

Tarsos Manual

Joren Six
joren.six@hogent.be

Royal Academy of Fine Arts & Royal Conservatory,
University College Ghent

November 24, 2011

Contents

1	Introduction	2
1.1	What is Tarsos?	2
1.2	Organization of this Manual	3
2	Getting Started	4
2.1	Requirements	4
2.2	Installation	4
2.3	Detecting a Tone Scale With Tarsos	5
2.4	Scala Files and Tarsos	6
2.5	Export Results	9
2.5.1	Annotations	9
2.5.2	Pitch Histogram	9
2.5.3	Pitch Class Histogram	9
2.5.4	Pitch Class Data	9
3	Advanced Use	11
3.1	Connecting a MIDI Keyboard	11
3.2	External Pitch Detectors	11
3.3	Alternative Peak Detection Schemes	11
4	Processing Datasets with Tarsos	12
4.1	Directory and file naming structure	12
4.2	Command line applications	13
4.3	Scripting Tarsos	13
4.3.1	The Tarsos API	13
4.3.2	Example scripts	14
A	Pitch, Pitch Interval & Pitch Ratio Representation	20
A.1	Pitch & Pitch Interval Representation	20
A.2	Pitch Ratio Representation	21
A.3	Conclusion	22
B	Maqams	23

Chapter 1

Introduction

This is the manual for Tarsos a modular software platform to extract and analyze pitch organization in musical audio. Tarsos can be used to automatically detect the tone scale of most music. The manual is geared towards:

- Musicologists: to identify tone scales in musical pieces. Either detailed, manually assisted analysis of one piece or automatic analysis of a large number of pieces.
- Musicians: to improve their intonation.
- Composers: to experiment with (micro) tonality.

1.1 What is Tarsos?

Tarsos is a modular software platform to extract and analyze pitch and scale organization in music, especially aiming at the analysis of non-Western music. Tarsos aims to be a user-friendly, graphical tool to explore tone scales and pitch organization in music of the world. With Tarsos pitch annotations are extracted from an audio signal that are then processed to form musicologically meaningful representations. These representations cover more than the typical Western 12 pitch classes, since a fine-grained resolution of 1200 cents is used.

The Tarsos API creates opportunities to analyse large sets of - ethnic - music automatically. With the Application Programmers Interface tasks can be automated by programming scripts. For examples see 4.3.1. The graphical user interface can be used for detailed, manually adjusted analysis of specific songs. Section 2.3 contains details on how to work with the interface. Several output modalities make Tarsos an interesting tool for musicological analysis, educational purposes and even for artistic productions. The different output options are listed in section 2.5.

Tarsos is open source and runs on any recent Java Runtime. JRE (Java Runtime Environment) 5 is supported, 6 or higher advised. Tarsos is available

on the Tarsos website¹ and is developed at the Royal Academy of Fine Arts & Royal Conservatory, University College Ghent, Belgium² To cite our work please refer to [8]:

```
@inproceedings{six2011tarsos,
  author    = {Joren Six and Olmo Cornelis},
  title     = {{T}arsos - a {P}latform to {E}xplore {P}itch
              {S}cales in {N}on-{W}estern and
              {W}estern {M}usic},
  booktitle = {Proceedings of the 12th ISMIR Conference},
  year      = {2011}
}
```

1.2 Organization of this Manual

This text is structured as follows: after this introduction the next chapter contains a down to earth explanation on how to get started. It features some basic use cases. More advanced use is explained in the following chapter. In the final chapter the graphical interface is left behind: it covers how to process large sets of audio from the command line or by writing scripts yourself.

Text that is present somewhere on the GUI (Graphical User Interface) is represented by a mono spaced font. E.g. the File-Open... menu. If you reading this manual digitally the URL's used throughout this text are clickable E.g. the Tarsos Website³, the full URL is embedded as a footnote, for reading on paper. Bold is used to give a quick summary of a paragraph: the TL;DR. E.g. **typographical conventions**. Source code is embedded as in Listing 1.1.

Listing 1.1: Source code example

```
1 if(true) {
   Console.println(      )
}
```

¹<http://tarsos.0110.be>

²<http://cons.hogent.be>

³<http://tarsos.0110.be>

Chapter 2

Getting Started

2.1 Requirements

Tarsos is programmed in Java. To run Tarsos you need a **recent Java Runtime Environment (JRE)** on your computer. JRE version 5.0 or newer is required, JRE version 6 is advised. To check if Java is installed on your machine use the tools provided on the Java website¹. There you can also find which version of Java is installed on your machine. If Java is not installed you can find installation instructions, for different operating systems, on the same website.

Apart from a Java Runtime, there are little other requirements. Tarsos should work on **any operating system** and is regularly tested on Ubuntu 11.04, Windows XP, Windows 7 and Mac OS X 10.7. As to be expected analysis on large files will go faster on a more recent, beefier computer.

2.2 Installation

To run Tarsos you need a recent JavaRuntime Environment (JRE) on your computer. See 2.1 to check if you have Java installed or for installation instructions.

The installation procedure for Tarsos is straightforward: you need to **download one executable file** (a JAR-file) and double click it. The executable can be found on the Tarsos website². If double clicking does not work, please check if Java is properly installed on your system.

Linux or Unix systems using the PulseAudio sound system do not cooperate very well with the Sun (Oracle) Java 6 Runtime Environment. To alleviate this annoyance it is recommended to install the PulseAudio to Java bridge. Documentation on how to get PulseAudio Support³ for Sun Java 6 an Ubuntu

¹<http://java.com>

²<http://tarsos.0110.be/attachment/tarsos.jar>

³http://tarsos.0110.be/artikels/lees/PulseAudio_Support_for_Sun_Java_6_on_Ubuntu

system with an AMD64 processor architecture is available. The procedure for other processor architectures or other Unix-like operating systems is similar.

2.3 Detecting a Tone Scale With Tarsos

Once Tarsos is correctly installed and up and running you can start detecting tone scales. The following procedure documents how you can find the tone scale of a piece of music. A more formal description of this task is how to detect the most commonly used *pitch classes* within a piece of music. This set of pitch classes can be a subset of the complete scale. The procedure is based on peak detection on a pitch class histogram, which is only useful for music with pitch organized in octaves. Music without octave equivalence is much less common but can be analysed using the pitch histogram which contains the same information, not reduced to one octave. In [3] the following is stated about pitch relationships and octave equivalence:

“... the use of discrete pitch relationships, as well as the concept of octave equivalence seem, while not universal in early and pre-historic music (Nettl 1956; Sachs 1962), rather common to current musical systems (Burns 1999).”

To be able to interpret the results one needs to fully understand both the limitations of the pitch detection algorithm used and how a pitch (class) histogram is constructed. Knowledge about the music under analysis is also a boon.

To execute this task **you need an audio file** with an interesting tone scale. An example audio file with a tone scale based on an octave division in 10 equal parts of 120 cents⁴ can be found here⁵. Examples using this file will be returning throughout this text.

1. Using the File-Open... menu to **choose a file** is the first step to detect a tone scale. Tarsos is also capable to open audio files by drag and drop. The audio can be in almost any format. In the background FFmpeg⁶ is used to convert audio to WAV. This is indicated by 1 in Figure 2.1.
2. The second step is simple: wait until **the file is analysed** by one or more pitch detection algorithms. By default a platform independent implementation of YIN[1] is used, it should 'just work'. See 3.2 if you want to try out other pitch detectors. After the file is analysed it starts to play and a pitch class histogram emerges. This histogram represents how many times each pitch class is detected. This is indicated by 2 in Figure 2.1.

⁴See appendix A for more information about the cents unit. There you can also find conversion formulas and pitch representation conventions used within Tarsos.

⁵http://tarsos.0110.be/attachment/cons/200/bach_BWV_1007_120.mp3

⁶FFmpeg is used by Tarsos as a cross-platform solution to convert audio. It has support for MP3, FLAC, Monkey's Audio, WMAv2, Theora, MusePack, etc. The supported audio formats depend on which version of FFmpeg is used by your system.

3. The third step is to **detect the pitch classes** that are most frequently used. In practice this means peak detection on the pitch class histogram. This can be done automatically and is visualised in Figure 2.1 as step 3. The sliders modify parameters of the peak detection strategy. Immediate visual feedback shows the detected locations in the pitch class histogram, the detected pitch classes. Manual adjustment is sometimes required and is therefore possible:

- To *add a pitch class* hold down `alt` while hovering the mouse pointer over the pitch class histogram. Pressing the left mouse button while hovering fixes it to the current location.
- To *change a pitch class* hold down `ctrl` while hovering over the pitch class histogram, this selects the closest pitch class. Pressing the left mouse button while hovering fixes it to the new location.
- To *remove a pitch class* hold down `ctrl` while hovering over the pitch class histogram, this selects the closest pitch class. Pressing `delete` or `d` removes the selected pitch class.

Auditory feedback is possible by clicking the pitch class histogram. This produces a sound that can be compared to the actual pitch classes present in the analysed audio.

2.4 Scala Files and Tarsos

Scala files are descriptions of tone scales. They are defined and used by the Scala software program. To quote the Scala website⁷:

“Scala is a powerful software tool for experimentation with musical tunings, such as just intonation scales, equal and historical temperaments, microtonal and macrotonal scales, and non-Western scales. It supports scale creation, editing, comparison, analysis, storage, tuning of electronic instruments, and MIDI file generation and tuning conversion. All this is integrated into a single application with a wide variety of mathematical routines and scale creation methods. Scala is ideal for the exploration of tunings and becoming familiar with the concepts involved. In addition, a very large library of scales is freely available for Scala and can be used for analysis or music creation.”

The Scala program comes with a data set of over 3900 scales ranging from historical harpsichord temperaments over ethnic scales to scales used in contemporary music. This data set has been used, amongst many other applications, in a study trying to find universal properties of scales[3]. Tarsos can parse and export Scala scale files. This feature makes Tarsos great in tandem with Scala.

⁷<http://www.huygens-fokker.org/scala/>

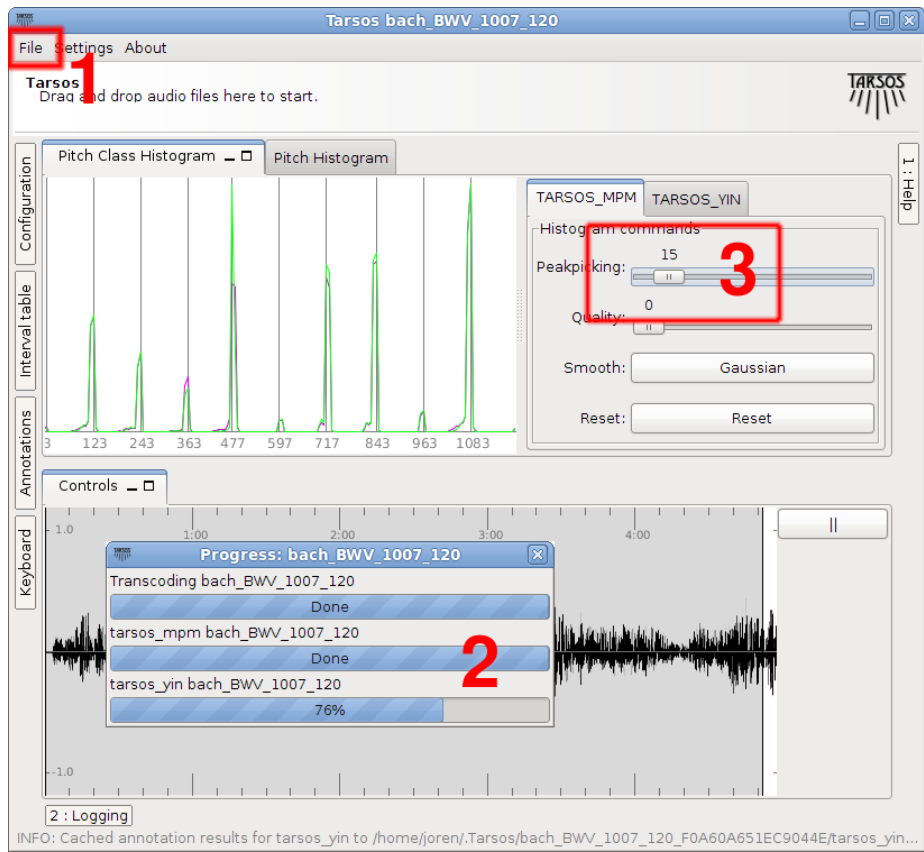


Figure 2.1: How to detect a tone scale with Tarsos 1 use the file menu to open a file, 2 the file is converted and pitch is analysed automatically, 3 use the peak picking slider to automatically detect pitch classes in the pitch class histogram.

The Scala scale file format is a simple, human readable format which uses plain text files. A complete definition can be found on the Scala scale file format reference page⁸. This is an example of a heptatonic Indonesian Pelog scale, encoded using the file format rules. All values, except for the last one, are in cents: the last one uses a fraction. Understanding the file format can help you to create your own scales and gain insight in the results of Tarsos.

```
! pelog_me1.scl
!
Gamelan Kyahi Kanyut Mesem pelog. 1/1=295 Hz
7
!
124.521
271.058
522.809
688.207
787.819
954.515
2/1
```

There are a couple of things you can do with Scala files and Tarsos.

- **Compare the current scale with a scala file.**
 1. Detect the tone scale of a song. See section 2.3. Important: peak detection needs to be done.
 2. Open a scala file using the File-Open... menu.
 - 3.
- **Find the most similar scale to the current scale from a folder of Scala files.**
 1. Detect the tone scale of a song. See section 2.3. Important: peak detection needs to be finished.
 2. Create a folder with a number of scala files.
 3. Open the folder using the File-Open... menu.
 4. The closest scale is detected from the folder and overlaid on the detected pitch class histogram.
- **Tune a MIDI Synthesizer with a Scala file:** Doing this is simple. Choose the Scala file using the File-Open... menu and the internal MIDI Synthesizer is tuned immediately according to description found in the Scala file. If you want to tune another MIDI synthesizer you need to check if it supports MIDI TUNING DUMP-messages⁹ and select the correct output in the Settings-MIDI Devices-Output menu.

⁸http://www.huygens-fokker.org/scala/scl_format.html

⁹It probably does not. There are very few synthesizers with support for these types of messages. Even most software synthesizers do not implement them. Notable exceptions are: Gervill (used by default within Tarsos),

Start (seconds)	Pitch (Hz)	Probability 0-1	Source
11.0875	363.2373	0.80101	TARSOS_YIN
12.0744	341.1381	0.89111	TARSOS_YIN
⋮	⋮	⋮	⋮

Table 2.1: Example CSV output for annotations

2.5 Export Results

Tarsos contains export capabilities for every step in the process, from raw pitch annotations until the pitch class interval matrix. The built-in export functions are ordered from low to higher level musicological meaning. Here an overview is given of the export modalities, using the same order.

2.5.1 Annotations

- CSV file see 2.1
- Audio

2.5.2 Pitch Histogram

- CSV
- PNG
- EPS

2.5.3 Pitch Class Histogram

- CSV
- PNG
- EPS
- PNG octave splits
- \LaTeX see figure 2.2

2.5.4 Pitch Class Data

- Scala file
- Interval matrix \LaTeX see table 2.2

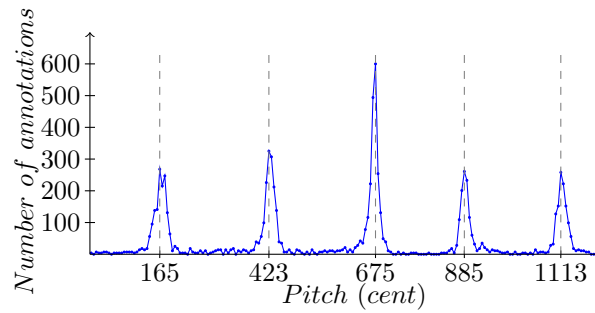


Figure 2.2: Example of a \LaTeX -export of a pitch class histogram

P.C.	165	423	675	885	1113
165	0	258	510	720	948
423	942	0	252	462	690
675	690	948	0	210	438
885	480	738	990	0	228
1113	252	510	762	972	0

Table 2.2: Example of a \LaTeX -export of a pitch class interval matrix. P.C. stands for pitch class, the values are given in cents.

Chapter 3

Advanced Use

3.1 Connecting a MIDI Keyboard

MIDI routing , MIDI keyboard. Alternative synths.

3.2 External Pitch Detectors

- How to configure vamp plugins
- How to configure ipem six
- MPM[5].

3.3 Alternative Peak Detection Schemes

Chapter 4

Processing Datasets with Tarsos

General tips and tricks (md 5, ...) configuration of directories, shares server processing.

4.1 Directory and file naming structure

Since transcoding and detecting pitch are computationally expensive operations Tarsos contains facilities to prevent needless work. Imagine the following: you are in possession of a large data set (10000+ files) of music using all sorts of scales and want to do research on pitch use in those files, together with a couple of colleagues, lets call them Franz-Jozeph, Gerhard and Edwin - you are Edwin. To make things easy there is a central server with a file share of the data set available for everybody on the network. As always Edwin, the computer science guy, has made this possible. The central server also contains a file share writable by Franz-Jozeph, Gerhard and, of course, our protagonist Edwin. Everybody has Tarsos running on their machine with the data directory of Tarsos configured to this writable share.

Every time a file - `file_name.ogg` - is analysed by one of the three friends a new directory is created on the share. The new directory has a name looking like this: `file_name_5878ef547f0942ec`. This directory contains a transcoded version of the file `file_name_transcoded.wav` and zero or more pitch detection result files: `tarsos_yin_file_name.txt`. The `5878ef547f0942ec` part of the directory name is an MD5[6] calculated using the first 256kB of the original file. When the exact same file is analysed first by Edwin - on Linux - and later by Gerhard - on Windows - transcoding and pitch detection is only executed once. This scheme even allows that Gerhard, who uses Mac OS X, can copy and rename the file on his local hard drive and analyse it quickly: it is still recognised as the same file and transcoded, pitch detected only once. Getting rid of the waiting time almost entirely is possible by analysing each file once on

the server.

4.2 Command line applications

A couple of command line apps. Annotator app.

4.3 Scripting Tarsos

This section explains how to write batch scripts that process many audio files or how to use the inner functions of Tarsos in a new way. Tarsos is written in Java, and is extendable using scripts in any language that targets the JVM (Java Virtual Machine). The functions exposed by the Tarsos API can be called using e.g. JRuby, Jython, Scala, Clojure, Groovy, and of course plain old Java¹. In this text we will use Scala² for its concurrency support, concise syntax and seamless interoperability with Java. Although the Scala programming language is used here, the concepts explained apply to scripts in any language.

4.3.1 The Tarsos API

Full documentation for the Tarsos API³ is available. The API is rather extensive but the main abstractions are documented here. This should help you to get started.

- `AudioFile` An encapsulation for a sound file. It has methods to transcode audio, query its length, location, calculate hashes, . . .
- `Configuration` Manages configuration settings for Tarsos. It uses the default Java configuration manager. It is an utility class to read and write configuration settings.
- `PitchHistogram` A (non octave reduced) histogram.
- `PitchClassHistogram` An octave reduced histogram.
- `KernelDensityEstimate` A different kind of histogram, constructed using kernels to prevent the influence of bin widths. When using Gaussian kernels it also creates a smoother curve.

Instead off a tiresome description of all components the API provides the remainder of this chapter contains some task oriented examples. The examples should make the possibilities of the API clear and illustrate how to leverage those capabilities. If the task you want to achieve is not similar to one of the examples you can still consult the full documentation.

¹The website of each language: <http://www.jruby.org/>, <http://www.jython.org/>, <http://www.scala-lang.org/>, <http://clojure.org/>, <http://groovy.codehaus.org/>

²Please do not confuse the general purpose Scala programming language with the tool to experiment with tunings, the Scala program.

³<http://tarsos.0110.be/attachment/doc/>

4.3.2 Example scripts

The example scripts use the Scala programming language version 2.9, to install Scala correctly on your platform, please see the Scala website⁴. The scripts can be executed from a command line when they have the correct header. For the Bash shell under Unix compatible systems the header looks like this:

Listing 4.1: Pitch Class histo creation

```
#!/bin/sh
2 exec scala -cp tarsos.jar -savecompiled @
!#
import be.hogent.tarsos.util._
//other import statements
```

The header makes the Tarsos functions available in the script by including `tarsos.jar`, located in the same directory as the script. The `savecompiled` option saves the compiled version so the start up time of the second run is limited. The first statement tells Bash which interpreter to use - `scalac` -, this makes running it by calling `./scala_script.sh` possible. Note that the file needs to be executable `chmod +x scala_script.sh`.

Iterate and transcode files

The first task is strictly utilitarian. This is the scenario: over the years I have collected music with different tone scales. All the music is nicely digitized but, as it often happens, I was a bit sloppy. There files are stored in various formats (MP3, WAVE, WMA, APE, FLAC, ...) and some duplicates are present in my botched directory structure. The aim of this task is to identify duplicates and transcode all music to one, lossless, format. Also all the files are copied to one directory.

Configuring the target directory is the first step. Luckily this is as simple as calling `Configuration.set (ConfKey.data_directory, "/home/user")`.

Listing 4.2: Pitch Class histo creation

```
def handleAudioFile(audioFile: AudioFile) = {
  val detectorYin = PitchDetectionMode.VAMP_YIN_FFT.getPitchDetector
    (audioFile)
  detectorYin.executePitchDetection()
}
5

def handleSomeAudioFiles(audioFiles: List[String], modulo: Int, thread
  : Int){
  var counter = 0;
  audioFiles.foreach{ file =>
10   if(counter % modulo == thread){
      val audioFile = new AudioFile(file)
      handleAudioFile(audioFile)
    }
```

⁴<http://scala-lang.org>

```

    }
    counter = counter + 1;
15 }
}

def handleAudioFilesConcurrently(numberOfThreads: Int) = {
    val directory =
20 val audio_pattern = Configuration.get (ConfKey.
    audio_file_name_pattern)
    val audioFiles = FileUtils.glob(directory, audio_pattern, true).
        toList
    for( i <- 0 until numberOfThreads) {
        actor {
            self ! handleSomeAudioFiles(audioFiles, numberOfThreads, i)
25 }
        Console.println(
            + i);
    }
}

30 handleAudioFilesConcurrently(6)

```

Tone Scale estimation

This script implements following task: find the tone scales most similar to the one used in recorded music. To complete this task you need a small set of theoretical scales and a large set of music, each brought in one of the scales. To make it more concrete, an example of Turkish classical music is used.

In an article by Bozkurt[2] pitch histograms are used for - amongst other tasks - makam⁵ recognition. The task is to identify which of nine makams is used in a specific song. A simplified, generalized implementation of this task is shown here. In this implementation there is no tonic detection step. Also here we use only theoretical descriptions of the tone scales as a template and do not construct a template using the audio itself, as is done by Bozkurt.

Listing 4.3: Tone Scale Estimation

```

val makams = List(
    ,
    ,
    ,
    ,
    ,
    ,
    ,
    ,
    )
val audio_pattern = Configuration.get (ConfKey.
    audio_file_name_pattern)

//parse scala files, store KDE's in a hash map
5 var theoreticKDEs = Map[java.lang.String, KernelDensityEstimate]()
makams.foreach{ makam =>
    val scalaFile =
        + makam +
    val scalaObject = new ScalaFile(scalaFile);
    val kde = HistogramFactory.createPichClassKDE(scalaObject, 35)
10 kde.normalize
    theoreticKDEs = theoreticKDEs + (makam -> kde)

```

⁵A maqam defines rules for a composition or performance of classical Turkish music. It specifies melodic shapes and pitch intervals.

```

}

//print an sorted result (guess, correlation) for each file
15 Console.println(
Console.println(
    + theoreticKDEs.toList.map{ r =>
    }.mkString(
)

var index = 0;
makams.foreach{ makam =>
    val directory =
    makam
20 val audioFiles = FileUtils.glob(directory, audio_pattern, true).
    toList
    audioFiles.foreach{ file =>
        val audioFile = new AudioFile(file)
        val detectorYin = PitchDetectionMode.TARSOS_YIN.getPitchDetector
            (audioFile)
        val annotations = detectorYin.executePitchDetection()
25 //only large files
        if(annotations.size > 3000) {
            val actualKDE = HistogramFactory.createPichClassKDE(
                annotations, 15);
            val resultList = List[Tuple2[java.lang.String, Double]]()
            actualKDE.normalize
30 for ((name, theoreticKDE) <- theoreticKDEs){
                val shift = actualKDE.shiftForOptimalCorrelation(
                    theoreticKDE)
                val currentCorrelation = actualKDE.correlation(theoreticKDE,
                    shift)
                resultList = (name -> currentCorrelation) :: resultList
            }
35 //order by correlation
            resultList = resultList.sortBy{_. _2}.reverse
            index = index + 1
            //print the ordered result
            Console.println(index +
                + makam +
                + resultList.map{e
                    => e._1 +
                    + e._2}.mkString(
                ) +
            )
40 }
    }
}

```

With this script it is possible to correctly identify 39% of the makams using a data set of 800 files. Some makams look very much alike: if the first three guesses are evaluated the correct makam is present in 75% of the cases. This method is very general and directly applicable to e.g. harpsichord tuning estimation as done, using another approach, by Tidhar et al[9].

Find by example

Resynthesize files

Bibliography

- [1] Alain de Cheveigné and Kawahara Hideki. Yin, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, 111(4):1917–1930, 2002.
- [2] Ali C. Gedik and Barış Bozkurt. Pitch-frequency histogram-based music information retrieval for turkish music. *Signal Processing*, 90(4):1049–1063, 2010.
- [3] Aline Honingh and Rens Bod. In Search of Universal Properties of Musical Scales. *Journal of New Music Research*, 40(1):81–89, 2011.
- [4] Philip McLeod. *Fast, accurate pitch detection tools for music analysis*. PhD thesis, University of Otago. Department of Computer Science, 2009.
- [5] Phillip McLeod and Geoff Wyvill. A smarter way to find pitch. In *Proceedings of International Computer Music Conference, ICMC*, 2005.
- [6] Ronald L. Rivest. The MD5 Message-Digest Algorithm (RFC 1321). <http://www.ietf.org/rfc/rfc1321.txt?number=1321>.
- [7] William Sethares. *Tuning Timbre Spectrum Scale*. Springer, 2 edition, 2005.
- [8] Joren Six and Olmo Cornelis. Tarsos - a Platform to Explore Pitch Scales in Non-Western and Western Music. In *Proceedings of the 12th ISMIR Conference*, 2011.
- [9] Dan Tidhar, Matthias Mauch, and Simon Dixon. High precision frequency estimation for harpsichord tuning classification. In *Acoustics Speech and Signal Processing (ICASSP), 2010 IEEE International Conference on*, pages 61–64, march 2010.

List of Figures

2.1	How to detect a tone scale with Tarsos 1 use the file menu to open a file, 2 the file is converted and pitch is analysed automatically, 3 use the peak picking slider to automatically detect pitch classes in the pitch class histogram.	7
2.2	Example of a \LaTeX -export of a pitch class histogram	10

List of Tables

2.1	Example CSV output for annotations	9
2.2	Example of a \LaTeX -export of a pitch class interval matrix. P.C. stands for pitch class, the values are given in cents.	10

Appendix A

Pitch, Pitch Interval & Pitch Ratio Representation

Since different representations of pitch are used by Tarsos and other pitch extractors this section contains definitions of and remarks on different pitch and pitch interval representations. For good measure we need a definition of pitch, here the definition from [4] is used:

The pitch frequency is the frequency of a pure sine wave which has the same *perceived* sound as the sound of interest.

For remarks and examples of cases where the pitch frequency does not coincide with the fundamental frequency of the signal see [4].

A.1 Pitch & Pitch Interval Representation

Since we are interested in a frequency or frequency interval Hertz (Hz), oscillations per second, seems the most appropriate unit. When working with sound this is not always the case. For humans the perceptual distance between 220Hz and 440Hz is the same as between 440Hz and 880Hz. A pitch representation that takes this logarithmic relation into account is more practical for some purposes. Luckily there are a few:

MIDI Note Number

The MIDI standard defines note numbers from 0 to 127, inclusive. Normally only integers are used but any frequency f in Hz can be represented with a fractional note number n using equation A.1.

$$n = 69 + 12 \log_2\left(\frac{f}{440}\right) \tag{A.1}$$

$$n = 12 \times \log_2\left(\frac{f}{r}\right) ; r = \frac{440}{2^{(69/12)}} = 8.176\text{Hz} \tag{A.2}$$

Rewriting equation A.1 to A.2 shows that MIDI note number 0 corresponds with a reference frequency of 8.176Hz which is C_{-1} on a keyboard with A_4 tuned to 440Hz. It also shows that the MIDI standard divides the octave in 12 equal parts.

To convert a MIDI note number n to a frequency f in Hz one of the following equations can be used.

$$f = 440 \times 2^{(n-69)/12} \tag{A.3}$$

$$f = r \times 2^{(n/12)} \text{ with } r = 8.176\text{Hz} \tag{A.4}$$

Using pitch represented as fractional MIDI note numbers makes sense when working with MIDI instruments and MIDI data. Although the MIDI note numbering scheme seems oriented towards western pitch organization (12 semitones) it is conceptually equal to the cent unit which is more widely used in ethnomusicology.

Cent

Ellis introduced the nowadays widely accepted cent unit. To convert a frequency f in Hz to a cent value c relative to a reference frequency r also in Hz.

$$c = 1200 \times \log_2\left(\frac{f}{r}\right) \tag{A.5}$$

With the same reference frequency r equations A.5 and A.2 differ only by a constant factor of exactly 100. In an environment with pitch representations in MIDI note numbers and cent values it is practical to use the standardized reference frequency of 8.176Hz.

To convert a frequency f in Hz to a cent value c relative to a reference frequency r also in Hz.

$$f = r \times 2^{(c/1200)} \tag{A.6}$$

Savart & Millioctaves

Divide the octave in 301.5 and 1000 parts respectively, which is the only difference with cents.

A.2 Pitch Ratio Representation

Pitch ratios are essentially pitch intervals, an interval of one octave, 1200 cents equal to a frequency ratio of 2/1. To convert a ratio t to a value in cent c :

$$c = \frac{1200 \ln(t)}{\ln(2)} \tag{A.7}$$

The natural logarithm, the logarithm base e with e being Euler's number, is noted as \ln . To convert a value in cent c to a ratio t :

$$t = e^{\frac{c \ln(2)}{1200}} \tag{A.8}$$

Further discussion on cents as pitch ratios can be found in appendix B of [7]. There it is noted that:

There are two reasons to prefer cents to ratios: Where cents are added, ratios are multiplied; and it is always obvious which of two intervals is larger when both are expressed in cents. For instance, an interval of a just fifth, followed by a just third is $(3/2)(5/4) = 15/8$, a just seventh. In cents, this is $702 + 386 = 1088$. Is this larger or smaller than the Pythagorean seventh $243/128$? Knowing that the latter is 1110 cents makes the comparison obvious.

A.3 Conclusion

The cent unit is mostly used for pitch interval representation while the MIDI key and Hz units are used mainly to represent absolute pitch. The main difference between cent and fractional MIDI note numbers is the standardized reference frequency. In our software platform Tarsos we use the exact same standardized reference frequency of 8.176Hz which enables us to use cents to represent absolute pitch and it makes conversion to MIDI note numbers trivial. Tarsos also uses cents to represent pitch intervals and ratios.

Appendix B

Maqams