# Bachelorthesis

Marian Tietz

A Concept Study for the Measurement of Musical Similarity Based on Audio Signal Features

Marian Tietz

# A Concept Study for the Measurement of Musical Similarity Based on Audio Signal Features

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung

im Studiengang Angewandte Informatik
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer : Prof. Dr. Wolfgang Fohl
Zweitgutachter : Prof. Dr. Andreas Meisel

Abgegeben am 25.03.2013

**Marian Tietz**

## Thema der Bachelorthesis

Konzeptstudie zur Messung von Musikähnlichkeit basierend auf Audiosignaleigenschaften

## Stichworte

Musikähnlichkeit, Audiosignalanalyse, Auditory Image Model, Psychoakustik, Umfrage zu Musikähnlichkeit

## Kurzzusammenfassung

Diese Thesis untersucht, basierend auf dem aktuellen Stand der Wissenschaft, neue Wege zur Messung von Ähnlichkeit zwischen Musikstücken. Zur Verifizierung dieser Methoden werden Referenzdaten von Menschen benötigt. Aus diesem Grund wurde eine Umfrage durchgeführt, welche aussagekräftige Resultate, sowie eine intuitive Benutzerschnittstelle bietet. Einige der vorgeschlagenen Distanzverfahren wurden unter Verwendung von Zeichenketten-Umwandlungs-Distanzverfahren umgesetzt. Mit Hilfe des Auditory Image Models, welches den menschlichen Hörprozess simuliert, konnte die Aussagekraft der Distanzverfahren verbessert werden.

**Marian Tietz**

## Title of the paper

A Concept Study for the Measurement of Musical Similarity Based on Audio Signal Features

## Keywords

Musical similarity, audio signal analysis, Auditory Image Model, psychoacoustics, musical similarity survey

## Abstract

This thesis investigates new ways to measure similarity, or distances, between musical pieces based on current research. Doing this also requires retrieving reference data to test the results against. Therefore, a survey was conducted, which lead to suitable results and a new intuitive interface for acquiring reference data in larger scales. A number of new distance measurement methods were proposed and some of them were implemented based on string edit distances with the help of the Auditory Image Model, which provides psychoacoustic features by simulating the human ear.

# Contents

# 1 Introduction

Some people may say that music is the closest that human beings come to expressing our understanding of the natural complexities around us. They may even claim that music is the most abstract of the arts because it has no meaning or purpose other than to be itself (Adams, 1987). Regardless of that, one can not deny that music plays an important role in our everyday life and is, in combination with the Internet, ubiquitous and manifold. A huge amount of new music is circulating in the Internet everyday, by new artists and established ones.

Because the huge amount of variety in music is impossible to manage for a single person, tools were invented to categorize and weight tracks by several social and technical aspects like their genre and popularity. One of the most popular approaches to categorization is the use of the users themselves, who associate similar sounding songs while hearing them. But as it is often the case, opinions diverge, especially in taste of music. So what the crowd says about a piece of music is not necessarily the same as what a single person would say about the same track, even if there is a consensus for music classification in general.

In the real world, for example in a record store, the situation is different. The staff listens to music and is building their very own, predictable opinion about pieces of music. As a consequence, their recommendations are much more precise in reference to the individual. One goal of the Music Information Retrieval community is to provide a system which does the same as the record store staff, but personalized and over a much larger database of songs. A small step towards realization of such a task is to start measuring similarities between musical pieces, or, measuring distances between them and thus measuring dissimilarities. This thesis builds upon the works of (Ness et al., 2011) and others to search for effective distance measurements between musical pieces. Additionally, and, as a consequence of the lack of ground truth data, a survey was conducted to test whether there is a consensus of musical similarity and to gather reference data for distance method implementations.

This thesis is structured in five parts which build upon each other. While this chapter gives an introduction, explaining the motivation, the following chapter introduces the actual problem. It analyzes the problem of musical similarity, possible use cases for working solutions, documents the execution of the conducted survey and proposes new distance measurement features. The implementation chapter discusses the implementation details and measurements for some of the proposed distance methods, as well as the tool chain used to implement said methods. Summarizing the measurements, the results chapter presents the test results of the implemented features and compares the distance methods to the survey data. As there are several tasks which are still to do, as well as new tasks which emerged during the implementation and testing, the future work chapter documents these. Finally, the conclusion chapter summarizes the things done, encountered problems and lessons learned.

# 2 Analysis

As already mentioned in the introduction, this thesis emphasizes on the measurement of similarity between musical pieces. Musical similarity can be defined in many ways, one example would be to define musical similarity using melody only, i.e., when both songs have the same melody, they are equal. In this thesis, **musical similarity** is defined as a measurement between two musical pieces that is proportional to the perception of the similarity a person perceives between these two. This similarity can be translated to a distance, which then resembles a measurement for **dissimilarity**. This thesis operates mainly on distances instead of similarities.

From the similarity definition follows, that human perception is the reference for musical similarity. Therefore it is necessary to acquire ground truth data from humans and to test whether the definition can be expanded to a broader audience than just a single human. For this reason, a survey was conducted which is documented in section 2.2 of this chapter.

Before the topic of similarity and distance measurement is expanded further, the first section of this chapter discusses use cases and implications for a working distance measurement. After that follows the survey documentation and evaluation. The actual definition of new distance methods is introduced by subsection 2.3.1, which discusses the general approach to these distance measurements, while the following section, section 2.4, describes them in detail.

## 2.1 Use Cases

The introduction of this thesis already discussed music recommendation as a popular use case for musical similarity or musical distance measurements and indeed more and more services today rely on musical or audio similarity measures and music information retrieval. With the rise of on-demand audio streaming services and social music sites, as well as online music shops, the need for recommendation systems arises. Online shops want to recommend similar music to their customers to sell more music. Audio streaming services may generate playlists based on the similarity of the songs. Artists could benefit from music recommendation as well: with a good, balanced musical similarities measurements, the long-tail effect (Celma, 2010) could be compensated.

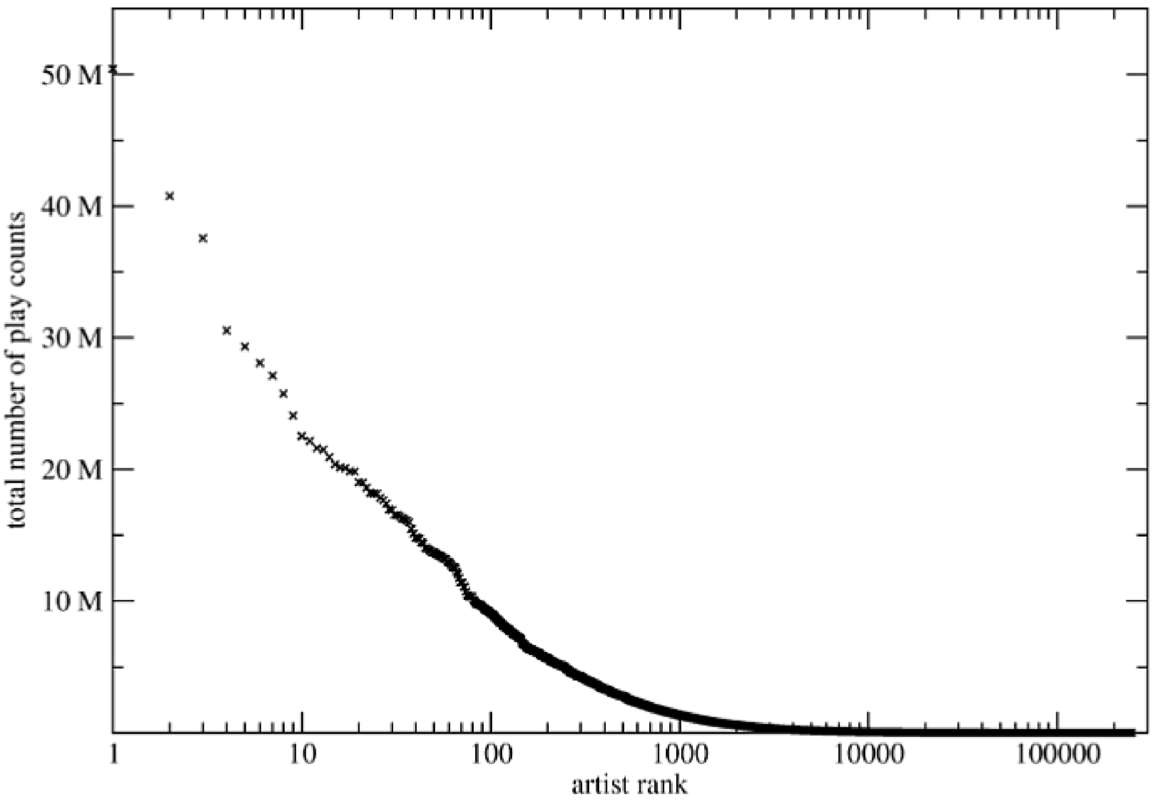


Figure 2.1: “The Long Tail for artist popularity. A log–linear plot depicting the total number of plays. Data gathered during July, 2007, for a list of 260,525 artists.” (Celma Herrada, 2009). Image credit: (Celma Herrada, 2009).

The long-tail effect in music described by (Celma, 2010) states that only a very small amount of all digital music tracks accounted for a very large number of plays, due to the biased recommendation systems of today's online shops and recommendation systems. The effects can be seen in Figure 2.1 from (Celma Herrada, 2009), which represents artist popularity for a list of 260,525 artists. It can be clearly seen that there is a bias towards a small group of artists, which make up most of the plays. With good recommendation systems, less known artists could be recommended to the user, making them more popular, thus balancing the sales rates over the available music tracks. Search providers could offer similar services like query by humming, textual search for songs given by tags or finding similar songs by a given reference.

With the ever more growing amount of music available on the internet, it is possible to create fully automated systems which build indexes of music to setup services as mentioned above. It is apparent that these kind of service can be used for good and to help users find more music they like.

## 2.2 Analyzing musical similarity

This chapter discusses why the concept of musical similarity is actually applicable to human beings and evaluates how to retrieve test data. Said test data can then be used to compare musical similarity as experienced by a person to different computer generated similarity measurements.

After all, it is possible, that every human has a different perception of musical similarity, e.g. due to differences in musical taste, which would render any attempt to create a universal musical similarity measurement impossible and useless.

The question, whether humans have a comparable judgment in musical similarity or not, can be formulated as a null hypothesis, which then may be proven wrong:

**Hypothesis 2.1** *The correlation of distances between songs, as judged by people, is zero.*

This null hypothesis states, that people have no shared sense of musical similarity, as the distances chosen by every human does not correlate with the rest of the distances chosen by others.

If the null hypothesis is rejected, the alternative hypothesis is true:

**Hypothesis 2.2** *The correlation of distances between songs, as judged by people, is significantly higher than zero.*

The alternative hypothesis implies that there is a common sense for musical similarity, as people judge distances between songs similarly.

The presented approach for testing the perception of similarity in this chapter, is to ask a sufficient number of persons to sort or categorize a list of songs by their similarity. If the results of this study are not evenly distributed, e.g. there are spikes for specific musical pieces, the presented hypothesis is proven to be wrong. However, doing a survey like this introduces a number of problems:

- How to present a measurement for similarity to the attendee?
- How many songs can a user judge in a reasonable amount of time?
- How to represent the songs to the attendee?

This chapter discusses these problems, along with implementation details in subsection 2.2.3, and the results of the study that was carried out in the scope of this thesis in subsection 2.2.5. The last part of this section consists of some words about future work and a conclusion.

For clarity, the terms **user** and **attendee** used in this chapter refer to a participant of the study.

### 2.2.1 Problem of a proper presentation

It is vital to choose the right presentation for the attendee, so that he can judge the similarities of the music as intuitively as possible for an accurate result of the survey. Furthermore, it is important that the presentation does not limit the informational value of the attendees data too much.

Possible ways of presenting musical pieces to the attendee are discussed and compared in this section, looking for an optimal trade-off between usability, being intuitive and precision of the data.

For the study, the Drag and Drop presentation was chosen as it was relatively easy to implement and use.

### Ranking

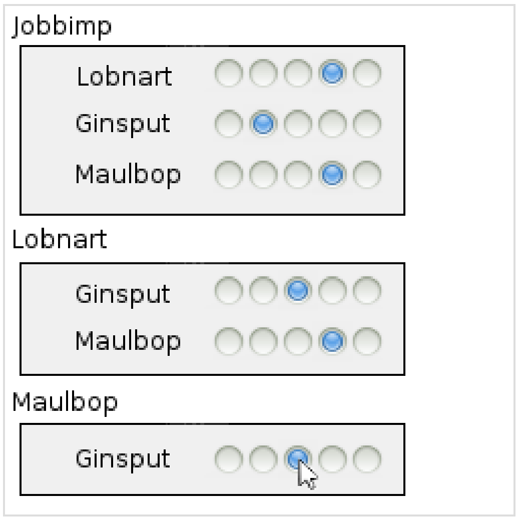


Figure 2.2: The user is presented with all necessary combinations of songs and can rate each song, in relation to the reference song, from not similar to very similar using radio buttons.

The ranking attempt represents all songs as a list, with each list item containing a further list of all other songs which were not yet compared to the current song.

The user needs to assign a ranking to the songs in the sub-list, which then states the relation of the selected song to the parent song, that is, the song containing the sub-list.

For the ranking a range from zero to five was chosen, with zero being the least similar rating and five the most similar.

The total number of songs the user has to rate can be cropped down to $\sum_1^{n-1}$ due to the symmetry of similarity.

This approach is similar to the approach used by (Lee, 2010), which offers a ranking of ‘Not similar’, ‘Somewhat Similar’ and ‘Very Similar’.

However, this sort of presentation turned out to be not very usable. Test persons found it difficult to express similarity of songs in a fixed range of numbers. Also, fixed classes are to vague for comparison with the result of algorithms, as these tend to generate distance values and not classes, which then results in a loss of accuracy when comparing the values.

### Drag and Drop

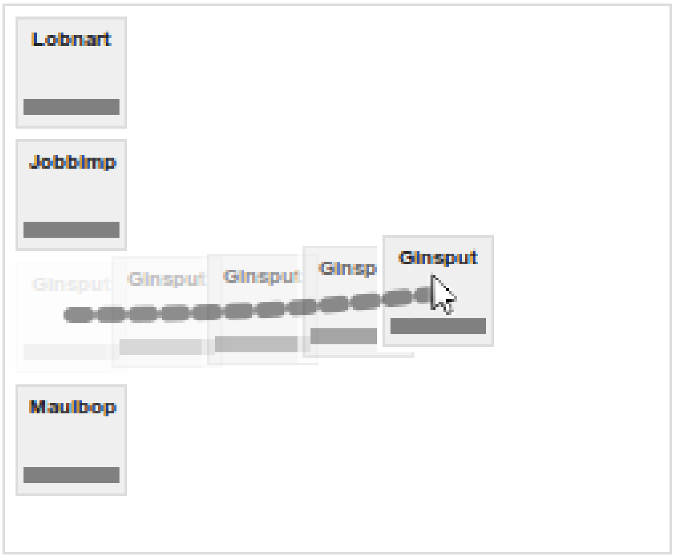


Figure 2.3: The songs are represented as movable boxes which can be dragged around using the mouse cursor. The distance between the songs represents the similarity. A far distance means that the song is not similar, near means similar.

Another way of expressing similarity is, to move similar things together. This can be done with music as well. The drag and drop rating method represents every song as a movable object, which can be dragged on the screen using the mouse. The music plays as long as the user's mouse cursor stays in the boundaries of the object.

By placing the squares close together, the user is expressing that those songs sound similar to her. Due to the symmetry of similarity, moving one object declares similarity for two or more songs, which results in quicker and more intuitive decisions made by the user.

### Drag and Drop with connections

The concept introduced in this section is based upon the Drag and Drop concept, applying the concept of dragging the songs around, but adds the ability to choose what connections are actually relevant and which are not.

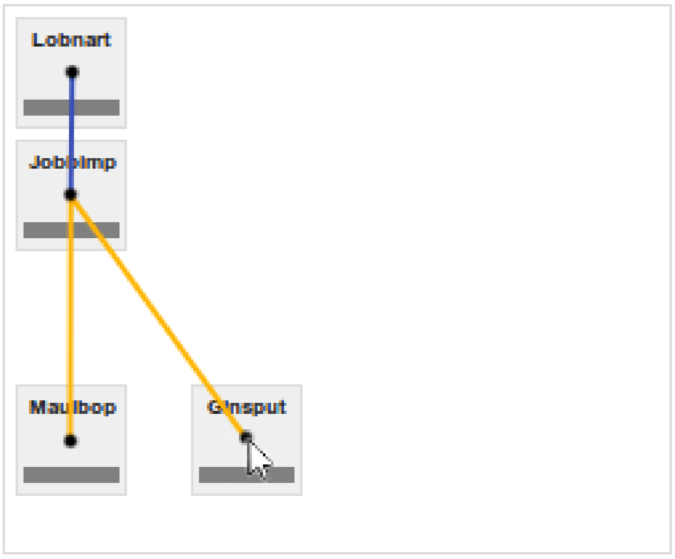


Figure 2.4: Songs can be moved around like before but the user can specify groups of songs for which the distances have meaning. Songs not in the group are not affected by the distances.

As the user moves songs around in the containment area, every time she places a song somewhere, this song has a distance to every other song, even if the user does not find any similarity between these. By letting the user choose on which songs the placement has influence on, the problem of unintentional similarities can be circumvented.

A drawback of this layout is the rather complex usage and the fact that too many connections between songs may confuse the user too much.

### 2.2.2 Chosen Songs

All songs were manually selected from the creative commons music distribution website jamendo.com[1] so that they fulfill the following criteria:

- There should be at least two songs of the same artist to test for artist similarity
- At least one representative of the following genres should be included: Jazz, Hip-Hop, Metal, Techno, Rock, Classic

[1] http://www.jamendo.com/

It was clear, that the attendees are not able to offer much more than 30 minutes of time for the study, on average probably not more than 15 minutes. Because of this, the number of songs was limited to 15.

To prevent any influence of the song or artist name, the survey only uses one-time randomly generated nick names for each song.

| Artist | Title | Nick name | Genre |
|---|---|---|---|
| AFTER WORK | Giù | Lobnart | Jazz |
| Brad Sucks | Dropping out of School | Jobbimp | Pop/Rock |
| Brad Sucks | Making Me Nervous | Pacsua | Pop/Rock |
| BS | Sin | Maulbop | Techno |
| Distemper | Одеяла Из Крыш | Mefpiis | Ska |
| Gianluca Luisi | Partita No. 1 - 4. Sarabande | Mujschi | Classic |
| Imprintz | Imprintz In the Limo | Cig | Techno / DnB |
| KEPAY | Cule Cule Bom Bom | Rasch | Ska |
| Marc Teichert | Inspiration | Raalneur | World |
| Nesto | Sales histoires | Spaspet | Hip-Hop |
| Nex2012 & DJ H | Funk y Salsa por favor | Meicje | Latin / Electronic |
| OnClassical | Partita in c-minor, I. Sinfonia | Jein | Classic |
| S.U.N. | Another Recipe | Dam | Hip-Hop |
| The Gasoline Brothers | Stardust, Baby! | Ginsput | Rock |
| Wasted Heroes | Keep It In Your Arms | Yadstis | Metal |

### 2.2.3 Implementation

#### General technical design

For presenting the study to the user, using web technology seemed to be the best choice, because web browsers are available almost everywhere and therefore makes the study available to a broader audience. Also there is a big set of software libraries ready to use.

The following software components were needed:

- Web Server, serving the content, namely the music and the study itself, to the participant
- Study, presented as a web page, which describes the purpose of the survey, introduces the participant and acts as a form to accept the input of the user.

The following implementation decisions were made:

- Web back-end (Serving, Evaluation, Storage) written in Go[2]
- HTML5 /JavaScript front-end using the jQuery and jQuery UI[3] framework
- Audio playback using the jQuery based jPlayer[4]
- Storage of survey data using the JSON format

#### Layout

The web page holding the content was sectioned in two parts, the description and the survey. The description contains an introduction into the purpose of the study and how to use it. The second part is the area containing the movable song elements, fitted to the user's screen width.

The songs the user judges are presented as squares with pseudo-names as descriptions. The squares are contained in a bigger square, which limits the area in which the squares may be moved. The containment area may vary in size to support different screen sizes. Figure 2.5 shows a visual example.

If the mouse cursor is hovering over a song square, the respective music plays using an HTML5 audio element. As soon as the mouse cursor leaves the square, the playback stops. By performing the common drag gesture with the mouse, the

[2]http://www.golang.org/
[3]http://www.jquery.org/
[4]http://jplayer.org/

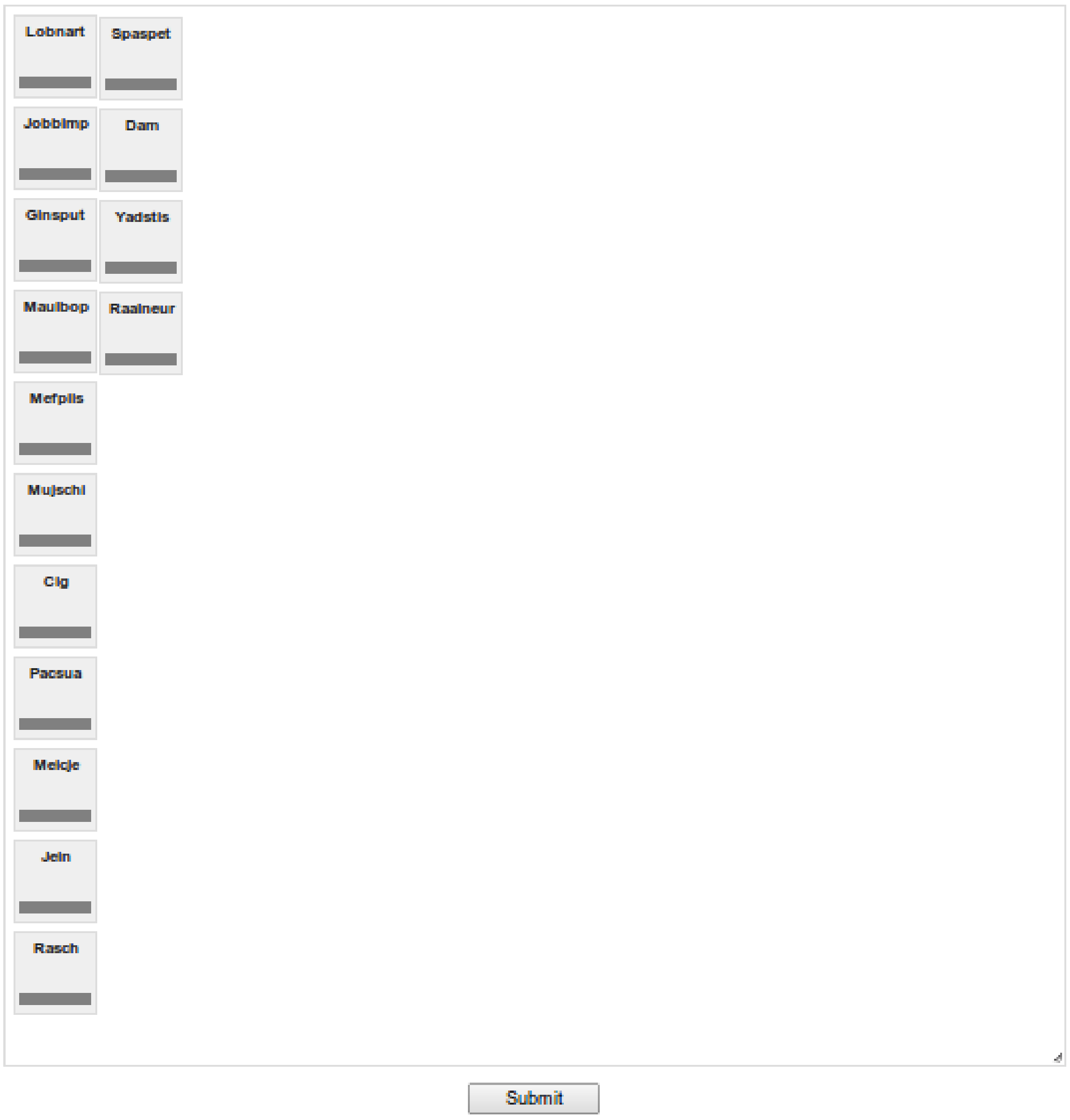


Figure 2.5: General layout of the study. Each box represents a song, which is identified by a nick name. Hovering the mouse cursor over a box plays the song. Dragging the box moves it. The gray bar can be used to seek in the song.

user is able to move the square to a position in the containment area. To support users who want to hear specific sections of a song, a clickable seek bar is included in every song object.

To help the user identify the songs that are identified as similar by the survey, the moved song square changes its color, depending on the position in the containment area.

The color distribution in the containment depends on the following rules:

- The lower on the y-axis, the greener is in the color
- The lower on the x-axis, the less red is in the color
- The higher on the y-axis, the bluer is in the color

At the end, when the user has finished adjusting the distance of the player objects, she needs to press a submit button. The positions of the song objects, as well as the dimensions of the containment area, are sent to the server software, which then stores the result on the server side.

After a successful submission, the user sees a greeting page that contains an acknowledgment, as well as a link to a page where she can review the overall results.

### Stored Data

After the user submits her result of the survey, the following data is transmitted to the web server:

- Dimensions of the containment area
- Positions of each song object relative to the containment area

The server stores the user's data in a unique file using the JSON format for data encoding. Listing 1 gives an example of how such a result may look like at the end.

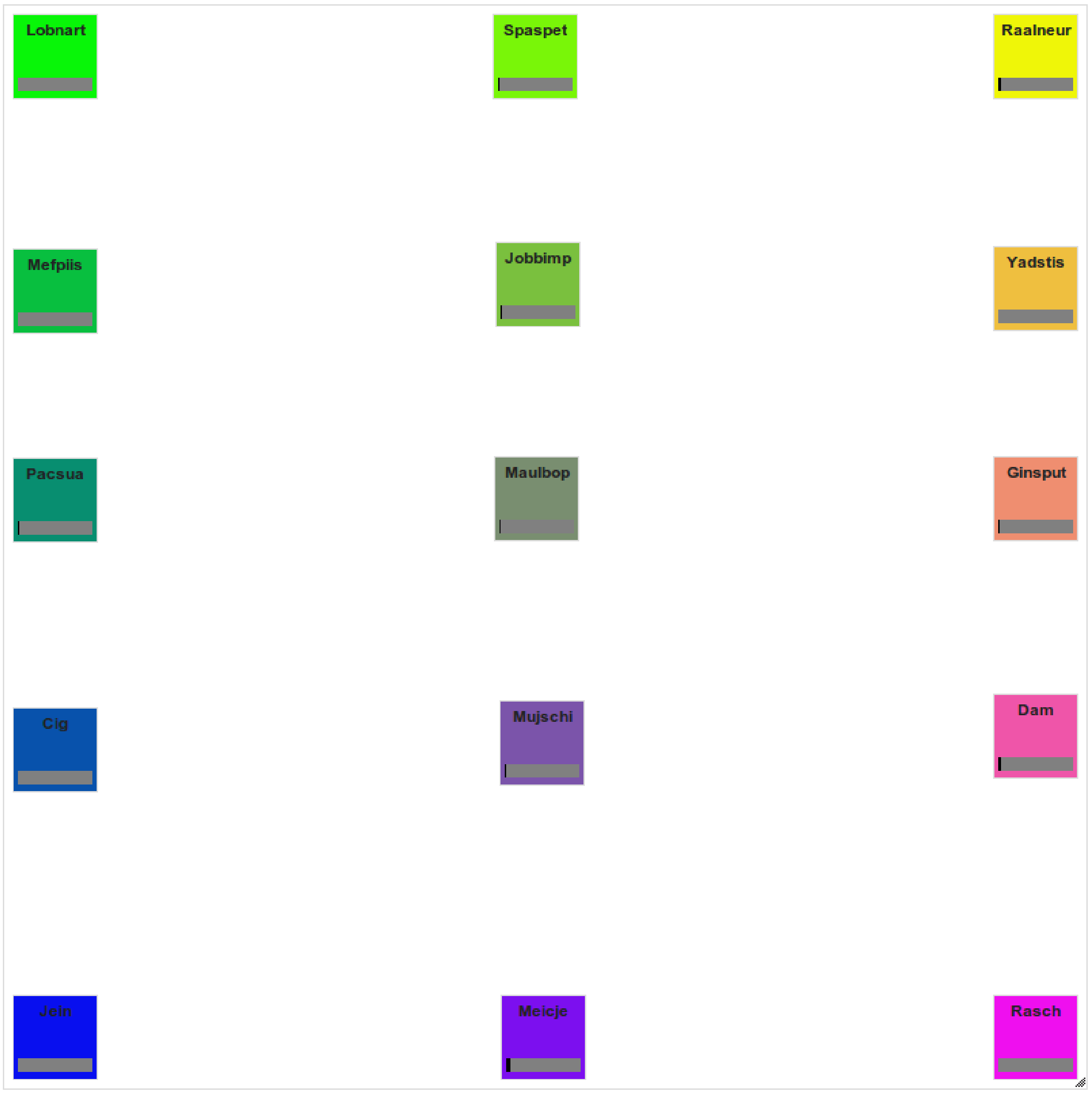


Figure 2.6: The movable song boxes are colored to help the user identify what is seen as similar and what is not.

Listing 1: Example of a user's survey result with two songs in JSON representation.

```
{
  "Positions": [
    {
      "Name": "Ginsput",
      "Position": [
        675.4999847412109,
        162.5
      ]
    },
    {
      "Name": "Lobnart",
      "Position": [
        678.4999847412109,
        27.5
      ]
    },
  ],
  "Dimensions": [
    730,
    730
  ]
}
```

### Computing the distance

The distance is computed using euclidean distance:

$$d(x, y) = sqrt(x^2 + y^2)$$

with

$$x = x_1 - x_0$$

and

$$y = y_1 - y_0 \tag{2.1}$$

The point taken for measurement is the center of the song square.

Due to the usage of squares, objects naturally have a higher distance to each other, if they are positioned diagonal.

### Normalization

Because of the variable size of computer screens, the containment area of the movable songs resizes itself to the browser window's width to offer as much space as possible. Due to this, the containment areas are not equally sized and **area normalization** must applied to the coordinates of each song.

As not every participant uses all the space of the containment area, some participant will place items near together, while others may use all the available space of the containment area. To reduce the effects of such placing habits, **distance normalization** needs to be applied for each computed distance.

**Area Normalization** Each point $(x, y)$ is normalized using the width and the height of the area the movable objects are in, giving us:

$$anorm(x, y) := (\frac{x}{width}, \frac{y}{height}) \tag{2.2}$$

It is important that $width = height$ , or otherwise the coordinates would lose their relationship in scale. The jQuery UI framework ensures, that this constraint is maintained.

**Distance Normalization** The distance normalization is achieved by using a min-max stretch:

$$dnorm(distance) = \frac{distance - min(distances)}{max(distances) - min(distances)} \tag{2.3}$$

This ensures, that the result is independent of the used containment area and the space used between the song objects.

### 2.2.4 Analysis of the user data

For proper analysis, the results of the study need to be presented to give as much information as possible. The distances are presented as a triangular matrix with song names as column and row descriptions. This way, it is very easy to see which song has what distance to another song. Additionally to the overall distances, the average distances between all songs, as well as the biggest and the smallest distance was computed.

The same matrix presentation used for the distances, is used to present the standard deviation. Along with the deviation data there is a table with the average deviation and the biggest /smallest deviation. The benefit of having the results displayed as a table is, that it is possible to join the distance table with the deviation sample, making it easier to see how certain the users were about the specific distance value.

To support the analysis, the mentioned tables are colored according to their minimum and maximum values. Cells of the table containing smaller values were colored greener whereas cells with higher values were colored redder.

### 2.2.5 Results

This section presents the cleaned and reviewed results and compares them with the uncleaned results. The first part of this section summarizes the study and introduces the further course through this section.

A statistical analysis of the results is carried out in the middle part of this section, followed by the features of the distances and deviations. At the end, the results of the two children are examined in comparison to the rest of the attendees.

### Introduction

The study was carried out as a qualitative one. The participants were hand picked and each participant received a personalized link, which however did not reveal the identity of the person.

A total of 42 persons with different expertise in music were asked for their opinions. The average age of the participants was 33 years. 40 persons were from Germany and two from the USA. The two children, siblings, were 8 and 10 respectively.

Some of the results needed to be omitted, as some attendees misunderstood the goal of the study and simply created clusters of songs which seemed similar to them. An example result with this properties can be seen in Figure 2.7. The problem with these results is, that they are not defining any distances between the songs in the clusters.

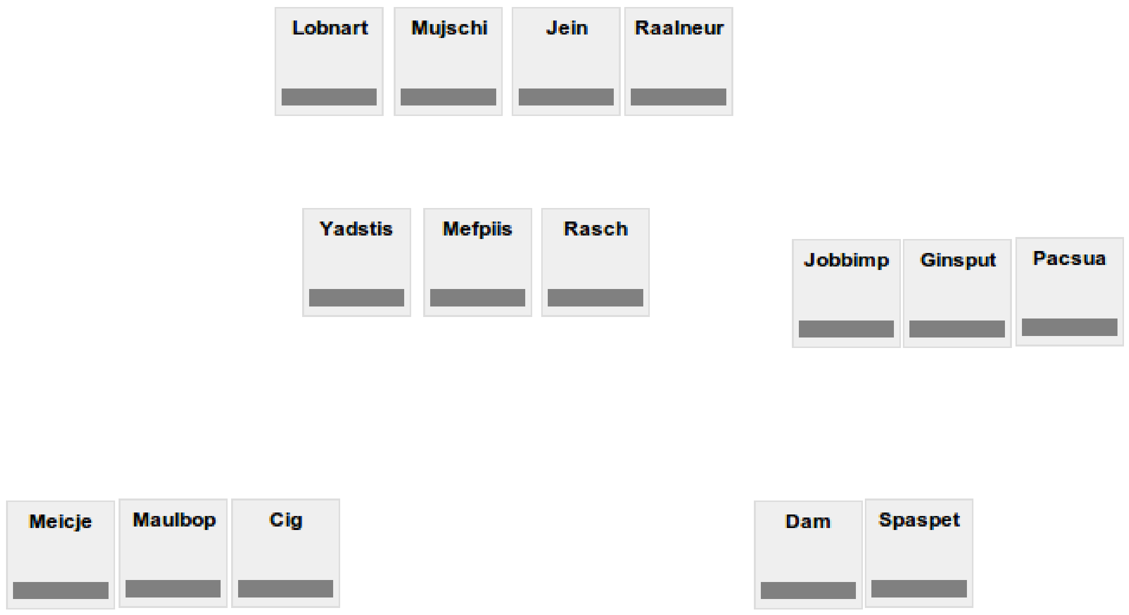


Figure 2.7: Some attendees tended to form clusters of similar sounding songs, regardless of the neighbors, resulting in unintentional similarities. In this figure, the Metal song “Yadstis” is very close to the Jazz song “Lobnart”. Also, there are no notable differences between the songs in the clusters.

As mentioned, some users did not complete the survey successfully and the results turned out to be unusable. For documentary purposes, this list shows the IDs of the unusable results:

- `1442443444`
- `TT98283905`
- `WF75882414`
- `NF75661670`
- `NF55759641`
- `TT44636702`
- `YH84375849`
- `YH66547383`
- `TT10565066`
- `TT67152040`
- `TT36831498`

Most of the time, the reason for the results being unusable was that the user did not understand the purpose of the survey, misinterpreting it as a task for classification. One result (`NF75661670`) was rendered unusable by a malfunction in the survey interface, which led to songs being placed outside the containment area. In total, having 10 partially useful and 1 broken, the remaining **31** results were examined as ‘cleaned results’. All 41 results were evaluated as ‘uncleaned results’ in Equation 2.2.5, which concludes that the uncleaned results are, in average, 10% more vague and thus are not significantly worse than the cleaned results. Due to this rather small difference the analysis of the results is based solely on the cleaned results.

The distance table shown in Figure 2.9 disproves the initial null hypothesis 2.1 visually. Statistical analysis of the results is done in the following section to verify that the null hypothesis can be rejected and the result is significant.

### Distribution and statistical analysis

In this section the statistical properties of the results are examined. The main tool used is the statistical programming language R (R Core Team, 2012), which is fed with the data from the survey using an export routine in the survey software that can be called using HTTP. An example call is shown by Listing 2 later in this section.

According to the null hypothesis from the survey introduction section 2.2, the correlation of the distance between the users is the most important data of the survey. If the mean correlation is zero or insignificantly higher than zero, one can say that the null hypothesis is true and there is no musical similarity for humans. The null hypothesis also holds in case of negative correlations. Negative correlation values occur when high values correspond to low values, meaning that some users tend to have high distances for songs and others think of the opposite. Clearly such a behavior would speak against a common sense for musical similarity.

In Figure 2.8 one can see, that most of the correlations gather around the mean correlation of 0.3269, displayed by the vertical line in the plot, following a normal distribution. The code written in R used to produce this histogram can be reviewed in Listing 2.

Listing 2: R code used to generate the histogram in Figure 2.8

```
> library(Hmisc)
> t <- read.table('http://localhost:8080/results?export=tabularfull')
> mt <- matrix(unlist(t), ncol=length(t))
> rs <- rcorr(mt)$r
# Strip the diagonal 1.0 values from the self-correlations
> rs <- rs[rs != 1.0]
> hist(rs, col="lightblue", breaks=20, main="",
+ xlab="Correlation Values", ylab="Occurences")
> abline(v=mean(rs))
```

As already mentioned in this section, negative mean correlation would indicate that the results are contradictory. Because the mean value of the correlations is not negative, the null- and alternative hypothesis can be formalized in Equation 2.4 and Equation 2.5 respectively. Notable is, that the hypotheses are one-sided as only the positive correlations matter.

$$H_0 : \mu = \mu_0 = 0 \tag{2.4}$$

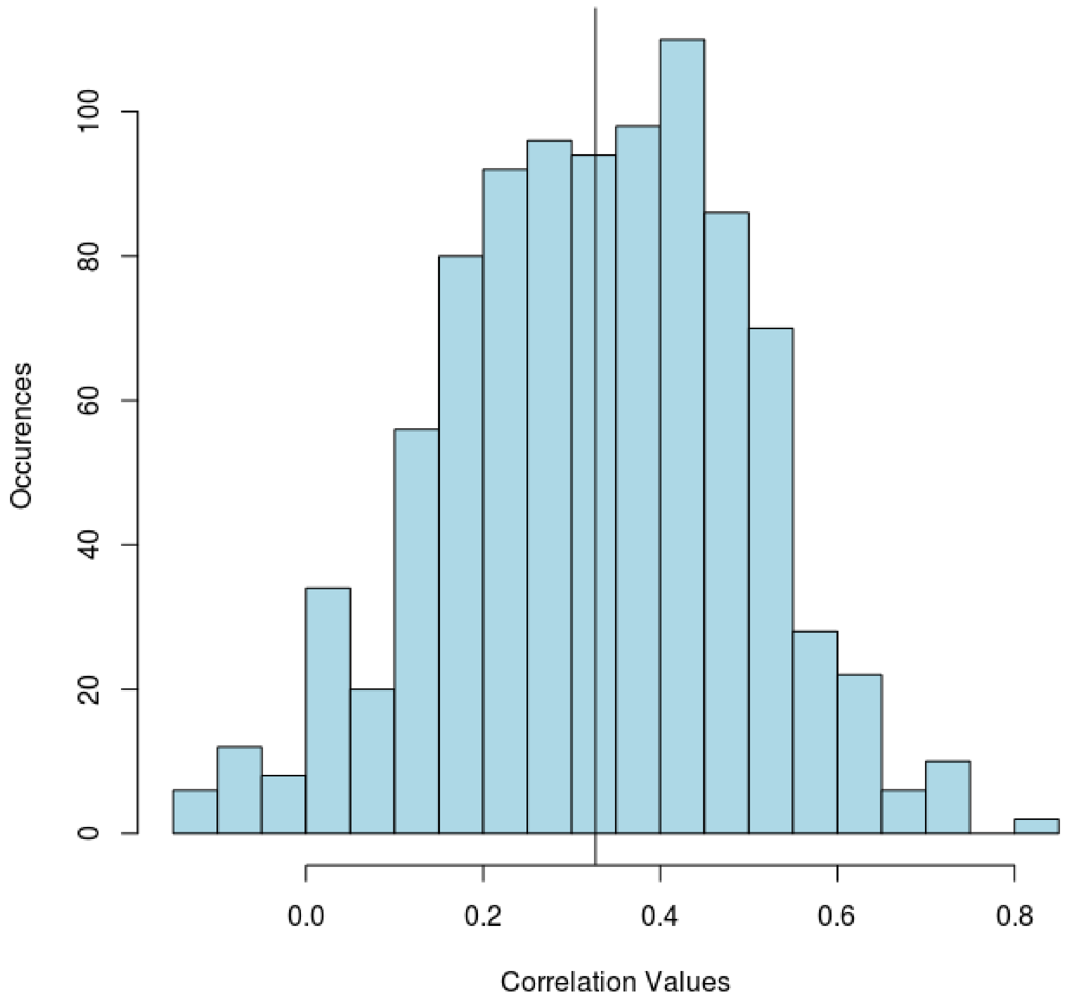


Figure 2.8: The correlations of the distances between the users seem to follow a normal distribution. This histogram, made with 20 bins, shows that most of the results are near the mean correlation of 0.3269, marked with a vertical line in the plot. The amount of negative correlations is in comparison to all positive values insignificantly low.

$$H_1 : \mu > \mu_0 \tag{2.5}$$

Hypothesis $H_0$ states, that the sampled mean correlation is equal to the value proposed in the null hypothesis, which is zero. The alternative hypothesis $H_1$ on the contrary, claims that this is not the case, so the mean correlation is significantly bigger than the proposed value of zero.

The mean correlation is already known to be $r = 0.3269$ from the calculation shown in Listing 2. To test if this value is significantly higher than zero and not random noise, the Student's t-statistics is used. This statistic needs a parameter to guess the actual t-distribution of the data, which is the degree of freedom (df). The actual t-Value, which is used along with the degrees of freedom, can be computed using R as follows, while the formula to retrieve the t-value is the one used in the `pspearman`[5] module from R:

```
> t <- r / sqrt((1 - r^2)/(n-2))
> t
[1] 1.862752
> p <- pt(t, df = 31-2, lower.tail = F)
> p
[1] 0.03632779
```

The results of the statistical analysis are shown in the R console listing above. The t-value of 1.863 was calculated with 31 degrees of freedom. This t-value corresponds to a one-tailed p-value of 3.6%. Therefore, it is safe to say that the null-hypothesis can be **rejected** with p $<$ 5%.

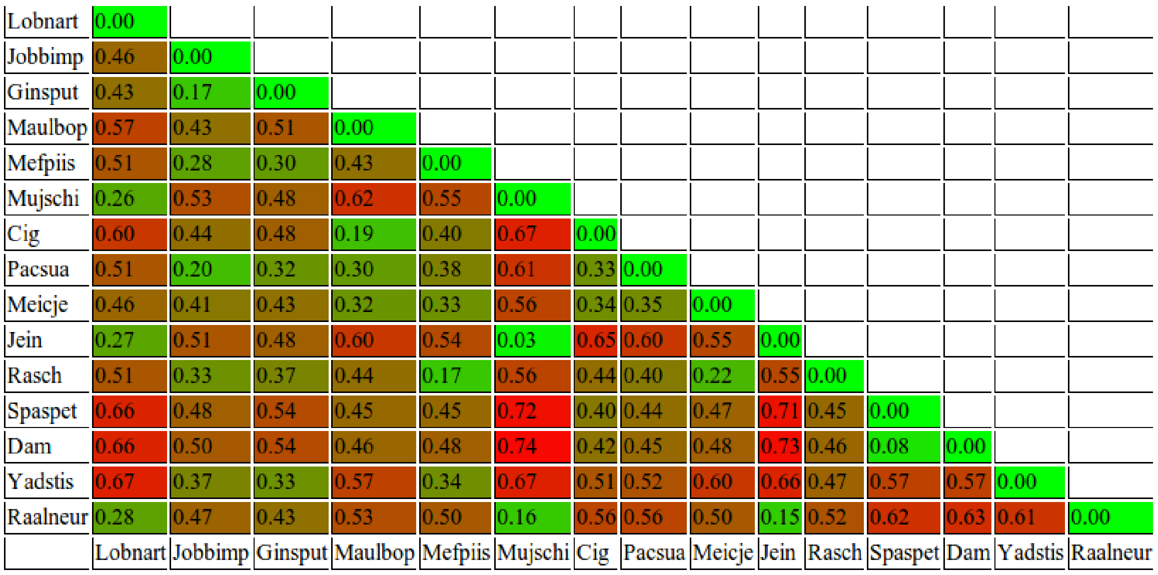
### Distances

The distances of the cleaned results are analyzed first in this section, followed by the distances of the uncleaned results, as described in the introduction of the results section subsection 2.2.5.

Several songs have very small distances, for example the two classical /piano songs, `Jein` and `Mujschi`. Also very close are `Mujschi`, `Jein` and `Raalneur`, which are rather quiet and chilling. The second closest pair in the table is `Spaspet` and `Dam`. Both are from the same genre and have a very typical Hip-Hop beat.

Pretty much all songs which share a genre have low distances to each other. Further examples are the three songs classified as Ska: `Mefpiis`, `Meicje` and `Rasch`, as well as the songs from the techno-ish genres: `Cig` and `Maulbop`.

[5] http://cran.r-project.org/web/packages/pspearman/index.html



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lobnart | 0.00 | | | | | | | | | | | | | | |
| Jobbimp | 0.46 | 0.00 | | | | | | | | | | | | | |
| Ginsput | 0.43 | 0.17 | 0.00 | | | | | | | | | | | | |
| Maulbop | 0.57 | 0.43 | 0.51 | 0.00 | | | | | | | | | | | |
| Mefpiis | 0.51 | 0.28 | 0.30 | 0.43 | 0.00 | | | | | | | | | | |
| Mujschi | 0.26 | 0.53 | 0.48 | 0.62 | 0.55 | 0.00 | | | | | | | | | |
| Cig | 0.60 | 0.44 | 0.48 | 0.19 | 0.40 | 0.67 | 0.00 | | | | | | | | |
| Pacsua | 0.51 | 0.20 | 0.32 | 0.30 | 0.38 | 0.61 | 0.33 | 0.00 | | | | | | | |
| Meicje | 0.46 | 0.41 | 0.43 | 0.32 | 0.33 | 0.56 | 0.34 | 0.35 | 0.00 | | | | | | |
| Jein | 0.27 | 0.51 | 0.48 | 0.60 | 0.54 | 0.03 | 0.65 | 0.60 | 0.55 | 0.00 | | | | | |
| Rasch | 0.51 | 0.33 | 0.37 | 0.44 | 0.17 | 0.56 | 0.44 | 0.40 | 0.22 | 0.55 | 0.00 | | | | |
| Spaspet | 0.66 | 0.48 | 0.54 | 0.45 | 0.45 | 0.72 | 0.40 | 0.44 | 0.47 | 0.71 | 0.45 | 0.00 | | | |
| Dam | 0.66 | 0.50 | 0.54 | 0.46 | 0.48 | 0.74 | 0.42 | 0.45 | 0.48 | 0.73 | 0.46 | 0.08 | 0.00 | | |
| Yadstis | 0.67 | 0.37 | 0.33 | 0.57 | 0.34 | 0.67 | 0.51 | 0.52 | 0.60 | 0.66 | 0.47 | 0.57 | 0.57 | 0.00 | |
| Raalneur | 0.28 | 0.47 | 0.43 | 0.53 | 0.50 | 0.16 | 0.56 | 0.56 | 0.50 | 0.15 | 0.52 | 0.62 | 0.63 | 0.61 | 0.00 |
| | Lobnart | Jobbimp | Ginsput | Maulbop | Mefpiis | Mujschi | Cig | Pacsua | Meicje | Jein | Rasch | Spaspet | Dam | Yadstis | Raalneur |

Figure 2.9: Distances computed from the cleaned results of the survey. The greener the cell, the lower the distance. The row and column labels name the songs used in the survey while the values are the average distance between the songs.

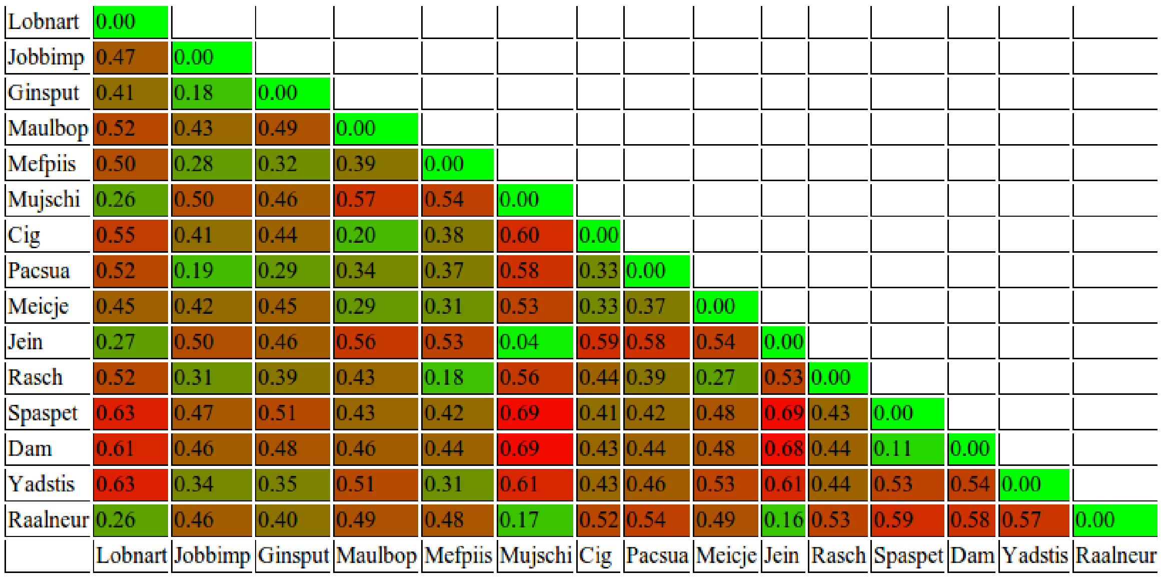
The two songs from the same artist, `Pacsua` and `Jobbimp`, have an average distance of 0.2, which is also relatively low. This short distance was predicted, as the songs do not differ in genre and are from the same artist, which inherently results in a similar style.

Regarding the Jazz song, `Lobnart`, strong similarities exist to the classical pieces, as well as the world song `Raalneur`, which might have something to do with the piano being played in all songs and the relatively chilling mood. To finish the analysis, the Metal song `Yadstis`'s only strong similarities are with the other more rocky songs, like the pop/rock song `Jobbimp`, the Ska song `Mefpiis` and the rock song `Ginsput`.

The following table shows key data about the distances shown in Figure 2.9:

| | | |
|---|---|---|
| Biggest distance | Mujschi - Dam | 0.741 |
| Smallest distance | Mujschi - Jein | 0.029 |
| Average distance | N/A | 0.461 |
| Median distance | N/A | 0.475 |

In comparison to the uncleaned samples shown in Figure 2.10, the cleaned results are spread a bit broader, with the average distance being 0.46 opposed to 0.44 in the uncleaned distances. However, the overall song relations, as discussed above, do not change.



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lobnart | 0.00 | | | | | | | | | | | | | | |
| Jobbimp | 0.47 | 0.00 | | | | | | | | | | | | | |
| Ginsput | 0.41 | 0.18 | 0.00 | | | | | | | | | | | | |
| Maulbop | 0.52 | 0.43 | 0.49 | 0.00 | | | | | | | | | | | |
| Mefpiis | 0.50 | 0.28 | 0.32 | 0.39 | 0.00 | | | | | | | | | | |
| Mujschi | 0.26 | 0.50 | 0.46 | 0.57 | 0.54 | 0.00 | | | | | | | | | |
| Cig | 0.55 | 0.41 | 0.44 | 0.20 | 0.38 | 0.60 | 0.00 | | | | | | | | |
| Pacsua | 0.52 | 0.19 | 0.29 | 0.34 | 0.37 | 0.58 | 0.33 | 0.00 | | | | | | | |
| Meicje | 0.45 | 0.42 | 0.45 | 0.29 | 0.31 | 0.53 | 0.33 | 0.37 | 0.00 | | | | | | |
| Jein | 0.27 | 0.50 | 0.46 | 0.56 | 0.53 | 0.04 | 0.59 | 0.58 | 0.54 | 0.00 | | | | | |
| Rasch | 0.52 | 0.31 | 0.39 | 0.43 | 0.18 | 0.56 | 0.44 | 0.39 | 0.27 | 0.53 | 0.00 | | | | |
| Spaspet | 0.63 | 0.47 | 0.51 | 0.43 | 0.42 | 0.69 | 0.41 | 0.42 | 0.48 | 0.69 | 0.43 | 0.00 | | | |
| Dam | 0.61 | 0.46 | 0.48 | 0.46 | 0.44 | 0.69 | 0.43 | 0.44 | 0.48 | 0.68 | 0.44 | 0.11 | 0.00 | | |
| Yadstis | 0.63 | 0.34 | 0.35 | 0.51 | 0.31 | 0.61 | 0.43 | 0.46 | 0.53 | 0.61 | 0.44 | 0.53 | 0.54 | 0.00 | |
| Raalneur | 0.26 | 0.46 | 0.40 | 0.49 | 0.48 | 0.17 | 0.52 | 0.54 | 0.49 | 0.16 | 0.53 | 0.59 | 0.58 | 0.57 | 0.00 |
| | Lobnart | Jobbimp | Ginsput | Maulbop | Mefpiis | Mujschi | Cig | Pacsua | Meicje | Jein | Rasch | Spaspet | Dam | Yadstis | Raalneur |

Figure 2.10: Distances computed from the uncleaned results of the survey. The greener the cell, the lower the distance.

The change in the biggest distance, from `Mujschi-Dam` to `Mujschi-Spasbet`, as seen in the table below, is not of great importance. The distances of the songs `Dam` and `Spaspet` are very close. So with a little fluctuation, as it happens in the uncleaned results, both swap places.

As the general direction of the distances remains the same but the deviances are, in average, 10% higher with the uncleaned results, the uncleaned results are ignored in further sections.

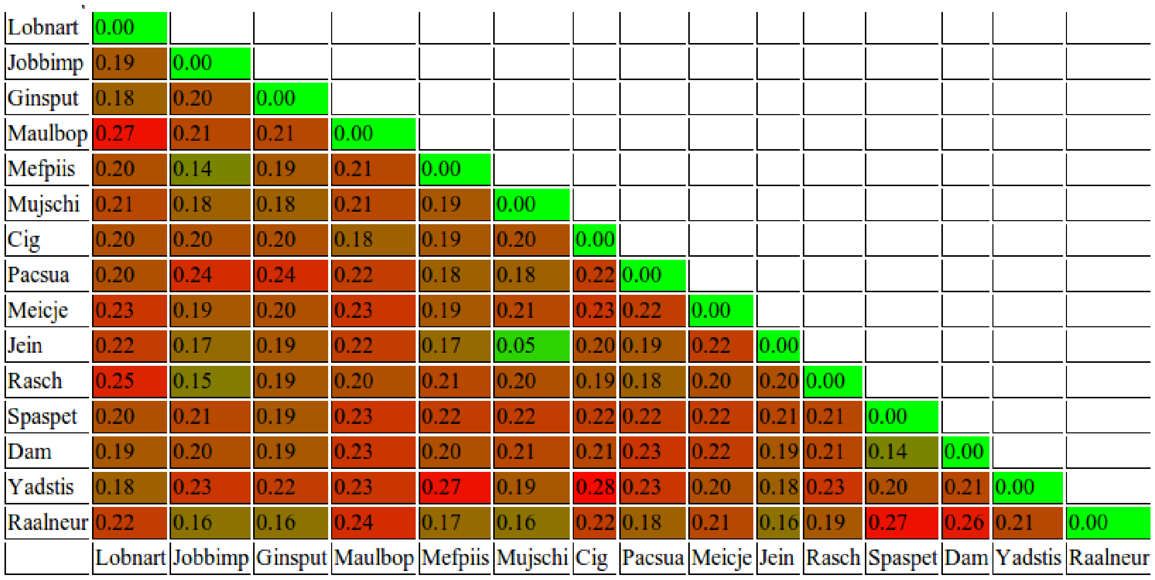
| | | |
|---|---|---|
| Biggest distance | Mujschi - Spaspet | 0.691 |
| Smallest distance | Mujschi - Jein | 0.037 |
| Average distance | N/A | 0.443 |
| Median distance | N/A | 0.459 |

### Deviations

While there are different deviation measurements, such as the mean deviation, the standard deviation was used to compare the results. The standard deviation emphasizes outliers so that these can be detected quickly.

The formula used for the Standard Deviation is the following, where $x_{avg}$ is the average distance of the song, $x_i$ is the user supplied distance and $n$ is the number of results.

$$\sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_{avg}-x_i)^2} \tag{2.6}$$



| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lobnart | 0.00 | | | | | | | | | | | | | | |
| Jobbimp | 0.19 | 0.00 | | | | | | | | | | | | | |
| Ginsput | 0.18 | 0.20 | 0.00 | | | | | | | | | | | | |
| Maulbop | 0.27 | 0.21 | 0.21 | 0.00 | | | | | | | | | | | |
| Mefpiis | 0.20 | 0.14 | 0.19 | 0.21 | 0.00 | | | | | | | | | | |
| Mujschi | 0.21 | 0.18 | 0.18 | 0.21 | 0.19 | 0.00 | | | | | | | | | |
| Cig | 0.20 | 0.20 | 0.20 | 0.18 | 0.19 | 0.20 | 0.00 | | | | | | | | |
| Pacsua | 0.20 | 0.24 | 0.24 | 0.22 | 0.18 | 0.18 | 0.22 | 0.00 | | | | | | | |
| Meicje | 0.23 | 0.19 | 0.20 | 0.23 | 0.19 | 0.21 | 0.23 | 0.22 | 0.00 | | | | | | |
| Jein | 0.22 | 0.17 | 0.19 | 0.22 | 0.17 | 0.05 | 0.20 | 0.19 | 0.22 | 0.00 | | | | | |
| Rasch | 0.25 | 0.15 | 0.19 | 0.20 | 0.21 | 0.20 | 0.19 | 0.18 | 0.20 | 0.20 | 0.00 | | | | |
| Spaspet | 0.20 | 0.21 | 0.19 | 0.23 | 0.22 | 0.22 | 0.22 | 0.22 | 0.22 | 0.21 | 0.21 | 0.00 | | | |
| Dam | 0.19 | 0.20 | 0.19 | 0.23 | 0.20 | 0.21 | 0.21 | 0.23 | 0.22 | 0.19 | 0.21 | 0.14 | 0.00 | | |
| Yadstis | 0.18 | 0.23 | 0.22 | 0.23 | 0.27 | 0.19 | 0.28 | 0.23 | 0.20 | 0.18 | 0.23 | 0.20 | 0.21 | 0.00 | |
| Raalneur | 0.22 | 0.16 | 0.16 | 0.24 | 0.17 | 0.16 | 0.22 | 0.18 | 0.21 | 0.16 | 0.19 | 0.27 | 0.26 | 0.21 | 0.00 |
| | Lobnart | Jobbimp | Ginsput | Maulbop | Mefpiis | Mujschi | Cig | Pacsua | Meicje | Jein | Rasch | Spaspet | Dam | Yadstis | Raalneur |

Figure 2.11: Standard deviation computed from the cleaned results of the survey. The greener the cell, the lower the deviation.

Figure 2.9 shows the deviances of the distances from the cleaned results. The most notable feature of the table is, that the two classical pieces, `Mujschi` and `Jein` are very definite with a deviance of 0.04. A possible explanation for the high uncertainty of the song pair `Cig` and `Yadstis` is, that both songs are dark /aggressive sounding but the genre of `Cig`, Drum and Bass, is not well known, which could lead to uncertainties.

| | | |
|---|---|---|
| Biggest deviance | Cig - Yadstis | 0.275 |
| Smallest deviance | Mujschi - Jein | 0.049 |
| Average deviance | N/A | 0.203 |
| Median deviance | N/A | 0.204 |
| Standard deviation | N/A | 0.145 |

The table shown above states some basic facts of the deviations as seen in Figure 2.9. In this case, average and median deviance is, the average and median pair deviance. That is, taking all the pair deviances and averaging /building the median. The median deviation is at $0.20$ , which equals 20% of the distance. The standard deviation of the distances is at $0.145$ , where higher values mean more variation in similarities. Higher variation in similarities is a good thing, as this means that some songs are more similar than others, which proves the null-hypothesis wrong, as mentioned before.

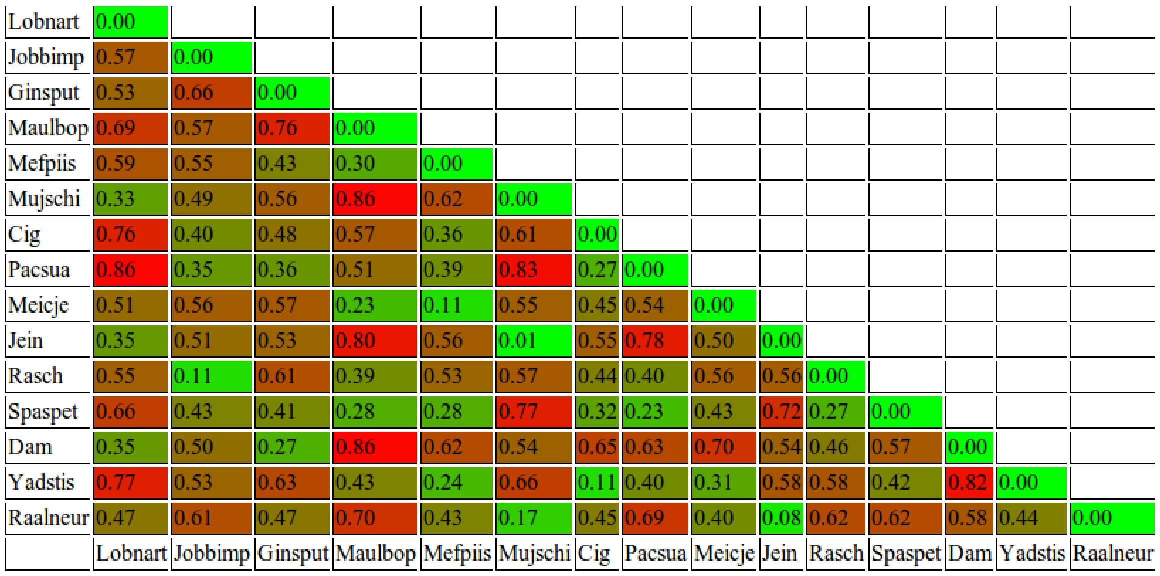
Almost all relations (71%) between `Jein` and other songs are below or equal the deviance median. The other classical song, `Mujschi`, is not far behind with 64% relations lower or equal to the median. This means that most of the relations between the classical songs are fairly certain for all attendees. The songs that can be clearly identified as rock songs, `Jobbimp`, `Ginsput` and `Mefpis`, are also on the top of the table.

The following table shows the distribution of songs below the median deviation in percentage. A lower percentage means higher uncertainty, as more values lie over the median deviation.

| | |
|---|---|
| Maulbop | 14.29% |
| Meicje | 21.43% |
| Spaspet | 28.57% |
| Yadstis | 35.71% |
| Dam | 42.86% |
| Pacsua | 42.86% |
| Raalneur | 50.00% |
| Rasch | 57.14% |
| Lobnart | 57.14% |
| Cig | 57.14% |
| Mujschi | 64.29% |
| Mefpiis | 71.43% |
| Ginsput | 71.43% |
| Jein | 71.43% |
| Jobbimp | 71.43% |

### Results of children

This section shows, that the two children, who took the survey, did not land off too much from the results of the grownups. There are some differences but, for example, the classical and rocky songs are classified equally to the grownups.



| | Lobnart | Jobbimp | Ginsput | Maulbop | Mefpiis | Mujschi | Cig | Pacsua | Meicje | Jein | Rasch | Spaspet | Dam | Yadstis | Raalneur |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lobnart | 0.00 | | | | | | | | | | | | | | |
| Jobbimp | 0.57 | 0.00 | | | | | | | | | | | | | |
| Ginsput | 0.53 | 0.66 | 0.00 | | | | | | | | | | | | |
| Maulbop | 0.69 | 0.57 | 0.76 | 0.00 | | | | | | | | | | | |
| Mefpiis | 0.59 | 0.55 | 0.43 | 0.30 | 0.00 | | | | | | | | | | |
| Mujschi | 0.33 | 0.49 | 0.56 | 0.86 | 0.62 | 0.00 | | | | | | | | | |
| Cig | 0.76 | 0.40 | 0.48 | 0.57 | 0.36 | 0.61 | 0.00 | | | | | | | | |
| Pacsua | 0.86 | 0.35 | 0.36 | 0.51 | 0.39 | 0.83 | 0.27 | 0.00 | | | | | | | |
| Meicje | 0.51 | 0.56 | 0.57 | 0.23 | 0.11 | 0.55 | 0.45 | 0.54 | 0.00 | | | | | | |
| Jein | 0.35 | 0.51 | 0.53 | 0.80 | 0.56 | 0.01 | 0.55 | 0.78 | 0.50 | 0.00 | | | | | |
| Rasch | 0.55 | 0.11 | 0.61 | 0.39 | 0.53 | 0.57 | 0.44 | 0.40 | 0.56 | 0.56 | 0.00 | | | | |
| Spaspet | 0.66 | 0.43 | 0.41 | 0.28 | 0.28 | 0.77 | 0.32 | 0.23 | 0.43 | 0.72 | 0.27 | 0.00 | | | |
| Dam | 0.35 | 0.50 | 0.27 | 0.86 | 0.62 | 0.54 | 0.65 | 0.63 | 0.70 | 0.54 | 0.46 | 0.57 | 0.00 | | |
| Yadstis | 0.77 | 0.53 | 0.63 | 0.43 | 0.24 | 0.66 | 0.11 | 0.40 | 0.31 | 0.58 | 0.58 | 0.42 | 0.82 | 0.00 | |
| Raalneur | 0.47 | 0.61 | 0.47 | 0.70 | 0.43 | 0.17 | 0.45 | 0.69 | 0.40 | 0.08 | 0.62 | 0.62 | 0.58 | 0.44 | 0.00 |

Figure 2.12: Distances computed from the childrens' results of the survey. The greener the cell, the lower the distance.

The following table shows the songs which are very similar, according to both children. Every song which has a distance smaller than half of the median distance, 0.265 in this case, is considered very similar.

| | | |
|---|---|---|
| Jein | Mujschi | 0.01 |
| Jein | Raalneur | 0.08 |
| Mefpiis | Meicje | 0.11 |
| Rasch | Jobbimp | 0.11 |
| Yadstis | Cig | 0.11 |
| Mujschi | Raalneur | 0.17 |
| Spaspet | Pacsua | 0.23 |
| Maulbop | Meicje | 0.23 |
| Yadstis | Mefpiis | 0.24 |

Like in the results of the grown ups, there are strong similarities between the two classical pieces and the world song, `Raalneur`. The two ska songs `Mefpiis` and `Meicje` are also in the tops, so are the rocky songs `Rasch` and `Jobbimp`. As already discussed in the analysis of the grown ups' results, there is a high deviation between `Cig` and `Yadstis`, which can be seen in the results of the children as well. The children seem to find both songs similar despite the fact, that both are using completely different instruments and the metal song even uses voice while the other song does not have voice in it.

Distances:

| | | |
|---|---|---|
| Biggest distance | Maulbop - Dam | 0.865 |
| Smallest distance | Mujschi - Jein | 0.011 |
| Average distance | N/A | 0.503 |
| Median distance | N/A | 0.529 |

### 2.2.6 Conclusion

The initial null hypothesis 2.1 for this study is rejected in the statistical analysis section, backing that there is something like musical similarity for humans. This result is also backed by the results of (Lee, 2010).

While it may seem unnecessary to conduct this study, especially with the limited amount of participants, the lack of freely available test data to test own features, and the interest in creating another interface for a similarity survey, justify this study. (Berenzweig et al., 2004) states, that there is still a lack for ground truth data in the Music Information Retrieval Community. While there are projects like the Million Song Database ongoing, there is still further research to be done. Sites like jamendo.com, which offer freely accessible music, are very well suited to use them in the quest for ground truth data. Test data for features in form of distance pairs between the songs, like the data successfully acquired in this study, is still needed.

The interface used in the study has proven itself to be usable and quite intuitive. Less technically experienced users, like the children, had no problems using it. However, as 26% of the users misunderstood the **concept** of the study, it would have been better to communicate the concept using, for example, introductory images or videos instead of a textual description.

### 2.2.7 Future work and open questions

This study has lots of room for improvement and there are questions that cannot be answered with the given data due to missing information or due to the limited amount of available data.

For example, this study does not provide information about the distances of songs in a shared genre. While the question of how humans judge similarities across genres was examined (e.g. Jazz vs. Heavy Metal), the question of how they judge songs in the same genre (e.g. Bach vs. Beethoven) is left untouched.

It would have been interesting to have test data for this case, as genres often have sub-genres which are very distinct to another but different enough to form a sub-genre.

So, while this study provides test data for inter-genre similarity, inner-genre similarity cannot be tested with it.

A further problem is the limited amount of data due to the number of participants. Running this study using crowd sourcing tools like Amazon Mechanical Turk, as done by (Lee, 2010), looks promising. However, the process of the survey needs a bit tweaking for that task, to prevent, or at least to detect, fake entries. This may be achieved by using a timer, to count how much time the attendee used, or by checking whether all songs were listened to and how long.

When releasing the survey to a bigger audience, one might also want to add more songs, which in turn makes it necessary for each user to take part in the survey to cover all songs. For this to happen, some sort of motivation must be given, e.g. some sort of game. One example for such a game would be, that users may get achievements for special actions, for example taking part the second time or rating the same song twice. However, the options for such games are very limited, as such games must not influence the decision making of the attendee regarding the similarities, which forbids comparisons of results and other participants to some extent.

## 2.3 Finding similarity

### 2.3.1 Approach

Defining a measurement for similarity based on audio signal data is a difficult task. The used frequencies and the frequency spectrum bias of the audio signal may give information about a general sound direction, maybe enough to tell certain songs apart, but ignores tempo and changes in pitch completely. For a similarity measure it is insufficient to rely solely on frequency statistics. Amongst other things, the tempo of the songs as well as the limitations of the human ear and neurons are important. The feature vector for musical similarity can grow very large: for example, the 'genome' database used by the Music Gnome Project holds "up to 450 distinct musical characteristics" (Pandora, 2013) per song. The approach made in this thesis is not to match with a feature vector like the one from the Music Genome Project but to find suitable approximations for musical similarity while being reasonably scalable to large quantities of songs.

At the beginning of this thesis it was planned to extract as many features as possible by frequency analysis and by using an onset strength signal analysis for beat and tempo analysis, which was successfully applied by (Holzapfel and Stylianou, 2010). However, after further research, this was given up in favor to the Auditory Image Model (Patterson et al., 1995). This model resembles every important step from the ear to the brain based on empirical data, and, thus featuring the psychological and physiological effects described by the field of psychoacoustics. The generated images of the neuronal signals even captures some temporal information of how the audio signal evolves over time. It seems reasonable to choose the Auditory Image Model as a foundation for further research, as it looks promising and offers many features combined at once while trying to emulate human hearing.

Additionally to using the Auditory Image Model, the idea of using string representations for audio features as proposed by (Casey and Slaney, 2006) is appealing, as string analysis is well understood and performs well. With these two ideas, the general approach used in this thesis can be broken down to the following steps:

1. Compute auditory image from a song
2. Translate the auditory image into a string representation
3. Analyze strings and extract features
4. Measure distance between feature vectors

The following section shortly explains what the auditory image is, while the Features section section 2.4 and its subsections discuss the way how such an auditory image is translated into strings and the details of how these strings are processed.

### 2.3.2 Auditory Image Model

Auditory images are an attempt to simulate the images that our brain receives when confronted with audio signals. Firstly, the audio signal is processed by filters which simulate the mechanical motions that occur with the basilar membrane. After that, the conversion to neural signals is simulated. Eventually the result is stabilized to compensate the phase lag resulting from the filters responsible for the lower frequencies.

(Patterson et al., 1995) introduces the first popular implementation of the Auditory Image Model, called AIM-C, while (Walters, 2011) gives detailed information over the current state and additions made over time. The main semantical stages to produce an auditory image basically remained and can be described in the following steps:

1. Middle ear filtering
2. Spectral analysis $\rightarrow$ Basilar membrane motion
3. Neural encoding $\rightarrow$ Neural Activity Pattern
4. Time-interval stabilization $\rightarrow$ Stabilized Auditory Image

[][impl:aimvq] in the implementation chapter contains a more detailed explanation of each module and its responsibilities by the means of an actual implementation of the auditory image. However, the Stabilized Auditory Image, the end product of the Auditory Image Model as shown in Figure 2.13, is discussed here to give the reader an idea of what data is processed by the distance measurement features presented in the next sections.

The horizontal axis of the Stabilized Auditory Image (SAI), as shown in Figure 2.13, represents a time interval, sometimes called lag, while the vertical axis names the center frequencies of the available frequency channels. The SAI is a short snapshot of an audio signal. The input sound is split into several frequency channels where every channel, each representing a neuron, is responsible for a certain frequency range of which the center frequency is stated by the frequency axis. For example, the channel at 3.9kHz for this image may be responsible for 3.9 kHz $\pm$263 Hz.

As the filters of the channels introduce a phase lag, the signal needs to be stabilized, hence the name Stabilized Auditory Image. By searching for special points

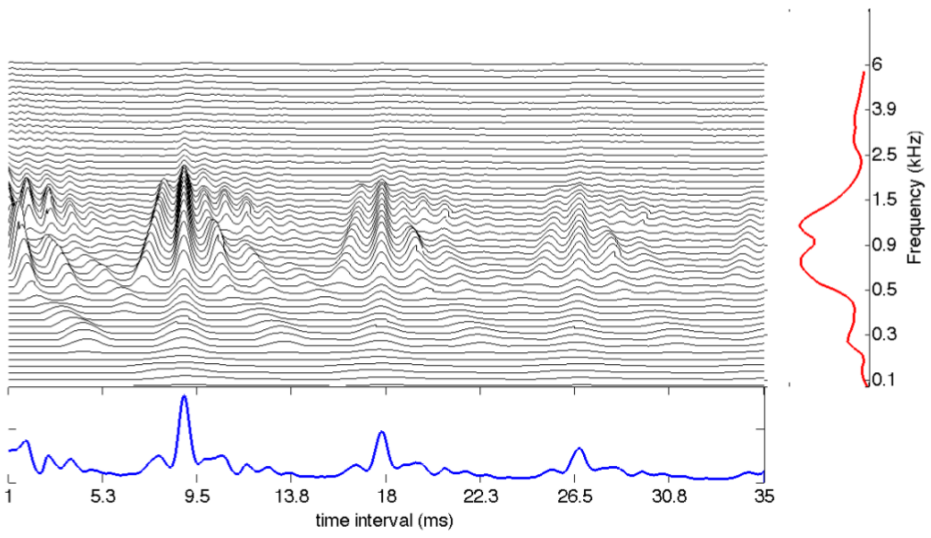


Figure 2.13: Example of a Stabilized Auditory Image. The sub-graph on the bottom shows the average neural activity for that time and the sub-graph at the right side shows the average of the neural activity over time. Every wave line represents the action in a single frequency channel. Image credit: (Walters, 2011).

in each frequency channel, for example local maxima, called strobes, the signal gets phase aligned along the found strobe points. This works by inserting channel values into the image only in case a strobe was detected. The time delay between the strobe and the other values behind it determines the place where that value will be inserted into the image. A value 10ms behind a strobe will be inserted to the image at $t = 0$ when the strobe is at $t = 10$. Because of this method the time axis represents an interval instead of continuous time. The interval time represents the time from a value to the last detected strobe. It also shows the time a data point was inserted into the image, for example, a data point at 30.8 ms was inserted in the image 30.8 ms ago. In the figure at $t = 8.975$ the effect of the phase aligning can be seen best. A comparison of how the unstabilized image would look like can be seen in Figure 3.2.

The additional graphs below and to the side of Figure 2.13 show the averaged frequency values for that time and the averaged frequencies for the whole interval respectively. One can see that at $t = 8.975$ the most channels were active. It can also be seen that the values are getting smaller over time, which is due to the decay factor for strobes and their values.

For further reading and details, Chapter 2 of (Walters, 2011) discusses in full detail the whole process of the Auditory Image Model chain in its current state. More compact information can be found in (Ness et al., 2011), which summarizes the process of the auditory image model.

The auditory image model was successfully used for melody recognition in (Walters et al., 2012) and for sound recognition tasks in (Ness et al., 2011).

## 2.4 Features

This section discusses various methods of getting features from vector quantized auditory images introduced in subsection 2.3.2. Not all of the discussed ideas were implemented, either due to lack of time or because basic analysis showed the features to be ineffective.

### Vector Quantization

Vector quantization (VQ) essentially associates blocks of data with a similar structured vector from another list of vectors called a codebook. The similarity is defined on a per number basis and determined using, for example, a k-nearest neighbor method. The result of this process is the index of the vector from the codebook. Figure 2.14 illustrates the simplified process of vector quantization. An input is matched against a codebook using a nearest neighbor method and the index with the nearest neighbor, having the smallest distance to the input, is the result.

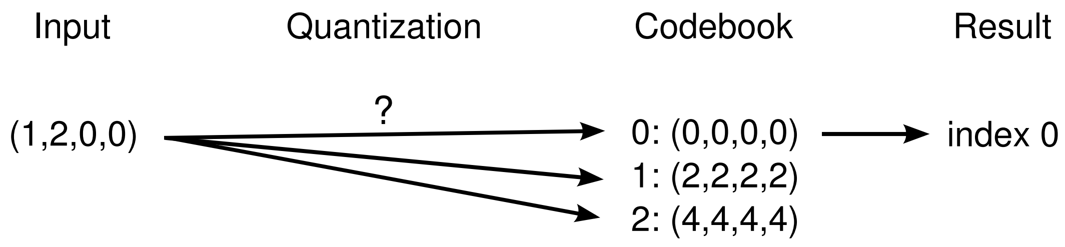


Figure 2.14: Visualization of vector quantization. A input vector is matched against a codebook in the proess of quantization. The index of the closest vector from the codebook is the result.

This thesis focuses on mapping the indices from the vector quantization to an alphabet and applying string analysis on the concatenated characters gathered by quantizing the input data. So, before the features can be described, firstly the alphabet has to be defined. The alphabet must be of at least the length of the used codebook, so that no information is lost. The number of characters is closely related to the number of codewords in a codebook. As the number of codebooks can grow bigger than 25 characters, a feasible solution is to use characters from the unicode character set for the alphabet. The actual alphabet used for the implemented features is discussed in the implementation chapter chapter 3.

### 2.4.1 String edit distance of vector quantized songs

The method presented in this section is inspired the works of (Casey and Slaney, 2006): using a string edit distance on a string of vector quantized audio. The method can be described by the following steps:

1. Compute vector quantizations of auditory images of songs
2. Compare characters using Levenshtein distance
3. The distance of the songs is the Levenshtein distance

In (Casey and Slaney, 2006) this method was used to show that temporal information is important. One experiment was to retrieve the used characters in a section, sort them in a stable manner to get rid of the temporal information and compare the resulting string using the Levenshtein distance (Levenshtein, 1966). The Levenshtein distance computes distances between two strings by measuring insertions, deletions and swaps of characters. For example the two strings “audo” and “auditory” have a Levenshtein distance of 5 as one swap and 4 insertions or deletions are necessary to convert one string into the other. The results presented in this paper have shown that this method performed poorly in comparison with methods which kept temporal information. However, this indicator function is used in this thesis to see what impact the Auditory Image Model may have on the results, as the image preserves temporal information. The method is described as follows:

1. Compute vector quantizations of auditory image of songs
2. Extract used characters
3. Sort characters to get rid of temporal information
4. Compute the distance using Levenshtein distance

The output of the vector quantization is mapped to an alphabet of some size. This output, a string, is then stripped of all content but the used characters. The resulting string is sorted, e.g. alphabetically, so that the result only depends on the characters occurring and not the order. Finally, a string edit distance method, e.g. Levenshtein distance, is used to generate the distance measure between the two songs.

This method is referred to as **string indicator** in the following sections.

The string indicator does not neither regard the temporal features of the symbols, nor when the characters are mentioned, in which pattern or with what frequency. For comparison a second method is introduced which regards temporal features.

The idea of the second string-edit distance method is, to compare the complete string of the song with the other song string:

1. Compute vector quantizations of auditory images of songs
2. Compute the distance using Levenshtein distance

This method is called **string distance** in the following sections.

To normalize the Levenshtein distance the following formula was used, despite being not a metric (De La Higuera and Mico, 2008):

$$d = \frac{d(s_1, s_2)}{max(s_1, s_2)} \tag{2.7}$$

The problem with the normalization above is that the triangle equality does not hold, as described by (De La Higuera and Mico, 2008). However, this may also be very well the case for musical similarity, as described by (Berenzweig et al., 2004):

> “The triangle inequality can be violated because of the multifaceted nature of similarity: for example, Michael Jackson is similar to the Jackson Five, his Motown roots, and also to Madonna. Both are huge pop stars of the 1980s, but Madonna and the Jackson Five do not otherwise have much in common.”

The results may or may not be improved by using normalizations as proposed by (De La Higuera and Mico, 2008) or (Marzal and Vidal, 1993).

### 2.4.2 Histogram of used characters in vector quantized songs

(Casey and Slaney, 2006) describes the use of histograms on the vector quantized data. (Foote, 1997) also uses a similar technique as well as (Ness et al., 2011). The basic method used is the following:

1. Compute vector quantizations of auditory images of songs
2. Count character or word frequency
3. Create histogram from frequencies
4. Compute distance of histograms

This feature makes use of the frequency of characters which is ignored by the string edit distance feature discussed previously, but still disregards temporal properties. This can be improved by using words rather than single characters when creating the histogram. However, the word approach is difficult for histograms to achieve, as all combinations of words have to be regarded in the vector which can grow quickly out of hand. Histograms as shown in Figure 2.15 are well suited for comparison using distance measures, as they are sorted by the alphabet and not too large.

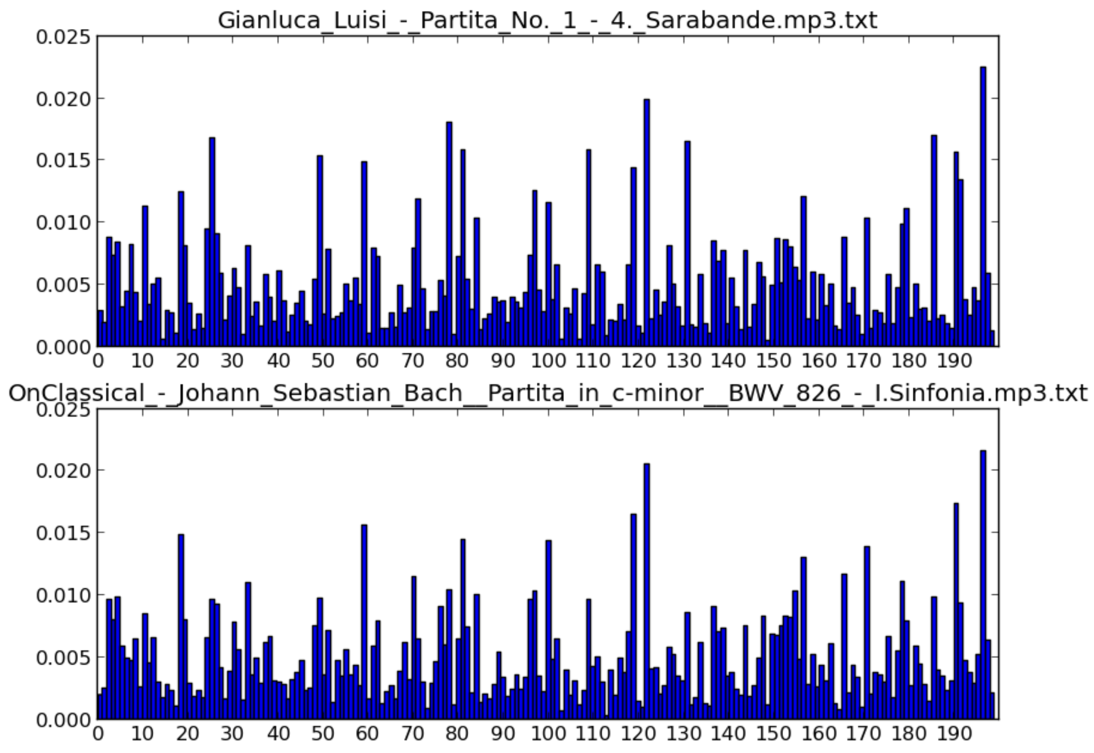


Figure 2.15: Histograms of the two least distant songs according to the survey, Jein and Mujschi. The x-axis represents the character index in the used alphabet.

There are certain distance measures throughout the literature which are promoted to give good results. Amongst them are the Euclidean distance, cosine distance and the Earth Mover's Distance of which the cosine distance performed best.

## 2.4.3 Pattern and redundancy in vector quantized songs

This section introduces the detection of patterns and sequences in the stream of characters produced by vector quantization. Finding those patterns may yield a way to detect the speed and rhythm properties of the song and may even result in catching, for example, the refrain or repeating guitar riffs or the like.

A general way to do this would be the following:

1. Compute vector quantizations of auditory images of songs
2. Find redundancies and patterns, e.g. by using a suffix tree
3. Compare pattern occurrences in other songs
4. Number of occurrences is a feature

Some algorithms described by (Crawford et al., 1998) could be used to detect patterns and repetitions. Once found, they may yield information about the beat. However, it is yet to be proven whether this may be necessary or even possible when using auditory images.

As this topic's outcome is not very clear, it is not further discussed in this thesis.

### 2.4.4 Mood State Machine

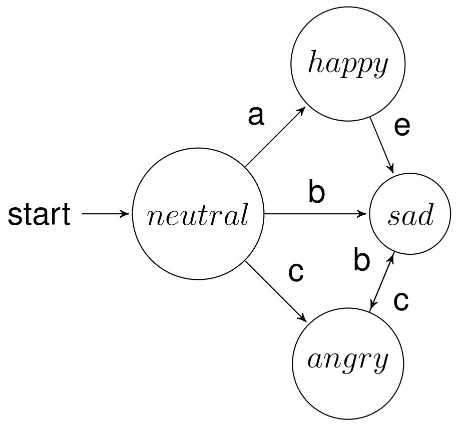


Figure 2.16: Examplary image of a mood state machine. States change to different moods due to the input characters.

This section introduces the idea of a mood state machine, which is a state machine constructed to react on inputs which would change the mood of the listener, as shown exemplarily in Figure 2.16. If, for example, the state machine is in neutral mood and an input classified as happy is given, the state machine would change it's state to 'happy'. To prevent permanent switches between the same states, thresholds, probabilities of transitions or stack based counts, which hold information about the amount of characters that led to the state, could be applied. A general use could be realized as follows:

1. Compute vector quantizations of auditory images of songs
2. Feed each generated character in a mood state machine

3. Read mood at the end of the song

As characters are generated from a window of auditory images, one can build a state machine, which maps single, or, combinations of characters to moods, when the following prerequisites are fulfilled:

1. Each character from the vector quantization process is comparable and adds information
2. It is known which character produces or influences the mood in what way
3. The mood of a song is not too much dependent on personality of the listener

The semantic of each character can be determined as it represents a certain time window in a song. Humans can then create ground-truth data for said time windows, which in turn is then used to either train a neural network for mood classification, or, train the quantizer directly so that the quantized characters carry mood information. This would solve point two as well.

Point three is optional, as there might exist a personalized mood state machine as well. However, a general idea of mood seems to exist amongst different human beings as well (Wood and O'Keefe, 2005) (Scherer and Zentner, 2001).

This idea was not realized in scope of this thesis, as it requires quite some time to even get the moods sorted out on a per song basis. The CAL500 (Turnbull et al., 2007) dataset may be a good resource for test data as it contains mood data.

# 3 Implementation

This chapter discusses the implementation of the features introduced in section 2.4 including the needed tools. It also gives an overview of Marsyas' auditory image model implementation in section 3.2 as this helps to understand the workings of the Auditory Image generation in general.

After describing the inner workings of Marsyas' auditory image model implementation, the process of test data acquisition and processing is discussed. Handled processing topics are, amongst others, the mapping of quantizations to characters and already available tools for analyzing the data. Following that, section 3.4 describes the general implementation layout, which tools were created and how the tools interact with each other. Finally, the implementation of the string distance, indicator and histogram features is discussed, including a performance analysis.

## 3.1 Evaluated Software Tools and Libraries

While writing the analysis software, certain audio and mathematical toolkits were examined. This section is an evaluation of tools suitable for implementing the features discussed in section 2.4.

### 3.1.1 Bregman

Bregman[1] is a toolkit for various music information retrieval tasks and supports basic psychoacoustics like the critical bands and the Bark scale. Support for the Bark scale was appealing as it was used successfully in (Wood and O'Keefe, 2005). It offers many examples and tutorials as well, which makes it easy to get into using the toolkit. However, in contrast to Marsyas the Auditory Image Model was not supported directly which renders Bregman unsuitable.

### 3.1.2 Marsyas

(Tzanetakis, 2007) describes Marsyas[2] as "an open source audio processing framework with specific emphasis on building Music Information Retrieval systems". It can be used for rapid prototyping and has a comprehensive selection of sound analysis modules, for example the Auditory Image Modules described in section 3.2. Benefits of Marsyas are its modular structure and the direct support of MP3 files, as well as the sufficient amount of examples. Another feature of Marsyas is the support for Python, which enables the use of suites like SciPy, which is introduced later in this section.

The used version of Marsyas at the time of writing is the Subversion revision r5064. Custom patches were applied to circumvent a bug which prevents double buffering of the auditory image, as described in section 3.2.5, and to prevent a bias to zero values when the song sample rate is to small as described in subsection 3.2.7.

[1] http://bregman.dartmouth.edu/bregman/
[2] http://marsyas.info/

### 3.1.3 AIM-C

*AIM-C*[3] is the main implementation of the works described by (Patterson et al., 1995). While it is the reference implementation for the Auditory Image Model and partly supports some compressed music file formats, it does not offer a Python interface and lacks the flexibility of Marsyas.

### 3.1.4 SciPy /NumPy

SciPy[4] is a rich open-source library for the Python language. It implements various methods suited for statistical analysis, mathematical problems, etc. The library depends on NumPy, which offers efficient processing of numerical arrays in Python. These Python modules alone are not sufficient for audio analysis. Thus the Python support for Marsyas was used to combine SciPy and Marsyas.

In this thesis, these tools are used for quick testing of distance measurements, visualization of results and for the main implementation of the AimVQ method, as described later in this chapter.

The version of SciPy used in this thesis was 0.10.1 and NumPy was at version 1.6.2.

### 3.1.5 Python Levensthein

python-Levenshtein[5] was used to compute the Levenshtein distance in the implementations of the features. As it offers a native implementation for the Levenshtein distance in C the efficiency exceeds Python implementations.

The version used was 0.10.2.

[3]http://code.google.com/p/aimc
[4]http://www.scipy.org/
[5]http://pypi.python.org/pypi/python-Levenshtein/

## 3.2 The AimVQ process in Maryas

In this chapter, the generation of an auditory image is explained using Marsyas' AIM-C implementation as an example. As Marsyas' auditory image model modules are used for the implementation of the features discussed in this thesis, this section also describes the data retrieval process for the features. The Auditory Image Model modules for Marsyas were ported from the original AIM-C software in scope of (Lyon et al., 2010).

### 3.2.1 Overview

The basic functionality of the auditory image generation process can be summarized by the following semantical tasks:

1. Simulate the hearing of humans using an auditory image
2. Cut the data into processable boxes
3. Use Vector Quantization on the data in the boxes

The process of analysis consists of chaining a list of modules, each module generating an output matrix from an input matrix, in the following order:

1. Sound File Source
2. AimPZFC
3. AimHCL
4. AimLocalMax
5. AimSAI
6. AimBoxes
7. AimVQ

The modules from the listing above are described in the following subsections, except for the sound file source, which has nothing to do with the actual processing. Before diving deeper into the module descriptions, the following code example aims to give an overview over how the AIM modules are used in combination with Marsyas. The presented code plots a histogram of the used codewords as done in (Ness et al., 2011). The implementation language is Python, as it is very easy to understand without much knowledge about the syntax.

The example program, split up in three parts for the sake of clarity, begins with importing the required modules. When Marsyas is compiled with Python support, two modules are available: `marsyas` and `marsyas_util`. The first being the bindings to the native library and the second being an utility module which offers, amongst other things, helper methods for creating a Marsyas network from lists and conversion methods for Marsyas vectors to NumPy vectors.

Listing 3: First part

```
import marsyas
import marsyas_util
import numpy
import sys
import pylab

# Network definition: 'net' is a series of modules.
aimNetwork =  ["Series/net", [
          "SoundFileSource/asrc",
          "AimPZFC/aimpzfc",
          "AimHCL/aimhcl",
          "AimLocalMax/aimlocalmax",
          "AimSAI/aimsai",
          "AimBoxes/aimBoxes",
          "AimVQ/vq",
        ]]
```

The Marsyas network is defined using lists, where each network component, for example the component of type `Series`, which is named `net`, has a list of modules or sub-networks. In this case there is only one `Series` module, which contains all the modules as described at the beginning of this section. The next step is to build the network from the list definition using `marsyas_util.create(aimNetwork)`, which registers and sets up the modules in Marsyas. After setting up the network, the code expects a filename as first parameter of the script, which is the input audio file. Said input filename is then fed to the `SoundFileSource` module using `net.updControl()`, which can set variables by supplying a string, which encodes the path to the module in the network as well as the type and name of the variable to update. In this case the string

`"SoundFileSource/asrc/mrs_string/filename"`

addresses a variable with the name `filename` of type `mrs_string` from the `SoundFileSource` module named `asrc`.

Listing 4: Second part

```
# Compile the network in Marsyas.
net      = marsyas_util.create(aimNetwork)
# Read a file from the command line (first parameter).
filename  = sys.argv[1]

# Use a smaller samples per window size to speed calculation up.
net.updControl("mrs_natural/inSamples", 512);
# Feed the file name to the SoundFileSource.
net.updControl("SoundFileSource/asrc/mrs_string/filename", filename)
```

This is the only setup the network needs to be run. The next step is initializing the histogram creation loop and the display of the results using the `pylab` module.

Listing 5: Third part

```
firstRun = True
histogram = []

# Loop as long as the sound file source has data available.
while net.getControl("SoundFileSource/asrc/mrs_bool/hasData").to_bool():
  # Run the network for one processing tick.
  net.tick()

  # Get data from the network and convert it to a numpy array.
  vq = net.getControl("mrs_realvec/processedData").to_realvec()
  vq = marsyas_util.realvec2array(vq)[0]

  # Accumulate the sparse vectors from the AimVQ module.
  if firstRun:
    histogram = vq
    firstRun = False
  else:
    histogram = histogram + vq

# Show a bar plot with the accumulated sparse vectors (a histogram).
pylab.bar(range(len(histogram)), histogram, width=1)
pylab.show()
```

The while-loop, which runs as long as the SoundFileSource in the network has data available, calls the network once per iteration and reads the available data. Said data is a realvec, a vector type used to pass information between Marsyas modules. The realvec, identified as vq in the code, is the sparse vector produced by the AimVQ module. The produced vector is 8800 entries long and represents the codewords of 44 codebooks, each having 200 codewords. The values of the vector are mostly zeros, a one indicates that the codeword at this index was used. For example, if every quantization results in the first codeword, every 200th index of the 8800 entries long vector would contain a 1 and every other value would be 0. This vector is then summed up, effectively creating a histogram of each codeword. At the end, this vector is printed using a bar plot provided by the pylab module. Figure 3.1 shows the resulting histogram of 8800 values.

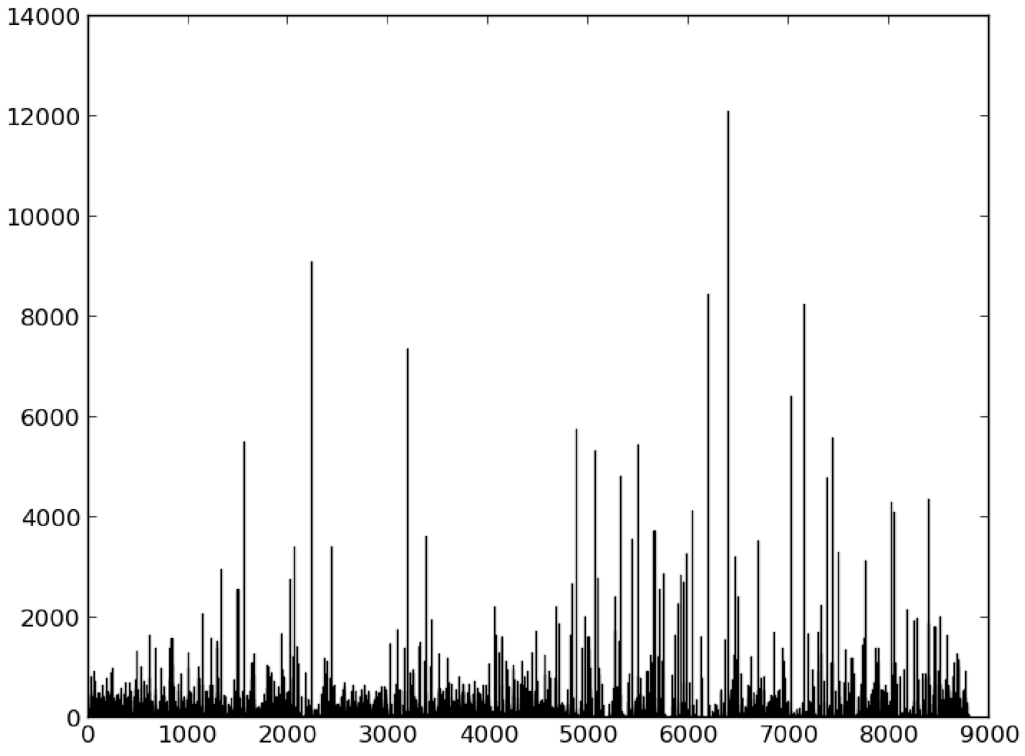


Figure 3.1: The output of the example program: a histogram of 8800 codewords for the song Spaspet used in the survey described in section 2.2.

### 3.2.2 AimPZFC

#### Dick Lyon's Pole-Zero Filter Cascade

Usually gammatone filters are used in AIM-C to simulate the basilar membrane motion (Patterson et al., 1995). (Lyon, 2011) introduces the pole-zero filter cascade which uses cascaded pole- and zero crossing filters and "provide a good match to human psychophysical and physiological data." (Lyon, 2011). The PZFC was introduced to the Auditory Image Model by (Walters, 2011). While there is an AimGamma module in Marsyas, the AimPZFC module is used because it was used successfully in related experiments like (Walters et al., 2012).

#### Expected Input Matrix

An array of samples from a sound file is expected to be the input, e.g. from the SoundFileSource module.

#### Produced Output Matrix

The output matrix is $n$-dimensional, with $n$ being the number of supported frequency channels. These channels are limited by the maximum and minimum center frequencies, which default to 6kHz and 100Hz respectively. However, the actual number of channels is computed from more parameters than that.

The structure of the output matrix is shown in the matrix below, filled with dummy values for illustration purposes. The rows labeled with $c$ refer to a frequency channel and the $cf$ rows refer to the respective center frequencies of the channels. The columns, labeled with $s$, correspond to the the samples from the song.

$$\begin{array}{c|cccc}
 & s_0 & s_1 & ... & s_n \\
c_0 & 1 & 1 & ... & 1 \\
c_1 & 1 & 0 & ... & 1 \\
... & 0 & 1 & ... & 1 \\
c_n & 0 & 0 & ... & 1 \\
cf_0 & 1 & 1 & ... & 1 \\
cf_1 & 1 & 0 & ... & 1 \\
... & 0 & 1 & ... & 1 \\
cf_n & 0 & 0 & ... & 1
\end{array}$$

### 3.2.3 AimHCL

#### Half-wave rectification, compression and low-pass filtering

In short, the Half-wave rectification Compression Low-pass filtering (HCL) module is used to generate a Neural Activity Pattern (NAP) from a gammatone filterbank (Patterson et al., 1995), or, in this setup, from the pole- zero filter cascade. In case of the pole-zero filter cascade, only half wave rectification and low-pass filtering are needed to simulate the motion of the hair cells in the inner ear that are responsible for translating the mechanical motions to neural excitement. This process then results in the neural activity pattern which represents the state of the neurons when excited by the mechanical motions of the basilar membrane at a given time, as shown in Figure 3.2. The used frequencies are Equivalent Rectangular Bandwidth (ERB) frequencies, which simulate the bandwidths of the filters in the inner ear using band-pass filters as simplification.

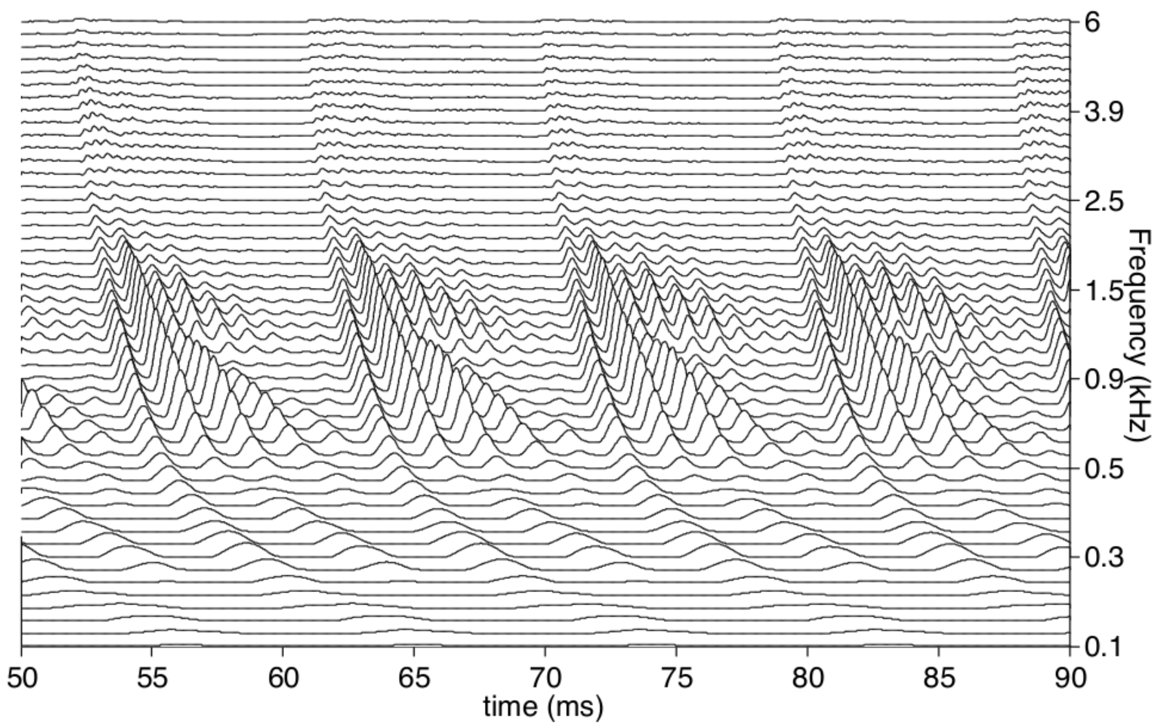


Figure 3.2: An example Neural Activity Pattern "for a short segment of the vowel /a/" (Walters, 2011). Each frequency channel corresponds to a neuron. The amplitudes per channel show the excitement of the corresponding neuron. Image credit: (Walters, 2011).

The neural activity pattern, as seen in Figure 3.2, is shifted to the right the lower the frequency channel gets. This phase shift is not wanted and has to be stabilized. For this purpose the following modules are needed.

#### Expected Input Matrix

The output matrix from AimPZFC or AimGamma is expected, which holds filtered values of an audio signal.

#### Produced Output Matrix

The output matrix has the same structure as the one produced by AimPZFC. The channel values are changed by this module, but the center frequencies are left untouched.

The number of channels is computed from the number of input rows divided by two. This works, as the output coming from AimPZFC has $channels \cdot 2$ rows (channel values and channel center frequency values).

### 3.2.4 AimLocalMax

#### Local maximum strobe criterion: decaying threshold with timeout

The NAP generated from the HCL module is not stable. That is, there is a phase shift which grows the broader the frequency bandwidth of the corresponding filter is. (Patterson, 1987) and the aim95 manual state that “this phase lag has to be enormous ($>$ 4ms) to affect what we hear; indeed, reversing the phase lag with synthetic stimuli does not change what we hear (Patterson, 1987)” (Patterson and Bleeck, 1995). As a result of these findings, “Phase information that appears in the basilar membrane motion but which we do not hear, is removed in the third module by the strobe mechanism of the temporal integration process.” (Patterson and Bleeck, 1995). This means that the phase shift is corrected using a temporal integration process, of which the first step is done by this module.

To align the phases, strobes are searched. Said strobes can then be used to align the frequency channels along the strobes. They are found by searching for local maxima in the signal of the channel. The details of this and others methods are discussed and compared deeply by (Walters, 2011). This module only finds the strobes, whereas the next module, AimSAI, correlates the strobes with the signal to produce a stable auditory image.

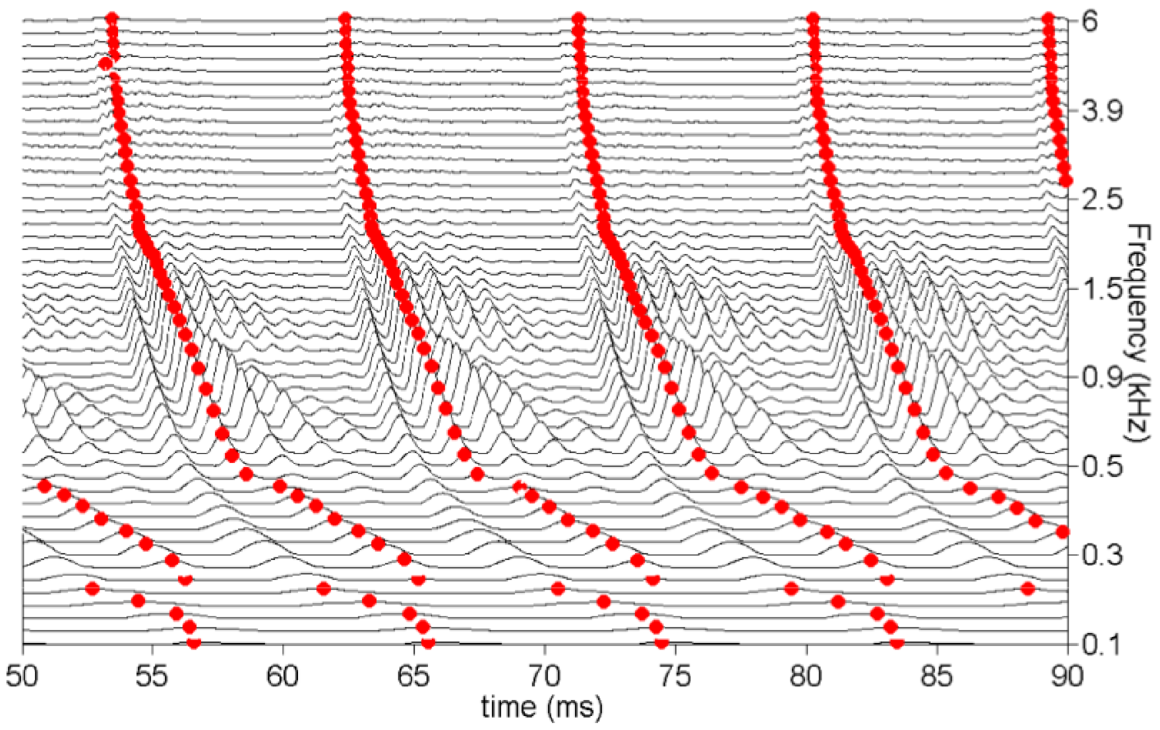


Figure 3.3: Found strobes in a NAP are highlighted red. Image credit: (Walters, 2011).

### Expected Input Matrix

The output matrix from AimHCL is expected, containing half-wave rectified and additionally filtered audio signal data.

### Produced Output Matrix

The following visualizes the generated output matrix. While the values are actually dummy values, it is to note that the strobe values are either zero or one, where a one indicates a strobe.

$$\begin{array}{c c}
 & \begin{array}{cccc} s_0 & s_1 & ... & s_n \end{array} \\
\begin{array}{c} c_0 \\ c_1 \\ ... \\ c_n \\ cf_0 \\ cf_1 \\ ... \\ cf_n \\ st_0 \\ st_1 \\ ... \\ st_n \end{array} &
\begin{bmatrix}
1 & 1 & ... & 1 \\
1 & 0 & ... & 1 \\
0 & 1 & ... & 1 \\
0 & 0 & ... & 1 \\
1 & 1 & ... & 1 \\
1 & 0 & ... & 1 \\
0 & 1 & ... & 1 \\
0 & 0 & ... & 1 \\
1 & 1 & ... & 1 \\
1 & 0 & ... & 1 \\
0 & 1 & ... & 1 \\
0 & 0 & ... & 1
\end{bmatrix}
\end{array}$$

The produced output matrix of the AimLocalMax module now has three times the number of frequency channels in rows. The first third is the signal ($c$ rows), the next third are the center frequencies ($cf$ rows) and the third block consists of the strobes calculated ($st$ rows) in this module.

## 3.2.5 AimSAI

### Stabilised Auditory Image

As already elaborated in the AimLocalMax (subsection 3.2.4) module description, the Neuron Activity Pattern generated by the AimPZFC (subsection 3.2.2) and

AimHCL (subsection 3.2.3) filterbanks, as seen in Figure 3.2, introduces a phase lag and is therefore not stable. In this case stable means that the loudness of a periodic sound is fixed and not fluctuating, which would not be the case with the phase lag.

> "In the high-frequency channels, the filters are broad and the glottal pulses generate impulse responses which decay relatively quickly. In the low-frequency channels is the `phase lag`, or `propagation delay`, of the cochlea, which arises because the narrower low-frequency filters respond more slowly to input." (Patterson et al., 1995)

Low-frequency channels are lagging behind the high-frequency channels, and therefore the image needs stabilization to compensate the lag. For this reason strobes are generated by AimLocalMax. These strobes are used in a way described in detail by (Walters, 2011, p. 100):

> "The stabilised auditory image (SAI) is a two dimensional representation of an input sound. A single SAI is a snapshot of the audio in a short window around a point in time. The SAI changes continuously with time, and successive snapshots can be concatenated to make a movie of these two dimensional frames. The first dimension of an SAI frame is simply the spectral dimension added by the filterbank. The second dimension comes from the strobed temporal integration process by which an SAI is generated. Strobed temporal integration works by locating prominent peaks, or 'strobes', in the incoming signal and calculating 'lags' relative to these times. These peaks are most commonly associated with the pulses in pulse resonance sounds, for example the glottal pulses in speech. When a strobe occurs in a channel, a short segment of the signal following the peak in that channel is added to a buffer, starting at zero lag. The signals following multiple strobe points add constructively in the buffer. This process leads to a stable spectrotemporal representation of the microstructure in the signal following each pulse in the input sound."

To summarize, the procedure of phase alignment is done to compensate the phase shift in the NAP. There is a buffer for each frequency channel which starts at zero lag and each time a strobe occurs. A segment of the signal with the strobe is added to the buffer. This is done for each frequency channel, so that strobes are synchronizing the auditory image. The buffer length in the implementation defaults to 11.63266 ms. Figure 3.4 shows how a NAP looks after the phase alignment.

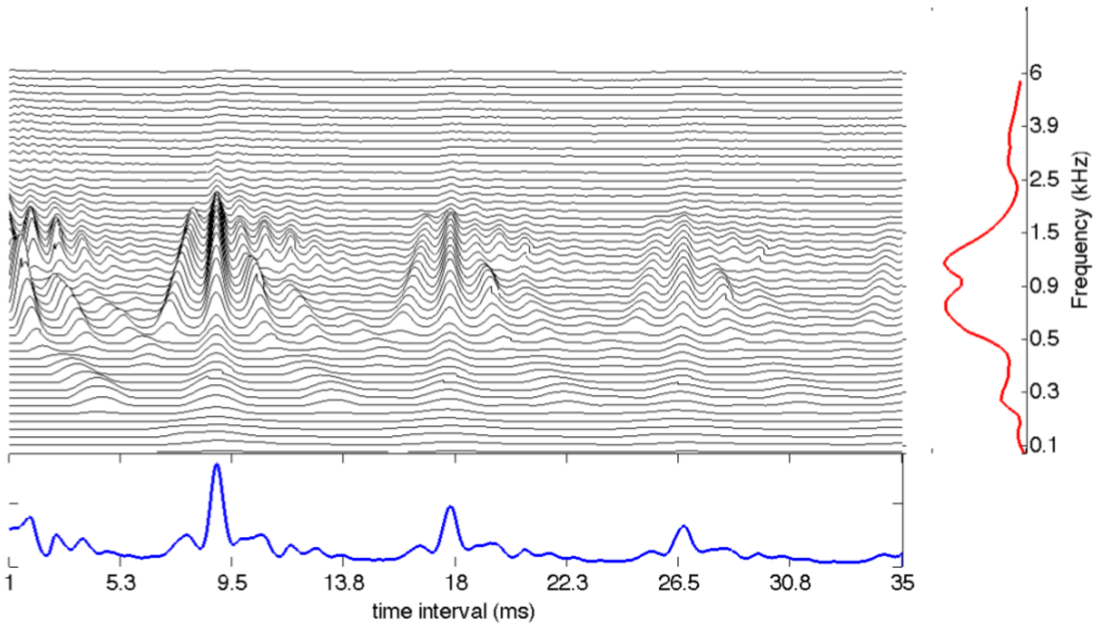


Figure 3.4: Stabilized Auditory Image. Image credit: (Walters, 2011).

### Expected Input Matrix

AimSAI expects the output matrix from AimLocalMax, which contains the filtered audio signal data as well as found strobes for correlation.

### Produced Output Matrix

The output is a $m \times n$ matrix where $m$ is the number of frequency channels and $n$ is the number of samples per frame, which depends on the number of input samples and the configured frame period in milliseconds, which defaults to 11.63266 ms, the same value as the maximum strobe delay.

$$\begin{array}{c|cccc|} & s_0 & s_1 & ... & s_n \\ \hline c_0 & 1 & 1 & ... & 1 \\ c_1 & 1 & 0 & ... & 1 \\ ... & 0 & 1 & ... & 1 \\ c_n & 0 & 0 & ... & 1 \end{array}$$

### Broken double buffering

At the time this thesis was carried out, there was a bug in AIM-C and Marsyas' AIM implementation, which prevented double buffering of the SAI. This leads to slightly different results, as there are less samples being generated from the same boxes. Figure 3.5, generated in scope of the string indicator implementation (subsection 4.2.2), compares the unbuffered and double buffered quality of performance. The figure is generated using the absolute difference between two matrices: the averaged distances of the participants of the survey and the distances calculated by the string indicator method. The absolute difference of the matrices is then averaged for each of the codebooks. Said average value is then used for comparison where lower values indicate better results, as the mean difference to the survey results is lower.

If the process works with double buffering, as it is intended to, results are only populated to the next module when a full image is rendered. Without the double buffering working properly the output matrix is constantly updated without waiting for a full image. This leads to more intermediate images or tearing and thus to noisy data. While the general direction of the results did not change significantly, the quality of some results improved, which can be seen by the local minimum at $x = 38$.

Detailed implementation specific information is available online in the AIM-C issue tracker[6].

## 3.2.6 AimBoxes

### 'Box-cutting' routine to generate dense features

The box-cutting is a very important step, as it reduces the high dimensional Stabilizied Auditory Image into a single vector of, in case of this thesis, $temporalSize + spectralSize = 32 + 16 = 48$ values, which can be matched easily afterwards using vector quantization. The dimensions are the same as they were used for base line testing in (Lyon et al., 2010):

> In our baseline we use rectangles of size 16 $\times$ 32 and larger, each dimension being multiplied by powers of two, up to the largest size that fits in an SAI frame. (Lyon et al., 2010, p. 29)

[6]http://code.google.com/p/aimc/issues/detail?id=4

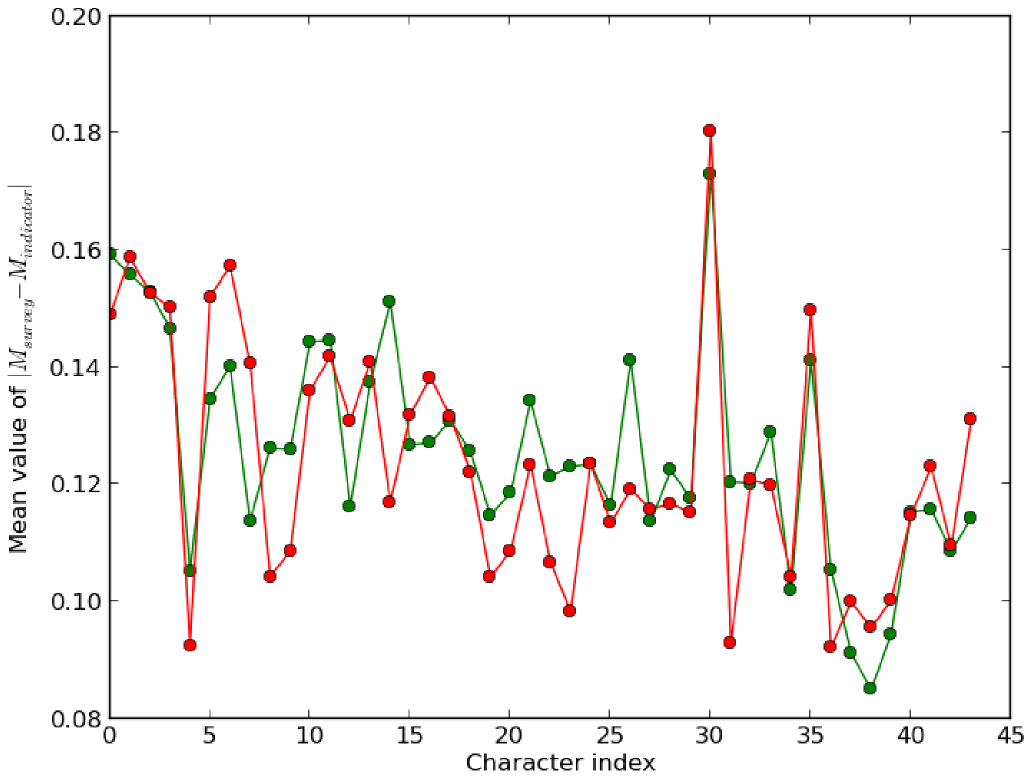


Figure 3.5: String indicator results for different codebooks. The absolute difference of the string indicator feature result matrix ($M_{indicator}$) to the survey results matrix ($M_{survey}$). The red line represents the unbuffered results whereas the green results are double-buffered.

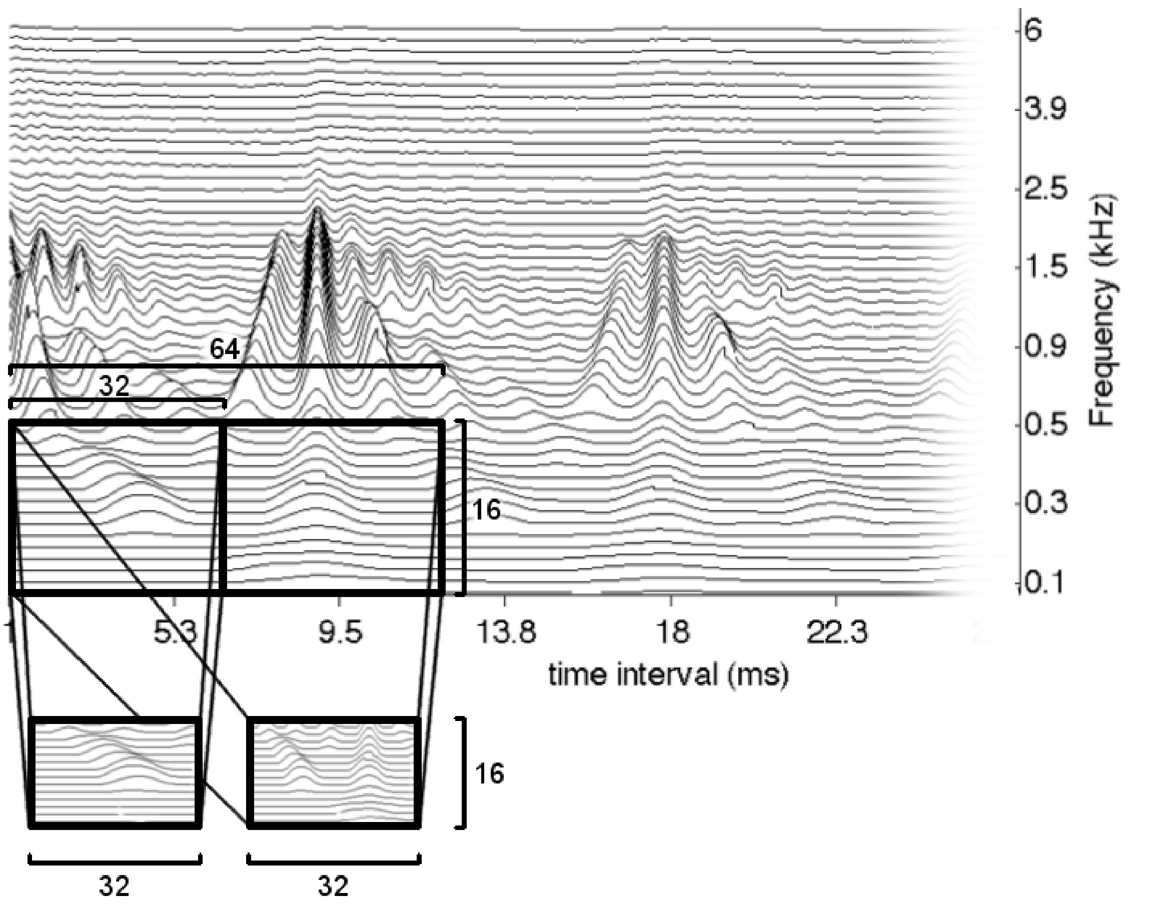


Figure 3.6: Box cutting of a Stabilized Auditory Image into two $32 \times 16$ boxes with different input areas. The first, most left, box uses $32 \times 16$ input values, whereas the second box uses $64 \times 16$ input values from the SAI.

The boxes itself are always fixed in width and height: $32 \times 16$ $(width \times height)$ values in this case. The width determines how many values in the lag/temporal dimension are covered. The height defines how many frequency channels, or neurons, are covered. However, to cover all possible details of the SAI, multiple box input areas with different resolutions are used, so that "different box sizes and shapes capture both different large-scale image structure, corresponding to pitch and temporal coherence, and the micro structure corresponding to the resonances following each pulse. Wide boxes capture long-term temporal patterns; [. . . ]" (Lyon et al., 2010, p. 30).

While, for example, small temporal coverage may suffice for tiny changes in the signal, the broader development of the signal gets lost. The idea is to create boxes with growing input areas, as shown in Figure 3.6. In the figure there is one box which uses an input area of $32 \times 16$ values and a second one, which uses twice as much area in temporal direction. The second area would catch twice as much temporal information and has therefore more information about the temporal development of the signal. Said input areas are always reduced to the box size of $32 \times 16$ values. The input area extends in temporal direction by powers of two and is limited by the length of the SAI, which itself is limited by the sample rate of the sound. Additionally to temporal growth, the spectral side of the input area grows as well. Instead of powers of two, the area is moved up in the spectral values by the size of itself. After the top of the SAI is hit by the input area, the process starts anew at the bottom but with the spectral input area doubled in size.

Because the values of the input area are fitted into the box, which is usually smaller than the input area, the overlapping values are reduced to a single value using the mean value. For example, an input area of $64 \times 16$ values does not fit into a $32 \times 16$ box as there are twice as much temporal values. The overlapping temporal values, two for each value in the target box, are then averaged and the resulting single value can be put into the box without problems.

### Expected Input Matrix

The output matrix from AimSAI is expected.

### Produced Output Matrix

In the following visualization, a row labeled with a $c$ represents a box column and a $r$ row represents a box row respectively. The column $b$ labels stand for box. As in the examples before, the values in the matrix are just dummy values.

$$\begin{array}{c|cccc|} & b_0 & b_1 & ... & b_k \\ c_0 & 1 & 1 & ... & 1 \\ c_1 & 1 & 0 & ... & 1 \\ ... & 0 & 1 & ... & 1 \\ c_n & 0 & 0 & ... & 1 \\ r_0 & 1 & 1 & ... & 1 \\ r_1 & 1 & 0 & ... & 1 \\ ... & 0 & 1 & ... & 1 \\ r_m & 0 & 0 & ... & 1 \end{array}$$

With a box size of $32 \times 16$, $n$ is the amount of columns, which is equal to the temporal size of the box and therefore corresponds to the value 32. The value $m$ on the other hand equals 16, being the number of rows. The box count $k$ depends on the audio input. Different dimensions of the SAI, for example lower width due to less samples, shrinks the amount of possible boxes cut from the SAI.

Every box generated consists of $spectralSize + temporalSize = 16 + 32 = 48$ rows, first the averaged columns and then the averaged rows.

### 3.2.7 AimVQ

#### Vector Quantization via kNN trees

Box cutting and using more than one codebook for the different boxes is done to catch the temporal development of the SAI. By splitting the image in several smaller boxes, each emphasizing on a specific region, the temporal information is saved in various resolutions. Therefore, each quantization is influenced by different portions of the SAI. This technique is described in (Faundez-Zanuy and Pascual-Gaspar, 2011).

#### Expected Input Matrix

AimVQ expects as input a matrix like the one that AimBoxes generates.

## Produced Output Matrix

The output matrix is a series of concatenated sparse vectors generated from the input boxes. Each vector consists of zero values except for a single value which is one. (Ness et al., 2011) uses concatenated sparse vectors to build a histogram of the used words. The following example shows a possible output matrix. $m$ denotes the number of input boxes and $n$ the number of values per box. $n$ is a known value, as it is the spectral/temporal size mentioned above, being 48 values.

$$\begin{array}{c|c|}
 & sparsevector \\
b_{0,0} & 1 \\
b_{0,1} & 0 \\
... & 0 \\
b_{0,n} & 0 \\
b_{1,0} & 1 \\
b_{1,1} & 0 \\
... & 0 \\
b_{1,n} & 0 \\
... & ... \\
b_{m,0} & 1 \\
b_{m,1} & 0 \\
... & 0 \\
b_{m,n} & 0
\end{array}$$

## Zero Value Bias

The AimVQ module expects the input matrix to have at least as much boxes as there are codebooks. If this is not the case, which may be when having down-sampled music, AimVQ takes an vector filled with zeros, that the vector quantizer then translates to an index. This results in a bias on a specific character when not treated. In the scope of this thesis a patch to prevent this behavior and to just abort the computation was applied.

### Characteristics of the codebooks

The codebooks are the one presented in (Ness et al., 2011), which are optimized for music retrieval. The codebooks were generated from a sound database containing environmental sounds like crickets, audiences laughing, etc. The following quote describes the process of generating the codebooks:

> “We first preprocessed a collection of 1000 music files from 10 genres using a PZFC filterbank followed by strobed temporal integration to yield a set of SAI frames for each file. We then take this set of SAI and apply the box- cutting technique described above. The followed by the calculation of row and column marginals. These vectors are then used to train dictionaries of 200 entries, representing abstract “auditory words”, for each box position, using a k-means algorithm.” (Ness et al., 2011, p. 9–10).

## 3.3 Data Retrieval

Having data to test and to guide the direction of the implementation is important. Thus, this section explains which songs were used as test data and introduces the methods to retrieve the test data from said songs.

### 3.3.1 Test Data

For testing the quality of the features, the songs of the previous user study are used as a reference, as well as the similarities found in the study.

Furthermore, the covers80 dataset (Ellis, 2007) is used, which provides 80 pairs of cover songs. This dataset has the advantage of not needing human reference data, as song covers can be assumed to be similar. The disadvantage is that even covers can be so different that a human would not judge them as similar. The first ten songs from the covers80 database were used, resulting in 20 songs being compared. As there is no reference data but the cover songs themselves, which are supposed to have a high similarity for their covered partner song, an ideal matrix is used for comparison. Said matrix has value 0.0 for each cover song pair and 1.0 everywhere else. This is then compared to the result of the algorithm. In the string indicator implementation (subsection 4.2.2) the impracticality of this approach is discussed.

All songs used for testing were sampled with 44kHz and 512 samples per window.

The reference data and the data from the algorithms mostly differ in scale, that is, they have different maximum values, which makes them difficult to compare. This was compensated by fitting the algorithm data to the maximum value of the reference data. At first, the results were fitted to the matrix with the higher maximum value, but it soon became apparent that this does not yield comparable results, as the algorithm data varies and therefore does the source of the fitting. Consequently, the distance matrix produced by the algorithm was normalized to the distance matrix of the survey using the following formula:

$$M_{comparable} = M_{algorithm} \cdot \frac{max(M_{survey})}{max(M_{algorithm})} \tag{3.1}$$

### 3.3.2 Alphabet

As the features presented in section 2.4 mainly operate on strings, it is necessary to map the audio data, or more specifically, the Stabilized Auditory Image (SAI) to characters. To do this, an alphabet is required, which serves as a translation table between vector quantization indices and characters.

The length of the alphabet is determined by the amount of words per quantization codebook. The AimVQ module uses 44 codebooks with 200 words each. Consequently, the bare minimum to represent one codebook is an alphabet of 200 characters, one for each codeword. Characteristics of the used codebooks are described in subsection 3.2.7.

The formula for how many characters are needed for a given number of codebooks $n$ is the following:

$$n \cdot 200 \tag{3.2}$$

For example, the first codebook may be mapped to the characters in $[0; 200)$ where the second codebook uses the characters in $[200; 400)$ and so on. To map each quantization to it's own codebook, one would require an alphabet with the size of $|quantizations| \cdot |codebooks| = 44 \cdot 200 = 8800$ characters.

The alphabet is constructed from a string of printable unicode characters. This string is sorted alphabetically and consists of all characters in the unicode classes `Lu`, `Ll`, `Lt`, `Lm`, `Lo`, `Nd`, `Nl`, `No`, `Pc`, `Pd`, `Ps`, `Pe`, `Pi`, `Pf`, `Po`, `Sm`, `Sc`, `Sk` and `So`, which are printable and punctuation characters[7]. Using unicode characters has the benefit of being able to scale the size of the alphabet up to the maximum number of 8800 characters just by adjusting the upper limit of the alphabet. For example, when using only 200 characters for the alphabet, one would only use the 200 first characters of the unicode string.

The actual mapping of the codebook to the alphabet is described in each experiment.

[7] http://www.unicode.org/reports/tr44/#GC_Values_Table

### 3.3.3 Translation modes

As the alphabet described in the previous chapter is limited in length and the quantized indices range from zero to the length of the quantization vector (8800 entries with the used implementation, 44 codebooks and 200 codewords each), the output must be translated to the alphabet in some way. This section describes the methods to map the quantization output to the alphabet. The general process of translation is shown in Figure 3.7.

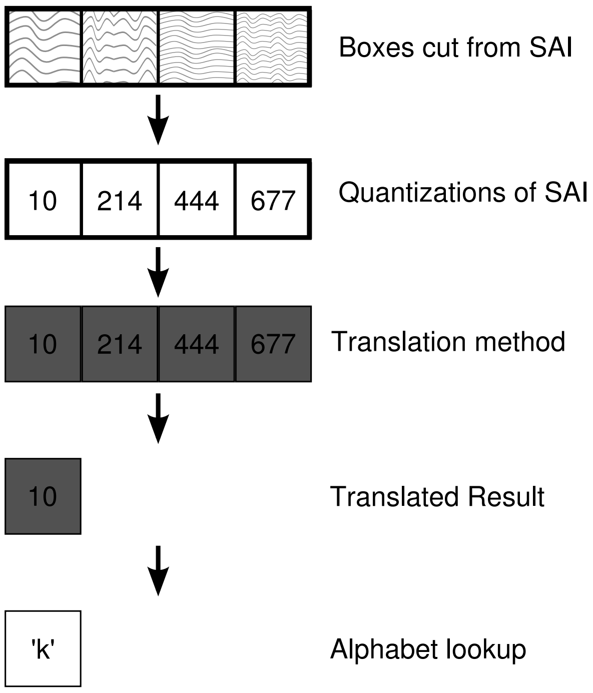


Figure 3.7: Visualization of the quantization vector translation as well as the mapping of the translated vectors to the alphabet. The translation method used in this example is the **single** method with box index 0, therefore only the first box is used.

Every translation mode used for the implementation of the various features is described in its own subsection below. The subsection title used in the following sections is the common name of the method.

#### Single

Use only one selected quantization from the available quantizations and discard the rest, as shown in Figure 3.8. This method maps each used quantization codeword to a character from the alphabet. The influence of this single feature depends on the selected index, as the index effectively represents a box cut from the SAI. Boxes from the SAI are differently sized before cut down to the actual box size as described in subsection 3.2.6, so it is likely that each quantization performs differently, depending on the box size.

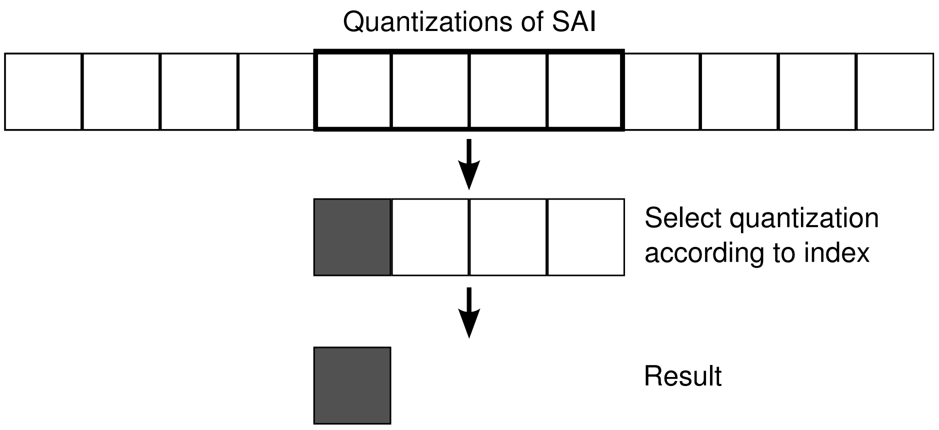


Figure 3.8: Example of the Single translation mode. Only one selected quantization, the first one in this case, is returned.

### Accumulate

Summarize all indices yielded by the quantization and take the result modulo alphabet size. The process is visualized in Figure 3.9. This method yields most or all characters from the alphabet, as the sum of the indices modulo alphabet size is very likely to repeat itself constantly, resulting in the same characters. Only an alphabet of $|quantizations| \cdot |codebook| = 44 \cdot 200$ would prevent the characters from reoccurring. This method distributes all quantizations, and therefore all different views on the SAI, to one string. Therefore, this method unifies all different box sizes and quantizers and should perform well.

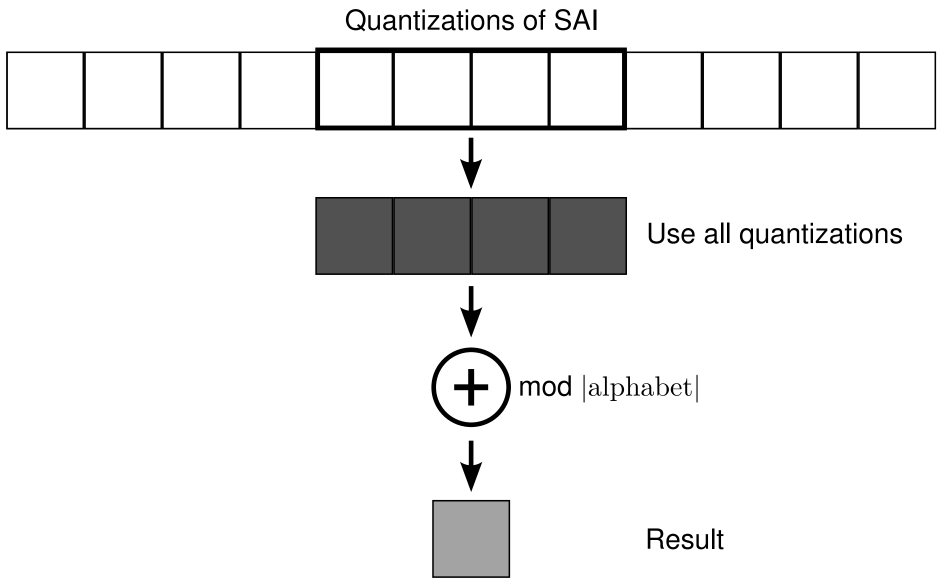


Figure 3.9: Example of the Accumulate translation method. The resulting indices from the quantization process are summarized and the result is taken modulo the size of the alphabet.

### EachIndex

Instead of generating only one character per quantization, all quantized characters are added. This is done by iterating over the found indices and getting the character from the alphabet at position $index$ mod $alphabetSize$. A visualization of this process is given by Figure 3.10. This method has the same problem as Accumulation, i.e., most of the characters from the alphabet will be used and therefore an alphabetically sorted string of used characters has little information when compared to equally extracted strings. However, using this method to pre-process the songs the Single method as well as the Accumulate method can be derived from the produced data. Therefore, this method is used for calculating the quantizations, as described in the next section.

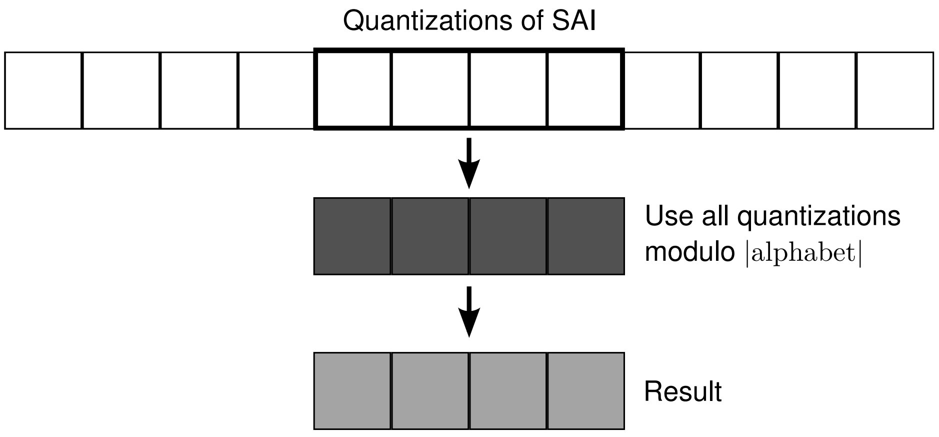


Figure 3.10: Example of the EachIndex translation method. All quantizations are returned.

The data is precomputed from the audio using the EachIndex method, with which all other methods can be derived from. Single can be derived from EachIndex by taking the n-th character of each word. Accumulate can be achieved by summarizing the index of each character in each word and getting the new character from the alphabet using the accumulated index. The next section describes how this precomputation was done.

### 3.3.4 Data Extraction

This section brings together the methods and formalities discussed before and summarizes the process of converting a song into a dataset of characters from an alphabet.

The resulting implementation in Python utilizes Marsyas to create a network of the Auditory Image Components described in section 3.2. For completeness the network is listed again in compact form:

1. SoundFileSource - The song to analyze
2. AimPZFC - Simulating Basilar Membrane Motion
3. AimHCL - Generating a Neural Activity Pattern
4. AimLocalMax - Detecting stabilization strobes
5. AimSAI - Generating a Stabilized Auditory Image
6. AimBoxes - Cutting down to uniformed vectors

7. AimVQ - Quantize the vectors

The tool to extract the data accepts three important parameters: the input filename, the name of the output file where the analyzed data will be stored and the size of the alphabet. While processing a song it runs the Marsyas network and collects the output of the AimVQ module. The output is then mapped to the alphabet as described by subsection 3.3.3. The tool also supports printing the currently quantized characters while simultaneously playing the song back, so the listener can get a feeling for which sound caused the generation of which character.

Said tool writes its JSON formatted output to the destination file name given at start time. This is particularly helpful because the computation is not optimized and takes some time. The JSON format consists of a mapping with five keys. An example file can be seen below:

Listing 6: Examplatory JSON output of the vector quantization.

```
{
  "Encoding": "eachIndex",
  "Alphabet": "ABCDEFGHIJKL",
  "WordSize": 1,
  "Usage": [["A",2],["B",5]],
  "Data": "ABBABBB"
}
```

In detail, the JSON format consists of the following keys with the given semantics types of the data they hold.

- `"Encoding"`: The translation method used described by a string. The tool is capable of changing the translation method, as the each index translation method was introduced last, and therefore, first attempts used the single method. This is merely a legacy feature. Possible values are: `"eachIndex"`, `"accumulate"` and `"ignore"`, whereas `ignore` refers to the single method.
- `"Alphabet"`: This value holds the used alphabet as a alphabetically sorted string.
- `"WordSize"`: Every translation mode has a number of characters generated from each quantization. `accumulate` and `single` generate words consisting of one character while `eachIndex` creates words where the length depends on the number quantizations. The survey data, for example, has a word size of 44 characters, so this field holds the integer number 44 in that case.

- `"Usage"`: The usage value is a histogram of the used characters of the alphabet. It is a list of lists, representing a list of tuples. Each tuple consists of the character, a string, and the number of times it occurs in the complete output string, an integer.
- `"Data"`: As every translation yields a certain amount (see `"WordSize"`) of characters, these characters are accumulated to a string. This field represents said string. It is the complete output of the whole process.

All features operate on the JSON format described in this section. As discussed before, only the eachIndex method was used in the experiments because all other methods can easily be derived from that and one does not have to run the lengthy extraction process for each method and parameter.

## 3.4 General implementation layout

Essentially, the whole implementation consists of three tools: the data extraction tool which uses Marsyas to retrieve the vector quantized data, the feature implementations and a tool to compare the results of the features to the test data. This section describes the data- and work flow of the tools as well as the intermediate formats.

### Data acquisition tool

The data acquisition tool analyzes a music file and stores the results in a JSON file, as described in subsection 3.3.4. After that, the song can be analyzed using the analysis tool, which implements the features. The feature tool generates distances from two sets of files and writes the results in a new JSON file. The third tool can now compare these results with the results of the survey or with other results generated by the distance tool. The whole simplified process is visualized in Figure 3.11.

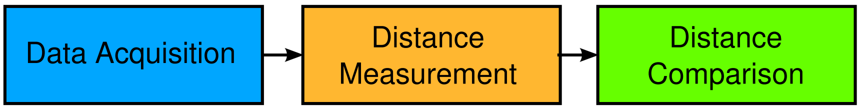


Figure 3.11: Visualization of the tool chain and the purpose of the single tools. At first, the results are computed using the Marysas network, then the second tool measures the distances between the results. Finally the measured distance can be compared to the results of the survey.

### Distance measurement tool

The feature implementations are not that lengthy, which made it possible to unify the distance computation of all features in a single tool. Three parameters define the behavior of the distance measurement tool. The first parameter, which is optional, is the character index to use. When specified, it defines which character is used of each word in the data. As the word size is known, those words can be found easily. This parameter makes it possible to use `eachIndex` data as a base for `single` data by selecting the wanted index. The second parameter determines whether to use the Levenshtein distance or the histogram distance. Finally the third parameter describes the specific method of distance computation to use, e.g., `indicator` or `cosine`.

For the Levenshtein distance measurements, there is the `normdistance` method, which takes the Levenshtein distance and normalizes it as described earlier in subsection 2.4.1 and there is the `distance` method, which just computes the Levenshtein distance without normalization.

The histogram distance has several methods, which are all related to the distance measurement used. The string supplied regarding the method defines the SciPy method to be used. Examples are `cosine` for cosine distance and `euclidean` for euclidean distance between the histogram vectors.

After generating the distances, the results are written to a JSON file, which is a $O(n^2)$ mapping of songs, holding the distance values of each song relation. The comparison tool then can use this file to compare the results with the results of the survey or other results. An example of this JSON file is shown in Listing 7.

Listing 7: Examplary JSON output of distance measurement tool. The values depend on the parameters of the distance measurement.

```
{
        "Song1": {
                "Song2": 0.10,
                "Song3": 0.20
        },
        "Song2": {
                "Song1": 0.10,
                "Song3": 0.80,
        },
        "Song3": {
                "Song1": 0.20,
                "Song2": 0.80
        }
}
```

### Tool usage

To visualize how the tools are used, the following examplatory calls are listed below:

- `$ aimvq.py -method eachIndex Nesto_-_Sales_histoires.mp3 results_eachIndex/ spaspet.json`

Use the Marsyas network to generate a JSON file as described in Listing 6 from an input audio file.

- `$ ./distances.py -levenshtein -method indicator -char-index 3 results_eachIndex/results_eachIndex/indicator.png`

Use the Levenshtein distance to compute the distances between the results in the supplied directory `results_eachIndex` by building an indicator string from the fourth character of every quantized word. A quantized word represents all quantization indices translated to characters that is acquired for one and only one SAI. The result is plotted and written as an image to `indicator.png`. The result can be seen in Figure 3.12.

- `$ ./distances.py -method cosine results_eachIndex/results_eachIndex/ cosine.png`

Generate a histogram distance plot using cosine distance between the histograms and write it to `cosine.png`.

- `$ ./distances.py -method cosine -export results_eachIndex/results_eachIndex/ cosine.json`

Generate the same histogram distance plot as before but export results to an intermediate JSON file called `cosine.json`.

- `$ ./compare.py survey.json cosine.json`

Compare the exported histogram results to the survey results. The result of this command can be seen in Figure 3.13.

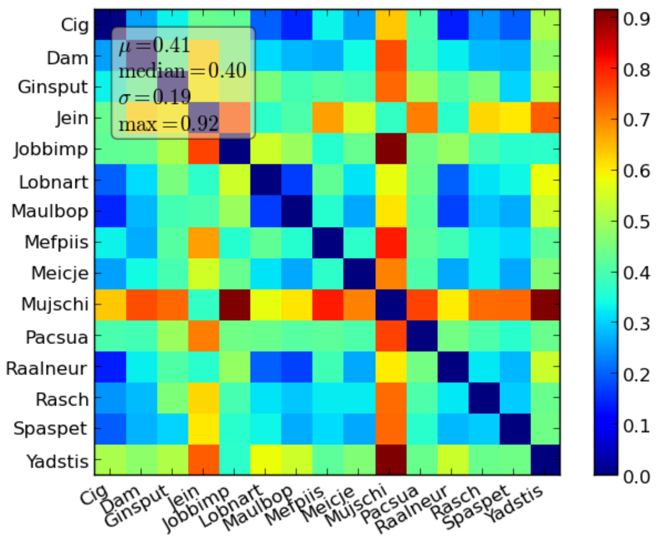


Figure 3.12: The result of the indicator example command.

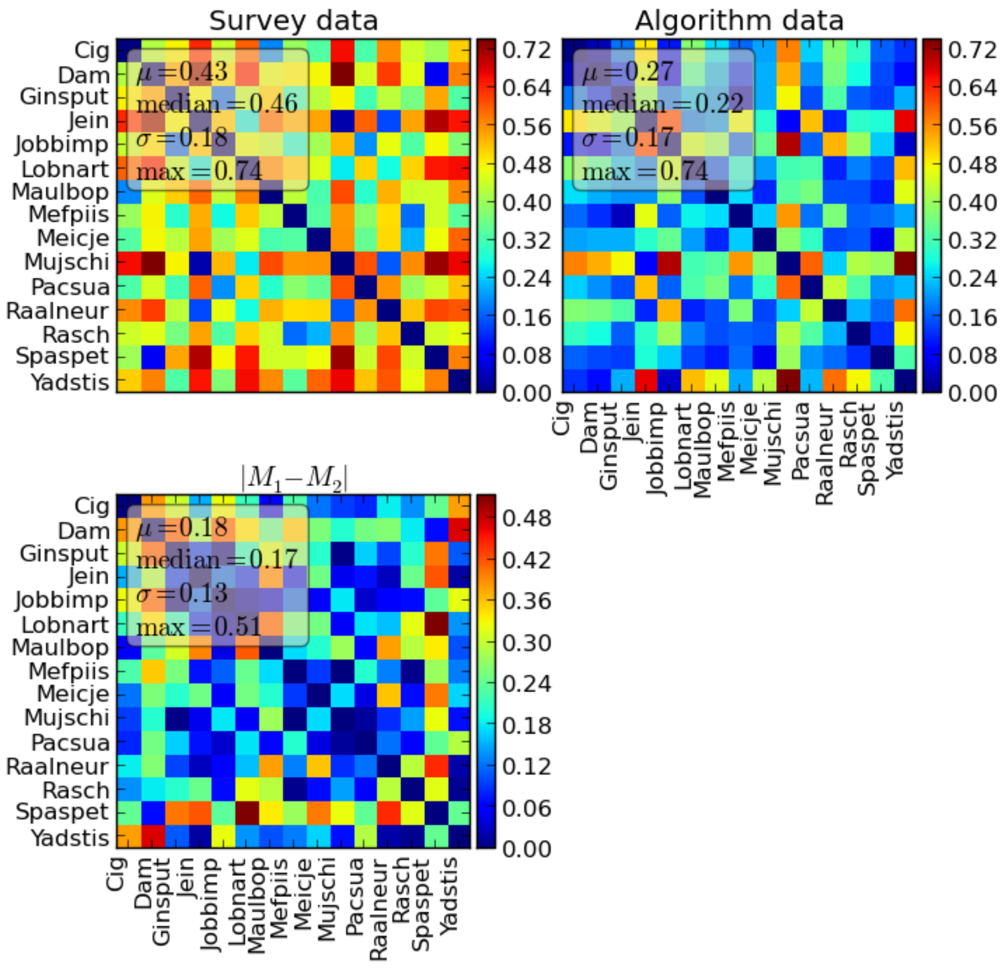


Figure 3.13: The result of the cosine histogram comparison example command.

## 3.5 String distance

The Python code in Listing 8 explains the basic implementation of this feature. The `levenshtein` method is provided by the Python-Levenshtein package described in the tools section (section 3.1).

Listing 8: Python code to illustrate the distance algorithm.

```python
def distance(vqString1, vqString2):
  # Normalized Levenshtein distance between two vqString strings.
  distance = levenshtein(vqString1, indicator2)

  return float(distance) / max(len(vqString1), len(indicator2))
```

As computations of the Levenshtein distance over large strings is expensive, with $\mathcal{O}(|s_1| \cdot |s_2|)$, it is not feasible to compute strings with lengths of more than 10,000 characters. As acquisition methods like `eachIndex` tend to generate these amounts, this distance method was only tested by extracting certain characters and ignoring the rest, as described by the `single` and `accumulate` method.

With the `single` method the best result in comparison to the distances acquired in the survey (section 2.2) is achieved with the 35th codebook. Each SAI is split into boxes with varying dimensions, which are then fed to the 44 quantizers and their underlying codebooks. Figure 3.14 compares the performances of the codebooks, and thus, of the box dimensions by using the mean of the absolute difference between the distances of the survey and the algorithm. The smallest mean value is the best, as the distance differences are small.

Clearly, certain boxes and codebooks perform better than others. The boxes from index 0 to 39 cover 16 frequency channel values of the SAIs. The covered temporal values for each SAI can be described by $2^{5+(i\%5)}$ with $i$ being the box index. As of box index 40, the spectral size is doubled to 32 and the temporal value continues as before. This process is described in detail by subsection 3.2.6.

Figure 3.14 shows that smaller box sizes, and therefore less coverage of the SAI, perform better. A temporal value of 32 produces both, the smallest mean difference, as well as the smallest overall mean difference. On the other hand, temporal value coverage of 256 and 512 values perform almost equally bad, with 128 temporal values being slightly better. Figure 3.15 compares the best result, performed using the 35th codebook, to the survey results. In the absolute difference graph, shown in the third panel of Figure 3.15, it can be seen that the classical songs Raalneur, Mujschi

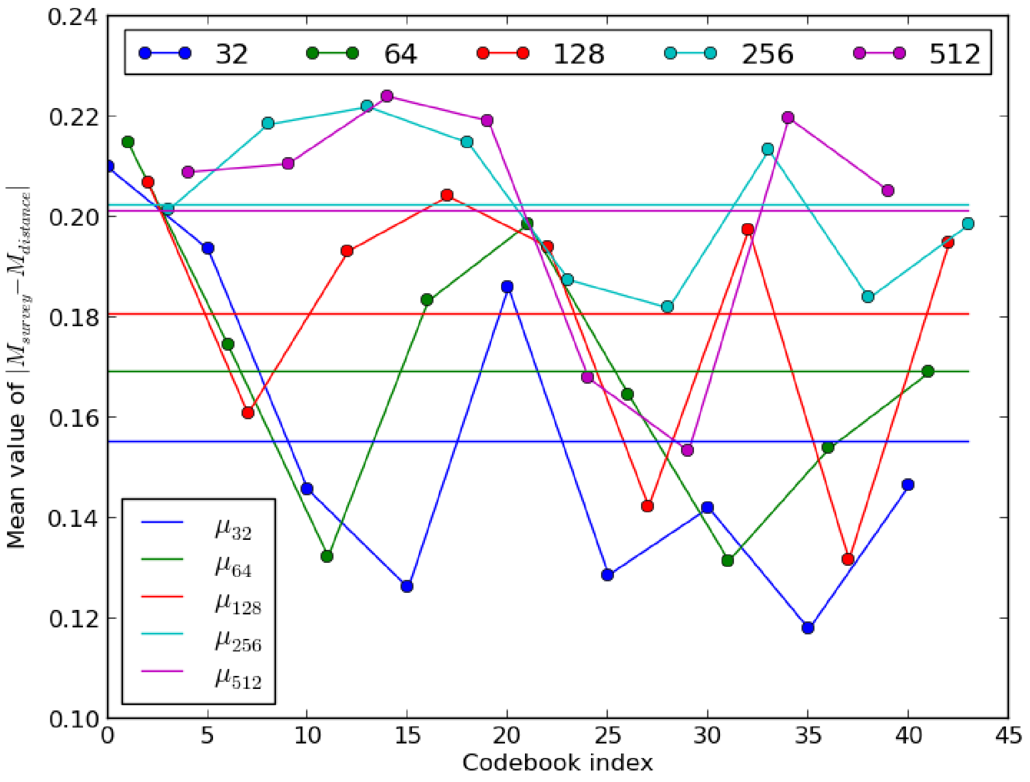


Figure 3.14: The result of each codebook is compared to the survey results presented in subsection 2.2.5. The lines constant to the y-axis are the mean values for a given temporal box size, to see which temporal box size performs best.

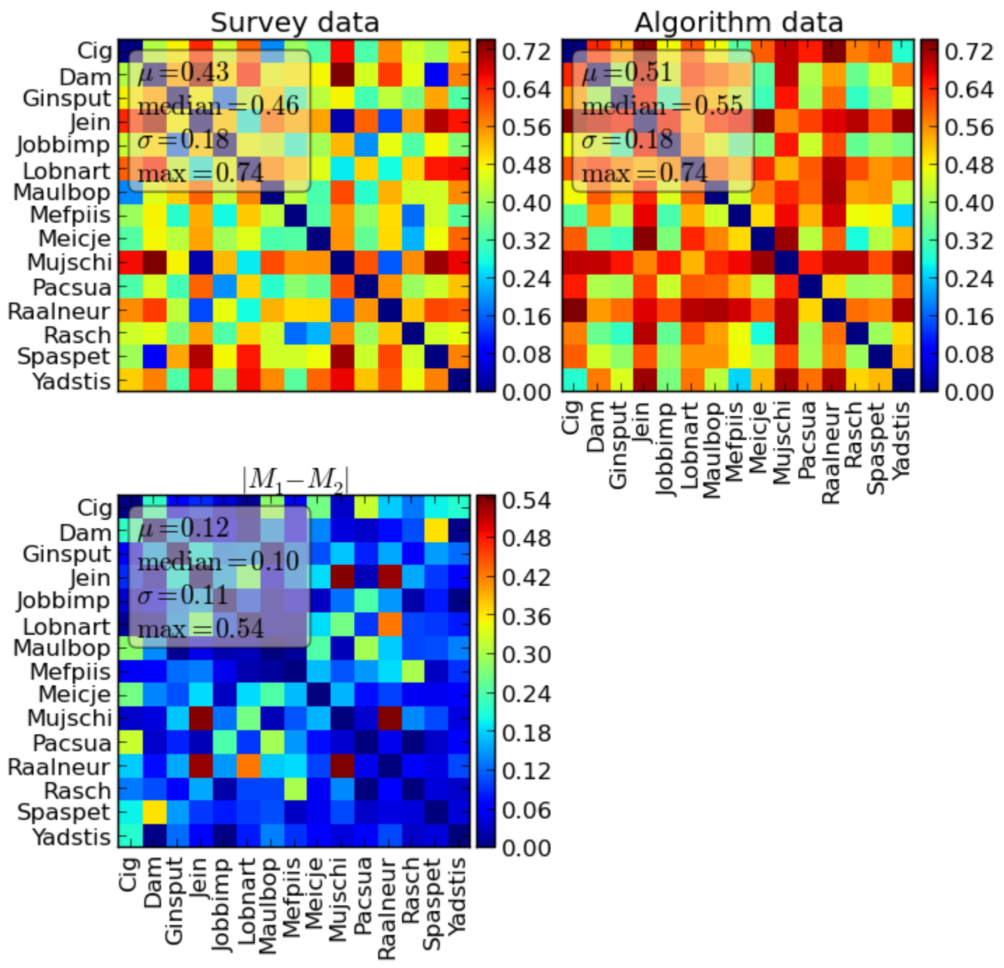


Figure 3.15: Comparison of the distances generated by using the characters from the codebook 35 to the survey distances using the string distance method.

and Jein are the most prominent outliers while the rest of the songs matches well with the survey results.

The same mean difference analysis was done with the selected songs from the covers80 dataset, resulting in the graph shown in Figure 3.16. Because of the lower sampling rates of the covers80 songs, there are less temporal values per SAI and thus less temporal values available for the boxes.

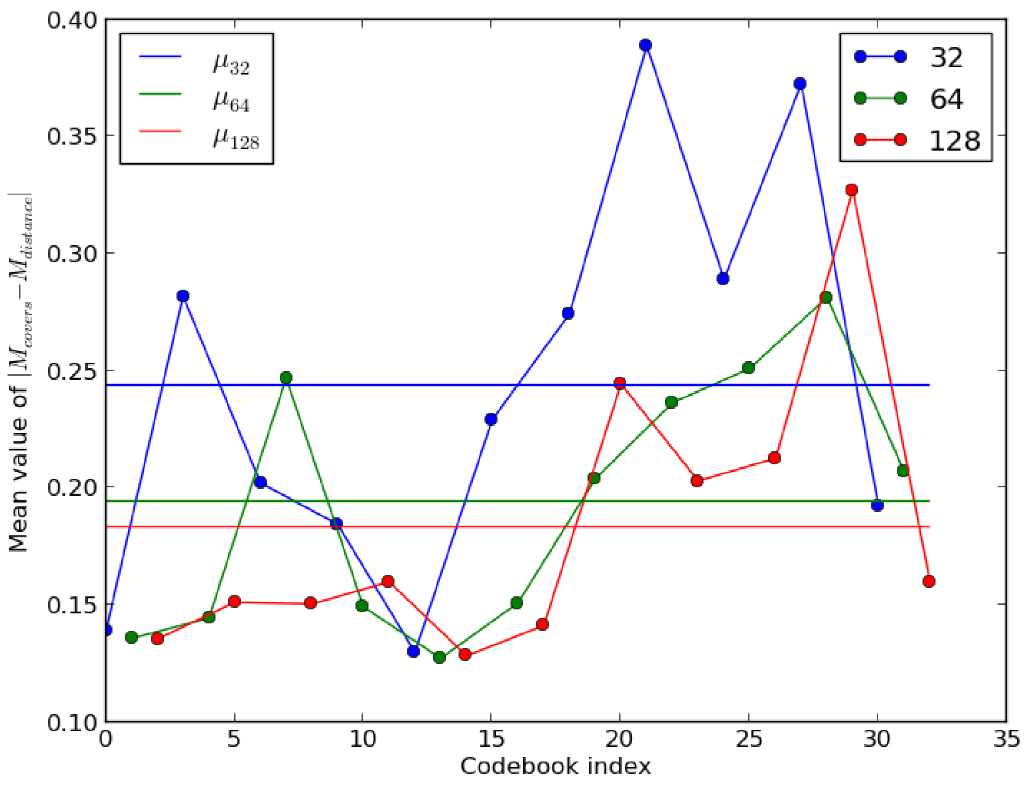


Figure 3.16: Performance of each codebook measured to the ideal reference matrix mentioned in subsection 3.3.1. The mean performance for every temporal box size used are shown by the lines with constant y-value.

In the case of the covers80 dataset, the parameters of the box at each index is different, as the music has a lower sample rate (16 kHz). The temporal size can be described by $2^{5+(i\%3)}$, ranging from 32 to 128. The spectral size is 16 for all indices below 24, otherwise 32. The mean values shown in Figure 3.16 show clearly that in the case of the covers80 songs, higher temporal window sizes perform better.

The `accumulate` method was not successful. The accumulation with a codebook size of 200 led to too many repetitions of characters, thus giving no distinguishable

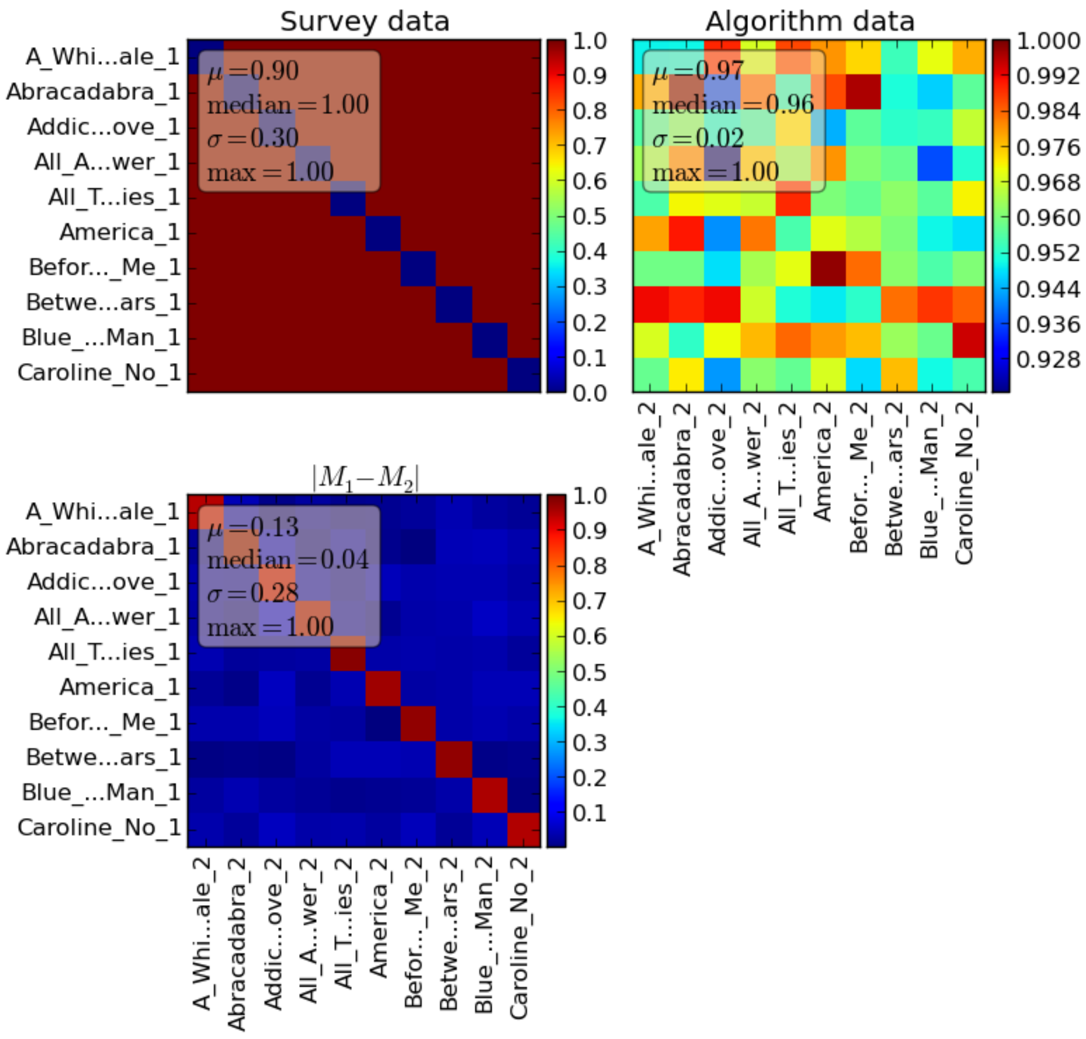


Figure 3.17: Comparison of the ideal comparison matrix to the results of the algorithm with the 14th codebook.

result. The effect can be seen in the algorithm result shown in Figure 4.3. However, this might be compensated by using a bigger alphabet. The main task to accomplish would then be to find the optimal alphabet size. This was not investigated in the scope of this thesis, though.

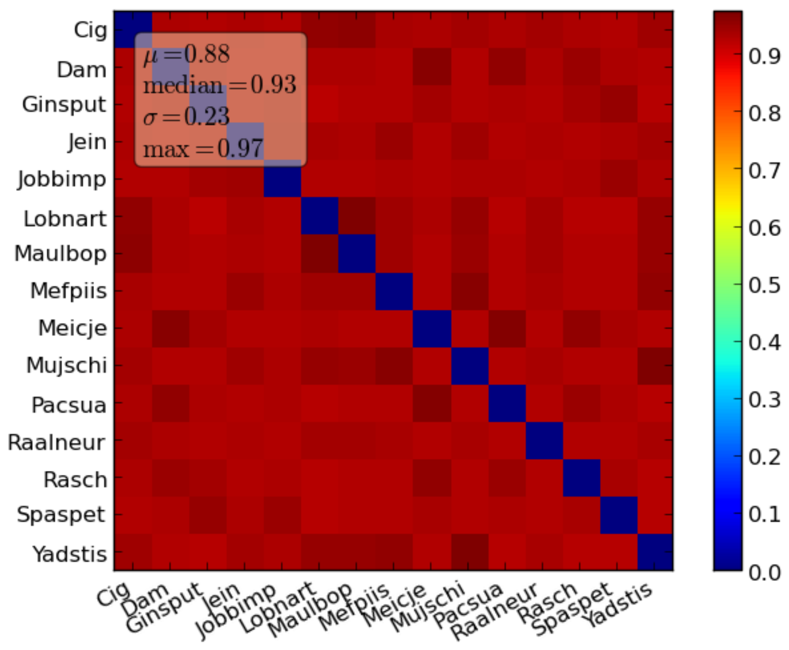


Figure 3.18: Results of the string distance using accumulated VQ data. Almost all songs have the same distance to each other due to the extreme re-usage of characters from the alphabet.

## 3.6 String indicator

The implementation of this method is straight forward. The following code describes the used algorithm which is embodied by the `distance` function. The two string parameters holding the quantized and translated output are acquired as described in the song conversion section (subsection 3.3.4). The `levenshtein` method is provided by the Python-Levenshtein package described in the tools section (section 3.1).

Listing 9: Python code to illustrate the indicator algorithm.

```python
def extractUsedCharacters(string):
  # Store each unique character of the string only once and return them.
  occurrences = dict((char, True) for char in string)
  return occurences.keys()

def indicator(vqString):
  # Return alphabetically sorted indicator strings in ascending order.
  return sort(extractUsedCharacters(vqString))

def distance(vqString1, vqString2):
  # Normalized Levenshtein distance between two indicator strings.
  indicator1 = indicator(vqString1)
  indicator2 = indicator(vqString2)

  distance = levenshtein(indicator1, indicator2)

  return float(distance) / max(len(indicator1), len(indicator2))
```

### 3.6.1 Single

In Figure 3.19 it can be seen which codebook indices result in the smallest average difference between the survey results and the results of the string indicator algorithm. As the figure shows, the codebook and therefore the box configuration at index 38 yields the best result.

The area of values covered at the best performing box, which has the Codebook index 38 shown in Figure 3.19, is 256 on the temporal scale and 16 in the spectral scale. On the other hand, the worst results were performed using a box with 32

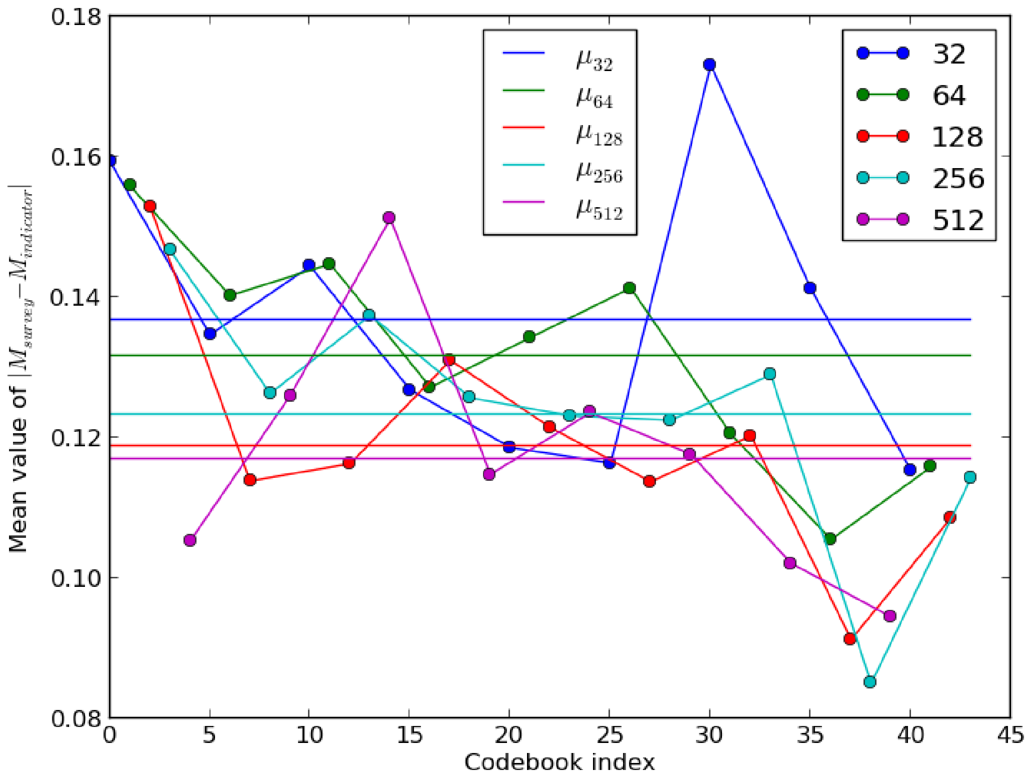


Figure 3.19: This figure shows the mean values of $|M_{survey} - M_{indicator}|$ with different codebook indices. The mean values of the differences are blended in as straight lines for each temporal window size.

temporal and 16 spectral values. The following table lists the box area parameters of notable indices in the figure.

| index | spectral | temporal |
|---|---|---|
| 0 | 16 | 32 |
| 4 | 16 | 512 |
| 31 | 16 | 64 |
| 32 | 16 | 128 |
| 34 | 16 | 512 |
| 35 | 16 | 1 |
| 38 | 16 | 256 |
| 42 | 32 | 128 |

The table above and the graph Figure 3.19 show that a high area in the temporal direction yields better results. This makes sense, as on the one hand this method generally discards temporal information by sorting the indicator string and on the other hand the higher the temporal window is, the more temporal information holds each quantized character. The higher temporal information of the boxes therefore compensates the loss of temporal information by the string sorting.

Due to the way the box cutting is implemented, lower spectral sizes are computed in various configurations before switching to higher spectral sizes. Because of the limited amount of codebooks and because each box is mapped to only one codebook, the higher spectral values are not exhaustively tested here. It is also noteworthy that the boxes which cover more values in the spectral scale perform better than their pendants with the same temporal width but smaller spectral height. For example, index 32 and 42 share the same temporal width, however 42 performed better. It would be interesting to see the performance of the values with higher spectral scale, as noted in the future work chapter chapter 5.

Figure 3.20 presents a comparison of cover song reference data and distances computed using the string indicator. An example of such a comparison is presented in Figure 3.21. The following table represents notable box parameters for the different indices shown in Figure 3.20.

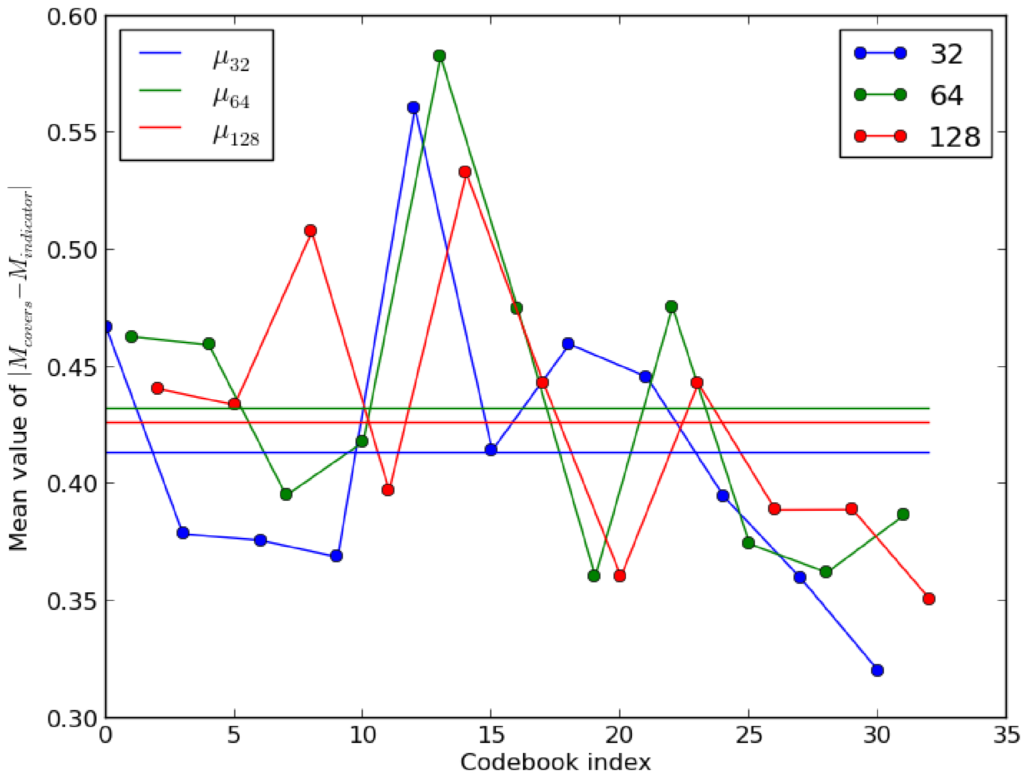


Figure 3.20: The smallest mean distance value to the comparison matrix of cover songs is at x=30. The worst identification can be found at x=13.

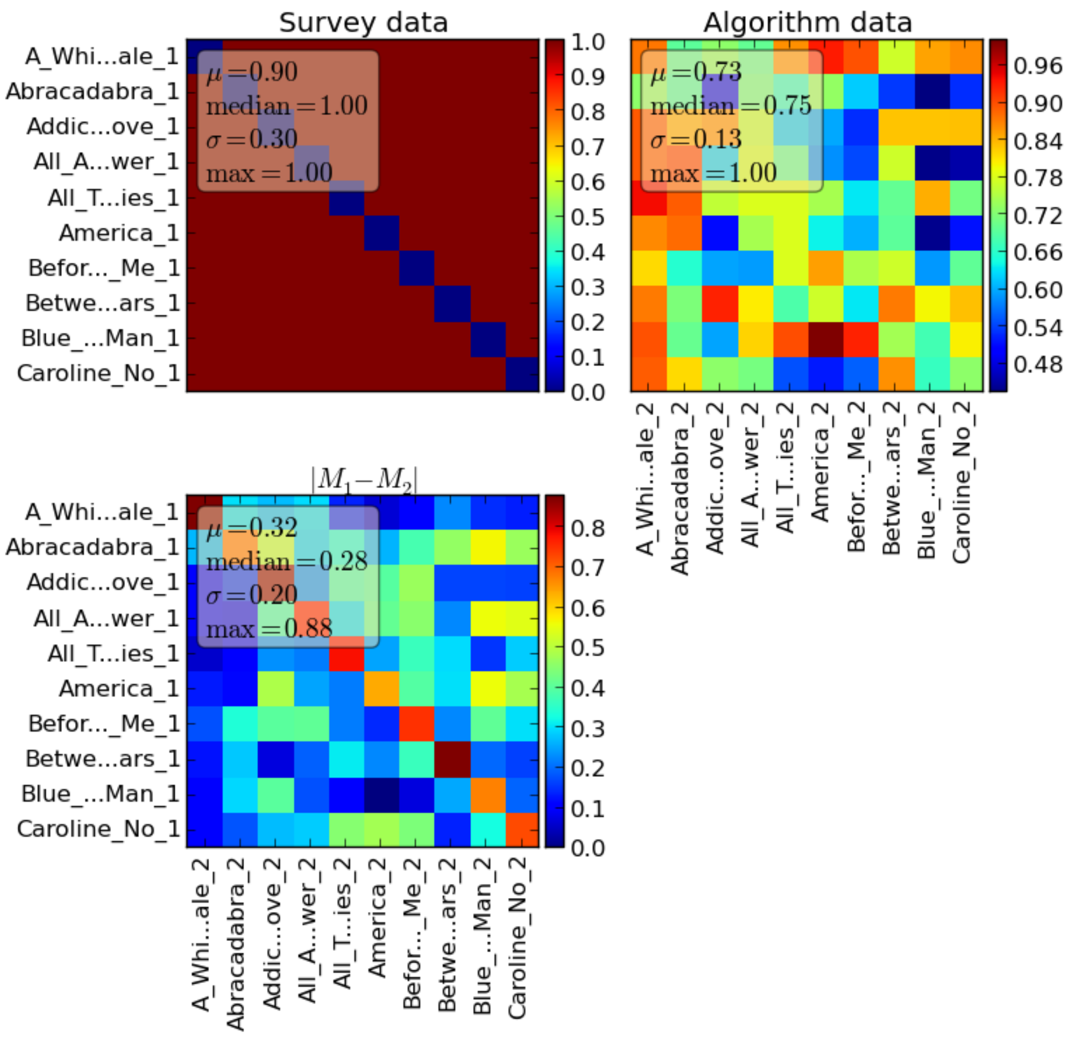


Figure 3.21: Comparison of the best string indicator results with the ideal results where all but the matched cover songs have the maximum distance. This result is achieved using the 30th codebook.

| index | spectral | temporal |
|---|---|---|
| 0 | 16 | 32 |
| 5 | 16 | 128 |
| 8 | 16 | 128 |
| 10 | 16 | 64 |
| 12 | 16 | 32 |
| 13 | 16 | 64 |
| 14 | 16 | 128 |
| 20 | 16 | 128 |
| 30 | 32 | 32 |

While with the survey data higher temporal sizes yield better results, the opposite is the case with the cover songs. This behavior was already noticed in section 3.5. A possible explanation is that these algorithms are not focused enough on cover detection as much as the intervalgram is (Walters et al., 2012). Another problem is that almost all values in the reference data are 1.0 and only a small fragment of the values is 0.0. Therefore, the best result is where most of the values are 1.0 and not 0.0. The results with overall high distances are mostly generated using low temporal sizes in case of the string indicator method and high temporal sizes using the string distance method. Therefore, the 'good' results favor bad configurations. It can be concluded, that the proposed method of comparing cover songs is not suitable for this sort of algorithms and it is an interesting task for the future to perform a suitable comparison.

Figure 3.22 shows the best comparison with the indicator string made from box 38. The two classic songs Mujschi and Jein have high distances to the other songs while being similar to each other. The two mostly electronic songs, Maulpop and Cig, are close and so are the two hip/hop songs Dam and Spaspet.

### 3.6.2 Accumulate

Using the `accumulate` method to generate the indicator string does not yield a usable result. With an alphabet of size 200 and 44 quantizations for each SAI, the whole alphabet gets covered with every song, resulting in a distance of zero for each song. A feasible workaround is to increase the alphabet size so it can hold the maximum value of quantization words, as mentioned in the alphabet section subsection 3.3.2.

This method grants every quantization, and therefore every box size, the possibility to participate in the resulting indicator string. This method performs on aver-

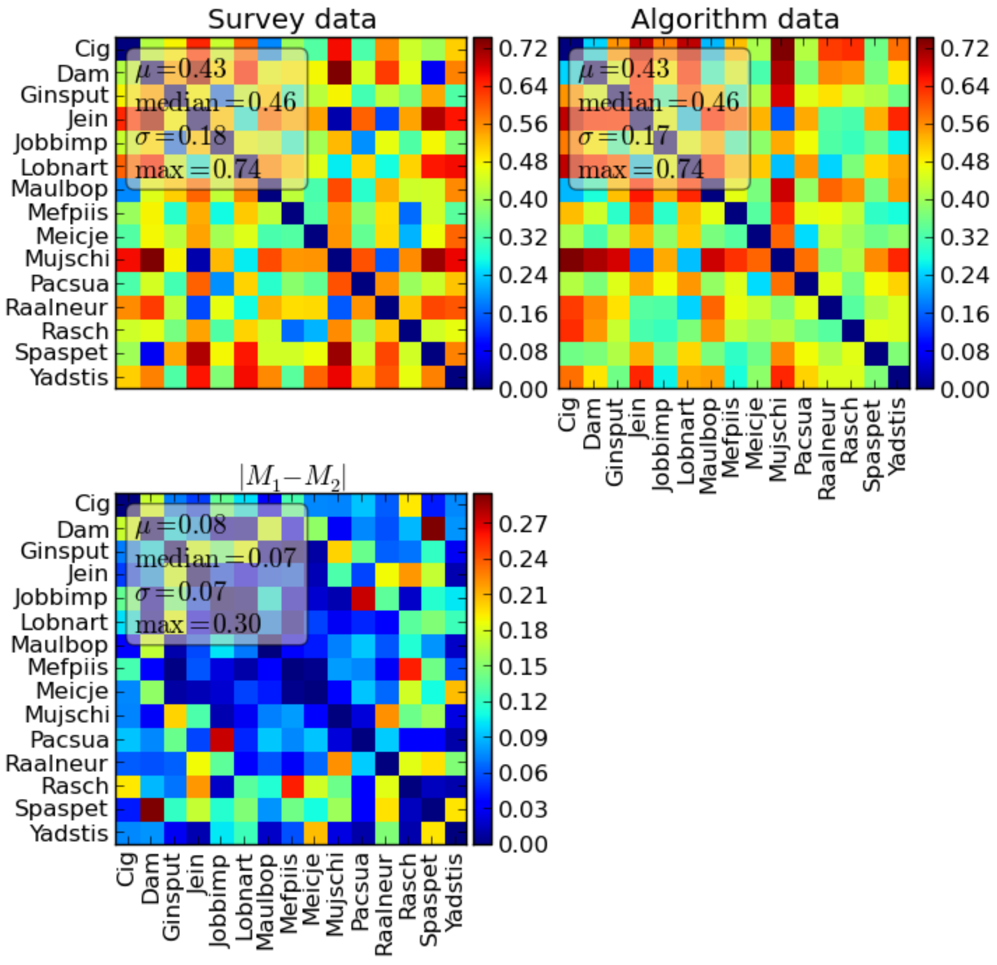


Figure 3.22: Comparison of best indicator results to the survey results. The classical pieces, Mujschi and Jein distinguish themselves best from the rest and are very close. This result is achieved by using the 38th codebook.

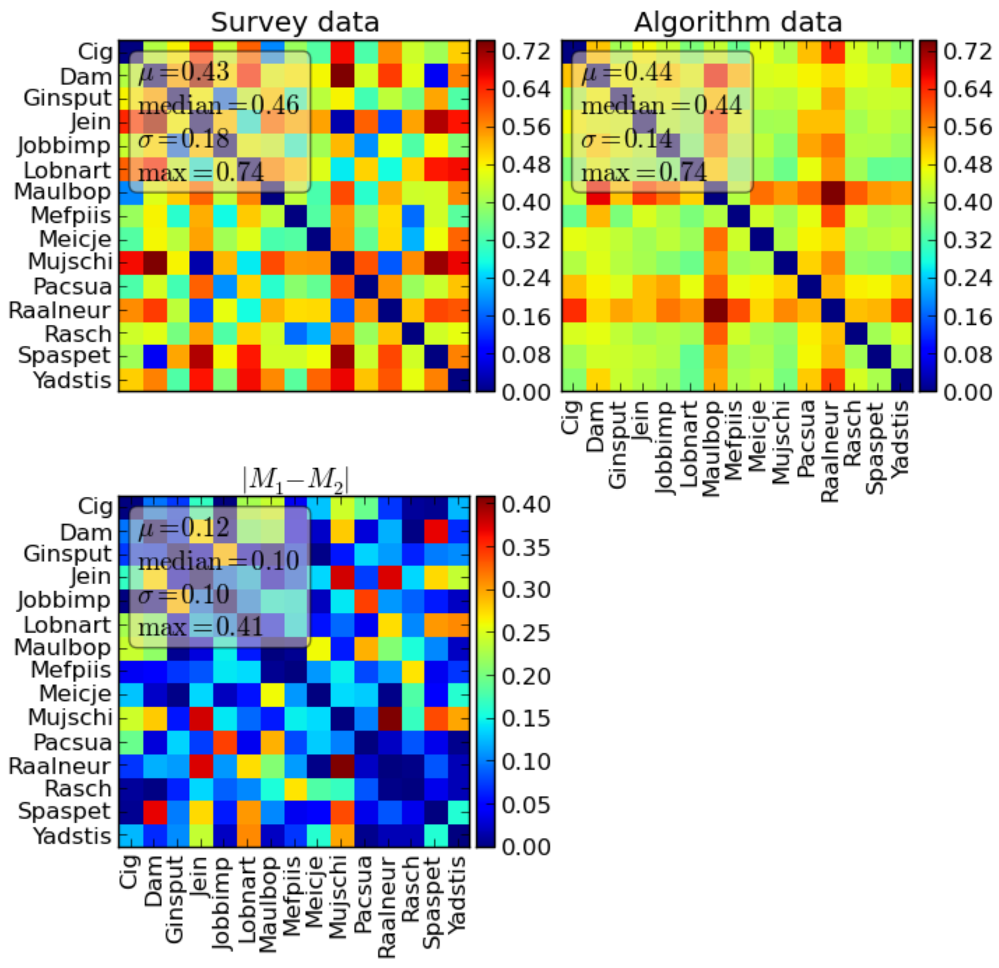
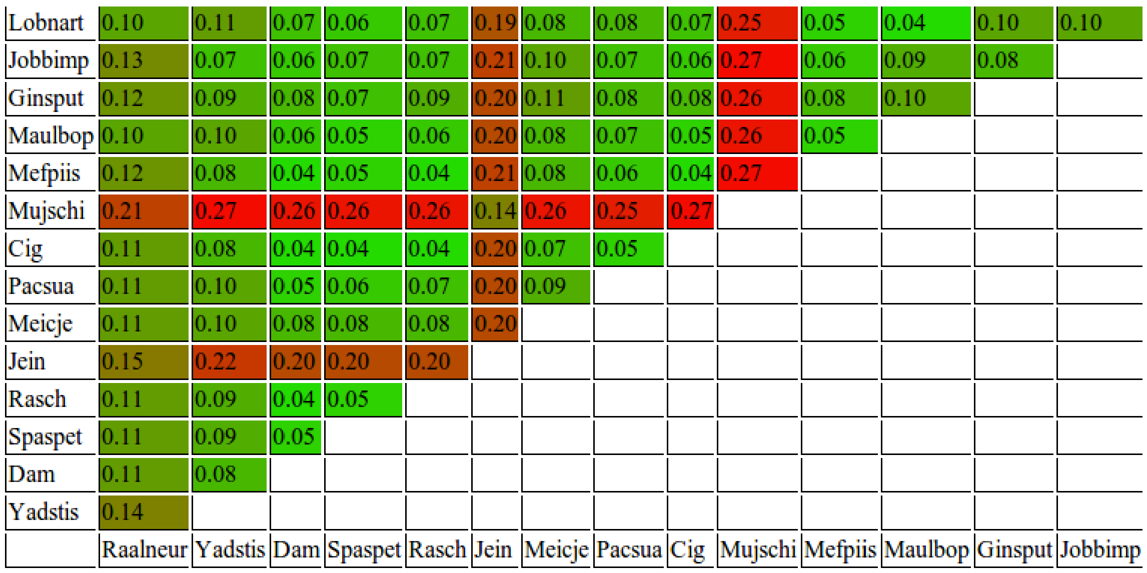


Figure 3.23: Comparison of the survey data to the indicator string generated from accumulated data over the biggest alphabet of 8800 characters.

age about as good as the best single character, however strong nuances like the distances of the classical songs or the distance between the two hip-hop songs Spaspet and Dam seem to vanish.

The effect of using more than one character to generate a string from which the indicator string is derived, e.g. using two characters from two codebooks, can be seen in Figure 3.24 and Figure 3.25. The first uses the output of two codebooks and the latter uses 20 characters where there was only one in the original experiment. It becomes clear that the generalization properties become weaker the more characters are used. This behavior is expected, as each indicator string gets more and more specialized to the song and is harder to compare to other indicator strings.



| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lobnart | 0.10 | 0.11 | 0.07 | 0.06 | 0.07 | 0.19 | 0.08 | 0.08 | 0.07 | 0.25 | 0.05 | 0.04 | 0.10 | 0.10 |
| Jobbimp | 0.13 | 0.07 | 0.06 | 0.07 | 0.07 | 0.21 | 0.10 | 0.07 | 0.06 | 0.27 | 0.06 | 0.09 | 0.08 | |
| Ginsput | 0.12 | 0.09 | 0.08 | 0.07 | 0.09 | 0.20 | 0.11 | 0.08 | 0.08 | 0.26 | 0.08 | 0.10 | | |
| Maulbop | 0.10 | 0.10 | 0.06 | 0.05 | 0.06 | 0.20 | 0.08 | 0.07 | 0.05 | 0.26 | 0.05 | | | |
| Mefpiis | 0.12 | 0.08 | 0.04 | 0.05 | 0.04 | 0.21 | 0.08 | 0.06 | 0.04 | 0.27 | | | | |
| Mujschi | 0.21 | 0.27 | 0.26 | 0.26 | 0.26 | 0.14 | 0.26 | 0.25 | 0.27 | | | | | |
| Cig | 0.11 | 0.08 | 0.04 | 0.04 | 0.04 | 0.20 | 0.07 | 0.05 | | | | | | |
| Pacsua | 0.11 | 0.10 | 0.05 | 0.06 | 0.07 | 0.20 | 0.09 | | | | | | | |
| Meicje | 0.11 | 0.10 | 0.08 | 0.08 | 0.08 | 0.20 | | | | | | | | |
| Jein | 0.15 | 0.22 | 0.20 | 0.20 | 0.20 | | | | | | | | | |
| Rasch | 0.11 | 0.09 | 0.04 | 0.05 | | | | | | | | | | |
| Spaspet | 0.11 | 0.09 | 0.05 | | | | | | | | | | | |
| Dam | 0.11 | 0.08 | | | | | | | | | | | | |
| Yadstis | 0.14 | | | | | | | | | | | | | |
| | Raalneur | Yadstis | Dam | Spaspet | Rasch | Jein | Meicje | Pacsua | Cig | Mujschi | Mefpiis | Maulbop | Ginsput | Jobbimp |

Figure 3.24: Distances calculated using indicator strings with two quantizations instead of one. All songs but the classical songs Mujschi and Jein have very low values and are mostly indistinguishable.

### 3.6.3 EachIndex

Creating the indicator string from used words instead of single characters was not implemented in this thesis. The general idea is to use all quantizations from a run as a word and build an indicator string from that. It is doubtful whether this has any practical value, as it would generate very large strings due to the many combinations possible with, in this case, 48 codebooks.

| | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Lobnart | 0.06 | 0.06 | 0.07 | 0.05 | 0.05 | 0.06 | 0.07 | 0.07 | 0.05 | 0.06 | 0.05 | 0.05 | 0.06 | 0.07 |
| Jobbimp | 0.07 | 0.04 | 0.04 | 0.07 | 0.05 | 0.05 | 0.05 | 0.07 | 0.06 | 0.05 | 0.05 | 0.08 | 0.05 | |
| Ginsput | 0.06 | 0.04 | 0.04 | 0.05 | 0.05 | 0.05 | 0.05 | 0.07 | 0.05 | 0.04 | 0.04 | 0.06 | | |
| Maulbop | 0.07 | 0.07 | 0.08 | 0.06 | 0.06 | 0.06 | 0.08 | 0.06 | 0.06 | 0.06 | 0.06 | | | |
| Mefpiis | 0.06 | 0.04 | 0.05 | 0.05 | 0.04 | 0.04 | 0.05 | 0.06 | 0.04 | 0.04 | | | | |
| Mujschi | 0.06 | 0.04 | 0.04 | 0.05 | 0.05 | 0.04 | 0.05 | 0.05 | 0.04 | | | | | |
| Cig | 0.05 | 0.05 | 0.05 | 0.05 | 0.04 | 0.04 | 0.05 | 0.06 | | | | | | |
| Pacsua | 0.07 | 0.06 | 0.06 | 0.08 | 0.07 | 0.05 | 0.06 | | | | | | | |
| Meicje | 0.06 | 0.04 | 0.04 | 0.07 | 0.05 | 0.05 | | | | | | | | |
| Jein | 0.06 | 0.04 | 0.05 | 0.06 | 0.05 | | | | | | | | | |
| Rasch | 0.06 | 0.05 | 0.05 | 0.05 | | | | | | | | | | |
| Spaspet | 0.06 | 0.06 | 0.06 | | | | | | | | | | | |
| Dam | 0.07 | 0.04 | | | | | | | | | | | | |
| Yadstis | 0.06 | | | | | | | | | | | | | |
| | Raalneur | Yadstis | Dam | Spaspet | Rasch | Jein | Meicje | Pacsua | Cig | Mujschi | Mefpiis | Maulbop | Ginsput | Jobbimp |

Figure 3.25: Distances calculated using indicator strings with 20 quantizations instead of one. Even the classical songs, that were still distinguishable using two codebooks, are vanished now.

## 3.7 Distance of Used Character Histograms

This section discusses the implementation of the histogram comparison feature discussed in subsection 2.4.2. The only significant translation method is `eachIndex`, which is defined previously in this chapter. The accumulate and ignore methods were unsuccessful, as there was either not enough correlation in case of `ignore` or too much repetition in the codewords, due to the limited alphabet size in case of `accumulation`.

Each row of the box cut from the stabilized auditory image is matched against a different codebook, resulting in an index of the used codeword. This index is then matched against the alphabet and translated into a character. The resulting string of characters is then converted into a histogram which is then used for distance measurement. The process is described by the following Python code, beginning with the `distance` function.

Listing 10: Python code to illustrate the histogram algorithm.

```python
import scipy.spatial.distance

def histogram(alphabet, string):
  # Return the occurences of each alphabet character as a vector.
  return [string.count(char) for char in alphabet]

def distance(alphabet, vqString1, vqString2):
  # Return the cosine distance between the histograms of each
  # supplied vector quantized string.
  histogram1 = histogram(alphabet, vqString1)
  histogram2 = histogram(alphabet, vqString2)

  return scipy.spatial.distance.cosine(histogram1, histogram2)
```

This method is very similar to the method presented in (Lyon et al., 2010) and (Ness et al., 2011) but without the classifiers, as this thesis focuses on distances only:

> “The VQ codeword index is a representation of a 1-of-N sparse code for each box, and the concatenation of all of those sparse vectors, for all the box positions, makes the sparse code for the SAI image. The resulting sparse code is accumulated across the audio file, and this histogram (count of number of occurrences of each codeword) is then used as input to an SVM [5] classifier[3]” (Ness et al., 2011, p. 7)

The method presented in this section also determines a codeword index for each row of the box and concatenates the result into a larger vector, from which a histogram is generated. However, instead of resolving a number for each box row, a character is resolved, using the VQ index as an index for the alphabet.

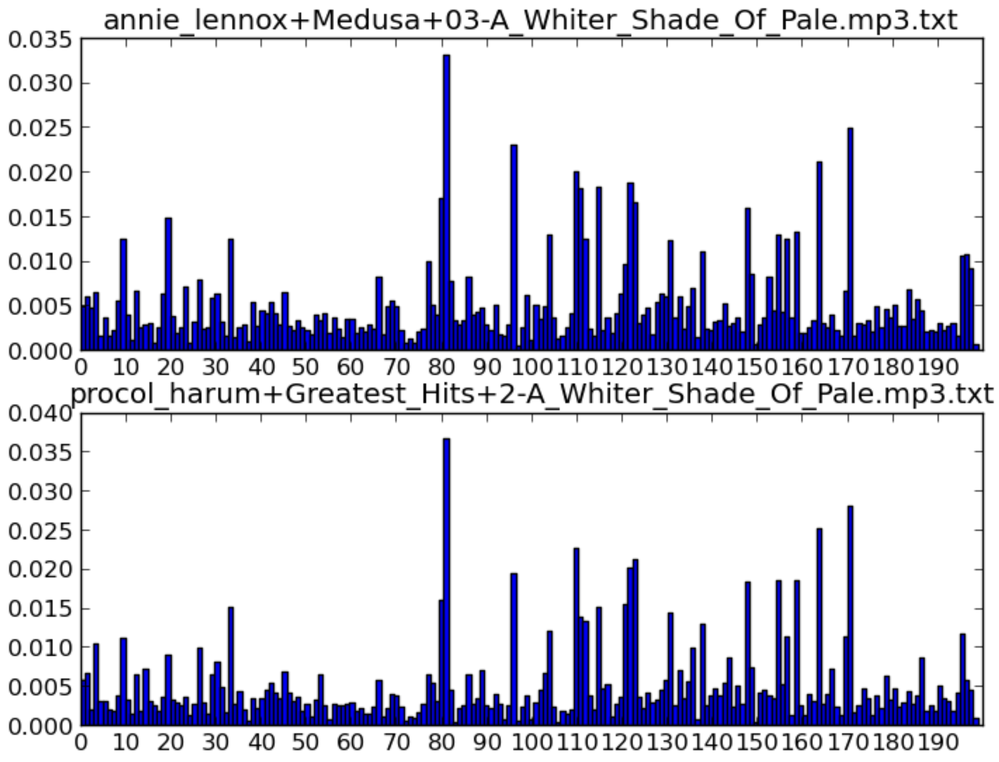


Figure 3.26: Histogram of two cover songs, both analyzed using the `eachIndex` method. One can clearly identify matching sections. The numbers on the x-axis represent the index of the used character from the alphabet.

The example histogram shown in Figure 3.26 presents promising similarities between the histograms of the two cover songs. This is backed up by the distance map shown in Figure 3.27, where 80% of the cover song pairs are at least considered similar and 30% of the pairs have the cover song as closest partner. Only two cover song pairs are not considered similar, `Abracadabra` and `Between the Bars`.

Figure 3.28 represents the comparison of the algorithm to the survey results. There are some consistencies with the survey results, for example that the classical songs Mujschi and Jein are most similar to each other and that they are similar to the world song Raalneur. However, overall coverage is worse than with the string indicator method.

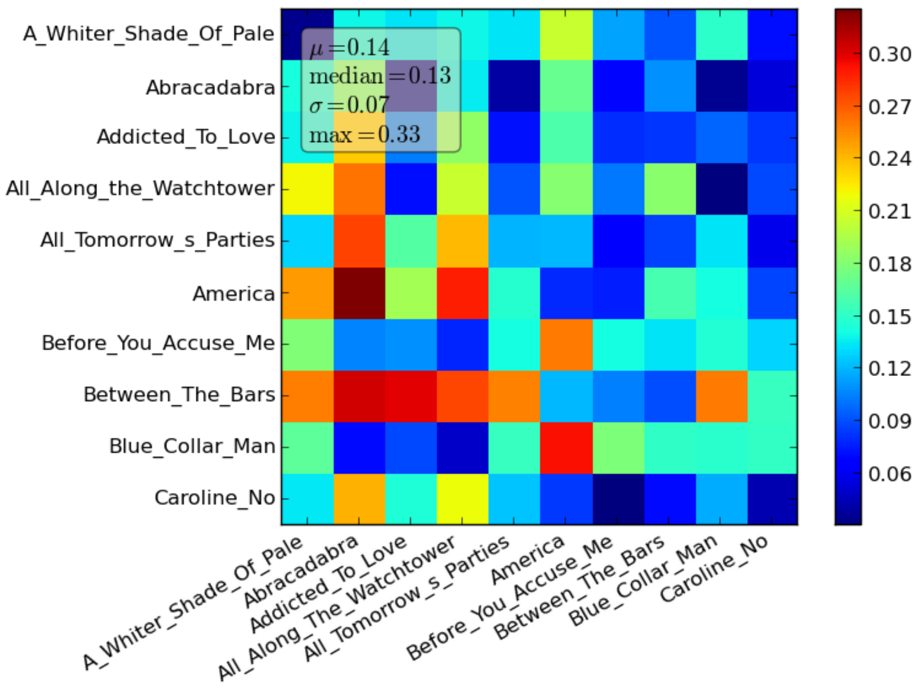


Figure 3.27: Distance map of the covers80 songs using cosine distance. Three of ten pairs match closest with the cover song. Eight of ten songs are rated at least similar.

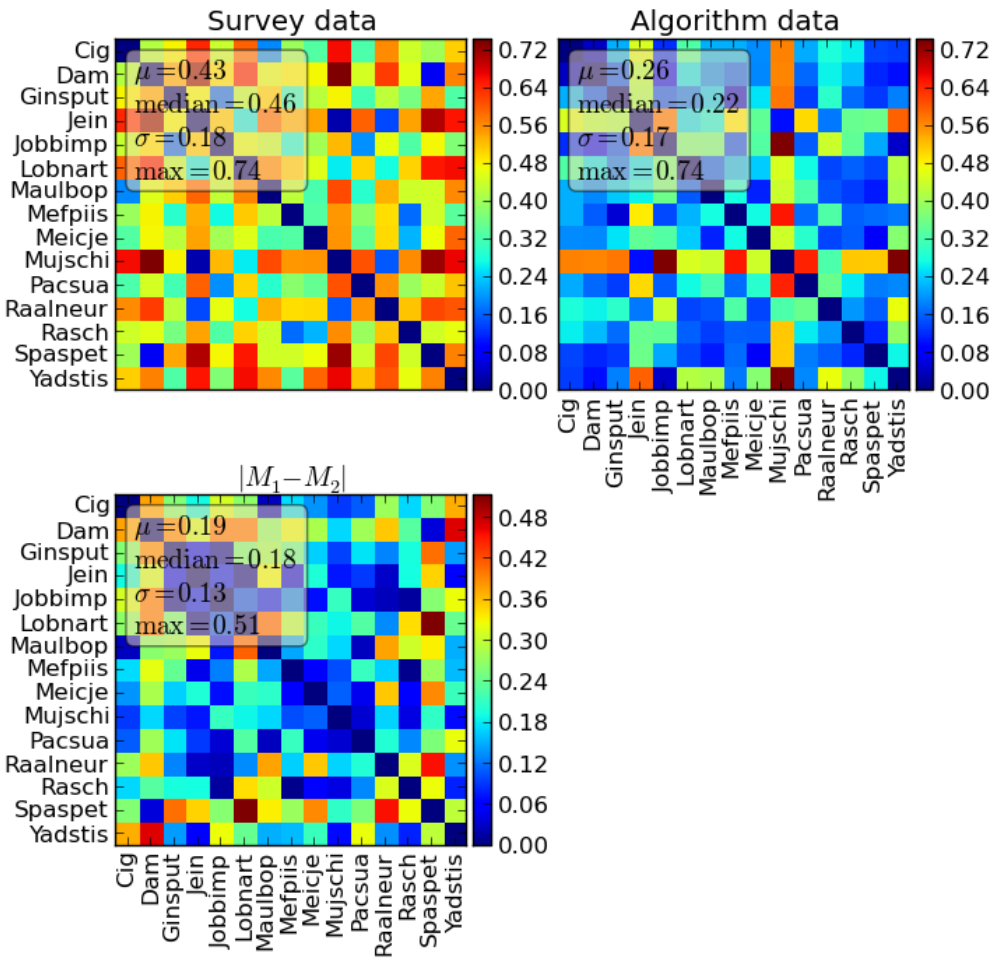


Figure 3.28: Results of comparing the songs used in the survey to each other by using the histogram method and the cosine distance with an alphabet of 200 characters. The classical songs Mujschi and Jein are only similar to each other and a bit similar to the World song Raalneur.

Using bigger alphabets decreases the performance as shown by Figure 3.29. The bins of the histogram are getting smaller while the histogram gets wider, which reduces recognizability. This approach is similar to the one used in (Ness et al., 2011).

While several distance methods were tested due to the variety of ScyPy's (subsection 3.1.4) `spatial.distance` module[8], the cosine distance performed best. This corresponds to the findings of (Foote et al., 1997) as well.

[8]http://docs.scipy.org/doc/scipy/reference/spatial.distance.html

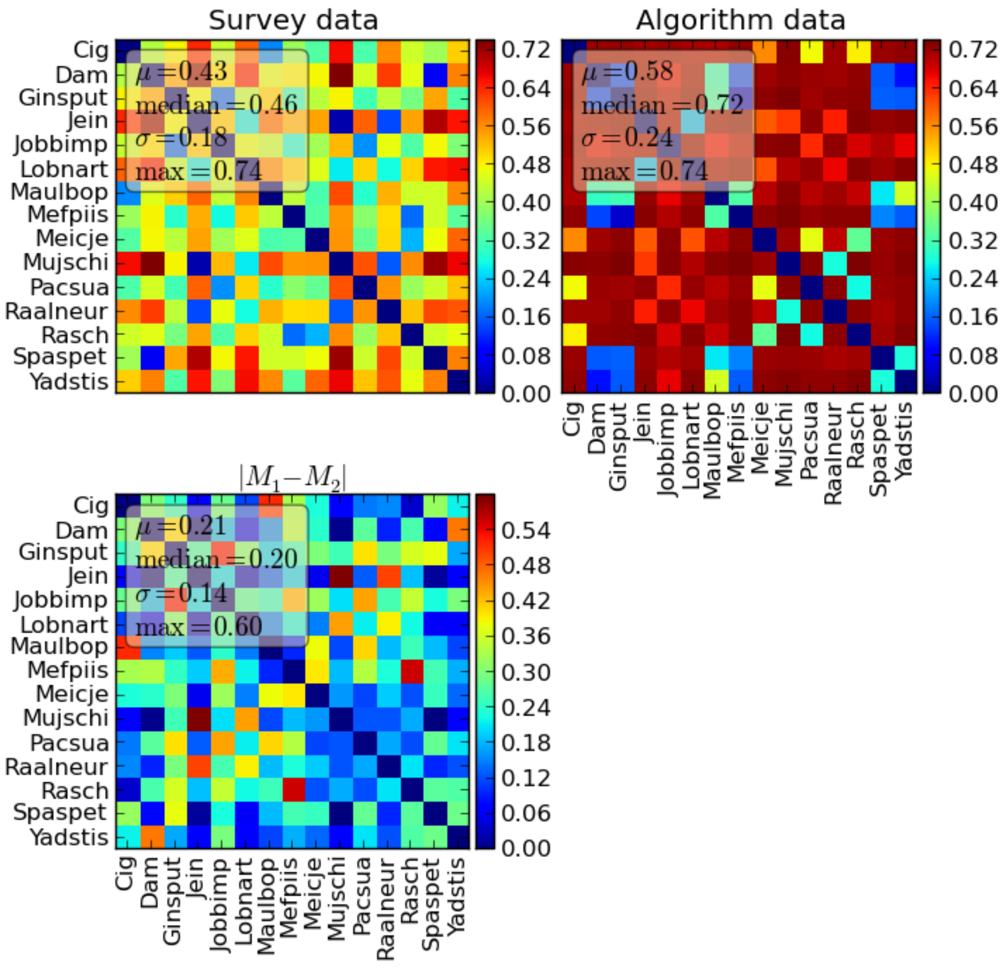


Figure 3.29: In comparison with the survey data, the approach of using a full alphabet $(44 \cdot 200$ characters) was, as expected, no success.

# 4 Results

Some results were already discussed in length, for example the study results and some results of the features implementations. However, to summarize this chapter gathers the key results and presents them in comparison to the other results. The survey results are also summarized in this chapter, along with references to the in-detail results section of the survey.

## 4.1 Survey

As already described in subsection 2.2.5, the survey introduces a new interface to ask humans for music similarity measurements and successfully provides test data for the algorithms featured in this thesis. Furthermore, the initial alternative hypothesis, that musical similarity for humans exists, is backed by this study, which was proven in the statistical analysis Figure 2.2.5.

There is a great potential for the survey interface to be combined with services like Amazon Mechanical Turk for data collection on a grand scale as further ground truth data is highly appreciated and needed in the MIR community (Berenzweig et al., 2004). This is also discussed in the survey's future work section subsection 2.2.7.

## 4.2 Musical Distance Measurement

This section compares and summarizes the results of the methods implemented in the scope of this thesis. Each implementation and their variations are discussed in the respective subsections starting with the string distance feature.

It turns out that the best results are achieved with the string indicator method, followed by the string distance. The least effective of the methods is therefore the histogram distance. Furthermore, the method used with the string distance and indicator method for cover song comparison proved itself to be ineffective. This method favors bad results over good ones to match the ideal reference matrix. As a result, the cover song distances are not representative for the tested methods.

### 4.2.1 String distance

Generally, this method does not scale well in comparison to the string indicator method, as it takes longer the longer the song is. Additionally, the EachIndex variation of this feature is not applicable as it would lengthen the string even further, making it ineffective to compute with the Levenshtein distance. The single method performs best but is still behind the best result of the string indicator feature.

#### Single

As already discussed in the implementation of this feature in section 3.5, the best result is achieved with a box size of $32 \times 16$ while the worst result uses boxes of $512 \times 16$. The performances of the box sizes in comparison to the survey were shown in Figure 3.14. Figure 4.1 and Figure 4.2 compare the best and the worst results are to the survey results.

The best result, shown in Figure 4.1, has a mean absolute difference of 0.12 and the highest outliers are the classical songs. It is notable that the classical songs Muschji, Jein and Raalneur have a high distance to every other song, including themselves, albeit these songs should have a low distance to each other. This indicates that the single string distance method does not perform well with classical music because it does not seem to be able to compare classical music to one another.

Having only a very small range of distances, the worst result shown in Figure 4.2 is very different to the survey results. There are notable differences between the distances with the same characteristics as shown in the best results, however due to the small range the result is not comparable. While a min-max stretch could have

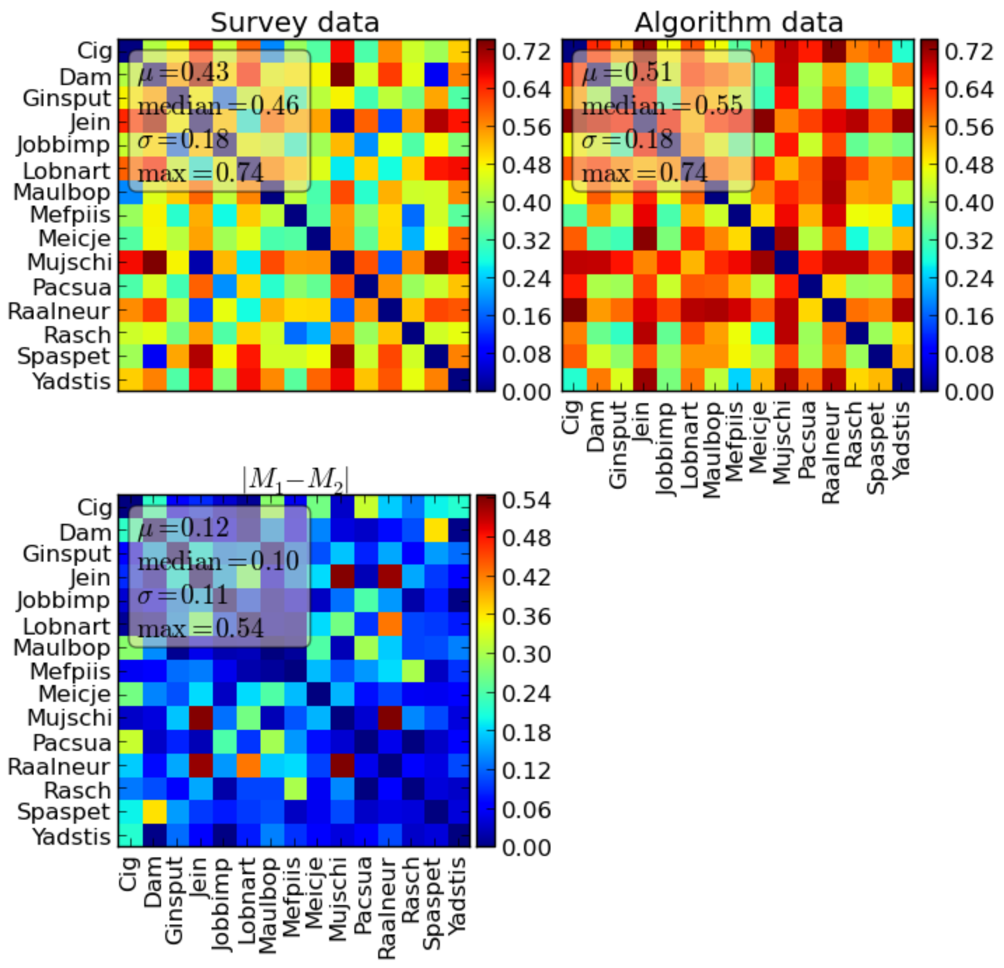


Figure 4.1: The best string distance results were achieved using the 35th codebook. Here they are compared to the survey results.

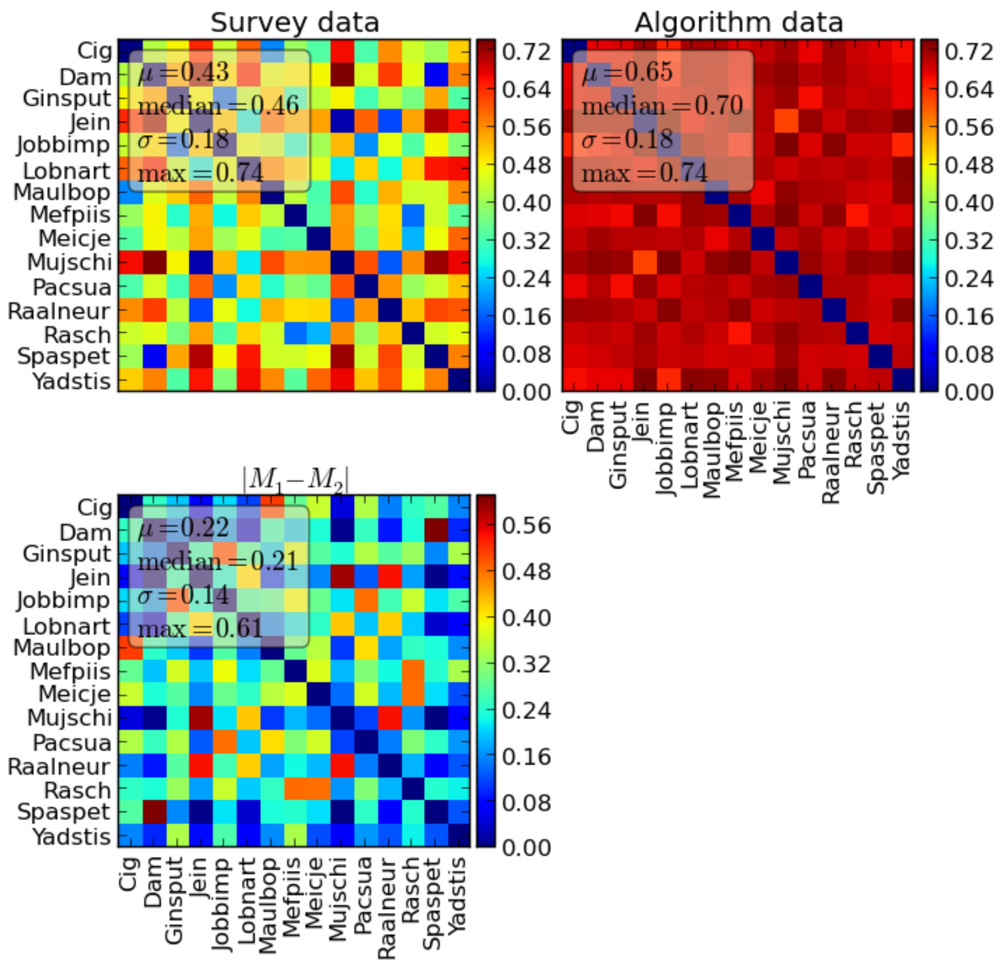


Figure 4.2: The worst string distance results using codebook 14 in comparison with the survey results.

compensated that, these results indicate a big amount of character swaps between the strings. Thus, the codebook does not seem to classify well enough.

### Accumulate

The 200 character sized codebook is too small for the accumulate method. Too many repetitions of characters make the resulting string nearly indistinguishable from other strings created with this method. Figure 4.3 shows the result in comparison to the survey distances. The low range of the values of the algorithm and the low variance underline the indistinguishableness of the results.

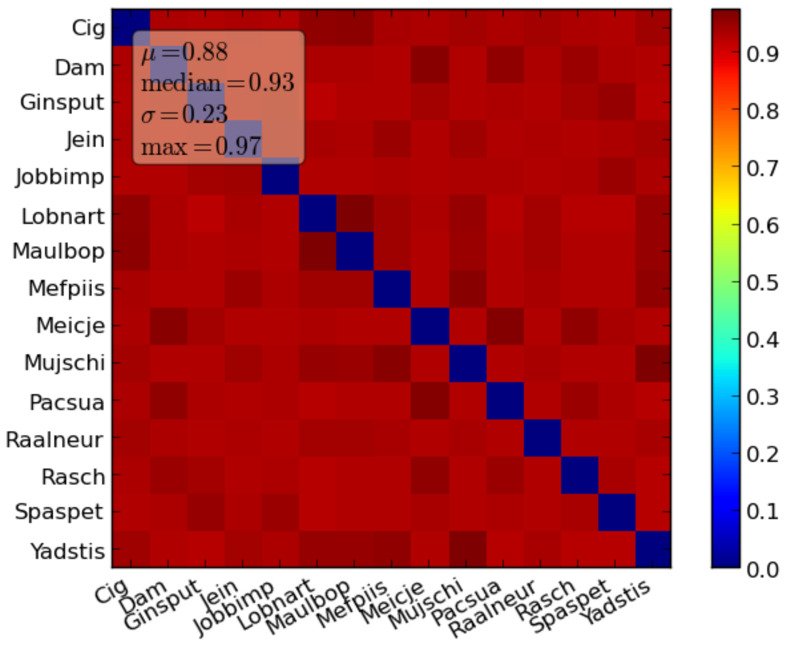


Figure 4.3: Results of the string distance using accumulated VQ data. Almost all songs have the same distance to another due to the extreme re-usage of characters from the alphabet.

Using a bigger alphabet is likely to give better results, however this was not tested in the scope of this thesis.

### EachIndex

As stated before in subsection 4.2.1, computation of the Levenshtein distance over a large string as generated by this method is simply not feasible. Therefore, a direct comparison of whole songs is not possible.

However, for small amounts of data such as in speech recognition or matching song snippets this method may perform well. This was not analyzed in the scope of this thesis but is noted in the Future Work chapter in chapter 5.

### 4.2.2 String indicator

This method produces the best results of all the features implemented in this thesis. It is also computationally efficient as it limits the string which is computed to a length significantly shorter than the actual number of characters.

#### Single

As not every codebook yields the best results, the best performing character and the worst performing characters were selected for comparison. According to the graph presented in Figure 3.19, these values correspond to $x = 38$ for the best and $x = 30$ for the worst result. With the best result shown in Figure 4.4, the songs from the same artist, Jobbimp and Pacsua, are not particularly similar according to the algorithm. This conflicts with the results from the survey. Rasch is the only song which does not match with the survey results at all. According to the algorithm results, Rasch has low distances with Muschji, Lobnart and Jein, which is contradictory to the survey results. Even if more ground truth data is used to compare the results, it is unlikely that Rasch will ever be judged similar to one of said songs, so it can be concluded that this song creates many **false positives**.

Interestingly, with the worst results, shown in Figure 4.5, the roles of Rasch and Yadstis swap with Yadstis being the song having the most differences with the survey results. Looking at the algorithm results, the mild rating of most of the songs may be the cause: almost all songs receive a rating around 0.4 which, by chance, matches up with the survey results, except for Yadstis.

For the future, it would be interesting if the same character, or more specifically, the same box configuration yields the best results in a larger test as well so it is sure that this result is not generated by chance.

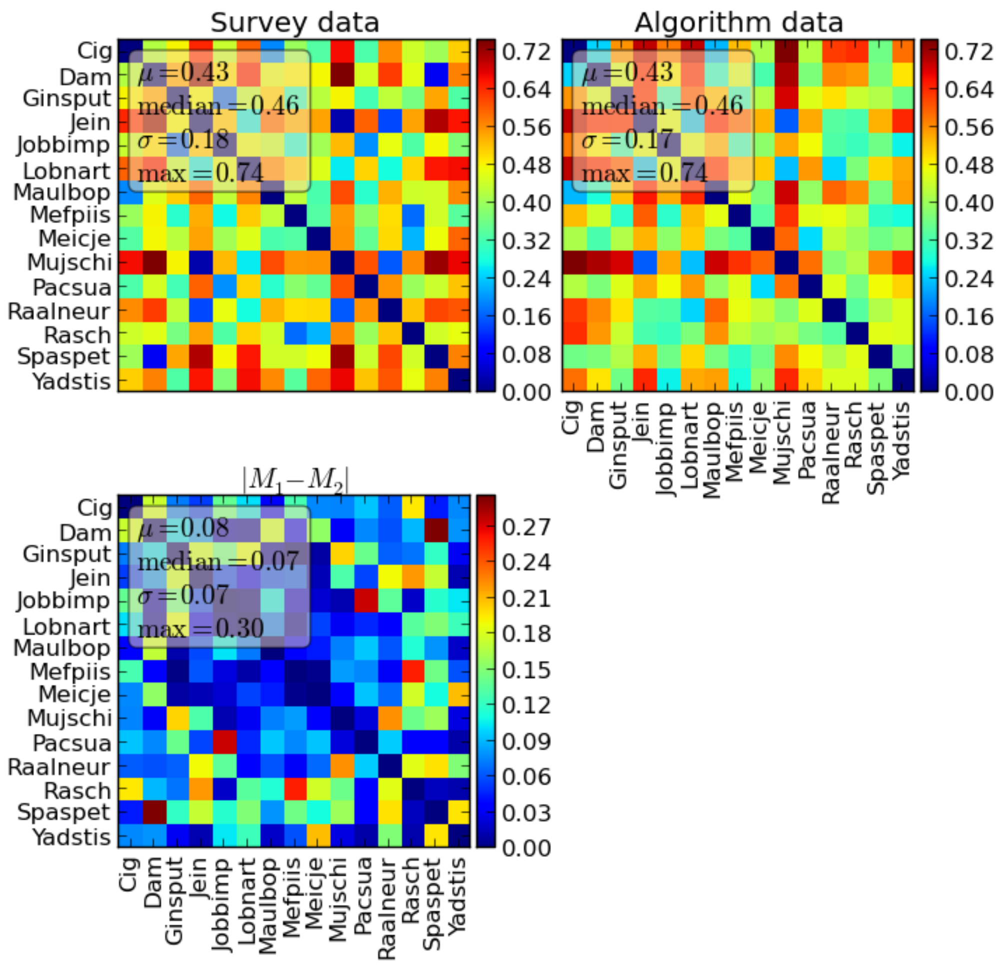


Figure 4.4: Results of the best indicator run using the 38th codebook and a $128 \times 16$ (temporal $\times$ spectral) sized box. The indicator results match the survey results very good, the only consistent outlier is the Ska song Rasch.

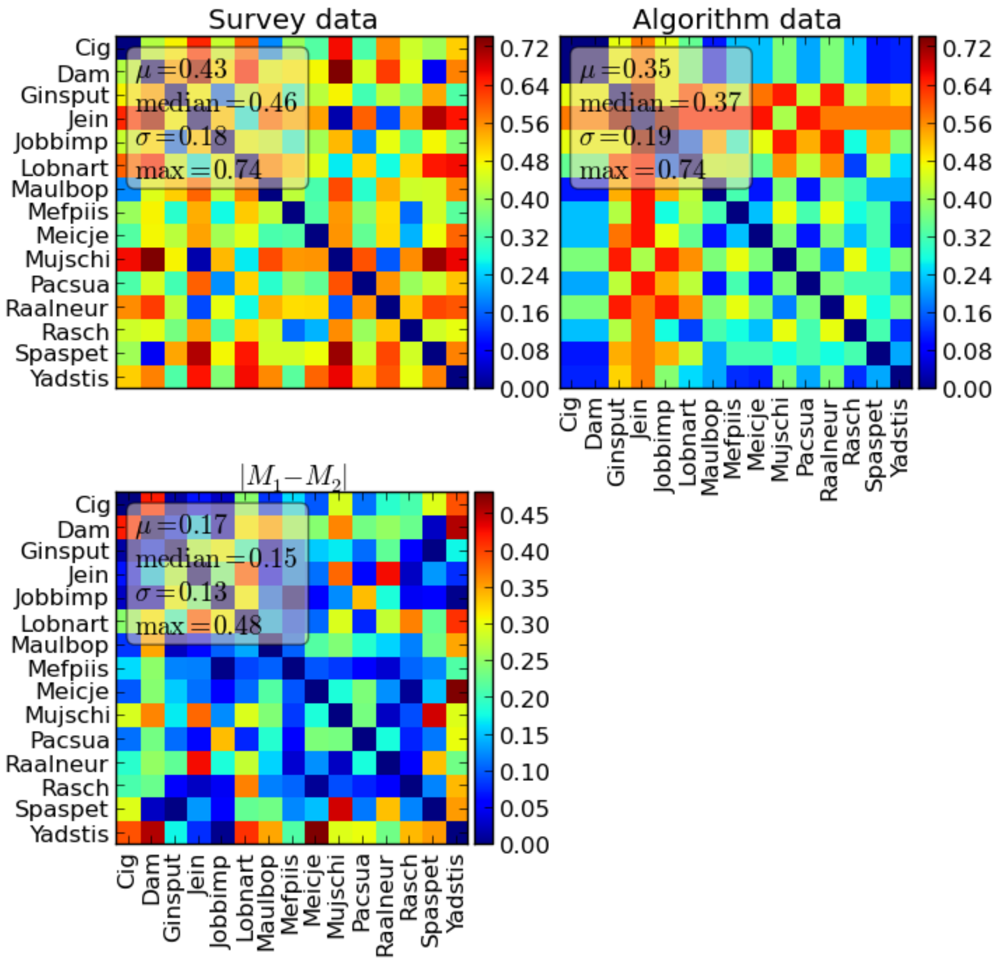


Figure 4.5: Results of the worst indicator run using the 30th codebook and a $32 \times 16$ (temporal $\times$ spectral). Yadstis is the song having the most contradictions with the survey results.

**Accumulate**

The worst result encountered was with the 200 character sized alphabet. As already discussed in the implementation of this features, the alphabet is simply too small and therefore the indicator string of every song contains all available characters of the alphabet. Of course, this leads to a overall distance of zero amongst all songs. A solution to this, is to increase the alphabet size. Figure 4.6 shows the comparison of the results with the accumulated results using the full alphabet of $44 \times 200$ characters.

While the results in Figure 4.6 tend to overlap widely with the survey results, the main properties tend to get lost. Like in the string distance results discussed before, the relationship of the classical songs are very different to the survey results. While in the survey the classical songs have a low distance between each other, Jein, Muschji and Raalneur are not very similar to each other. In fact, the classical songs have a high distance to every other song in general. It is very likely that the high correspondence with the survey result is just a result of the fact that many survey results are rated with distances around 0.5 and only some values differ very explicitly.

### 4.2.3 Distance of Used Character Histograms

The histograms of the used characters presented in this section are produced from two configurations of the survey songs. One configuration uses an alphabet of 200 characters while the other uses the full alphabet of 8800 characters. Using the full alphabet equals the histograms used in the MIREX 2010 comparison presented in (Ness et al., 2011). The first result presented is that of the 200 character alphabet shown in Figure 4.7. In addition to comparing to the survey results, a cover song detection is attempted using the first 10 songs of the covers80 database. All distances between the song histograms are computed using cosine distance.

Some of the dominant factors from the survey results can be found in the result shown in Figure 4.7. The close relationship between the classical songs Jein, Muschji and Raalneur are present as well as some of the Rock relationships, for example Pacsua, Jobbimp and Meiscje. However, the median of the values is very low, meaning that most of the values are in the lower third of the range, making all songs somewhat equal to another and just a few really distinguishable.

The second comparison features the histogram created using the 8800 character alphabet as shown in Figure 4.8. While the first comparison has a very low median, this comparison in contrast has a very high median. Combined with the relatively

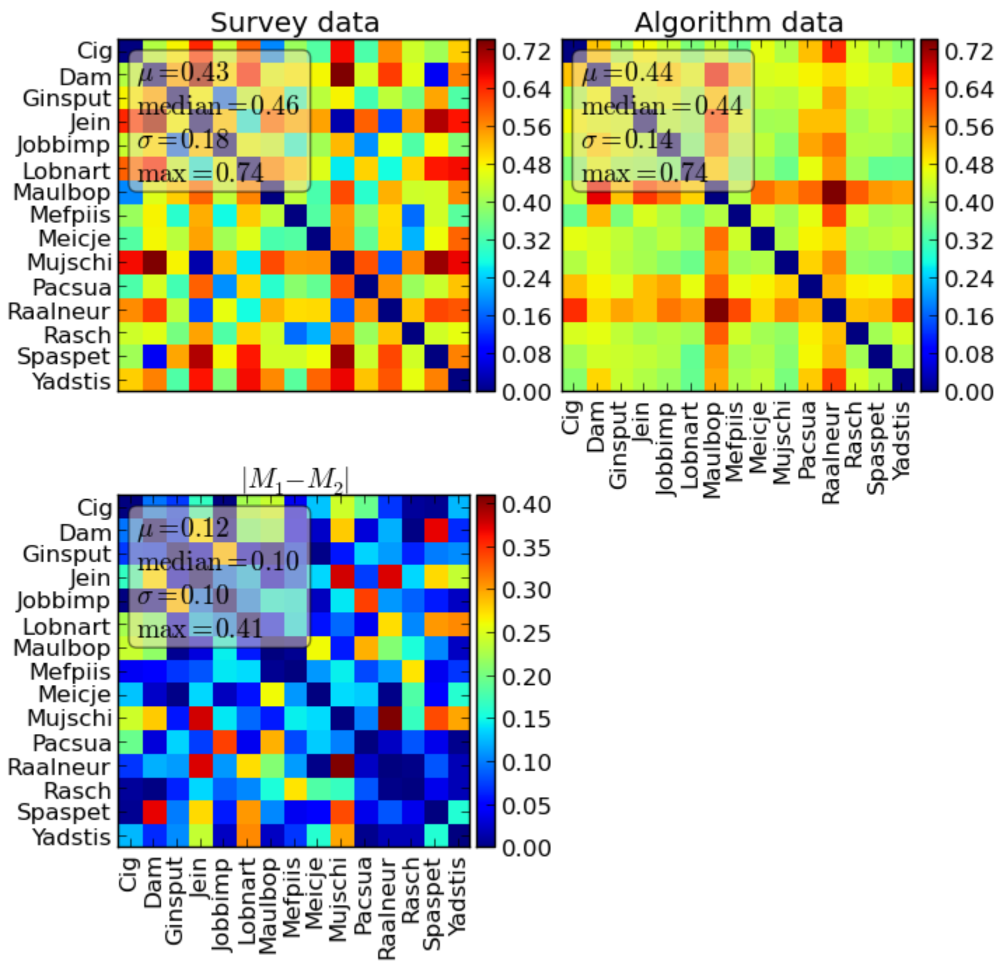


Figure 4.6: String indicator results generated using accumulated quantized indices in comparison to the survey results. The underlying alphabet consists of 8800 characters.

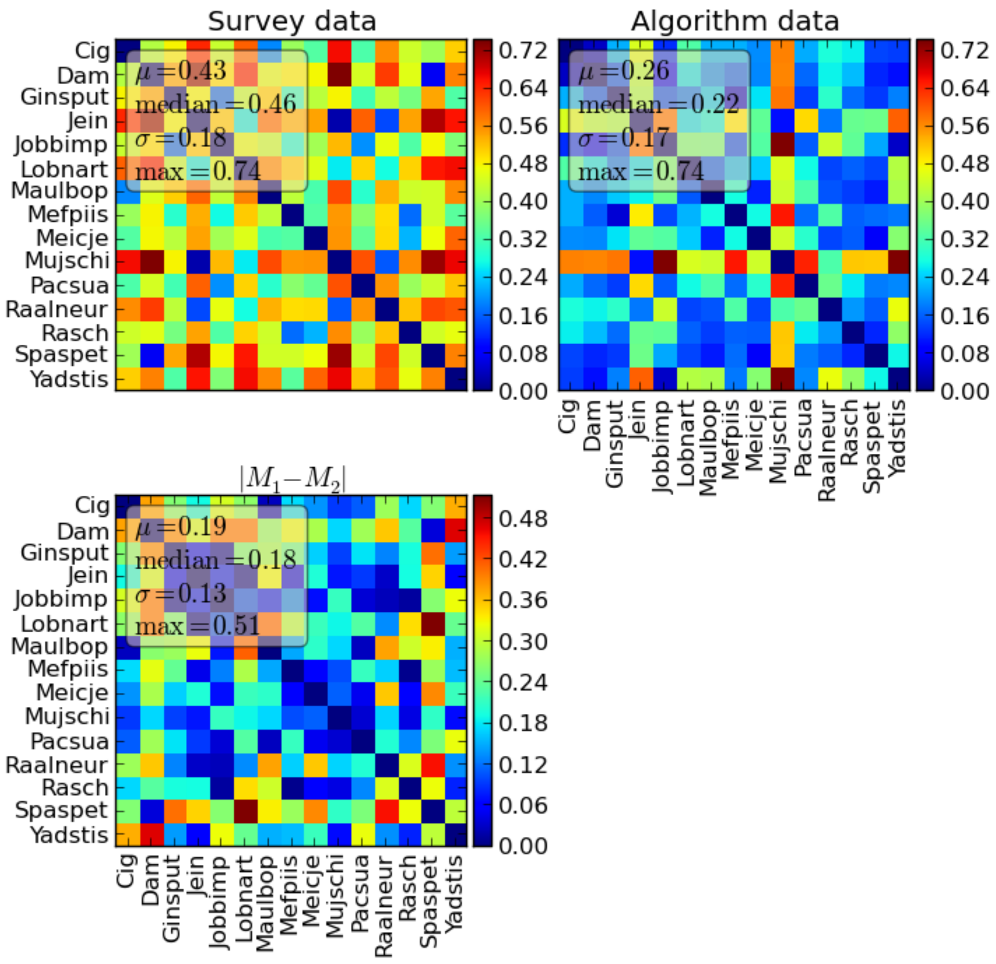


Figure 4.7: Comparison of the survey songs to themselves with the histogram method using cosine distance and an alphabet of 200 characters.

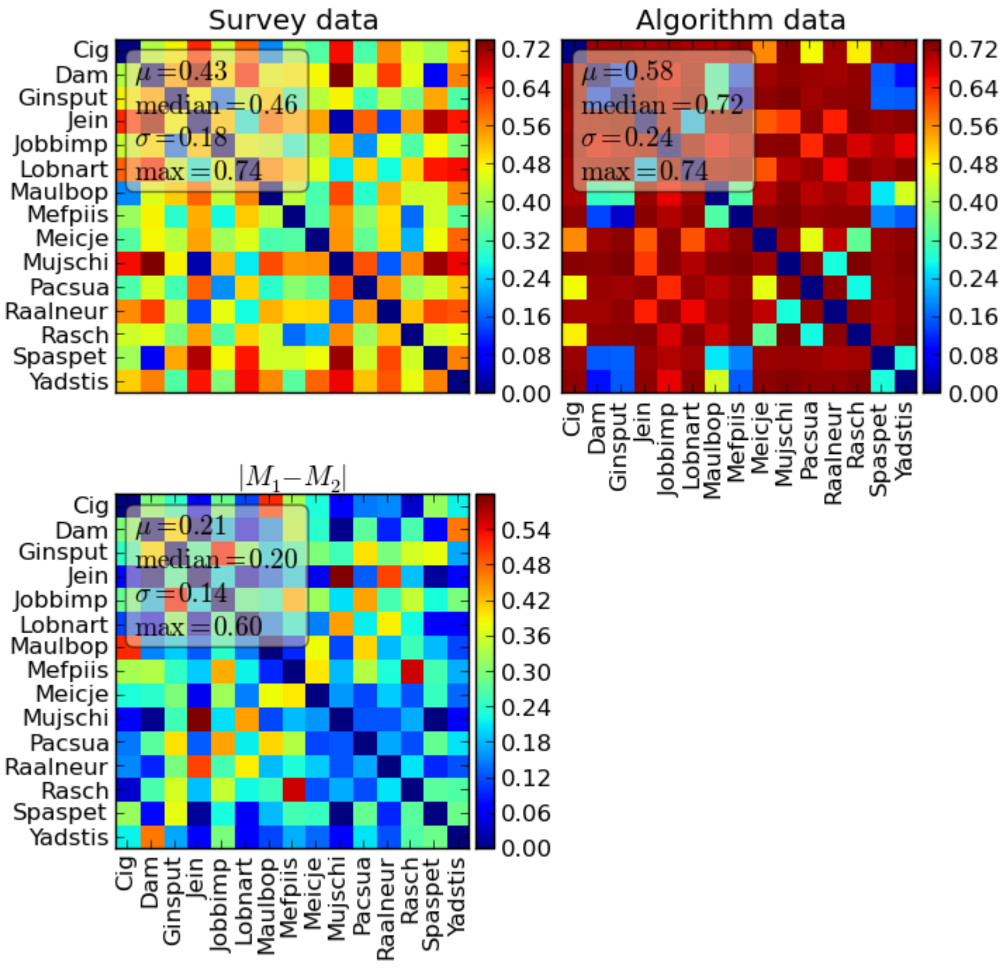


Figure 4.8: Comparison of the survey songs to themselves with the histogram method using cosine distance and an alphabet of 8800 characters.

high mean value, this means that most values have a high value $>0.5$, which makes them indistinguishable in the same ways as in the first comparison.

Concluding, it can be said that varying the alphabet size makes no big difference in the average absolute difference of the result matrices. The first histogram has a mean absolute difference of 0.19 while the second histogram's mean absolute difference is at 0.21. This leads to the conclusion that the better results are achieved with smaller alphabet sizes. Assuming that 200 is the best result, this method performs worse than the other methods discussed in this chapter. However, it would be interesting if this method can be used differently, for example by using only the characters from boxes which performed well in the string indicator or string distance methods.

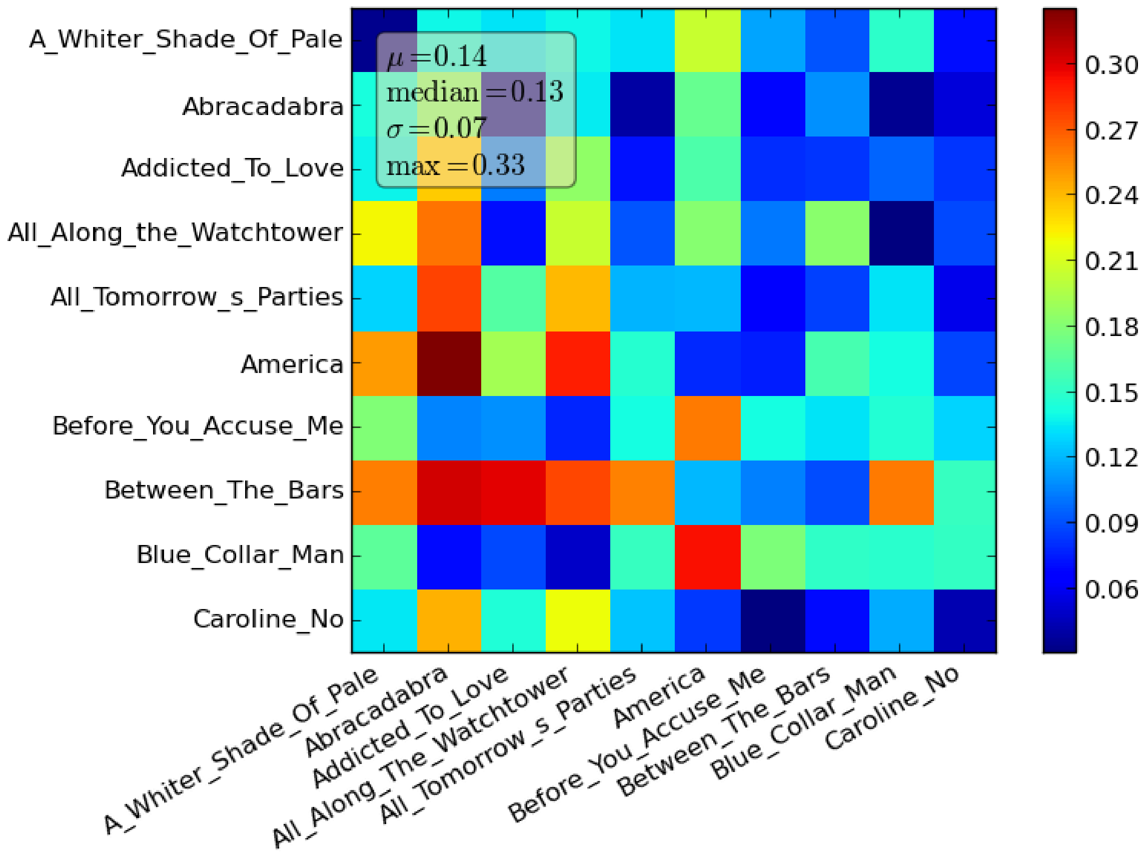


Figure 4.9: Comparison of cover song pairs using cosine distance and an alphabet of 200 characters.

Figure 4.9 shows the distances acquired comparing the covers80 songs. In contrast to the survey results, this compares pairs of cover songs instead of using the same songs. Ideally, all values on the diagonal would be 1.0. However, only for 4 of 10 songs it holds that the value on the diagonal is the smallest distance. In conclusion, the configuration of this method is not suitable for reliable cover detection with only 40% detection rate.

# 5 Future Work

While future work of the survey is already described in subsection 2.2.7, some tasks are worth mentioning again. For example the lack of freely available test data or tests. It would benefit the MIR community a lot if services existed which provide an automated testing based on ground truth for own feature implementations, or, at least a feature independent music database, which the million song database is not. While the Music Information Retrieval Evaluation eXchange (MIREX) is a step in the right direction, automated tests would speed up development and quality of the features submitted to the MIREX. Speaking of the MIREX, it would be interesting to see the features implemented in the scope of this thesis submitted to the MIREX to test them against the MIREX Evalutron 6000 dataset (Gruzd et al., 2007).

Because the implemented features were only tested roughly, further tests and improvements may be made in the future. The single modules used to create the Stabilized Auditory Image may be tuned to match the music recognition task more as proposed by (Ness et al., 2011). Even simpler tunings, for example adjusting the filter bandwidth and maximum frequencies or creating vector quantization codebooks using broader databases may improve the results. After all, the Auditory Image Model was mainly tested for speech recognition tasks and may need further tuning to better match music.

The feature configurations may be tuned as well. For example the string indicator feature mainly used boxes with 16 channel values instead of 32. A broader investigation of the box characteristics with music recognition could yield interesting results. With the histogram method it would be interesting to see whether certain boxes yield better results for the comparison task, as seen in the string indicator method comparisons. Moving on to tasks with smaller amounts of data, the string distance method combined with the `eachIndex` translation method may perform well for speech recognition or similar tasks.

Most of the cover song comparisons turned out to be ineffective. There is room to investigate different approaches to measure the match rate of cover songs, for example simply counting the amount of best matches.

Finally, new ways to operate on the SAIs like the intervalgram proposed by (Walters et al., 2012) show good results and it may be worth thinking of new ways to process the SAIs. Some ways were proposed in the feature section section 2.4 of this thesis, like the mood state machine which may be used as a way for future per-user song recommendations depending on the current mood setting of the user.

# 6 Conclusion

This thesis introduces several new ways to evaluate stabilized auditory images, building on top of the work of (Ness et al., 2011), while implementing three of them. The tests conducted during this thesis show that these features yield results that point in the right direction but lack fine tuning, further evaluation and testing, as discussed in the future work chapter, chapter 5.

In addition to the implementations done, the thesis provides results of a music distance measurement survey using freely accessible music, so that others may use these to test their features with. Furthermore, a successfully tested survey interface was created as a byproduct of said survey. This interface may be used in large scale surveys using, for example, Amazon's Mechanical Turk service.

The Auditory Image Model seems to perform well and the Marsyas implementation of the model is easy to use, inviting developers to build their own features and optimizations upon them.

# Bibliography

Adams, D. (1987). *Dirk Gently's Holistic Detective Agency*, William Heinemann Ltd., London (UK). ISBN: 0-671-69267-4.

Berenzweig, A., Logan, B., Ellis, D. and Whitman, B. (2004). A large-scale evaluation of acoustic and subjective music-similarity measures, *Computer Music Journal* **28**(2): 63–76.

Brothers, T. G. (n.d.). Stardust, baby!
**URL:** http://www.jamendo.com/de/track/423901/stardust-baby

BS (n.d.). Sin.
**URL:** http://www.jamendo.com/de/track/839286/bs-sin

Casey, M. and Slaney, M. (2006). The importance of sequences in musical similarity, *Acoustics, Speech and Signal Processing, 2006. ICASSP 2006 Proceedings. 2006 IEEE International Conference on*, Vol. 5, IEEE, pp. V–V.

Celma Herrada, Ò. (2009). Music recommendation and discovery in the long tail.

Celma, Ò. (2010). *Music Recommendation and Discovery: The Long Tail, Long Fail, and Long Play in the Digital Music Space*, Springer. ISBN: 9783642132865.
**URL:** http://books.google.de/books?id=bwVqHPxd3HMC

Crawford, T., Iliopoulos, C. S. and Raman, R. (1998). String-Matching Techniques for Musical Similarity and Melodic Recognition, *Computing in Musicology* **11**: 71–100.

De La Higuera, C. and Mico, L. (2008). A contextual normalised edit distance, *Similarity Search and Applications, 2008. SISAP 2008. First International Workshop on*, IEEE, pp. 61–68.

Distemper (n.d.). Odejala is krjsch.
**URL:** http://www.jamendo.com/de/track/211586/--

Ellis, D. (2007). The covers80 cover song data set.
**URL:** http://labrosa.ee.columbia.edu/projects/coversongs/covers80/

Faundez-Zanuy, M. and Pascual-Gaspar, J. M. (2011). Efficient on-line signature recognition based on multi-section vector quantization, *Pattern Analysis & Applications* **14**(1): 37–45.

Foote, J. (1997). A similarity measure for automatic audio classification, *Proc. AAAI 1997 Spring Symposium on Intelligent Integration and Use of Text, Image, Video, and Audio Corpora*.

Foote, J. et al. (1997). Content-based retrieval of music and audio, *Proc. SPIE*, Vol. 3229, pp. 138–147.

Gruzd, A. A., Downie, J. S., Jones, M. C. and Lee, J. H. (2007). Evalutron 6000: collecting music relevance judgments, *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '07, ACM, New York, NY, USA, pp. 507–507.
**URL:** http://doi.acm.org/10.1145/1255175.1255307

"H", N. . D. (n.d.). Funk y salsa por favor.
**URL:** http://www.jamendo.com/de/track/47168/funk-y-salsa-por-favor

Heroes, W. (n.d.). Keep it in your arms.
**URL:** http://www.jamendo.com/de/track/592278/keep-it-in-your-arms

Holzapfel, A. and Stylianou, Y. (2010). *Similarity methods for computational ethnomusicology*, Thesis, PhD thesis, University of Crete.

Imprintz (n.d.). In the limo.
**URL:** http://www.jamendo.com/de/track/330243/cck001-b-imprintz-in-the-limo

KEPAY (n.d.). Cule cule bom bom.
**URL:** http://www.jamendo.com/de/track/7055/cule-cule-bom-bom

Lee, J. H. (2010). Crowdsourcing music similarity judgments using mechanical turk, pp. 183–188.
**URL:** http://ismir2010.ismir.net/proceedings/ismir2010-33.pdf

Levenshtein, V. (1966). Binary Codes Capable of Correcting Deletions, Insertions and Reversals, *Soviet Physics Doklady* **10**: 707.

Luisi, G. (n.d.). Partita no. 1-4 sarabande.
**URL:** http://www.jamendo.com/en/track/259810/partita-no.-1-4.-sarabande

Lyon, R. F. (2011). A pole-zero filter cascade provides good fits to human masking data and to basilar membrane and neural data, *Mechanics of Hearing*.

Lyon, R., Rehn, M., Bengio, S., Walters, T. and Chechik, G. (2010). Sound retrieval and ranking using sparse auditory representations, *Neural computation* **22**(9): 2390–2416.

Marzal, A. and Vidal, E. (1993). Computation of normalized edit distance and applications, *Pattern Analysis and Machine Intelligence, IEEE Transactions on* **15**(9): 926–932.

Ness, S., Walters, T. and Lyon, R. (2011). Auditory sparse coding, *Music Data Mining* **21**: 77.

Nesto (n.d.). Sales histoires.
**URL:** http://www.jamendo.com/de/track/790241/sales-histoires

OnClassical (n.d.). Johann sebastian bach: Partita in c-minor.
**URL:** http://www.jamendo.com/en/track/146589/johann-sebastian-bach-partita-in-c-minor-bwv
sinfonia

Pandora (2013). About the music genome project.
**URL:** http://www.pandora.com/about/mgp

Patterson, F., Mike, H. and Giguere, C. (1995). Time-domaln modelling of peripheral auditory processing, *J. Acoust. Soc. Am* **98**(4).

Patterson, R. D. (1987). A pulse ribbon model of monaural phase perception, *The Journal of the Acoustical Society of America* **82**: 1560.

Patterson, R. D. and Bleeck, S. (1995). Auditory image model implementation manual.
**URL:** http://www.pdn.cam.ac.uk/groups/cnbh/aimmanual/

R Core Team (2012). *R: A Language and Environment for Statistical Computing*, R Foundation for Statistical Computing, Vienna, Austria. ISBN 3-900051-07-0.
**URL:** http://www.R-project.org

Scherer, K. R. and Zentner, M. R. (2001). Emotional effects of music: Production rules, *Music and emotion Theory and research* **emotion**: 361–392.
**URL:** http://psy2.ucsd.edu/~charris/SchererZentner.pdf

Sucks, B. (n.d.a). Dropping out of school.
**URL:** http://www.jamendo.com/de/track/210907/dropping-out-of-school

Sucks, B. (n.d.b). Making me nervous.
**URL:** http://www.jamendo.com/de/track/30058/making-me-nervous

S.U.N. (n.d.). Another recipe.
**URL:** http://www.jamendo.com/de/track/196155/another-recipe

Teichert, M. (n.d.). Inspiration.
**URL:** http://www.jamendo.com/de/track/945026/inspiration

Turnbull, D., Barrington, L., Torres, D. and Lanckriet, G. (2007). Towards musical query-by-semantic-description using the cal500 data set, *Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval*, SIGIR '07, ACM, New York, NY, USA, pp. 439–446.
**URL:** http://doi.acm.org/10.1145/1277741.1277817

Tzanetakis, G. (2007). Marsyas-0.2: a case study in implementing music information retrieval systems, *Intelligent Music Information Systems. IGI Global* .

Walters, T. (2011). Auditory-based processing of communication sounds.

Walters, T., Ross, D. and Lyon, R. (2012). The intervalgram: An audio feature for large-scale melody recognition, *9th International Symposium on Computer Music Modeling and Retrieval*.

Wood, G. and O'Keefe, S. (2005). On techniques for content-based visual annotation to aid intra-track music navigation, *Proceedings of ISMIR*, pp. 58–65.

WORK, A. (n.d.). Giù.
**URL:** http://www.jamendo.com/de/track/576761/gi

# Glossary

**AIM-C**
Auditory Image Model reference implementation written in C/C++.. 49

**AIM**
The Auditory Image Model is a model of human hearing which applies the limitations and pre-processing of the human ear and neural processing to audio signals.. 38

**Bandwidth**
The range defined by the difference of the upper frequency to the lower frequency of, e.g., a filter.. 55

**BMM**
Basilar Membrane Motion: The motion of the basilar membrane, a part of the human ear. It translates the sound waves into motion and, by that, affects tiny hair cells which generate a neural response.. 54

**ERB**
Equivalent Rectangular Bandwidth: The bandwidth of a specific center frequency of a simplified version of a human audio filter in the ear. Those filters are simplified to be rectangular band-pass filters.. 55

**Marsyas**
Comprehensive modular audio framework which supports rapid prototyping of audio signal processing applications.. 48

**NAP**
Neural Activity Pattern: A pattern generated by the movement of hair cells in the inner ear. Said hair cells are moved by the motions of the Basilar Membrane.. 55

**NumPy**
Fast numerical array computations for Python.. 49

**Samples**
Data points of the discrete signal acquired by the sampling function, for example the Discrete Fourier Transform.. 68

**SciPy**
Comprehensive scientific Python framework.. 49

**SAI**
The Stabilized Auditory Image is one possible end-product of the Auditory Image Model. It is the Neural Activity Pattern generated by the inner ear simulation in a stabilized form.. 38

**Temporal**
Something relating to time. A temporal shift, for example, can relate to a shift in time.. 37

**VQ**
Vector Quantization: The translation of an input vector to an index of a codebook. Said codebook contains vectors which the input vector is matched against using a nearest-neighbor method. The closest match determines the result, which is the index of the matched vector.. 41

**Window**
Window function: A function which limits the input to an interval and zeros the values not in the range. Often used to slice a longer signal into smaller blocks.. 68

# List of Figures

# List of listings

# Versicherung über die Selbstständigkeit

Hiermit versichere ich, dass ich die vorliegende Arbeit im Sinne der Prüfungsordnung nach §16(5) APSO-TI-BM ohne fremde Hilfe selbstständig verfasst und nur die angegebenen Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen habe ich unter Angabe der Quellen kenntlich gemacht.

| | |
|---|---|
| Hamburg, 25.03.2013 | |
| Ort, Datum | Unterschrift |