MUSICAL MICRO-TIMING FOR LIVE CODING

Max Johnson[1], **Mark Gotham**[1,2]

[1] Department of Computer Science and Technology, University of Cambridge
[2] Department of Computer Science, Durham University

ABSTRACT

Micro-timing is an essential part of human music-making, yet it is absent from most computer music systems. Partly to address this gap, we present a novel system for generating music with style-specific micro-timing within the Sonic Pi live coding language. We use a probabilistic approach to control the exact timing according to patterns discovered in new analyses of existing micro-timing data (jembe drumming and Viennese waltz). This implementation also required the introduction of musical metre into Sonic Pi. The new metre and micro-timing systems are inherently flexible, and thus open to a wide range of creative possibilities including (but not limited to): creating new micro-timing profiles for additional styles; expanded definitions of metre; and the free mixing of one micro-timing style with the musical content of another. The code is freely available as a Sonic Pi plug-in and released open source at `https://github.com/MaxTheComputerer/sonicpi-metre`.

1. INTRODUCTION

1.1 Metre *Versus* Rhythm

Metre is distinct from rhythm in that it primarily concerns a kind of mental representation for processing events in musical time; a common analogy casts metre as a "grid" or "template" for categorising rhythmic events [1–3].

Although metre often involves familiar notions such as "the beat", and definitions often emphasise intuitive ideas like regular periodicity, a clear-cut definition of metre is surprisingly hard to pin down. This is especially so when trying to capture the extremely wide range of musical-cultural contexts for which some concept of metre might be relevant. Nevertheless, notwithstanding the complexities of these terms, and putting any more specific definition of these terms to one side, it is reasonable to argue that some form of both "rhythm" and "metre" feature in almost all known musics: "rhythm" in the sense of events occurring in time, and "metre" in some form of semi-regular cycle of event expectation.

1.2 On Micro-Timing in Theory and Practice

"Micro-timing", in turn, refers to the specific timing of both those actual rhythmic "events" and of the metrical "grid" positions. While rhythm and metre are sometimes *modelled* in terms of a completely regular underlying pulse (e.g., 1-1-1-1-) and small integer combinations thereof (e.g., 2-1-1-), it is impossible in practice for a human performer to play with the mechanical precision of identical gaps between successive events. [1] Moreover, musicians make a virtue of this. A close look at the micro-timings in human performance reveals deeply sophisticated, style-specific micro-timing strategies, a.k.a. "groove". [2]

Even when distinguishing between a mental "grid" for events (metre) and the actual placement of those events (rhythm), it is appropriate to discuss micro-timing for both the rhythm and the metre. This distinction is sometimes cast in terms of a difference between "categorical" and "expressive" timing where events may be expressively altered from their expected (categorical) position [5], but note that the "categorical" position itself is also subject to micro-timing strategies because the "expected" positions are not spaced with equal, 1:1 regularity. In short, although micro-timing is sometimes described in terms of "small deviations" from simple (natural number) pulse relations, it is necessary also to consider micro-timing as part of the metre itself. To continue the "metre as grid" analogy: the gaps between grid lines are not evenly spaced.

In practice, these micro-timing durations are too short to learn in a declarative fashion (verbal or mathematical). Partly for that reason, they are also typically neglected by notation systems (including Western staff notation). Nonetheless, these micro-timing strategies clearly *are* taught and learned in the way that most music has been passed on: through listening, playing, and embodiment.

Computers enable us to achieve a level of timing regularity beyond our human capability. As ever, the technology not only extends what we can do, but invites us to consider new techniques, questions, and aesthetics. And the human's micro-timing can of course be combined with the computer's extreme timing precision, as when an MC raps over a beat. Computers are also used to perform the micro-timing *analysis* discussed here. However, computers are currently less exploited as a tool to help us engage in *creative uses* of micro-timing strategies.

[1] This has been systematically studied since Seashore [4].
[2] See, for example, Justin London's "many metres" [2].

2. RELATED WORK

A substantial research field has grown to analyse the micro-timing strategies in different musical styles. This has included work on the Viennese waltz (from Bengtsson and Gabrielsson's landmark 1977 work to Yang's 2022 analysis of 'The Blue Danube' [6, 7]), jembe music from Mali (notably through Rainer Polak's career-long focus on this repertoire, [8–10]), and a recent surge of work on Afro-Cuban and Latin musics (see, for instance, [11, 12]).

Creating music with synthetic micro-timing has also been explored as part of the broad field of MIR, but has concentrated on attempting to model human-like expressive timing [13]. For example, Flossman et al. use probabilistic models for expressive performance rendering [14, 15]. There has been very little academic work on integrating style-specific timing analyses into computational settings. The closest examples are in the commercial sphere: Ableton Live, for instance, features "grooves" which shift MIDI events from quantised positions according to micro-timing styles, including a probabilistic element and the option to create new "grooves" from any human performances (via MIDI).

Musical live coding is a way of creating and performing music by writing and modifying code in real time. While there may or may not be pre-made materials, live manipulation of the material is a given. Given the inherent "liveness" of live coding, [3] it is arguably an ideal part of the computer music pantheon to integrate human micro-timing. Yet most live coding languages lack not only micro-timing functionality, but even a full representation of musical metre, typically encoding only events in time, or at most an anaemic representation for beats and/or time signatures. For example, the commercial Max/MSP language uses its `transport` object to allow access to bar and beat numbers for the current time signature. An exception is McLean's open-source Tidal Cycles which uses a cyclic notion of time that can be subdivided to achieve more complex hierarchies [17].

In summary, although there has been much research into the analysis of micro-timing in different musical styles, and the application of expressive timing to computer-generated music, we still lack implementations of style-specific micro-timing in most computer-music software, and even foundational notions of musical metre in most live coding environments. This project seeks to address those issues through an implementation of both metre and style-specific micro-timing for Sonic Pi: a popular live coding language and IDE designed to support a range of creative possibilities while being simple enough to use as an educational tool for use in schools [18]. [4] We aim to improve not only how "life-like" the generated music sounds in general (and thus, arguably the "liveness" of that live coding), but also to do this in a style-specific way. In this paper we report on implementation of two case studies as well as a general framework for integrating further styles.

[3] This is discussed in Chapter 5 of [16], for instance.

[4] Sonic Pi's domain-specific language is written in Ruby and uses the SuperCollider sound synthesis server to produce sounds [19].

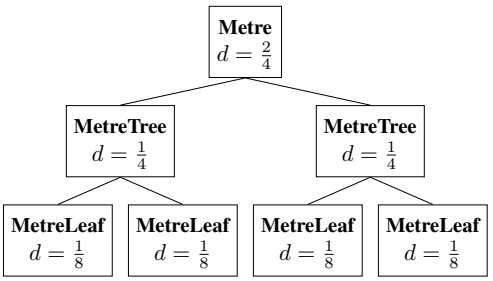

Figure 1: An example of how MetreTree and MetreLeaf objects are nested to construct a metrical hierarchy for $\frac{2}{4}$. The total duration d of each node is also displayed, and the duration of a parent node is the sum of the durations of its children [22, 23].

3. IMPLEMENTING METRE

This section describes the model of metre we have implemented for Sonic Pi. We argue that this is useful for a range of applications including (but not limited to) use as a basis for micro-timing as described below (§4). As discussed, metre is a very widespread phenomenon in general but specific aspects differ. We can broadly distinguish here between the specifically *hierarchical aspects* (important for some styles but not all) and the more general notion of *categorical positions* in a metrical cycle (much more widespread, and axiomatic for the kind of micro-timing systems discussed and implemented here).

3.1 Modelling Metrical Hierarchy with Trees

A favoured method for encoding metrical hierarchy in a data structure is through trees. See Forth [20] for a detailed mathematical treatment of trees used in this context and the `music21` Python library [21] for a popular implemenation. Our approach shares some high-level ideas with `music21` (and indeed Forth, and others), but differs in the specific implementation.

The tree structure is implemented by the `MetreLeaf` and `MetreTree` classes. Figure 1 shows the default tree structure formed by these objects and their durations for a Western $\frac{2}{4}$ time signature. Note how the duration of a parent node is the sum of the durations of its children.

To model tree data structures of any depth with a succinct, finite representation, we follow the Western notational assumption of diving each MetreLeaf into two equal parts to get the next level where not specified otherwise. [5] Users can specify the full depth of a tree as necessary against this assumption.

3.2 The MetreLeaf Class

A MetreLeaf object is the leaf node of the metrical tree structure. It has an instance variable `fraction` which represents the duration of the MetreLeaf as a fraction of a whole note. For example, a leaf node with the duration of one quarter note will have the value $\frac{1}{4}$.

[5] See [24] for discussion of this point, of metrical "well-formedness", and the notion of "binary", "ternary" and wider metrical structures.

The class contains a `subdivide()` method, which divides the MetreLeaf by two a given number of times, s. It returns a new MetreTree with 2^s MetreLeaf children, each of value $f/2^s$ where f is the fraction of the original MetreLeaf.

3.3 The MetreTree Class

A MetreTree object represents the hierarchical tree or subtree of a metre. The instance variable `sequence` is an ordered list representing this node's children and contains any combination of MetreLeaf objects and other MetreTree objects. For example, the hierarchy in Figure 1 could also be written in list form as:

$$\left[\left[\frac{1}{8}, \frac{1}{8}\right], \left[\frac{1}{8}, \frac{1}{8}\right]\right]$$

Each list is a MetreTree, and each fraction is a MetreLeaf. The MetreTree class contains several methods for manipulating and extracting information from the metrical hierarchy it represents. The two most important of these are explained in more detail below.

3.3.1 Getting Metrical Levels

Metrical level refers to the depth level of a metrical hierarchy. Here, we base our representation on the *beat level*, which is divided to get *division levels*, and grouped to get *grouping levels*.[6] The `get_level()` method allows a user to "flatten" the tree structure to a specified depth, accessing the sequence of events at a given metrical level.

For flattening to a division level ($l > 0$) or to the beat level ($l = 0$), we perform a recursive depth-first search on the tree. For each child in the sequence list, if it is a MetreTree, the method is recursively called until the base case of $l = 0$ is reached. At this point, all the children of that node are combined into one MetreLeaf equal to the sum of their durations. If the child is instead a MetreLeaf, it is subdivided l times to reach the desired metrical level.

For a grouping level ($l > 0$), we find an estimate of the structure of higher metrical levels by clustering nodes together. It is an estimate because this information is not in the MetreTree's representation of the metre, so is just one possibility for the higher structure. The algorithm recursively clusters nodes until the desired metrical level l is reached. The number of nodes combined in each cluster is determined by the smallest prime factor of the number of nodes at the level below. For example, if level $l + 1$ has four nodes, they will be clustered in groups of two. If it has nine nodes, they will be clustered in groups of three.

Some examples of the output of the flattened tree for the following complex hierarchy are shown in Table 1:

$$\left[\left[\frac{1}{8}, \frac{1}{8}\right], \left[\frac{1}{16}, \frac{3}{16}\right], \frac{1}{8}, \left[\frac{1}{4}, \left[\frac{5}{16}, \frac{3}{16}\right]\right]\right]$$

[6] Centring the beat level in this way reflects the psychology of metre better than alternative "top down" and "bottom up" approaches.

l	`get_level`(l)
-2	$\left[\frac{11}{8}\right]$
-1	$\left[\frac{1}{2}, \frac{7}{8}\right]$
0	$\left[\frac{1}{4}, \frac{1}{4}, \frac{1}{8}, \frac{3}{4}\right]$
1	$\left[\frac{1}{8}, \frac{1}{8}, \frac{1}{16}, \frac{3}{16}, \frac{1}{16}, \frac{1}{16}, \frac{1}{4}, \frac{1}{2}\right]$

Table 1: Examples of the output of `get_level`(l) at different metrical levels l for an example hierarchy. Note how level $l = -1$ is formed by the clustering of level $l = 0$.

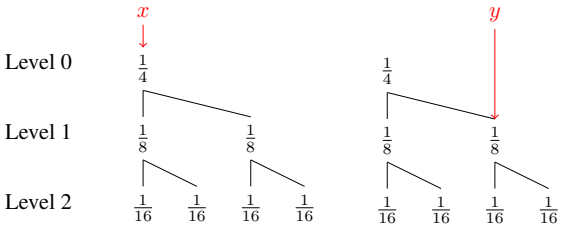

Figure 2: An example metrical hierarchy for $\frac{2}{4}$ showing those metrical events at each level which coincide with offsets x and y.

3.3.2 Getting Exact Metrical Events

We define an *offset* as a position in the metric cycle represented by the quarter length duration to have elapsed since the beginning of the cycle.[7] The `metrical_level_indices()` method of the MetreTree class finds any metrical events occurring at a given offset, and returns their index.

Consider the example shown in Figure 2. Offset x occurs on the first event of all three levels, so the function would return $L_0(x) = L_1(x) = L_2(x) = 0$, where $L_l(x)$ is the index of an event at level l that offset x occurs on. Offset y occurs only on the last event of Level 1 and the second-to-last event of Level 2, so the function would return $L_1(y) = 3, L_2(y) = 6$.

This method is important because it is used later to determine the "categorical" position to link an event to and, through that, which micro-timing probability distribution to apply.

3.4 Bar Class

The Bar class is a representation of a single metrical cycle,[8] and each instance of it has an associated metre. A Bar object is responsible for:

[7] "Quarter length" is a semi-standard measurement for a length of time in symbolic values where the unit length is one "quarter note" duration (UK: "crotchet").

[8] A more precise definition would account for *hypermetre* where bars occur at the beat level [25], but the simple definition is sufficient for our purposes. Note that "bar" is the UK English equivalent of "measure" (USA).

```
play :C4
sleep(1)
play :E4
sleep(1)
play :G4
sleep(0.5)
play :E4
sleep(0.5)
play :C4
```

(a) Old (above left)

```
use_metre '4/4'

bar do
    add_note :C4, 0, 1
    add_note :E4, 0, 1
    add_note :G4, 1, 1
    add_note :E4, 1, 1
    add_note :C4, 0, 1
end
```

(b) New (above right)

(c) Western musical notation:

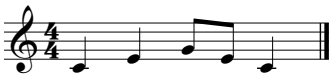

Figure 3: A bar of music represented by (a) the original Sonic Pi syntax, (b) our new metre commands, and (c) traditional Western music notation. Note how the original Sonic Pi syntax loses information about the metre. The second and third arguments to `add_note` are the metrical level and duration (l and d in §3.4).

- Keeping track of playback position during the cycle.
- Converting a note length given as a metrical level and a duration into a quarter length.
- Checking if a note or rest fits in the remaining time in the cycle, and updating the bar's playback position accordingly.

A note's length is specified by a metrical level and a duration, where the duration is in the units of an event at the specified metrical level and acts as a multiplier. For example, if a note's length is defined as $(l, d) = (0, 3)$, its unit length is the duration of an event at level $l = 0$, and it lasts for $d = 3$ of these units.

The `add_note()` method checks if a note fits into the bar's remaining time; if it cannot, an exception is raised. This ensures the total ("actual") duration of the bar matches its metre's ("nominal") duration.

3.5 Playing Music

This framework for musical metre enabled the creation of new Sonic Pi commands. Figure 3 shows a comparison between the original Sonic Pi commands, our alternative commands, and traditional Western music notation.

There are two main commands for metre. The first is `use_metre(`m`)`, which changes the current thread's metre to m (using a thread-local variable). The second is `bar do ... end`, which creates a new Bar object, stores this to a thread-local variable, then executes a block of user code.

A user can use `add_note` to play a note on the current synthesiser. This works by first getting the current Bar object from the thread-local variables and calling the Bar's `add_note()` method to check if the note will fit in the bar. It then passes the note pitch to Sonic Pi's `play` function which creates the sound, and finally applies `sleep` for the remaining duration of the note.

4. MICRO-TIMING

4.1 Storing Micro-Timing Information

In order to add micro-timing functionality to our implementation, we first needed a way of representing and storing the micro-timing information for different musical styles. We implement this by storing each event in the metrical cycle, the theoretical (isochronous) position of that event in the cycle (e.g., 1), and the typical displacement of the from this position (the μ of the micro-timing, e.g., +0.004). Actual event occurrence is modelled by normal probability distribution around these μ values.

Samples can then be drawn from these distributions using the Box-Muller transform [26] on uniform random samples from Sonic Pi's random number generator. Sonic Pi's generator produces a deterministic, repeatable sequence of pseudorandom numbers, which means the output of a Sonic Pi program sounds the same each time it is run [27].

4.2 Applying Micro-Timing

When a user sets a metre with the `use_metre` command, they can optionally specify a style as well. This causes all music played with that metre to use the micro-timing of the chosen style.

At the start of each new bar, the `Metre` object samples new values from the `Style`'s probability distributions. When a note is played inside the bar, the `add_note` command requests the timing shift that should be applied to the note from the Metre. To calculate this, the Metre object calls its `metrical_level_indices()` method to determine which timing values from each level to use. The individual timing contributions of each metrical level are summed to produce an overall timing shift for the note. A positive value means the note should be played slightly late; a negative value means slightly early. This is returned to `add_note` which then uses Sonic Pi's `time_warp` function to adjust the timing of the call to `play`.

For example, if the sampled timings, T_l, for each level, l, are:

$$T_0 = [0, 0.1]$$
$$T_1 = [0.03, 0, 0, -0.02]$$

and the metrical level indices, L_l, for each level, l, are:

$$L_0 = 1$$
$$L_1 = 3$$

then the timing shift, t, would be calculated by:

$$\begin{aligned} t &= \sum_{i \in T.\text{keys}} T_i[L_i] \\ &= T_0[L_0] + T_1[L_1] \\ &= 0.1 + (-0.02) \\ &= 0.08 \end{aligned}$$

Therefore, the note will be played 0.08 quarter lengths after the reference value.

5. CASE STUDIES

Creating music with realistic micro-timing using the implementation we have described requires a set of probability distributions which accurately characterise a style of music. Clearly this is best implemented with data derived from real-life examples of the musical style in question. This project uses two different styles of music as contrasting case studies for evaluation: jembe (or "djembe") drum music from Mali and Viennese waltz music. These two styles both have robust, well-known micro-timing characteristics. The distributions derived here form the "preset" styles included in our Sonic Pi plugin.

5.1 Jembe Data Analysis

Jembe is a style of West African music involving a small ensemble of drummers (typically 3–4). It provides an ideal case study for our purposes because it has a highly consistent micro-timing strategy [8]. Malian drummers have been shown to exhibit some of the most consistent timing (lowest levels of variability) between performers in the world [28].

Moreover, jembe music is relatively constrained in terms of its pitch, timbre, and number of instruments. This also helps by enabling a clear focus on timing. Extensive research into the micro-timing of jembe music has included the release of high-quality datasets of processed live recordings [8–10].

The first dataset is from Jacoby et al. [10] and consists of 11 processed recordings of a piece called 'Suku', which is a very commonly played piece in this style. The second dataset is from the "Interpersonal Entrainment in Music Performance" (IEMP) Data Collection [29, 30]. This consists of 15 recordings across three different pieces: 'Manjanin', 'Maraka', and 'Woloso'. Both datasets here use recordings made by Rainer Polak in Mali. The datasets supply the following information:

- Onset of the drum stroke in seconds since the start;
- Phase: beats since the start of the *current cycle*;
- Cycle (bar) number: a natural number count;
- Categorical metrical position within the cycle associated with this event (integer, 0–11).

5.1.1 Micro-Timing Estimation

The pieces of jembe music in the dataset use a metre with four beats, [9] each of which divides into three, for a total of 12 metrical events at the first division level (similar to $\frac{12}{8}$ in Western classical notation). It is at this level, referred to as the "pulse", that the main micro-timing occurs.

Recall that the probability distributions described in §4.1 store the displacement of each event. We calculate this from the phase given by the datasets with the following equation:

$$\text{displacement} = \frac{(\text{phase} \times \text{beat division}) - \text{metric position}}{2}$$

[9] See Polak [8] for an ethnographically sensitive discussion of the extent to which metre applies in this context.

The phase is multiplied by the beat division (in this case, 3) to convert it into pulse units. The metric position at the pulse level is subtracted to get the displacement. The final division by 2 converts the displacement into quarter lengths (because each pulse unit is an eighth length).

For example, if an onset has metric position $= 6$ and phase $= 2.01$, the displacement would be calculated by:

$$\text{displacement} = \frac{(2.01 \times 3) - 6}{2} = 0.015 \text{ quarter lengths.}$$

Once the displacement has been calculated for each drum stroke, we were then able to estimate the distribution of displacements for each of the twelve metric locations using maximum likelihood estimation (MLE).

5.1.2 Tempo Estimation

Generating a synthetic piece of jembe music requires analysis of other musical features as well as the micro-timing to sound realistic. One of these is the *tempo*, which in jembe pieces of music typically increases substantially over the duration of the performance [10], with the last 15 seconds or so showing the tempo increasing at a much faster rate.

The *inter-beat interval* (IBI) is defined as the time between two consecutive beats in a piece of music, from which the instantaneous tempo can be calculated [31]. A moving average can be applied to the instantaneous tempo to obtain an estimate of the global tempo.

For the jembe data, we first filtered all the onsets to include just those played by Jembe 2 (because it plays on every beat as discussed in [10]), then filtered these to only consider onsets on the beats. We then calculated the inter-beat interval in bpm and applied a moving average with window size 10 to smooth the tempo estimate.

Inspection of the smoothed tempo graphs (§6.2) showed a logarithmic trend for the first ~95% of the piece. A sharper increase follows this which was modelled by a quadratic curve. To fit curves to the data, we used the `optimize.curve_fit` function from the SciPy Python library, which uses a non-linear least squares method [32]. The parameters estimated by the curve fitting are then used in Sonic Pi to control the tempo of a synthetic jembe piece during playback.

5.2 Waltz Data Analysis

The Viennese waltz is a style of fast waltz notated in $\frac{3}{4}$ (but often counted in 1), originally intended for ballroom dancing, and now often performed in concerts by Western classical orchestras.

The Viennese waltz provides a useful comparison to Malian jembe in evaluating this project's micro-timing implementation. The fast three beats ($\frac{3}{4}$) and typical hypermetrical grouping in 2s and 4s make that *metrical structure* somewhat similar to that of jembe music, but with a very different micro-timing profile. Distinctive micro-timing can be observed on (at least) the beat level, where it has a characteristic short-long-medium pattern [6, 33].

At the time this work was carried out, the micro-timing in Viennese waltz had not been studied in as much detail or

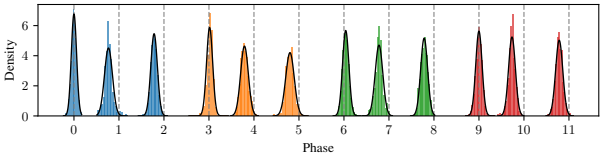

Figure 4: A histogram plot of the positions of each pulse within the cycle for Suku. Dashed lines show the isochronous division of the cycle for reference. Black curves show the PDF of the MLE-derived probability distributions and the colours distinguish to the four beat.

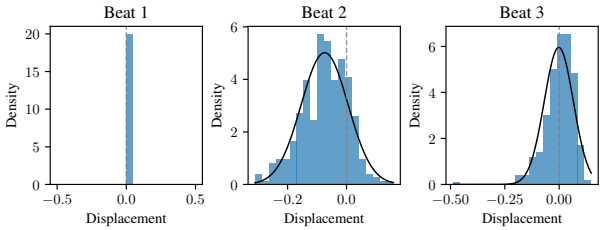

Figure 5: A histogram plot of the displacement of the second and third beats for the waltz dataset. Dashed lines show the metrical grid. Black curves show the PDF of the MLE-derived probability distributions.

as recently as jembe and there were no existing datasets of Viennese waltz performances with micro-timing. Therefore, we constructed a new dataset comprising of 30-second samples from seven waltz recordings performed by the Vienna Philharmonic Orchestra, all of which have noticeable and statistically significant micro-timing.

We then performed automatic beat tracking on this dataset using the libfmp Python library [34] (a dynamic programming approach introduced by Müller [35]), with some small manual corrections. Since the beat level is where the primary micro-timing in the Viennese waltz occurs, no additional onset detection was necessary.

Calculating the micro-timing displacement of each beat from onset times alone involves first identifying the start and end of each cycle, estimating the onset of each beat as if they were isochronous, then finding the difference between this and the actual onset to get the displacement. Once the displacements were derived, maximum likelihood estimation was again used to fit the probability distributions as discussed above for the jembe case.

6. RESULTS

6.1 Micro-Timing Estimation

Figure 4 shows the results of the micro-timing estimation for one of the jembe pieces in the datasets: "Suku". The histograms show the positions within the cycle where the 12 pulses occurred (phase). The dashed lines indicate where the event would occur if they were isochronous: from this the existence of the micro-timing can be seen clearly by the positions of the second and third pulses in each beat. By examining the positions of the histograms, we can see that the length of each pulse follows a short-medium-long pattern (SML), which is consistent across each beat. Also shown are the probability density functions (PDF) of the maximum-likelihood estimated normal distributions. The plots/data for each beats showed the same pattern which also corresponds to other jembe pieces and matches results previously reported by Polak [8].

Figure 5 shows the results for the waltz dataset. The calculations use the first beat as the definition for the start of the cycle, so every Beat 1 has a displacement of 0. The early onset of the second beat can be clearly seen in the plot ($\mu = -0.0743$, $\sigma = 0.0795$). A one-sample t-test confirms the micro-timing as significant ($t = -16.5$, $p = 0.000$). Beat 3 shows no significant deviation from a 3-part isochronous division of the cycle, so the overall pattern identified is the short-long-medium (SLM) discussed elsewhere [7].

6.2 Jembe Tempo Estimation

The results of the jembe tempo estimation showed the increase in tempo throughout the piece that is characteristic of Malian jembe music. The more dramatic speedup at the end is also reflected in this data – this is why we fit two different curves to the data. For example, in "Suku", the tempo starts at around 135 bpm at the beginning of the piece and ends at around 175 bpm. The tempo results match those found by Jacoby et al. [36], and each jembe piece showed the same trend.

7. CONCLUSION

In this project, we have investigated and implemented probabilistic style-specific micro-timing in a musical live coding language. To do this, we extended the Sonic Pi language with implementations of both musical metre and micro-timing, and we performed data analysis on recordings of music from two case study styles to generate music with realistic micro-timing.

In further work (not reported here but available on request), we conducted a user study to assess how "realistic" our synthesised micro-timing sounded for each of the case study styles. Significant results were obtained for the Viennese waltz, however participants struggled more with the jembe, likely due to their unfamiliarity with the style. Future work could conduct a new user study with expert participants, as in Neuhoff [37].

Naturally, other future work could focus on additional styles with well-documented micro-timing, such as jazz swing rhythms [38], candombe drum ensembles from Uruguay [11, 39], and Brazilian samba music [11, 40]. Likewise, larger datasets for the styles reported here would enable more accurate distributions – two notable datasets of Viennese waltz recordings have been released even since the work reported here: Weigl et al. [41] and Yang [7].

As for software functionality, we imagine extensions including new *variable gridline* positions in DAWs, and additional controls within Sonic Pi to dynamically adjust the *"strength" of the micro-timing*.

8. ACKNOWLEDGEMENTS

Thanks to Rainer Polak for his invaluable advice and feedback on this project, for his years of research on jembe music, and for releasing the data that made that case study possible. Thanks indeed to all who provided feedback on this project. This research was conducted in the context of author MG's time as an Affiliated Lecturer at the Department of Computer Science and Technology, University of Cambridge. We thank Alan Blackwell and others for their role in organising and supporting this.

9. REFERENCES

[1] J. London, "Rhythm. Grove Music Online," 2001. [Online]. Available: https://www.oxfordmusiconline.com/grovemusic/view/10.1093/gmo/9781561592630.001.0001/omo-9781561592630-e-0000045963

[2] ——, *Hearing in time: Psychological aspects of musical meter*, 2nd ed. Oxford University Press, 2012.

[3] R. Cohn, "Meter," in *The Oxford Handbook of Critical Concepts in Music Theory*. Oxford University Press, 01 2020. [Online]. Available: https://doi.org/10.1093/oxfordhb/9780190454746.013.9

[4] C. E. Seashore, "The psychology of music. ix. a performance score with phrasing score for the violin," *Music Educators Journal*, vol. 24, no. 1, pp. 28–29, 1937. [Online]. Available: http://www.jstor.org/stable/3385490

[5] E. F. Clarke, "Levels of structure in the organization of musical time," *Contemporary Music Review*, vol. 2, no. 1, pp. 211–238, 1987.

[6] I. Bengtsson and A. Gabrielsson, "Rhythm research in Uppsala," *Music, Room and Acoustics*, 1977.

[7] J. Yang, "Viennese style in viennese waltzes: An empirical study of timing in the recordings of the blue danube," *Musicologica Austriaca: Journal for Austrian Music Studies*, no. 2022, 2022.

[8] R. Polak, "Rhythmic feel as meter: Non-isochronous beat subdivision in jembe music from Mali," *Music Theory Online*, vol. 16, no. 4, 2010.

[9] J. London, R. Polak, and N. Jacoby, "Rhythm histograms and musical meter: A corpus study of Malian percussion music," *Psychonomic bulletin & review*, vol. 24, no. 2, pp. 474–480, 2017.

[10] N. Jacoby, R. Polak, and J. London, "Extreme precision in rhythmic interaction is enabled by role-optimized sensorimotor coupling: analysis and modelling of West African drum ensemble music," *Philosophical Transactions of the Royal Society B: Biological Sciences*, vol. 376, no. 1835, p. 20200331, 2021.

[11] M. Fuentes, L. S. Maia, M. Rocamora, L. W. Biscainho, H. C. Crayencour, S. Essid, and J. P. Bello, "Tracking beats and microtiming in Afro-Latin American music using conditional random fields and deep learning," in *Proceedings of the 20th Conference of the International Society for Music Information Retrieval*, 2019, pp. 251–258.

[12] M. E. P. Davies, M. Fuentes, J. Fonseca, L. Aly, M. Jerónimo, and F. B. Baraldi, "Moving in time: Computational analysis of microtiming in maracatu de baque solto," in *Proceedings of the 21st International Society for Music Information Retrieval Conference*, 2020, pp. 795–802.

[13] J. Bilmes, "Timing is of the essence: Perceptual and computational techniques for representing, learning, and reproducing expressive timing in percussive rhythm," Ph.D. dissertation, Massachusetts Institute of Technology, 1993.

[14] S. Flossmann, M. Grachten, and G. Widmer, "Expressive performance rendering: Introducing performance context," *Proceedings of the 6th Sound and Music Computing Conference (SMC)*, pp. 155–160, 2009.

[15] ——, *Expressive Performance Rendering with Probabilistic Models*. Springer London, 2013, pp. 75–98.

[16] A. F. Blackwell, E. Cocker, G. Cox, A. McLean, and T. Magnusson, *Live coding: a user's manual*. MIT Press, 2022.

[17] A. McLean and G. Wiggins, "Tidal–pattern language for the live coding of music," in *Proceedings of the 7th sound and music computing conference*, 2010, pp. 331–334.

[18] S. Aaron and A. F. Blackwell, "From Sonic Pi to Overtone: Creative musical experiences with domain-specific and functional languages," in *Proceedings of the first ACM SIGPLAN workshop on Functional art, music, modeling & design*, 2013, pp. 35–46.

[19] J. McCartney, "Rethinking the computer music language: Super collider," *Computer Music Journal*, vol. 26, no. 4, pp. 61–68, 2002. [Online]. Available: http://www.jstor.org/stable/3681770

[20] J. Forth, "Cognitively-motivated geometric methods of pattern discovery and models of similarity in music," Ph.D. dissertation, Goldsmiths, University of London, 2012.

[21] C. Ariza and M. S. Cuthbert, "Modeling beats, accents, beams, and time signatures hierarchically with music21 meter objects," in *Proceedings of the 2010 International Computer Music Conference, ICMC 2010*. Michigan Publishing, 2010. [Online]. Available: http://hdl.handle.net/2027/spo.bbp2372.2010.043

[22] H. C. Longuet-Higgins and C. S. Lee, "The rhythmic interpretation of monophonic music," *Music Perception: An Interdisciplinary Journal*, vol. 1, no. 4, pp. 424–441, 1984. [Online]. Available: http://www.jstor.org/stable/40285271

[23] G. Sioros, M. E. P. Davies, and C. Guedes, "A generative model for the characterization of musical rhythms," *Journal of New Music Research*, vol. 47, no. 2, pp. 114–128, 2018.

[24] M. Gotham, "The metre metrics: Characterising (dis)similarity among metrical structures," Ph.D. dissertation, University of Cambridge, 2015.

[25] J. Neal, "Songwriter's signature, artist's imprint: The metric structure of a country song," in *Country Music Annual 2000*, C. K. Wolfe and J. E. Akenson, Eds. Lexington, KY: University Press of Kentucky, 2000, pp. 112–140.

[26] G. E. P. Box and M. E. Muller, "A note on the generation of random normal deviates," *The Annals of Mathematical Statistics*, vol. 29, no. 2, pp. 610–611, 1958.

[27] S. Aaron, "Sonic Pi — reliable randomisation for performances," in *2016 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 2016, pp. 242–243.

[28] M. Clayton, K. Jakubowski, T. Eerola, P. E. Keller, A. Camurri, G. Volpe, and P. Alborno, "Interpersonal entrainment in music performance: theory, method, and model," *Music Perception: An Interdisciplinary Journal*, vol. 38, no. 2, pp. 136–194, 2020.

[29] R. Polak, S. Tarsitani, and M. Clayton, "IEMP Malian jembe," 7 2020.

[30] M. Clayton, S. Tarsitani, R. Jankowsky, L. Jure, L. Leante, R. Polak, A. Poole, M. Rocamora, P. Alborno, A. Camurri *et al.*, "The interpersonal entrainment in music performance data collection," *Empirical Musicology Review*, vol. 16, no. 1, pp. 65–84, 2021.

[31] S. Dixon, "Automatic extraction of tempo and beat from expressive performances," *Journal of New Music Research*, vol. 30, no. 1, pp. 39–58, 2001.

[32] J. J. Moré, "The Levenberg-Marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1977, pp. 105–116. [Online]. Available: https://www.osti.gov/biblio/7256021

[33] I. Bengtsson, "Empirische rhythmusforschung in Uppsala," *Hamburger Jahrbuch für Musikwissenschaft*, vol. 1, pp. 195–219, 1974.

[34] M. Müller and F. Zalkow, "libfmp: A Python package for fundamentals of music processing," *Journal of Open Source Software*, vol. 6, no. 63, p. 3326, 2021.

[35] M. Müller, *Fundamentals of music processing: Using Python and Jupyter notebooks*, 2nd ed. Springer, 2021.

[36] N. Jacoby, R. Polak, and J. London, "Supplementary material from extreme precision in rhythmic interaction is enabled by role-optimized sensorimotor coupling: analysis and modelling of West African drum ensemble music," 7 2021.

[37] H. Neuhoff, R. Polak, and T. Fischinger, "Perception and evaluation of timing patterns in drum ensemble music from Mali," *Music Perception: An Interdisciplinary Journal*, vol. 34, no. 4, pp. 438–451, 2017.

[38] C. Dittmar, M. Pfleiderer, S. Balke, and M. Müller, "A swingogram representation for tracking micro-rhythmic variation in jazz performances," *Journal of New Music Research*, vol. 47, no. 2, pp. 97–113, 2018.

[39] L. Jure and M. Rocamora, "Microtiming in the rhythmic structure of Candombe drumming patterns," in *4th Int. Conf. on Analytical Approaches to World Music (AAWM)*, 6 2016.

[40] L. Naveda, F. Gouyon, C. Guedes, and M. Leman, "Microtiming patterns and interactions with musical properties in samba music," *Journal of New Music Research*, vol. 40, no. 3, pp. 225–238, 2011.

[41] D. M. Weigl, C. VanderHart, M. Pescoller, D. Rammler, M. Grassl, F. Trümpi, and W. Goebl, "The vienna philharmonic orchestra's new year's concerts: Building a fair data corpus for musicology," in *Proceedings of the 9th International Conference on Digital Libraries for Musicology*. Association for Computing Machinery, 2022, pp. 36–40.