MIDI.jl: Simple and intuitive handling of MIDI data.

George Datseris\textsuperscript{1, 2} and Joel Hobson\textsuperscript{3}

\textsuperscript{1}Max Planck Institute for Dynamics and Self-Organization \textsuperscript{2}Department of Physics, Georg-August-Universität Göttingen \textsuperscript{3}Roadmunk Inc.

Introduction

MIDI.jl is a Julia Bezanson, Edelman, Karpinski, & Shah (2017) package for reading, writing and analyzing MIDI data. In this paper, we are briefly overviewing versions 1.1.0 or later for MIDI.jl.

MIDI (Music Instrument Digital Interface) is a data format created to transmit music data across devices and computers. The actual MIDI interface is low-level, directly translating all music information to and from byte chunks. MIDI.jl exposes all this low-level interface, but it also builds a useable high-level interface on top of that. This makes reading MIDI data intuitive and convenient, as we demonstrate in the following examples.

All functionality of MIDI.jl is well documented and hosted online: https://juliamusic.github.io/JuliaMusic_documentation.jl/latest/. Besides documentation of functions there are plenty of useful real world examples.

Intuitive and Simple Interface

The biggest strength of MIDI.jl is the ability to transform the raw MIDI data in a format that is human-readable, intuitive and simple to use and manipulate. In addition, the high-level interface does not require knowledge of which exact MIDI code corresponds to which exact MIDI command.

What makes this possible is the data structures we have designed in order to provide easier handling of MIDI files. The most important data structure is the Note/Notes. A music note can be (in its most basic level) deconstructed into just four numbers: the temporal position the note is played in, the duration, the pitch, and the intensity (strength with which the note is played, also called velocity). A Note is a data structure that has these four “quantities” as its “fields”. All of these are accessible immediately with e.g., Note.position and their values can be mutated in place.

These aspects can be deduced from the raw MIDI format with a lot of analyzing of bytes. However, in MIDI.jl we provide a simple function:

\begin{verbatim}
getnotes(midi, args...)
\end{verbatim}

which obtains all note-specific information and stores it as a vector of notes, which we call Notes. This is very convenient, as to “identify” a note in the MIDI format, one needs to first identify two different streams of bytes; one denotes the start and the other the end of the note. This quickly becomes tedious, but getnotes does not expose all these details to the user.
Extensions

The easy-to-use high-level interface allows MIDI.jl to be extendable. For instance, in another software package MusicManipulations.jl we provide general functions for manipulating (and further analyzing) music data. For example, the function quantize from the package MusicManipulations.jl allows the user to quantize any Notes instance to any grid.

This functionality is offered by Digital Audio Workstations, like the software Cubase, but we offer ways to do it programmatically instead. Many other helpful functions are contained in MusicManipulations.jl, and for further reading we point to the official documentation of the JuliaMusic GitHub organization, which hosts both MIDI.jl and MusicManipulations.jl, as well as other useful packages.

Scientific Application

Microtiming deviations are defined as temporal deviations below the phrase level, typically in the millisecond range. These have been studied extensively in the literature and their importance and influence are debated strongly, see Madison, Gouyon, Ullén, & Hörnström (2011), Butterfield (2010), Frihauf, Kopiez, & Platz (2013), Davies, Madison, Silva, & Gouyon (2013), Senn, Kilchenmann, Von Georgi, & Bullerjahn (2016), Hofmann, Wesolowski, & Goebel (2017) and references therein.

Qualitative studies of these microtiming deviations have been done extensively by Geisel and coworkers Holger Hennig et al. (2011), H. Hennig (2014), Rüsänen, Pulkkinen, Virtanen, Zollner, & Hennig (2015), Sogorski, Geisel, & Priesemann (2018). A crucial finding is that the sequence of such deviations is not random but power-law correlated. In addition, there is strong evidence that their distribution is normal (Gaussian).

In the following, we will compute the distribution of the microtiming deviations of a piano track (played by a professional pianist) and show that indeed it approximates a normal distribution.

We first load the notes of the piano track:

```julia
using MusicManipulations # Re-exports MIDI
midi = readMIDIFile(testmidi()) # test midi file

# Track number 4 is the piano track
piano = midi.tracks[4]
notes = getnotes(piano, midi.tpq)
```

533 Notes with tpq=960

<table>
<thead>
<tr>
<th>Note</th>
<th>vel</th>
<th>pos</th>
<th>dur</th>
</tr>
</thead>
<tbody>
<tr>
<td>F4</td>
<td>69</td>
<td>7427</td>
<td>181</td>
</tr>
<tr>
<td>A4</td>
<td>85</td>
<td>7760</td>
<td>450</td>
</tr>
<tr>
<td>D5</td>
<td>91</td>
<td>8319</td>
<td>356</td>
</tr>
<tr>
<td>D4</td>
<td>88</td>
<td>8323</td>
<td>314</td>
</tr>
<tr>
<td>G3</td>
<td>88</td>
<td>8327</td>
<td>358</td>
</tr>
<tr>
<td>A4</td>
<td>76</td>
<td>8694</td>
<td>575</td>
</tr>
<tr>
<td>C4</td>
<td>66</td>
<td>9281</td>
<td>273</td>
</tr>
<tr>
<td>A4</td>
<td>94</td>
<td>9594</td>
<td>666</td>
</tr>
<tr>
<td>F3</td>
<td>98</td>
<td>10189</td>
<td>307</td>
</tr>
<tr>
<td>C4</td>
<td>87</td>
<td>10206</td>
<td>285</td>
</tr>
</tbody>
</table>

We then compute their microtiming deviations. For the purpose of this article, we define the microtiming deviations of a note as the distance of the position of a note from its position when quantized on a 8th-note triplet grid (the pianist was playing triplets in the above midi file).

We now compute those microtiming deviations, using the function `quantize` from MusicManipulations.jl.

```julia
grid = 0:(1//3):1 # grid to quantize on, see documentation
qnotes = quantize(notes, grid)
mtds = positions(notes) .- positions(qnotes)
mtds_ms = mtds .* ms_per_tick(midi)
```

533-element Array{Float64,1}:

32.21150624999999
38.46149999999999
12.49999874999999
13.46152499999999

A plot of the histogram of these is presented in Figure 1. Even if produced with an extremely small pool of data, the plot follows the existing evidence that the distribution of the microtiming deviations follows a normal distribution.
Conclusions

In conclusion, MIDI.jl is a useful package with intuitive usage, as we have demonstrated by our simple application. In addition, it has plenty more use for scientific applications. We currently have a manuscript titled Does it Swing? Microtiming Deviations and Swing Feeling in Jazz under review, where we have used MIDI.jl and its extensions to not only read but also manipulate microtiming deviations of human recordings in order to inquire about the impact of microtiming deviations in the listening experience.

Related Software

There is existing software that offers functionality similar, but not identical, to MIDI.jl. Some of these software are:

- mido
- python-midi
- pretty-midi (Raffel & Ellis, 2014).

Notable differences between MIDI.jl and these libraries include (but are not limited to):

1. The Notes data structure and getnotes functionality that exists in MIDI.jl. Although the package pretty-midi contains similar functionality, it lacks the channel property.
2. MIDI.jl is extended further into higher level applications like the ones offered by MusicManipulations.jl or the module MotifSequenceGenerator that can create specially random sequences of notes.
3. The fact that MIDI.jl is written for the Julia programming language.
4. The fact that MIDI.jl is the only package for handling MIDI data for the Julia programming language.
5. MIDI.jl has proper, multi-page and multi-example documentation that is hosted online and is automatically updated with every commit to the repository.
6. MIDI.jl does not currently have the so-called “piano roll” functionality, which plots notes as in a Digital Audio Workstation.
7. Sequencer functionality, which has been implemented in the python-midi package, is currently lacking in MIDI.jl.
8. Basic tempo estimation, which has been implemented in pretty-midi, is also absent in MIDI.jl.

References


