t) = \begin{cases} \sum_{n=-\infty}^{+\infty} \delta(t - n \cdot \tau(t)), & if\ music\ if\ on \\ 0, & if\ music\ is\ off \end{cases}$$

Where *n* is an integer number and $\tau(t)$ is the detected period of the different pieces of music that the robot hears in time. This output signal will then be used to know at what time instants the robot will have to move.

This system or block is internally, at the same time, a set of other smaller blocks that have different input and output signals. These blocks correspond to the bubbles in the beat tracking subsystem in the figure of the overview of the global system (see figure 6). The onset analysis block takes the microphone signal as input and the onset strength signal *o(t)* as output. The music period

estimator block has this onset strength signal as an input and outputs the detected periods real-time $\tau(t)$. The beat tracker block takes both the onset strength signal *o(t)* and the detected periods $\tau(t)$ and outputs the signal *b(t)* introduced above.

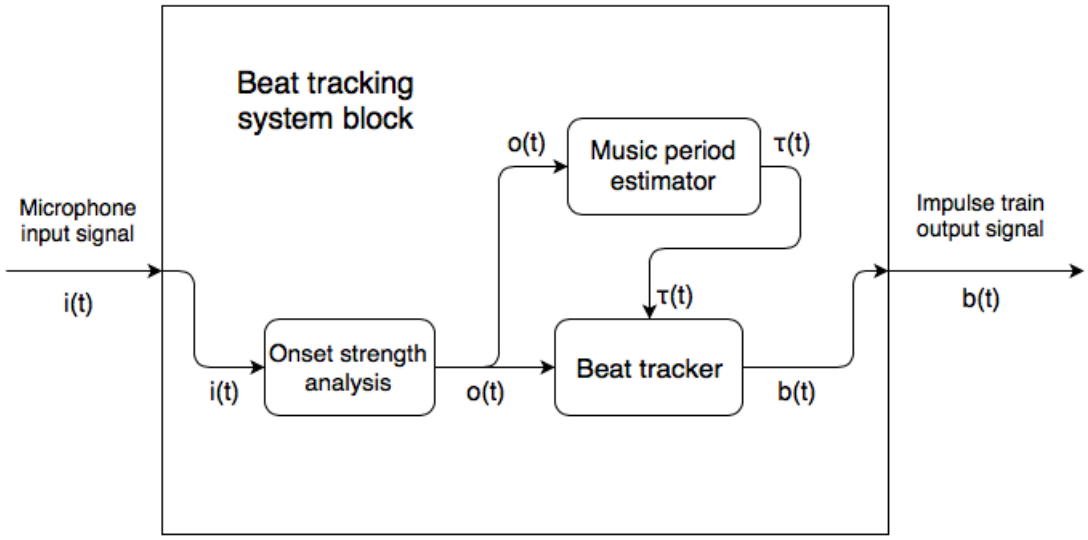


Figure 8: The internal blocks of the beat tracking system.

In the next sections, we are going to explain how each internal block works and how it is implemented.

3.4.1. Onset strength analysis

The raw microphone input signal is not appropriate for directly finding its beats. If we use the raw signal and we feed it into our beat tracking blocks, the system would be too unreliable. Therefore, we have to make some pre-processing and analysis so that we have a signal that better represents the probability that there exists a beat at a specific time instant. The output of this block would then be the onset strength signal.

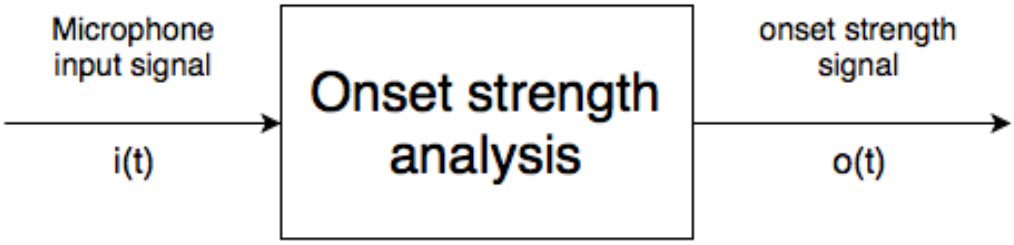


Figure 9: The onset strength analysis as an input-output block.

In the analysis of a musical signal and in determining exactly when a beat happens in the music, timing is essential. That is why we used the ChucK language for implementing this block, since with this language we can control time precisely and easily.

The basic structure of the implemented algorithm is as follows. We listen to the microphone at a 8 kHz sampling rate and we first wait for 4 ms of microphone input samples in order to have a sufficient amount of information. Then we calculate the energy at that time instant by making the quadratic sum of the samples and we check if music is on or off according to an algorithm explained later. Is music is off we go back to buffering the next samples and if it is on we process these samples to perform feature extraction, as commented in the previous work, and extract the onset strength at this 4 ms time instant (although it is not instantaneous in the physical sense, it is pretty much a time instant for our perception). All this results in a single float number, which is then sent via the OSC protocol to the next blocks of the beat tracking subsystem.

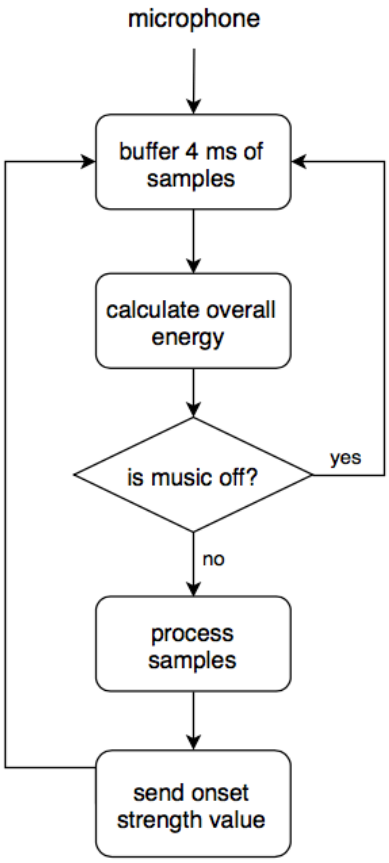


Figure 10: The general flowchart of the onset strength analysis block.

The “is music off?” block in the previous flowchart has two possible states that are saved in a boolean variable: music is off (the initial state) and music is on. To change between states we use the energy at that time instant and a counter. If the energy is below a threshold, we increment the counter; and in the opposite case, we decrement it by 6. If the counter is above a counter threshold, we change the state to “music is off” whereas if it is below 0, we change the state to “music if on”.

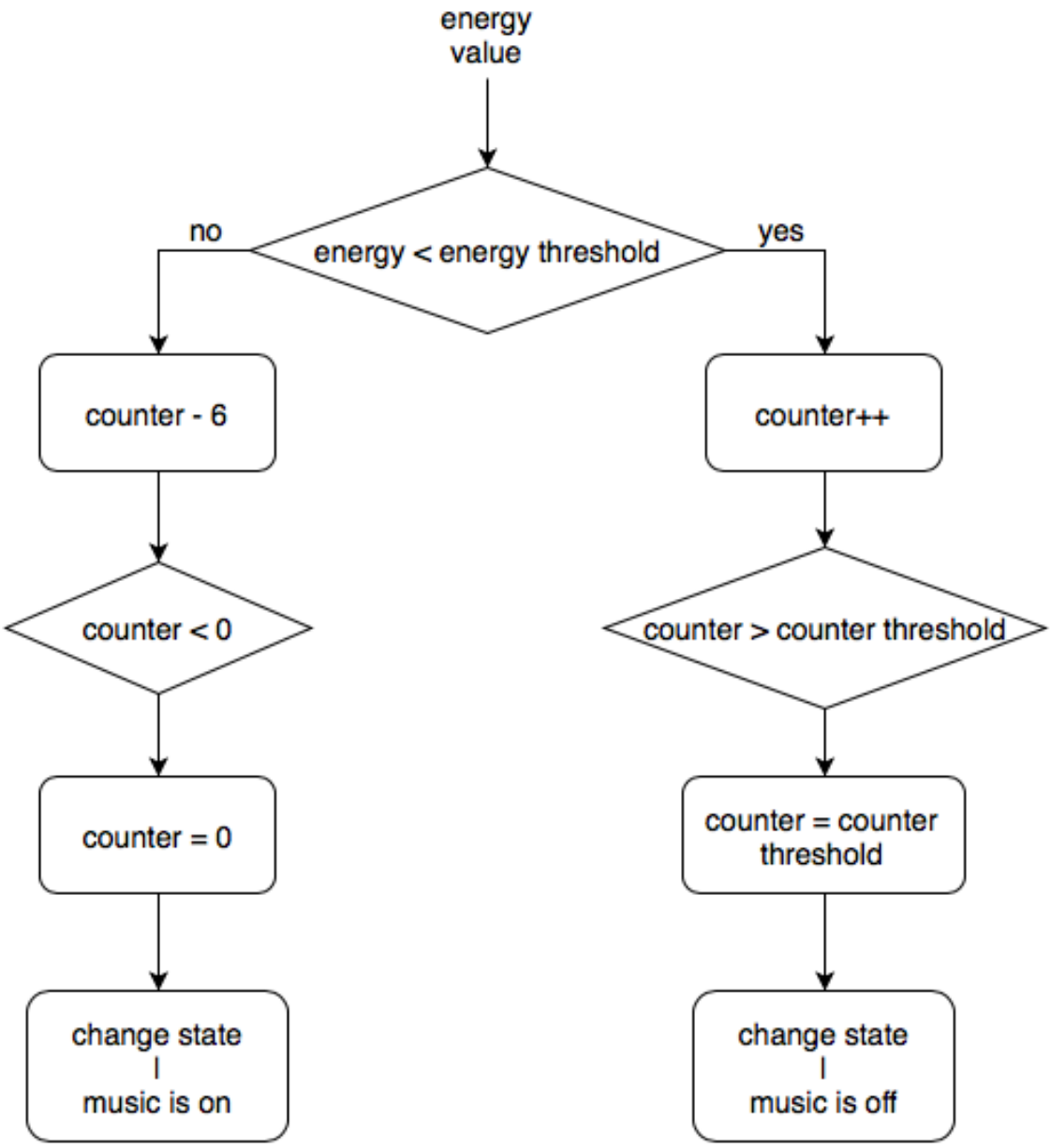


Figure 11: Flowchart of the algorithm used to change state between "music is on" and "music is off".

By preserving state and waiting for a counter and not changing state instantaneously when we detected energy was low, we avoid false alarms since it is very common that a piece of music has short periods of low energy even in more than 4 ms consecutively. The same way, this avoids that transitory noise triggers a state change and, as a consequence, that the system thinks wrongly that it is hearing to music. If the counter threshold is too low, it will not solve the former problem. However, if it is too high, it will not respond quickly to the start

and end of music. Through experimentation with different values, we found that a counter threshold of 150 works fine.

To tell the next blocks that music is off, we send them an onset strength value of -1042, which is an impossible value since onset strength is mainly positive and rarely negative but always near 0 (it can be negative due to the normalisation).

To find the onset strength value at the 4 ms time instant, we first perform the FFT of the samples of the buffer (strictly speaking, we actually perform the STFT since we do it each 4 ms) to divide the spectrum in different bands. Then, for each band, we calculate the difference with the value of the same band at the previous instant and we set it to 0 if the difference is negative. Next, we sum all the differences calculated for each band and we normalise the result with its temporal mean and variance. The result of this process is the onset strength value.

The onset strength is normalised so that it has a stable range of values and it does not vary if the music is softer or louder. For the same reason, if the music has a lot of variance so it is very unstable with lots of highs and lows, an unnormalised onset strength signal would have too many peaks, since it would think each high could correspond to a beat when in reality it is just the nature of the music signal. The normalisation is performed as follows:

$$norm\ onset = \frac{onset - \mu}{\sigma}$$

Where μ is the mean and σ the standard deviation. Since we never have the entire signal so as to perform the mean and standard deviation of it, we use a different definition of the mean and the standard deviation to adapt it to this online process.

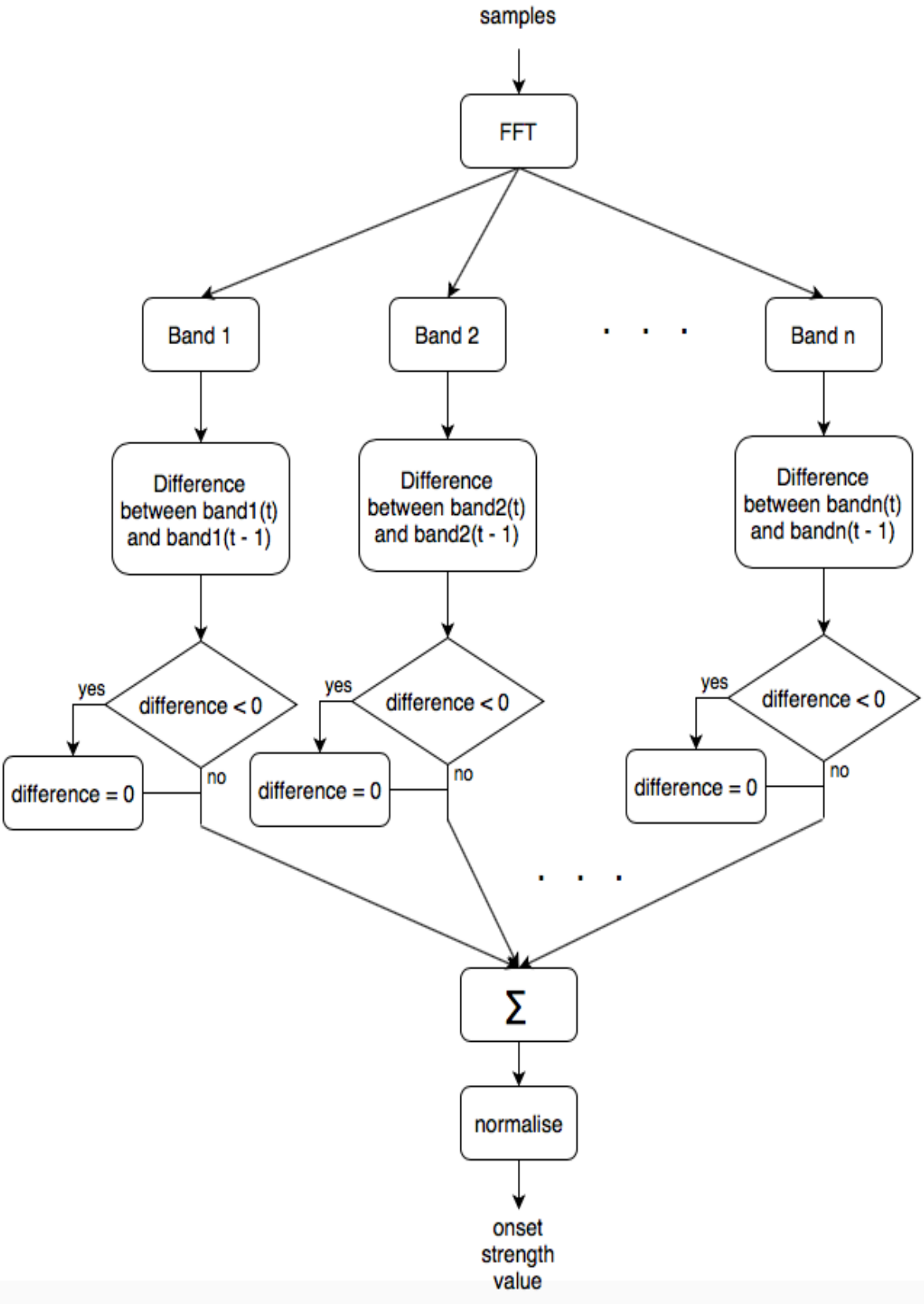


Figure 12: Flowchart of the processing of the microphone signal and extraction of the onset strength.

For the online mean, we just calculate the weighted sum of the mean calculated at the previous instant and the new onset strength value. The weighting affects the adaptability and variation of the online mean with respect to time.

$$online\_\mu[n] = (1 - adaptability) * online\_\mu[n-1] + adaptability * o[n]$$

For the online standard deviation, the concept is the same. We compute the weighted sum of the previous online standard deviation and the newly calculated standard deviation as the absolute value of the difference between the onset strength and the previously calculated online mean.

$$online\_\sigma[n] = (1 - adaptability) * online\_\sigma[n-1] +$$

$$adaptability * |o[n] - online\_\mu[n]|$$

The factor adaptability determines how fast the online mean and online standard deviation will change with a new different input. If the music gets suddenly much louder and the adaptability is high, the online mean will get bigger faster. However, if it is too high, it will have a lot of noise and vary too much with respect to time and the mean should be a relatively constant value. For this reason, in general a low adaptability has better results. In our case, we have used a 0.05 adaptability with good results.

Last of all, the onset strength value is sent to the next two blocks, the music period estimator and the beat tracker, which are both programmes written in C++. The way ChucK has to communicate with other external processes is via OSC, which is explained is section 3.1.2. ChucK supports OSC natively but the C++ programmes have to implement some code to parse the OSC messages.

In our case, since we always just need to send a single float value, we do not need to create a full OSC parsing system. Therefore, the message structure remains the same: it starts by the address pattern, which we have set to “/onset”, a type tag with a single float parameter definition, i.e. “,f”, and the value of the 4-byte float, which is in IEEE 754 single-precision binary floating-point format. As a result, they will have to reject any package which does not start with “/onset” and is not followed by “,f” and the main problem just lies on parsing the float value.

The structure and location of information in the 4-byte block is as follows. The first bit represents the sign of the float: if it is 0, the number is positive, in the opposite case, it is negative. The next 8 bits contain the exponent and the remaining bits hold what is called the *significand, mantissa* or fraction.

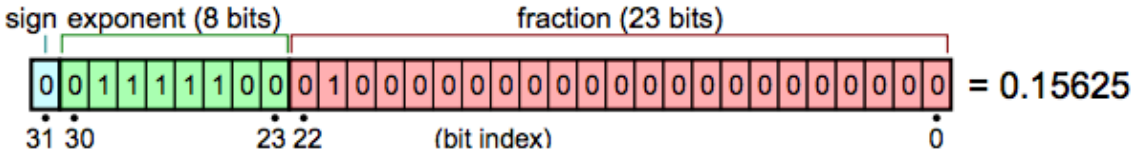


Figure 13: Example of the structure of an IEEE 754 single-precision binary floating-point number (Fresheneesz, 2007).

Finally, to decode the binary block into the actual floating-point number, we perform the following operation:

$$float = (-1)^s \cdot (1 + m \cdot 2^{-23}) \cdot 2^{exp-127}$$

Where *s* is the sign bit, *m* is the significand and *exp* is the exponent.

3.4.2. Music period estimator

The music period estimator receives the onset strength signal real-time from the previous analysis and examines its periodicities to extract the main beat frequency.

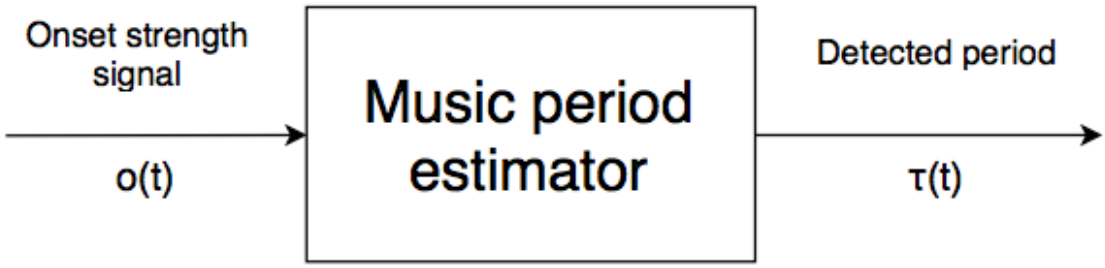


Figure 14: Music period estimator as an input-output block.

The find the period of the incoming onset strength signal, when we receive an onset strength value from the previous block, we first add it to a buffer and check if the music is off. If music is off, we restart the algorithm and otherwise, we continue after waiting to have a minimum amount of onset

strength values in order to be able to do computations. When our buffer is sufficiently loaded, we perform the 4-cycled autocorrelation, we get the three highest values, we add their corresponding periods to the histogram and we decide which period of the histogram would be the most probable period of the music. So as not to send unreliable periods too fast, we wait for the algorithm to stabilise and finally we send the period to the beat tracker. To avoid detecting irrelevant frequencies and to make the algorithm faster, we just search BPM between 40 and 250.

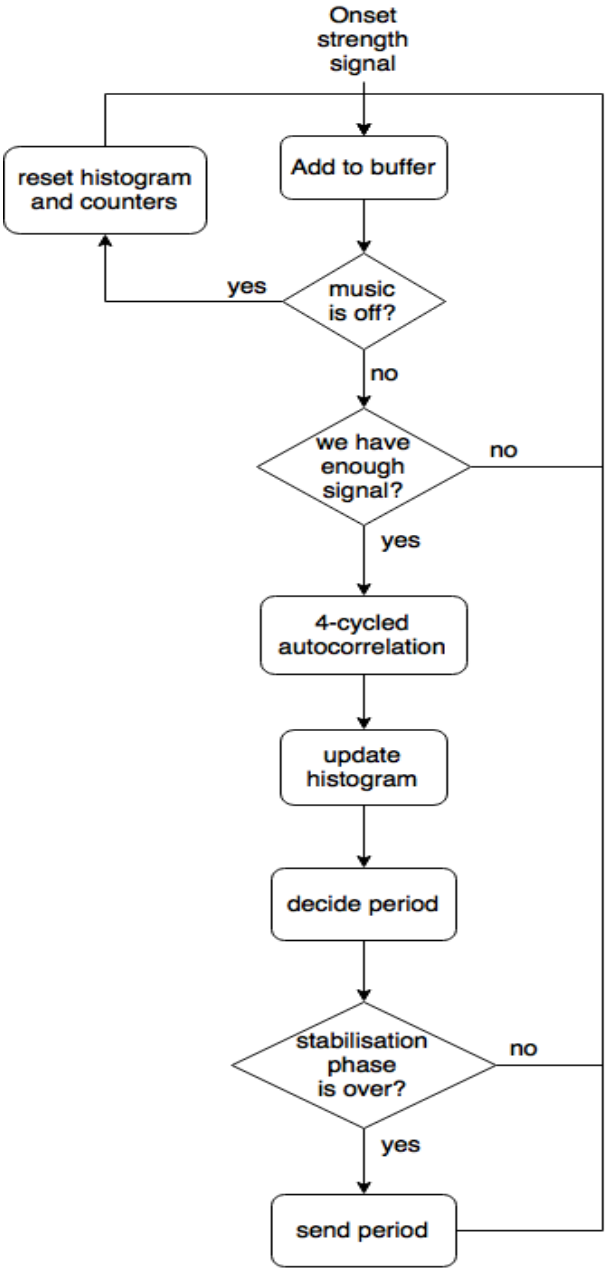


Figure 15: Flowchart of the general algorithm used to estimate the period of the incoming music.

For the stabilisation phase and to check if there are already enough values of the onset strength signal, we use counters. For the former, we just wait until we have received 80 more onset values. For the latter, we wait until we have received at least what corresponds in the time domain to two times the maximum period we want to find (which coincides with the lowest BPM, i.e. 40 BPM). That is, we wait for 750 onset strength values:

$$number\ of\ values\ to\ wait = 2 * \frac{1}{min\ BPM/60} * \frac{1}{receiving\ period} = 750$$

Where min BPM is 40, and the receiving period is the time it takes for a new onset value to be received, the "sampling period" of the onset strength signal, i.e. 4 ms (see section 3.3.1.).

In order to solve some of the disadvantages of the autocorrelation commented in section 2.2.4.3., we do not use the real definition of autocorrelation but a derived one, what has been called the *n*-th cycled autocorrelation. The problem is that if we want to know the weight of a period τ, if this period is a good estimation, the autocorrelation would just give the similarity of the signal which itself displaced one period τ since autocorrelation is one-cycled. But actually, if this period is indeed a good estimation, the signal should not only be similar to itself displaced one period τ, but also to 2τ, 3τ, and so on till $n\tau$. Since we have finite signals, we cannot take this to the infinity but, in any case, since the tempo of music could change with respect to time, we should not make comparisons with too delayed signals.

To spare unnecessary computations, we just compute the *n*-th cycled autocorrelation in the range from the minimum period (250 BPM) to the maximum period (40 BPM). The definition of the *n*-th cycled autocorrelation is:

$$N - th\ cycled\ autocorr\ (\tau) = \sum_{\min \tau}^{\max \tau} \sum_{\varphi=\tau}^{N \cdot \tau} \frac{o(t) \cdot o(t-\varphi)}{N}$$

Where φ is τ, 2τ, 3τ, etc. and N is the maximum number of cycles.

Once we know the weight of each possible period thanks to the *n*-th cycled autocorrelation, we add the best three periods to our histogram. The histogram serves to count real-time the occurrences of each period we detect. If a period has surpassed 500 occurrences, we decrement the counter of each

period by one, avoiding negative values. This is necessary because a new piece of music with another period may begin and if there is no maximum number in the counter of occurrences, it can be very difficult for the new period to surpass the previous one if the robot has listened to the previous piece of music for a long time.

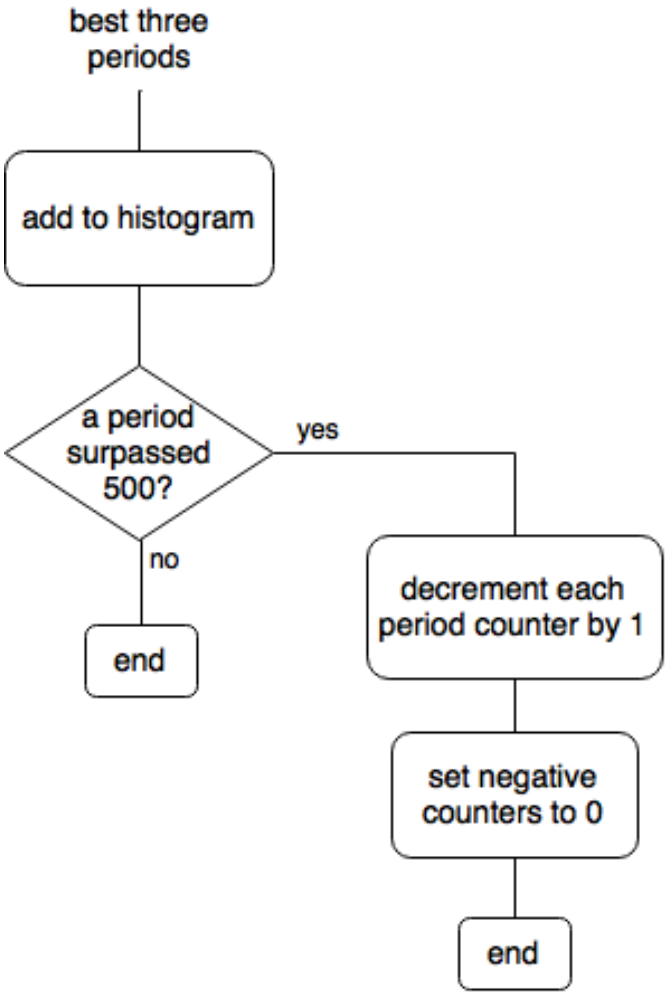


Figure 16: Summary of the use of the histogram when new three periods were detected.

After the last detected periods have been added to the histogram, we decide with the registered occurrences of each period in the histogram, which is the period that has most probability of being correct.

First, we filter out the periods that have less than 25 occurrences and among the remaining ones we get the three with the highest counter value. Then, we see if these periods have a simple relationship between them, that is, if one is double of triple the other one. To do that we divide the periods and we see if the result is near to 2 or ½, or 3 or 1/3. Since music has several periods at

the same time, which follow these relationships, we consider unrelated periods to be outliers, so we discard them. Finally, from the remaining periods, we get the nearest one to 120 BPM, since human perception tends to this rate (see section 2.2.1).

If all three periods were found to be unrelated, at least we compare their occurrences. We get the highest value of occurrences in the current histogram data and then we filter out all periods that have fewer occurrences than half this highest value. If there remains more than one, we get the nearest to 120 BPM.

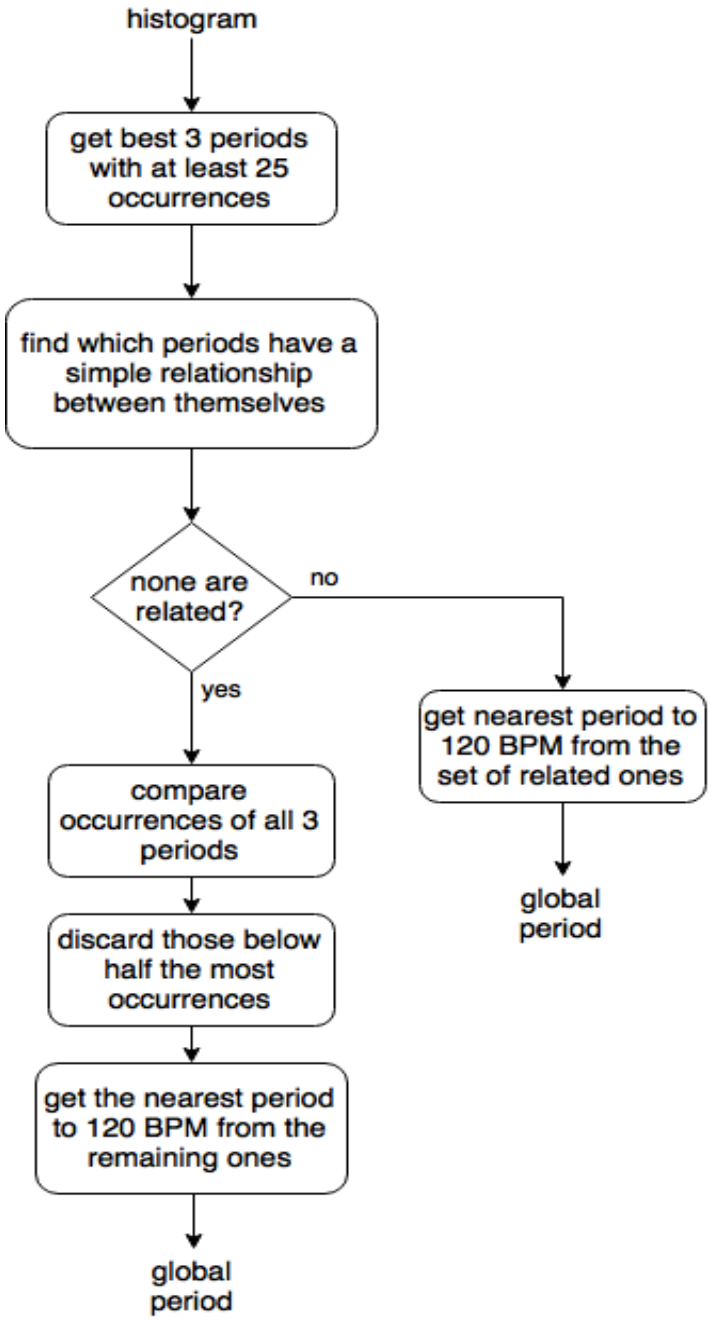


Figure 17: Algorithm to decide the best global period from the histogram data.

When the system decides which period it considers better, once the stabilisation phase has finished, it is sent to the beat tracker via ROS by publishing it to the topic “/music_period”, to which the beat tracker listens.

3.4.3. Beat tracker

With the period alone, we cannot synchronise to the song, we need to know the phase of the beats as well. The beat tracker gets the onset strength signal and the current detected period from the music period estimator and it outputs a beat signal with the frequency corresponding the last detected period and a phase according to the onset strength signal. With the beat signal we will then tell the robot to move at those instants.

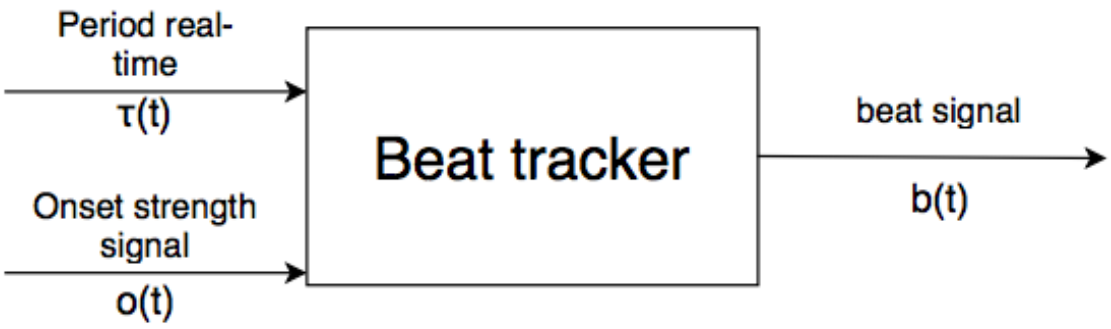


Figure 18: The beat tracker as an input-output system.

If we know reliably at least the time instant of one beat, that would be enough since we could then infer the time instants of the remaining ones with the BPM. Therefore, the way we have used to find the right phase of the beat signal is by looking for beats, in the onset strength signal, which have a much higher probability of being correctly detected. With the position of one beat, we can derive the other ones with the period.

A detected beat that has high probability of really corresponding to the song has two characteristics. It has a minimum onset strength at current time instant and at previous time instants displaced by a whole number of periods from the current instant as well. Like this, we ensured that our beat signal fits better the onset strength signal.

The general flow of the algorithm is as follows. When a new onset strength value arrives, we first check if the music is on to continue or wait for another value otherwise. We then adapt the threshold, which we our going to use to find beats later, to the incoming music if it is needed. Next we use this threshold to check if at the current time instant there is a beat that we can detect reliably, so as to estimate the phase of the beat signal. If not, by counting the time that has passed from the previous beat, we check if a period of time has already passed from the previous beat so a new beat should occur. We will command mini Maggie to move if we found a highly reliable beat or if according to the period we estimate that a beat should happen now.

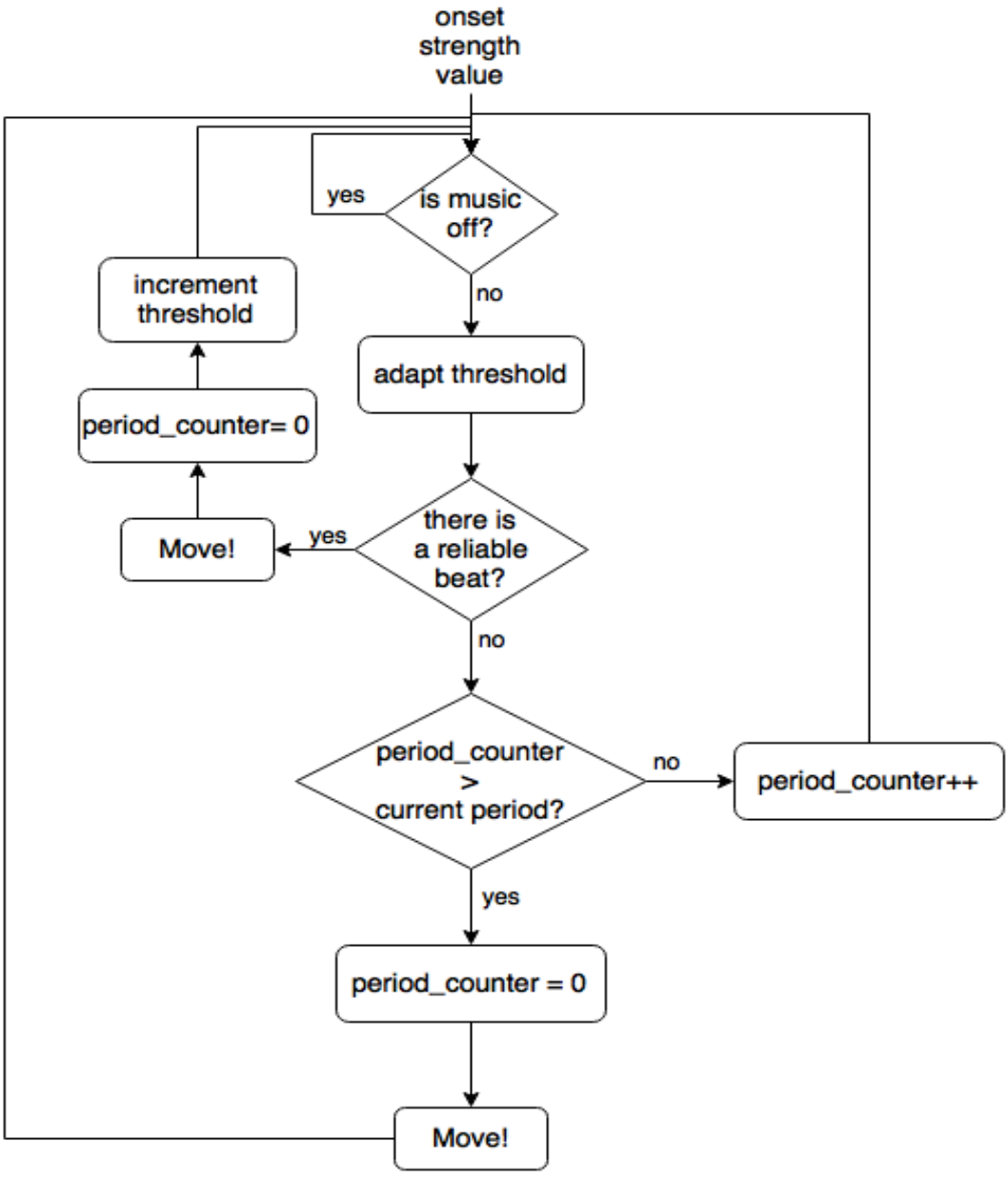


Figure 19: The general flowchart of the beat tracker.

We use a counter that counts the onset strength samples that we have received since the last beat occurred. When this counter surpasses the number of samples that correspond to a period, we reset it and tell the robot to move.

To find out if at current instant there is a reliable beat, we compare the last onset strength value to a threshold and the mean of the onset strength values at the three previous periods by 0.8*threshold. If the current instant satisfies these conditions then that means there is very probably a beat and, therefore, we tell Mini Maggie to move and we restart the period counter, so that we make current instant to correspond to the phase of the beat signal.

To try to make this separately detected beat as reliable as possible, we increment the threshold whenever one such beat was found so that next detected beat has more restrictive requirements and is thus more reliable. We cannot use a very high threshold from the beginning since, depending on the incoming music, that may be too high and that could cause that we may never find this reliable estimate of a beat. For the same reason, this threshold will be lowered if the onset strength signal has much lower values, to adapt to new music since we phase might change with respect to time.

Lastly, for the dance subsystem to know when music is off, in this case the beat tracker will send zeros instead of ones in order to make a distinction.

3.5. Dance subsystem

When the robot knows when it has to move in order to be synchronised with the music, it has to know as well what dance steps it has to perform. The dance controller or dance reader will be in charge of deciding a dance step whenever the dance publisher asks for it. When the beat tracker detects a beat, it will tell the dance publisher that it has to move right now, so the dance publisher will ask the dance controller for a dance step string. Then it will parse it and command specific movements to the dynamixel module of the robot, which automates transitions between different joint states, so we only have to care about the new desired joint angles.

As a summary, the beat tracker tells the dance publisher when it should move, the dance controller tells the dance publisher what movement it should perform, and then the dance publisher will actually execute that movement at that time instant.

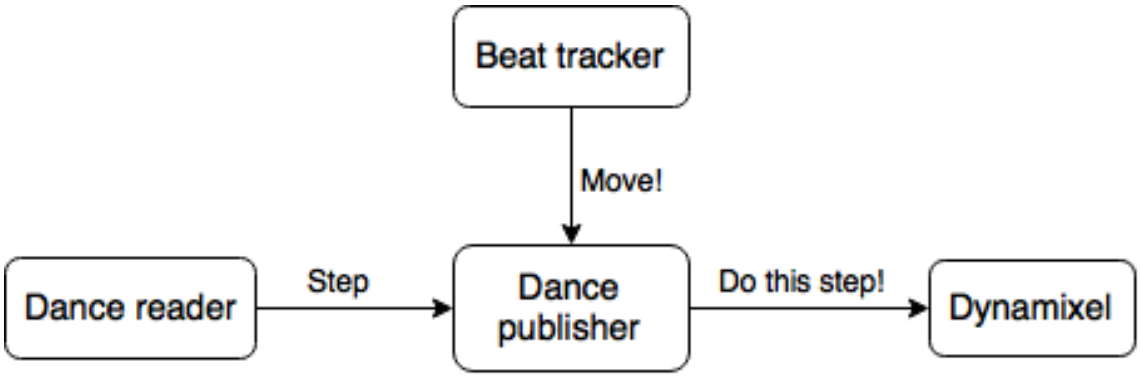


Figure 20: General scheme of the dance subsystem.

A dance is represented as a dance string. A collection of dance strings is stored in each line of a dance file, which the dance controller will read:

```
DANCE STRING 1
DANCE STRING 2
DANCE STRING 3
DANCE STRING 4
DANCE STRING 5
DANCE STRING 6
```

Each dance string is at the same time a sequence of step strings separated by the character '/', so the dance file would have this structure, where the line number would represent the dance number:

```
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8/STEP9/STEP10/STEP11/STEP12
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8/STEP9/STEP10
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8/STEP9/STEP10/STEP11
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8/STEP9/STEP10/STEP11/STEP12
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8/STEP9/STEP10/STEP11/STEP12
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8/STEP9/STEP10/STEP11/STEP12/STEP13
STEP1/STEP2/STEP3/STEP4/STEP5/STEP6/STEP7/STEP8
```

Each step string is represented at the same time by an angle in radians of each joint, separated by the character ';'. In our case, joints have one degree of freedom except from the head, which has two, and Mini Maggie has only four joints (left and right arm, base and head), so we only need four numbers separated by ';'.

As a consequence, in a dance file we have all the necessary information about the exact movements of all the dances. The dance controller will just

need to read a dance and keep control of the last dance step it has sent to the dance publisher in order to send consecutive steps when the dance publisher wishes.

When a dance step string is received, the dance publisher will parse each joint position by splitting the step string with ';' knowing its structure:

```
leftArm;rightArm;base;head1;head2
```

Figure 21: Structure of a dance step string.

Finally, when the desired position of a joint is parsed, it will be sent to the corresponding topics of the dynamixel module for each joint.

Due to the movement limitation of Mini Maggie, it is very difficult to programme choreographies that make the viewer think that she dances according to specific dance styles, such as hip-hop or tango. On the contrary, her choreographies have a general style and each dance is not meant to only fit a specific style but to fit as well as possible a general set of music. Therefore, instead of performing specific dances according to incoming music, a random dance will be performed when Mini Maggie detects a new song.

For this reason, Mini may not dance synchronously with the general feelings that we may perceive from the music. Nevertheless, so that Mini can somehow express minimally her emotions, she is able to express the desire to hear music when she detected music is off. That is done with a random 15 to 25 seconds timer, which is activated when music is off. Then, Mini says a random sentence from a set of previously saved sentences that express this desire for music.

Mini's speech is controlled by the etts package. If we want Mini Maggie to say something, first we have to set some parameters, mainly related to the language we want her to speak. Then, a publisher has to be set up, which sends messages to the topic "etts". The message should be a string with the desired text but it can also contain some annotations that specify how the text can be communicated, such as adding emphasis or making pauses at specific time instants.

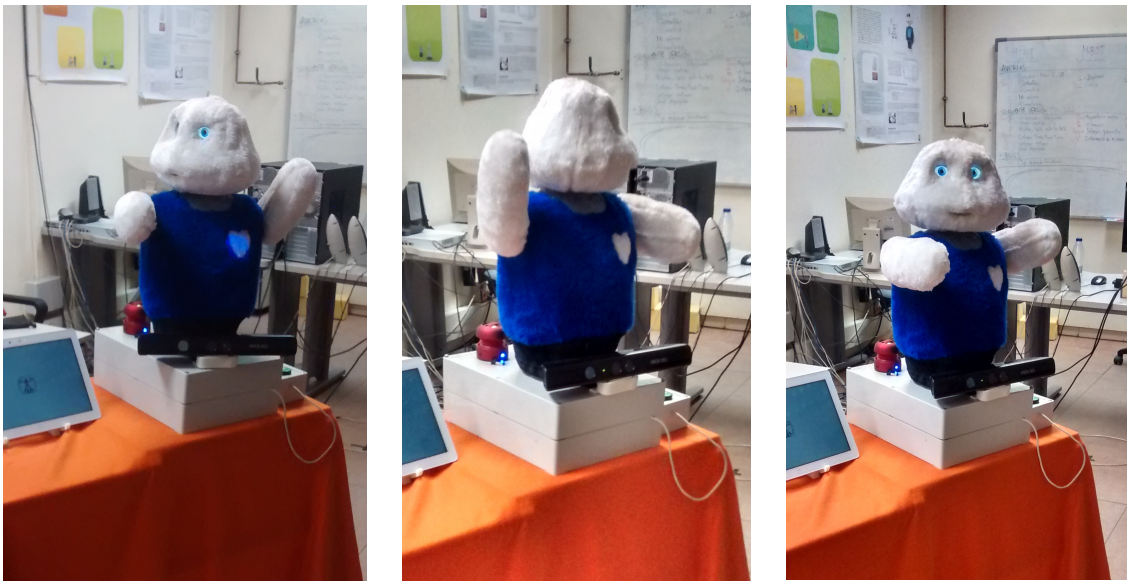


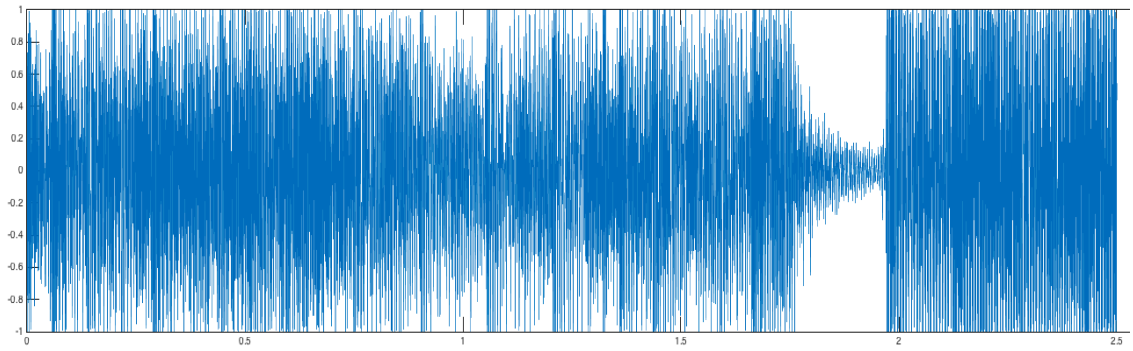
Figure 22: Mini Maggie dancing.

4. Experimentation and results

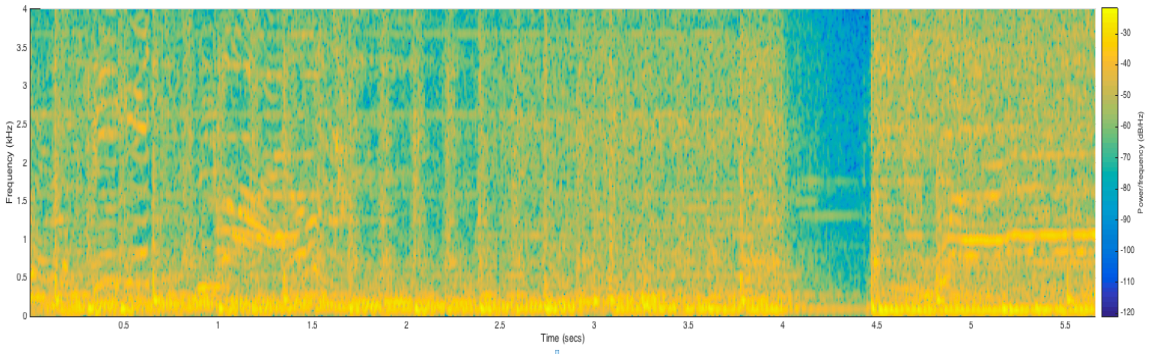
4.1. Onset strength analysis

In this section we are going to examine how is the onset strength signal of our analyser with different kinds of incoming music. For the analysis, all the graphs were plotted using an implementation of our algorithm in MATLAB instead of ChucK language, which is the language used in the original implementation. For simplicity purposes and easier control, the input signal did not come from the microphone but from computer files.

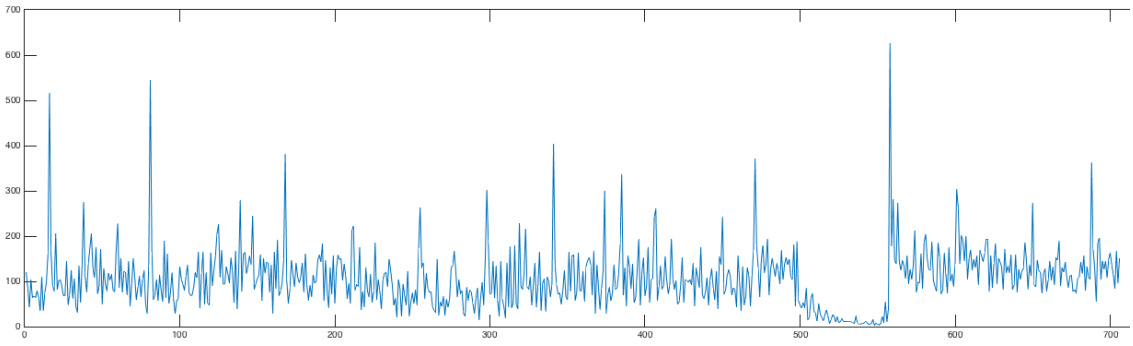
As a first example, we show the effect of each stage of our algorithm on a 5-6 seconds extract of a piece of music in figure 22. We can visually appreciate that the raw microphone signal is not appropriate to be directly used for our beat tracker since it is too noisy. After the FFT, with the spectrogram it is much easier to find several beats in some regions since we separate frequencies and, therefore, we can see each frequency band distinctively. The onset strength is then extracted from the spectrogram, which basically summarises in two dimensions the relevant beat information that we had in the spectrogram in three dimensions. After normalising it, we get a stable range for the onset strength signal, which approximately is from -1 to 1 for lower onset strength values and from 1 to 10 for time instants with higher probability of corresponding to beats.



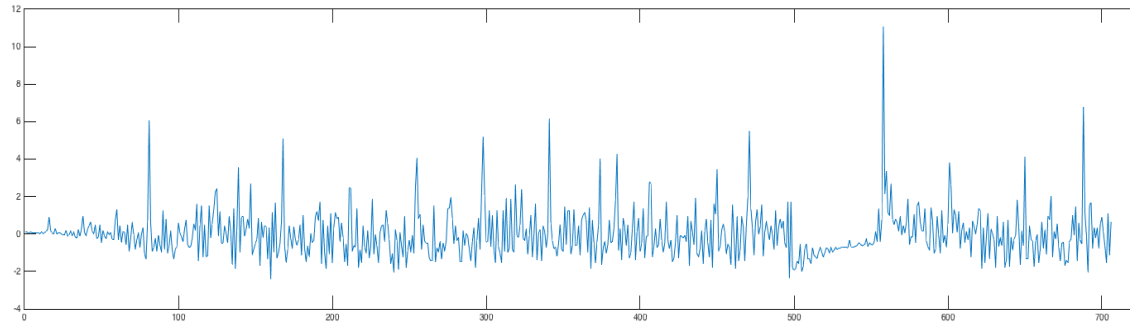
a)



b)



b)



c)

Figure 23: Graphs representing the input signal at different stages of the algorithm. a) contains the original raw signal with no processing. b) is a representation of the STFT, that is, the spectrogram of the signal. c) is the final onset strength signal that results of the processing with this algorithm.

The number of bins of the STFT marks the frequency resolution of the spectrogram and, thus, its clearness. In figure 23 we show the effect of different number of bins in the spectrogram with the same input signal. We can see that with just four bins, frequency resolution is that low that is very difficult to extract the beats. As the number of bins increases, we can observe that beats are much easier to identify but, as a side effect, our algorithm has a higher

computational cost. Since above 64-128 bins the spectrogram does not seem to become clearer anymore, we use 128 bins for our implemented beat tracking system.

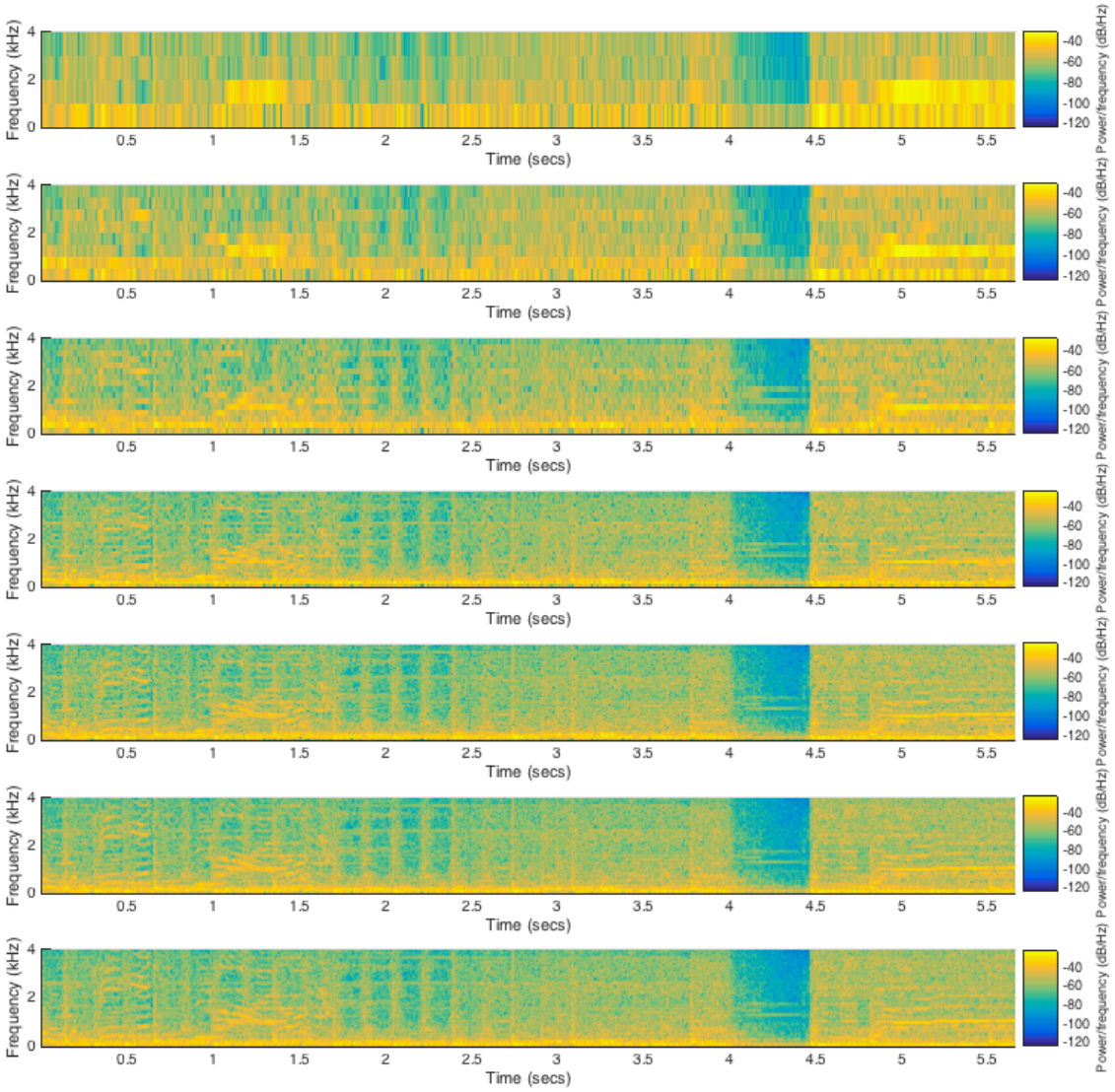


Figure 24: Spectrogram of the same input signal with different number of bins. The number of bins (we just graph the non-repeated bins instead of the really computed bins, which would be doubled) is from top to bottom 4, 8, 16, 32, 64, 128 and 256.

Since we have developed a real-time system, to normalise the onset strength signal we have to calculate the mean and standard deviation online. As we commented in the system description in section 3.3.1., that is when the adaptability comes into place. If adaptability is high, the system will respond quickly to changes in the mean and standard deviation, so they will change more often and vary much more with respect to time. As we see in figure 24

with an adaptability of 20% the onset strength signal is too noisy and varies too much.

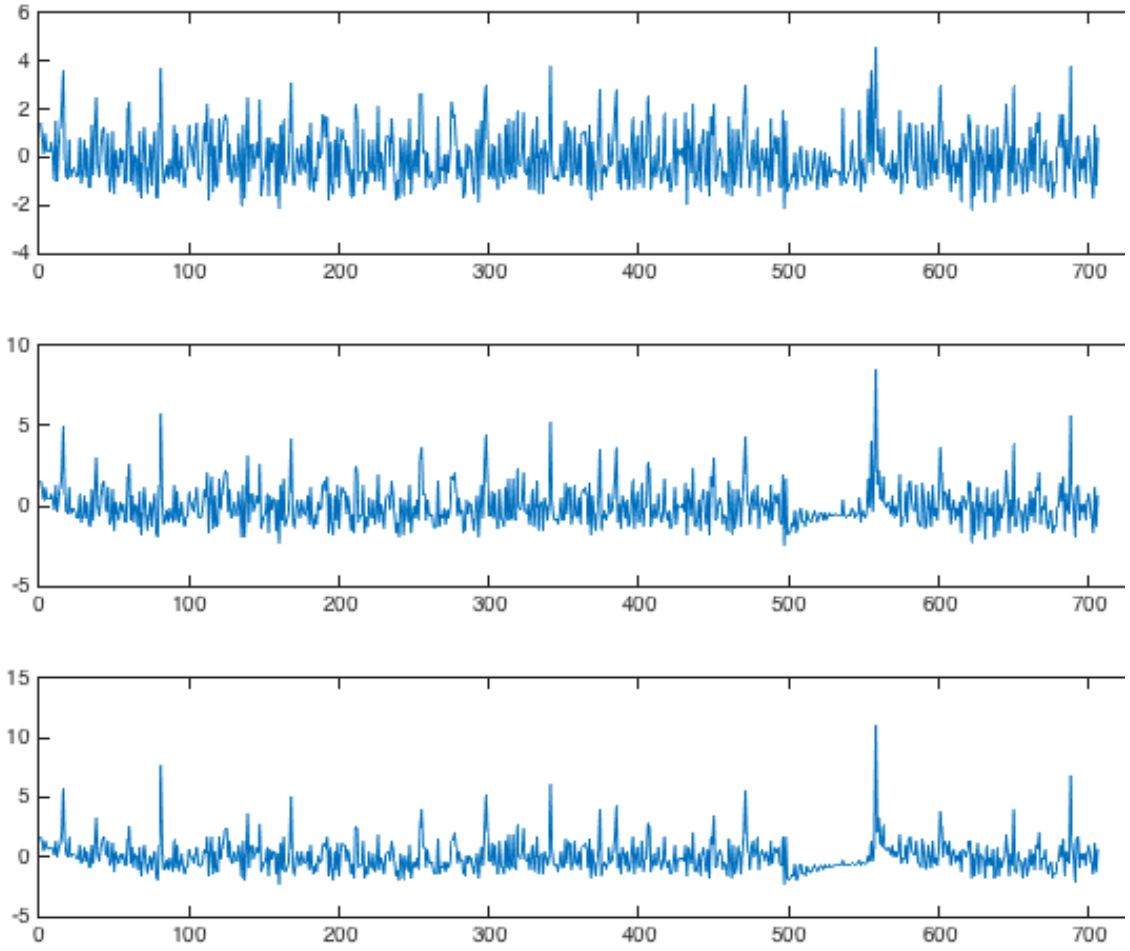


Figure 25: Effect of the adaptability on the normalised onset strength signal. Adaptability goes from top to bottom 20%, 10% and 5%.

Different styles of music have different characteristics that affect the form of the onset strength signal and, as a result, the reliability of the system. Experimenting with different music genres we have concluded that, in general, onset strength signals take three types of form depending on the music style. The one that results in a more reliable system happens generally with popular music, i.e. mainly rock, pop and electronic music. Beats are easy to identify and appear with stable intensity and frequency. Nonetheless, classical music, even if it is a piece of music that seems very rhythmic to our ears, produces a very unclear onset strength signal; beats are not distinct. However, with jazz music we obtain an onset strength signal with clear beat locations, but due to the nature of this music style, peaks do not appear in a stable manner, they are not

equally spaced and their intensity varies with respect to time because of syncopation and the use of more complex rhythmic patterns.

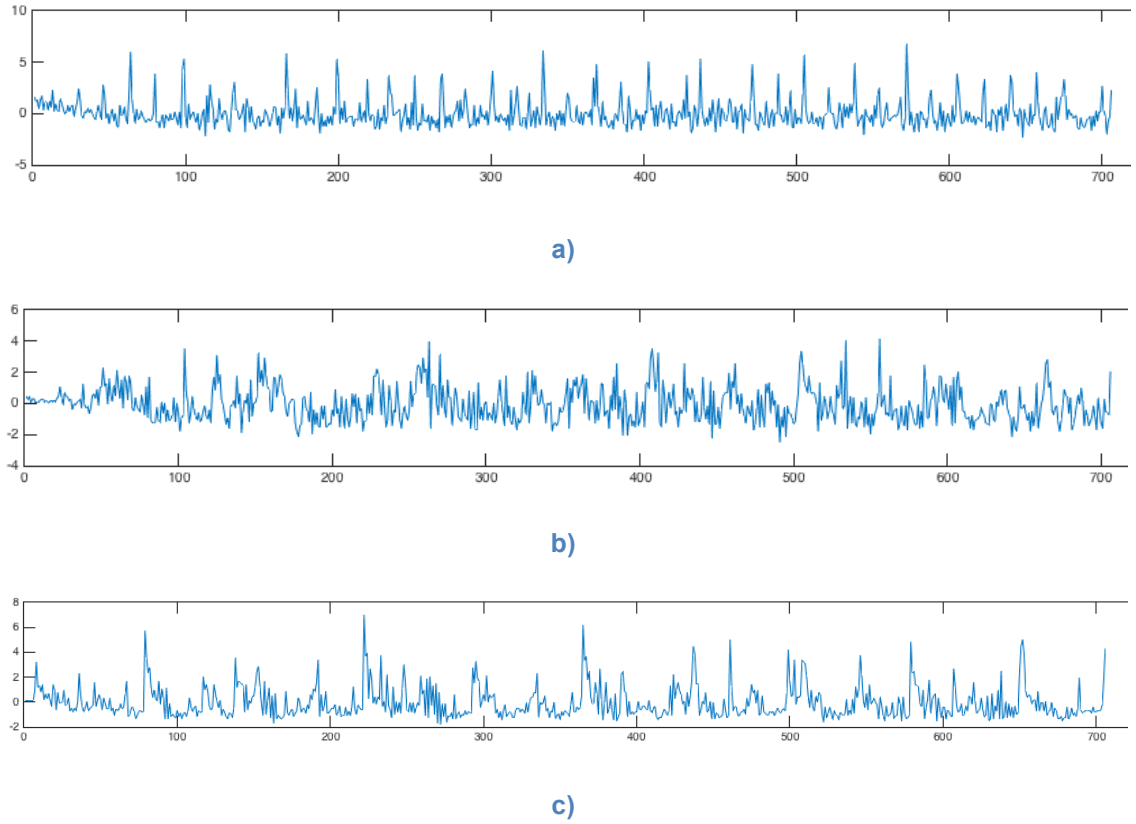


Figure 26: Resulting onset strength signal from extracts of representative pieces of music of different music styles. a) Rock music, b) classical music and c) jazz music.

4.2. Music period estimation

In order to test our music estimator, we have tried to evaluate three parameters: its error rate, its exactitude and its reaction time.

The resulting period estimation can be wrong due to three reasons: because the period of the music has suddenly changed and the estimator has not yet adapted to the new rate; due to inaccuracy in the result, even though it is very similar to the ground truth (if incoming music has a tempo of 156 BPM and our algorithm detects 159 BPM, would you consider the estimation right, wrong or inaccurate?); or just because the estimation was wrong. For this reason, we have separated error from accuracy, that is, we consider very near BPM estimations are right and then we calculate the accuracy as the deviation

of correct estimations from ground truth. Besides, our beat tracker does not necessarily need an exact BPM estimation since inexact BPM estimations just cause an increasing phase difference with the ground truth but our system resynchronises from time to time with incoming music due to phase detection. Nonetheless, accuracy is important and affects the robustness of the beat tracker, so we have evaluated it.

Wrong periods due to a slow adaptation to the new rate of the music will also be considered as right estimations in the error rate evaluation and reaction speed will be evaluated separately later.

We have defined the error rate as the number of wrong estimations (excluding the previously commented cases) divided by the total number of guesses. That coincides with the relation between the total time the music period estimator has been sending wrong estimations of the BPM divided by the total duration of incoming music.

$$error = \frac{wrong\ guesses}{total\ number\ of\ guesses} = \frac{time\ the\ estimator\ has\ been\ wrong}{total\ duration\ of\ music}$$

And accuracy is defined as the mean deviation of right estimations from the ground truth.

$$accuracy = \frac{\sum_i^N |estimation_i - GT_i|}{N}$$

Where $estimation_i$ is the detected BPM at time instant *i* and GT_i it the ground truth at the same time instant (BPM change with respect to time in some pieces).

To evaluate the error rate and the accuracy of our period estimator, we have separated music pieces into the three main music categories that we have differentiated in the previous section (section 4.1): general popular music, jazz music and classical music. We have selected 5 pieces from each category (see appendix for the list of pieces and the results for each song), some of them with varying tempo, and then calculated the error rate and accuracy as explained above.

Classical music has very often a continuously varying tempo due to rubato and other common practices and our algorithm is adapted to find sudden

tempo changes and not continuously varying tempi; therefore, with the choice of classical pieces we have been partially biased since we have rejected pieces that are interpreted with a lo of rubato and expression that vary significantly the tempo too often. As a consequence, there is no music from the Romantic period and we have included more Baroque music since it has rhythmically a more stable nature. Nonetheless, we have still evaluated the error and accuracy of these unstable pieces, but separately.

	Popular	Jazz	Classical	Unstable classical
Error rate	5.51%	9.87%	5.27%	44.20%
Accuracy	±1.75	±1.85	±3.14	±2.88

Figure 27: Rate of wrong estimations and exactitude of the implemented system. The category named "unstable classical" includes pieces from the Romanticism period that have a very unstable tempo.

We can observe that we generally achieve the best results with popular music, both in correct estimation rate and precision. The more complex rhythmic patterns of jazz music makes it harder for the algorithm to estimate correctly the right period, but precision is still acceptable. With classical music that is not too unstable temporarily, we can observe that the problem does not lie in approximately detecting correctly the period, but to be precise in the detection. This is generally due to the less rhythmic nature of this music and due to the practices in the interpretation of classical music, where tempo changes for expressive purposes are very common. If tempo variations are generalised and excessive, as in the "unstable classical" category, our algorithm is not able to estimate correctly the period of incoming music most of the time.

Next, we are going to evaluate how fast our algorithm detects a sudden change of tempo. To do that, we concatenate two pieces of music or use a piece of music that has parts with different tempi. We are going to use the pieces of music of the previous experiment that showed better results, to

separate reaction time evaluation with the evaluation of correct estimation and precision.

To compute reaction time we have repeated the experiment 10 times and for each experiment we counted the number of times the system still estimated the old BPM when a new BPM suddenly appeared and then translated that into time, knowing our system makes an estimation each 0.32 seconds.

	Reaction time
Mean count	15.2
Mean time	4.86 s

Figure 28: Mean reaction time of our algorithm as the number of old estimations and the duration of the detection of the new period.

4.3. Beat tracking

Now we will evaluate how our beat tracker is capable of tracking the beats of a song by synchronising to its phase and using the information of the music period estimator. To do that, we have examined the precision and recall of our system for each song and then, with those two values, we have calculated the F-score, which summarises that into a single value by computing the harmonic mean of the precision and recall:

$$F - score = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

Precision is a measure of the correctly tracked beats in relation to the total number of beats that our algorithm has outputted while recall is the ratio between the number of correctly tracked beats and the total number of correct beats (ground truth).

$$precision = \frac{\{correct\ AND\ detected\}}{\{detected\}}$$

$$recall = \frac{\{correct\ AND\ detected\}}{\{correct\}}$$

Due to the subjectivity of deciding if a detected beat was correctly tracked or not and to make the evaluation independent of changes between different correct periods, we have tried to simplify it to make it more deterministic making some assumptions. First, we will consider that undetected beats happen while period has been incorrectly estimated or if a there is a clear lack of phase synchronisation.

$$undetected\ beats = \{wrong\ period\ OR\ wrong\ phase\}$$

The number of undetected beats based on previous suppositions will contribute to the number of false alarms as well, since for each undetected beat there is a beat that was detected erroneously. However, during the experimentation phase, we realised that every time a phase resynchronisation occurs because the system has found a new highly reliable beat (see section 3.3.3.), our system detects two beats when there is in reality just one, so that will contribute to the number of false alarms as well.

$$false\ alarms = undetected\ beats + num\ phase\ resynchronisations$$

The programme counts by its own the number of beats it has detected and the number of phase resynchronisations. The number of correct beats (ground truth) will then be computed as the number of detected beats minus the number of phase resynchronisations, since phase resynchronisations count twice a correct beat and each incorrectly detected beat corresponds to an undetected beat.

$$num\ correct\ beats = num\ detected\ beats - num\ phase\ resynchronisations$$

Therefore, with all this information we can calculate the precision and recall as follows:

$$precision = \frac{num\ detected\ beats - false\ alarms}{num\ detected\ beats}$$

$$recall = \frac{num\ detected\ beats - false\ alarms}{num\ correct\ beats}$$

With music from the category named “unstable classical” in previous section, our beat tracker had rather random results since our algorithm was not able to track the beats so we have not made this quantitative evaluation.

Besides, ground truth was sometimes very difficult to determine, even by humans and, therefore, the evaluation would be too unreliable.

	Popular	Jazz	Classical
Precision	0.767	0.771	0.770
Recall	0.802	0.815	0.788
F-score	0.784	0.792	0.779

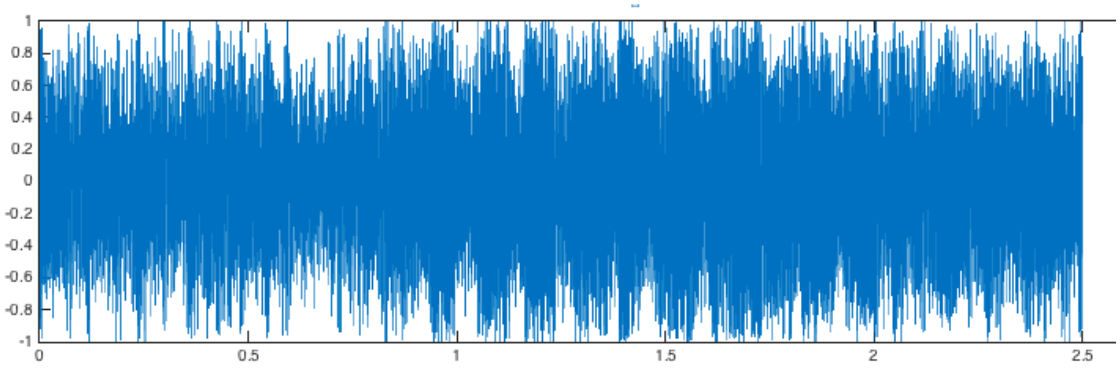
Figure 29: Recall, precision and F-score of our beat tracking system according to three main music categories of music genres.

Results about each particular piece of music are presented in the appendix.

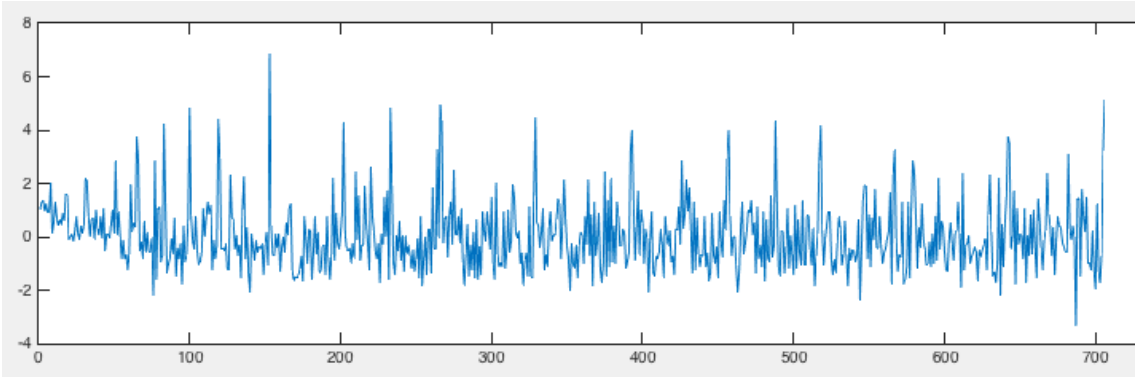
We observe that our beat tracking system generally works well enough if music is not too unstable rhythmically, independently of the music genre. Nevertheless, as already commented, if there is not a stable rhythmic pattern, our algorithm is not able to track beats.

4.4. Dancing

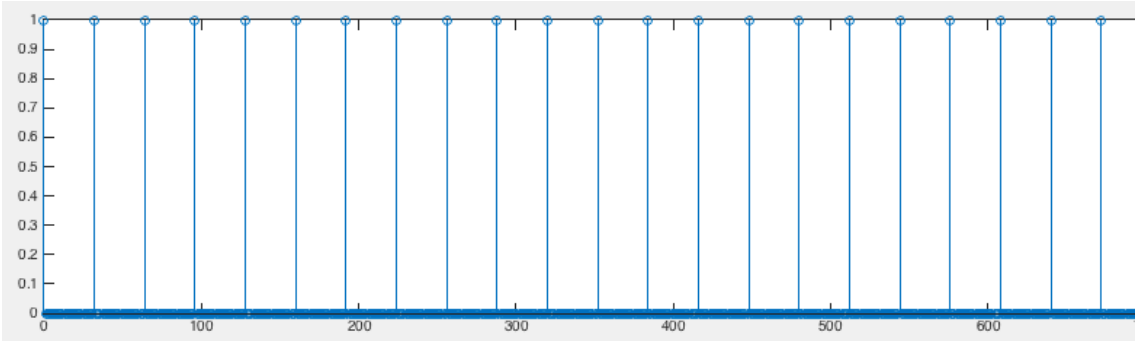
When Mini hears music, she analyses the microphone signal to get the onset strength signal, from which we can get the tempo with the music period estimator and the phase and final beat sequence with the beat tracker (see system description). Then Mini executes a movement (a single dance step) when the beat tracker detected a beat.



a)



b)



c)

d)

Figure 30: Representation of the stages of the whole system from microphone input signal to Mini's dancing. a) Microphone input signal, b) onset strength signal, c) beat sequence, d) Some dance steps of Mini's performed choreography.

To create choreographies and dance steps for the dance database, we experienced two limiting factors while making tests with the robot. On the one hand, the limited degrees of freedom of Mini did not allow to make many interesting choreographies that highly appeal to the viewer. On the other hand, with the choreographies we just decide the desired positions of each dance step at each beat instant but the displacement between consecutive dance

steps and velocity of transition was handed over to the dynamixel module of the robot, which sometimes was too slow if the beat sequence had a high frequency and at the end that made the impression that Mini did not dance along at the rate of the music or some dance steps were only partially executed.

To cope with this issue and make it less visible to the viewer, we first reduced the maximum joint displacement between consecutive dance steps so that dance step transitions could be fully executed with a slower velocity. Nevertheless, by trying so, we noticed that short displacements made the choreographies much less appealing and seem much more basic. For this reason, we introduced again long displacements in the choreographies but when a large displacement was foreseen, another joint would make shorter displacements that would clearly follow the music even when the beat signal had a high frequency. For example, for most written choreographies, the head moves with short displacements whereas the arms make longer movements and, thus, the result is a more appealing choreography that follows exactly the detected beat sequence.

5. Conclusions and future work

In recent years, dancing robots have become much more human and realistic and seem to be able to synchronise with the music very well and perform very flexible dances. Nevertheless, real-time dancing robots have still a long way for research, since music and choreography is not fixed in advanced by humans and, therefore, the robot has to make much more decisions and analysis, thus, making the dance ability more complex.

The processing of music can be a very difficult task, since it usually involves the extraction of high-level features and subjective emotions. For most high-level knowledge, even if it is deterministic information such as chord type, robots seem to have much more trouble than. This deterministic high-level music features also include tempo, time signature, tonality and many others, which robots still have to learn to extract more precisely.

Besides, since we wanted to implement a real-time system with a low delay, we did not have all the information of the song when we needed to make an estimation of the tempo or the beat instants and the microphone signal could consist in many consecutive songs or a song with tempo changes, what made music processing even more complex. Despite these issues, we have been able to create a system that extracts real-time and with a fairly high reliability the tempo of the music, extracted by the music period estimator.

Nevertheless, even if Mini Maggie can dance synchronously with the rhythm of the music, sometimes the user may perceive her not to dance according to the feelings that may arise with song or, for example, even with its intensity, since humans do not dance in the same way with louder and softer songs. The inclusion of a manner to coordinate Mini's dancing with feelings and other features is still to be done in the future. However, to include some liveliness and emotion in the dance ability, Mini Maggie is able to express the desire of having music when she hears none.

The extraction of emotions can be even more complex since we have to convert it to a deterministic process that robots can understand and that usually involves the extraction of many different features and the execution of many

different processes that might be complex as well at the same time. However, humans perceive determinism as the opposite of feelings and that forms a barrier for the research in emotion extraction. Besides, not every human being may feel exactly the same emotions when they listen to a piece of music.

That opens up an issue that has been considered for other abilities of social robots, but not for dancing robots, as far as we are concerned. Should a dancing system have different results in different robots to emulate subjectivity of humans? The effect of the memory of the robot could be introduced to modify the way a robot dances according to the specific experience of the robot, for example, when Mini watches people dancing, she learns new choreographies emulating humans. Not only experience but identity of each robot as well, to dance in a particular manner that makes humans remember this specific robot. That could have striking impact on how human beings perceive and remember Mini Maggie.

If robots may be able to feel in the future, is a question that has aroused many interest in the robotics community, especially in recent years. Investigation currently explores how to make humans think robots respond to moving events and feelings and how they can seem to be compassionate or empathic. However, at the current research stage, algorithms are written to trick people into thinking they really have feelings but developers generally do not deliberately code in order to really “create” feelings because that is currently impossible.

That may always remain impossible as well but I would personally say, since some future technologies have always been regarded as impossible by previous generations, that in the far future a robot, which we could not currently grasp, might be able to feel in a similar way humans do. As technologies are discovered and the world is better understood, some topics previously considered as metaphysics or philosophy become science. Besides, if the real deterministic world of science has been able to create humans with complex feelings and interactions, why wouldn’t it be possible to create another system with similar results?

A manner that some authors have been used to try to create a robot that seems to have similar feelings to us, is to add randomness to its processes like

the Markov chain we explained in the previous work section. Other options may include adding random noise, such as perlin noise, which has been recently used for making robots or characters in computer animations seem to move in a very lively way. Nevertheless, the control of perlin noise movement by dances, synchronising to music, has not been widely explored yet and it could be researched in the future in order to include it for Mini's dancing generation system.

There are other tasks in the field of music processing that are very simple for humans but for robots it seems too confusing. For example, if we listen to two songs at the same time it is very easy for us to identify that there is not just one song and we may identify each song without difficulty as well but that remains challenging for a robot. Besides, if a singer sings very badly, if a piece of music is performed artificially by a computer, we can clearly identify these situations while for a robot to detect that, we may have to implement complex algorithms and these algorithms generally cope with just one of those many issues.

5.1. Integration of the dance ability in Mini Maggie's dialog system

As we explained in this thesis, a robot can perform a varied set of abilities such as playing games, singing, helping the elders, etc., and that is the case of our robot Mini Maggie; she is a baby robot that still learns new abilities. For this reason, all abilities have to be managed by a higher-order system that controls when each one should be executed.

The implemented system is prepared it is possible to activate and deactivate it externally. Therefore, it is adjusted so that a higher-order system is able to control it without making any modifications in the code. However, modifications in the already existing code of the robot have to be made to prepare the robot to accept another ability, in our case the dance ability, and manage it among all the previously existing ones.

Mini Maggie has a dialog system and a status control that manages how different abilities are activated and switched in time. These statuses include some general ones that specify how the robot is reactive to new input: sleeping, ready, etc.; and each ability has an associated status as well, which indicates the ability that is currently being performed.

To switch between different states, a dialog system is generally used so that, for example, when Mini Maggie hears that someone tells her to play a game, she changes to the status “playing”; or if she is told to go to sleep, she changes her status to “sleeping”. The main way to change between different statuses is with this dialog system but it is not the only manner; for example, the robot can wake up from the sleeping status when it is touched.

In order to fully integrate the implemented dance ability to this high-order system, we would need to define a status associated with the dancing ability and manage transitions from and to this status. Since the dialog system works with *Loquendo*, it would be needed to write a set of grammars such as “I want you to dance”, “please, dance” (although it would actually be in Spanish, since that is the language Mini Maggie knows how to speak) and create a recipe in the dialog system in order to recognise the sentence.

References

[1] P. M. Brossier, M. Davies and M. F. McKinney. “2006: Audio Beat Tracking”, *music-ir.org*, 24 July 2006 [online]. Available at: http://www.music-ir.org/mirex/wiki/2006:Audio_Beat_Tracking [Accessed 16 May 2016].

[2] F. Patin. “Beat Detection Algorithms”, *gamedev.net*, 7 July 2003 [online]. Available at: http://www.gamedev.net/page/resources/_/technical/math-and-physics/beat-detection-algorithms-r1952 [Accessed 18 May 2016].

[3] F. Scholkmann, J. Boss and M. Wolf. “An Efficient Algorithm for Automatic Peak Detection in Noisy Periodic and Quasi-Periodic Signals”, *mdpi.com/journal/algorithms*, vol. 5, issue 4, pp. 508-603, November 2012 [online]. Available at: http://www.mdpi.com/1999-4893/5/4/588 [Accessed 18 May 2016].

[4] J. D. Touch. “A Statistical Method for Detecting Peaks in Electrocardiogram Signals”, *isi.edu/touch/pubs*, December 1986 [online]. Available at: http://www.isi.edu/touch/pubs/tri.pdf [Accessed 18 May 2016].

[5] M. I. Sezan. “A peak detection algorithm and its application to histogram-based image data reduction”, *Computer Vision, Graphics, and Image Processing*, vol. 49, issue 1, pp. 36-51, January 1990.

[6] K. S. Fu. Chapter 1. “Introduction to Syntactic Pattern Recognition”, in *Syntactic Pattern Recognition, Applications*. Berlin: Springer-Verlag, 1977, pp. 1-31.

[7] G. Palshikar. “Simple Algorithms for Peak Detection in Time-Series”, *researchgate.net*, January 2009 [online]. Available at: https://www.researchgate.net/publication/228853276_Simple_Algorithms_for_Peak_Detection_in_Time-Series [Accessed 18 May 2016].

[8] K. Jensen and T. H. Andersen. “Real-time beat estimation using feature extraction”, in *Proc. Computer Music Modeling and Retrieval Symposium, Lec- ture Notes in Computer Science*, 2003.

[9] A. Divakaran. Signals and Communication Technology, 1st ed. Princeton (New Jersey): Springer 2009.

[10] G. Peeters. “A Large Set of Audio Features for Sound Description (Similarity and Classification) in the CUIDADO Project”, Ircam, France, 23 April 2004 [online]. Available at:

http://recherche.ircam.fr/anasyn/peeters/ARTICLES/Peeters_2003_cuidado audiofeatures.pdf [Accessed 20 May 2016].

[11] P. Masri, A. Bateman. “Improved Modelling of Attack Transients in Music Analysis-Resynthesis”, *Digital Music Research Group*, University of Bristol, 1996 [online]. Available at: http://hans.fugal.net/comps/papers/masri_1996.pdf [Accessed 20 May 2016].

[12] A. Klapuri. “Sound Onset Detection by Applying Psychoacoustic Knowledge”, *Signal Processing Laboratory, Tampere University of Technology*, 1999 [online]. Available at: http://www.cs.tut.fi/sgn/arg/music/icassp99.pdf [Accessed 20 May 2016].

[13] E. Zwicker, “Subdivision of the Audible Frequency Range into Critical Bands", *The Journal of the Acoustical Society of America*, vol. 33, issue 2, pp. 248-248, 27 September 1961.

[14] Stevens, Stanley Smith; Volkmann; John; & Newman, Edwin B. (1937). “A Scale for the Measurement of the Psychological Magnitude Pitch”, *Journal of the Acoustical Society of America*, vol. 8, issue 3, pp. 185–190, 4 August 1936.

[15] T. Jehan. “Event-Synchronous Music Analysis/Synthesis”, *Proc. of the 7tth Int. Conference on Digital Audio Effects*, Naples, 5-8 October 2004.

[16] D. P.W. Ellis. “Beat Tracking by Dynamic Programming”, *LabROSA*, Columbia University, New York, 16 July 2007 [online]. Available at: https://www.ee.columbia.edu/~dpwe/pubs/Ellis07-beattrack.pdf [Accessed 21 May 2016].

[17] N. Collins. “A Comparison of Sound Onset Detection Algorithms with Emphasis on Psychoacoustically Motivated Detection Functions”, presented at *the 118th Convention of the Audio Engineering Society,* Barcelona, Spain, 28-31 May 2005 [online]. Available at: http://community.dur.ac.uk/nick.collins/research/comparison.pdf [Accessed 22 May 2016].

[18] Acoustics -- Normal Equal-Loudness-Level Contours, ISO 226:2003.

[19] M. Goto, Y. Muraoka. “A Real-Time Beat Tracking System for Audio Signals” ”, *School of Science and Engineering*, Waseda University, Tokyo, Japan [online]. Available at:

https://staff.aist.go.jp/m.goto/PAPER/ICMC95goto.pdf [Accessed 22 May 2016].

[20] M. Goto, Y. Muraoka. “A Real-Time Beat Tracking for Drumless Audio Signals: Chord Change Detection for Musical Decisions”, *School of Science and Engineering*, Waseda University, Tokyo, Japan, 29 December 1997 [online]. Available at: https://staff.aist.go.jp/m.goto/PAPER/SPECOM1999goto.pdf [Accessed 22 May 2016].

[21] A. Lukin, J. Todd. “Adaptive Time-Frequency Resolution”, presented at *the 120th Convention of the Audio Engineering Society*, Paris, France, 20-23 May 2006 [online]. Available at: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.6362&rep=rep1&type=pdf [Accessed 25 May 2016].

[22] G. Tzanetakis, G. Essl, P. Cook. “Audio Analysis using the Discrete Wavelet Transform”, Computer Science Department, Princeton (New Jersey), 2001 [online]. Available at: http://soundlab.cs.princeton.edu/publications/2001_amta_aadwt.pdf [Accessed 25 May 2016].

[23] W. Jenkins, A. W. Hull, J. C. Strait, B. A. Schnaufer, Xiaohui Li. *Advanced Concepts in Adaptive Signal Processing*, 1st ed. New York: Springer, 1996.

[24] W. H. Press, S. A. Teukolsky, W. T. Vetterling, B. P. Flannery. *The Art of Scientific Computing*, 3rd ed. England: Cambridge University Press, 2007.

[25] D. Eck. “A Tempo-Extraction Algorithm Using an Autocorrelation Phase Matrix and Shannon Entropy”, *Department of Computer Science*, University of Montreal, 2005 [online]. Available at: http://www.music-ir.org/mirex/abstracts/2005/eck.pdf [Accessed 28 May 2016].

[26] D. Böck, F. Krebs, G. Widmer. “Accurate Tempo Estimation based on Recurrent Neural Networks and Resonating Comb Filters”, presented at *Proceedings of the 16th ISMIR Conference*, Málaga, Spain, 26-30 October, 2015 [online]. Available at: http://ismir2015.uma.es/articles/196_Paper.pdf [Accessed 28 May 2016].

[27] E. D. Scheirer. “Tempo and Beat Analysis of Acoustic Musical Signals”, *The Journal of the Acoustical Society of America*, 1998 [online]. Available at:

http://www.iro.umontreal.ca/~pift6080/H09/documents/papers/scheirer_jasa.pdf [Accessed 28 May 2016].

[28] H. Cheng, S. Harris. “Tempo Estimation and Manipulation”, *Stanford University*, California, 2015 [online]. Available at: http://web.stanford.edu/class/ee264/projects/EE264_w2015_final_project_cheng_harris.pdf [Accessed 28 May 2016].

[29] G. Wang. *The ChucK Audio Programming Language: A Strongly-timed and On-the-fly Environ/mentality*. PhD Thesis. New Jersey: Princeton University, September 2008 [online]. Available at: http://www.cs.princeton.edu/~gewang/thesis.pdf [Accessed 1 June 2016].

[30] G. Wang. ChucK: Strongly-timed, Concurrent and On-the-fly Music Programming Language. *ChucK language main webpage*, 2002 [online]. Available at: http://chuck.cs.princeton.edu/ [Accessed 1 June 2016].

[31] G. Wang, P. R. Cook. “ChucK: a Concurrent, On-the-fly Audio Programming Language”, presented at *Proceedings of the 2003 International Computer Music Conference*, Singapore, 2003 [online]. Available at: http://soundlab.cs.princeton.edu/publications/chuck_icmc2003.pdf [Accessed 1 June 2016].

[32] A. Freed and M. Wright. OSC an Enabling Encoding for Media Applications. *OSC protocol main webpage* [online]. Available at: http://opensoundcontrol.org/ [Accessed 1 June 2016].

[33] ROS community. ROS documentation. *ROS wiki* [online]. Available at: http://wiki.ros.org [Accessed 2 June 2016].

[34] Open Source Robotics Foundation. *ROS main website* [online]. Available at: http://www.ros.org [Accessed 2 June 2016].

[35] Willow Garage. *Willow Garage main website* [online]. Available at: https://www.willowgarage.com/ [Accessed 2 June 2016].

[36] A. Gellius. *The Attic Nights*, translation into English by W. Beloe. London: The University of Michigan Libraries.

[37] G. Cox. “A History of Robotics: Da Vinci's Mechanical Knight”, *blog.salvius.org*, 1st May 2014 [online]. Available at: http://blog.salvius.org/2014/01/a-history-of-robotics-da-vincis.html [Accessed 6 June 2016].

[38] BBC. “Bristol's Robot Tortoises Have Minds Of Their Own”, *BBC newsreel*, 1950.

[39] Honda Motor Co. “History. Robot Development Process”, *Honda Worldwide Site*, 2016 [online]. Available at: http://world.honda.com/ASIMO/history [Accessed 6 June 2016].

[40] The Economist, “Better than people”, *The Economist*, 20th December 2005 [online]. Available at: http://www.economist.com/node/5323427 [Accessed 6 June 2016].

[41] T. Mizumoto, R. Takeda, K. Yoshii, K. Komatani, T. Ogata, H. G. Okuno. “A Robot Listens to Music and Counts Its Beats Aloud by Separating Music from Counting Voice”, Presented at: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 22-26 September 2008.

[42] K. Murata, K. Nakadai, K. Yoshii, R. Takeda, T. Torii, H. G. Okuno, Y. Hasegawa, and H. Tsujino. “A Robot Uses Its Own Microphone to Synchronize Its Steps to Musical Beats While Scatting and Singing”, Presented at: *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Nice, France, 22-26 September 2008.

[43] J. L. Oliveira, G. Ince, K. Nakamura and K. Nakadai. “Online Audio Beat Tracking for a Dancing Robot in the Presence of Ego-Motion Noise in a Real Environment”, Presented at: *2012 IEEE International Conference on Robotics and Automation*, RiverCentre, Saint Paul, Minnesota, USA, 14-18 May 2012.

[44] G. Kim, Y. Wang and H. Seo. “Motion Control of a Dancing Character with Music”, *6th IEEE/ACIS International Conference on Computer and Information Science*, 2007.

[45] S. Nakaoka, S. Kajita and K. Yokoi. “Intuitive and Flexible User Interface for Creating Whole Body Motions of Biped Humanoid Robots”, *The 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 18-22 October 2010.

[46] G. Xia, R. Dannenberg, J. Tay and M. Veloso. “Autonomous Robot Dancing Driven by Beats and Emotions of Music”, Presented at: *Proceedings of the 11th International Conference on Autonomous Agents and Multiagent Systems*, Valencia, Spain, 4-8 June 2012.

Appendix – Data used for the evaluation and specific results of each song

JAZZ MUSIC CATEGORY

1. “Little Lily Swing”, Tri-Tachyon.
2. “As Time Goes By”, H. Person.
3. “Boogies Blues”, D. Gaynor.
4. “Corcovado”, A.C. Jobim.
5. “Amado Mio”, D. Fisher and A. Roberts.

POPULAR MUSIC CATEGORY

1. “Take Me”, JiKay & MNKN ft. Gaby Henshaw.
2. “Spaceships”, AREA21.
3. “Billboard Killer”, Cheap Talk.
4. “We’re all to blame”, Sum 41.
5. “Hung up”, Madonna.

CLASSICAL MUSIC CATEGORY

1. “Eine Kleine Nachtmusik”, W. A. Mozart.
2. “Concerto Grosso in G Minor, 4th movement”, A. L. Vivaldi.
3. “Brandenburg Concerto No. 3 in G major, 1st movement”, J. S. Bach.
4. “String Quintet in E major G.275”, L. Boccherini.
5. “Messiah - Hallelujah”, G. F. Haendel.

JAZZ MUSIC CATEGORY

	1	2	3	4	5
Music period estimation					
Error rate	0.23%	42.95%	0.00%	3.20%	2.97%
Accuracy	±0.43	±2.13	±2.01	±1.52	±3.16
Beat tracker					
Detected beats	280	412	524	419	512
Phase resynchs	37	23	8	17	8
Not detected	8	158	102	53	79
Precision	0.839	0.561	0.790	0.833	0.830
Recall	0.967	0.594	0.802	0.868	0.843
F-Score	0.899	0.577	0.797	0.850	0.837

POPULAR MUSIC CATEGORY

	1	2	3	4	5
Music period estimation					
Error rate	2.77%	1.93%	7.30%	15.57%	0.00%
Accuracy	±1.20	±0.96	±1.88	±3.45	±1.25
Beat tracker					
Detected	237	221	174	119	441

beats					
Phase resynchs	9	4	5	10	18
Not detected	46	46	33	29	28
Precision	0.776	0.774	0.718	0.672	0.896
Recall	0.806	0.788	0.750	0.734	0.934
F-Score	0.790	0.781	0.734	0.702	0.914

CLASSICAL MUSIC CATEGORY

	1	**2**	**3**	**4**	**5**
Music period estimation					
Error rate	1.23%	11.15%	1.85%	2.06%	10.08%
Accuracy	±4.63	±2.52	±2.84	±2.45	±3.24
Beat tracker					
Detected beats	187	286	584	250	379
Phase resynchs	7	4	5	9	8
Not detected	48	45	124	41	94
Precision	0.706	0.829	0.779	0.807	0.731
Recall	0.733	0.840	0.786	0.836	0.747

F-Score	0.719	0.835	0.782	0.821	0.739