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

George Datseris[1, 2] **and Joel Hobson**[3]

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

DOI: 10.21105/joss.01166

Software

- Review
- Repository
- Archive

Submitted: 29 December 2018
Published: 14 March 2019

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:

```
getnotes(midi, args...)
```

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), Frühauf, Kopiez, & Platz (2013), Davies, Madison, Silva, & Gouyon (2013), Senn, Kilchenmann, Von Georgi, & Bullerjahn (2016), Hofmann, Wesolowski, & Goebl (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), Räsänen, 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
 Note F4  | vel = 69  | pos = 7427, dur = 181
 Note A♯4 | vel = 85  | pos = 7760, dur = 450
 Note D5  | vel = 91  | pos = 8319, dur = 356
 Note D4  | vel = 88  | pos = 8323, dur = 314
 Note G♯3 | vel = 88  | pos = 8327, dur = 358
 Note A♯4 | vel = 76  | pos = 8694, dur = 575
 Note G4  | vel = 66  | pos = 9281, dur = 273
 Note A♯4 | vel = 94  | pos = 9594, dur = 666
 Note F♯3 | vel = 98  | pos = 10189, dur = 307
 Note C4  | vel = 87  | pos = 10206, dur = 285
```

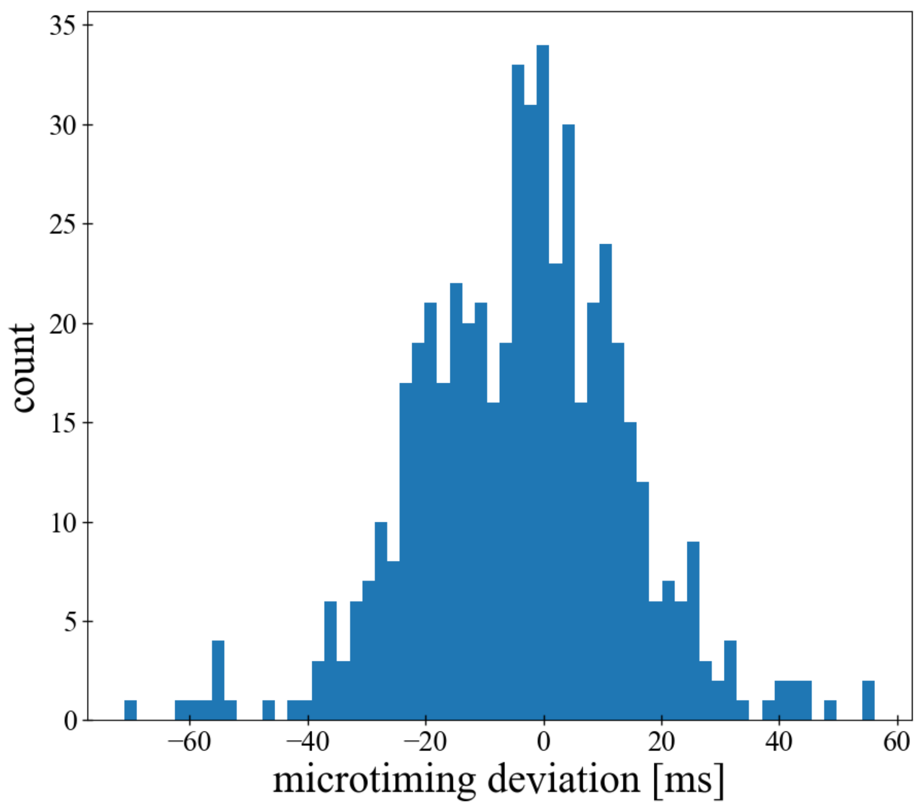

Figure 1: Histogram of the microtiming deviations of a simple piano recording.

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**

```
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
  38.461499999999994

  12.499987499999998
  13.461524999999998
```

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

Bezanson, J., Edelman, A., Karpinski, S., & Shah, V. B. (2017). Julia: A fresh approach to numerical computing. *SIAM Review*, *59*(1), 65–98. doi:10.1137/141000671

Butterfield, M. (2010). Participatory discrepancies and the perception of beats in jazz. *Music Perception: An Interdisciplinary Journal*, *27*(3), 157–176. doi:10.1525/mp.2010.27.3.157

Davies, M., Madison, G., Silva, P., & Gouyon, F. (2013). The effect of microtiming deviations on the perception of groove in short rhythms. *Music Perception: An Interdisciplinary Journal*, *30*(5), 497–510. doi:10.1525/mp.2013.30.5.497

Frühauf, J., Kopiez, R., & Platz, F. (2013). Music on the timing grid: The influence of microtiming on the perceived groove quality of a simple drum pattern performance. *Musicae Scientiae*, *17*(2), 246–260. doi:10.1177/1029864913486793

Hennig, H. (2014). Synchronization in human musical rhythms and mutually interacting complex systems. *Proceedings of the National Academy of Sciences*, *111*(36), 12974–12979. doi:10.1073/pnas.1324142111

Hennig, H., Fleischmann, R., Fredebohm, A., Hagmayer, Y., Nagler, J., Witt, A., Theis, F. J., et al. (2011). The nature and perception of fluctuations in human musical rhythms. *PLoS One*, *6*(10), e26457. doi:10.1371/journal.pone.0026457

Hofmann, A., Wesolowski, B. C., & Goebl, W. (2017). The tight-interlocked rhythm section: Production and perception of synchronisation in jazz trio performance. *Journal of new music research*, *46*(4), 329–341. doi:10.1080/09298215.2017.1355394

Madison, G., Gouyon, F., Ullén, F., & Hörnström, K. (2011). Modeling the tendency for music to induce movement in humans: First correlations with low-level audio descriptors across music genres. *J. Exp. Psychol. Hum. Percept. Perform.*, *37*(5), 1578–1594. doi:10.1037/a0024323

Raffel, C., & Ellis, D. P. (2014). Intuitive analysis, creation and manipulation of MIDI data with pretty_midi. In *15th international society for music information retrieval conference late breaking and demo papers* (pp. 84–93).

Räsänen, E., Pulkkinen, O., Virtanen, T., Zollner, M., & Hennig, H. (2015). Fluctuations of hi-hat timing and dynamics in a virtuoso drum track of a popular music recording. *PLoS One*, *10*(6), e0127902. doi:10.1371/journal.pone.0127902

Senn, O., Kilchenmann, L., Von Georgi, R., & Bullerjahn, C. (2016). The effect of expert performance microtiming on listeners' experience of groove in swing or funk music. *Front. Psychol.*, *7*, 1487. doi:10.3389/fpsyg.2016.01487

Sogorski, M., Geisel, T., & Priesemann, V. (2018). Correlated microtiming deviations in jazz and rock music. (E. Hernandez-Lemus, Ed.)*PLoS One*, *13*(1), e0186361. doi:10.1371/journal.pone.0186361