EAI Endorsed Transactions
on Creative Technologies Research Article ICST.ORG

Varianish: Jamming with Pattern Repetition

Jort Band[1], Mathias Funk[1,*], Peter Peters[1], Bart Hengeveld[1]

[1]Department of Industrial Design, Eindhoven University of Technology, Eindhoven, The Netherlands

Abstract

In music, patterns and pattern repetition are often regarded as a machine-like task, indeed often delegated to drum Machines and sequencers. Nevertheless, human players add subtle differences and variations to repeated patterns that are musically interesting and often unique. Especially when looking at minimal music, pattern repetitions create hypnotic effects and the human mind blends out the actual pattern to focus on variation and tiny differences over time. Varianish is a musical instrument that aims at turning this phenomenon into a new musical experience for musician and audience: Musical pattern repetitions are found in live music and Varianish generates additional (musical) output accordingly that adds substantially to the overall musical expression. Apart from the theory behind the pattern finding and matching and the conceptual design, a demonstrator implementation of Varianish is presented and evaluated.

Keywords: Musical patterns, rhythm, pattern detection, micro-focus, improvisation.

Received on 15 May 2014, accepted on 11 September, published on 14 October 2014

doi: 10.4108/ct.1.1.e3

1. Introduction

Rhythmic patterns play a fundamental role in almost all kinds of music. They structure a musical piece, give texture to harmonics, and they induce effects on listeners such as the feeling of energy and movement. Music with pronounced (rhythmic) patterns is widely used for dancing and marching – the synchronization of bodily movements to these patterns allows for groups moving as a whole, stimulating movements, emotions, and also joint music play [11].

Music that consists solely of (too few) repeating patterns, however, is often regarded as dull, mechanic, and "technoid", which can be the intention, but requires a certain state of mind to enjoy thoroughly. For human players, it is natural and unavoidable to introduce small variations into repeated patterns. Minimal music pieces that require the precise repetition of patterns over several minutes by one or more musicians are often hard to perform, but very interesting to listen to, which reveals a particular quality of human hearing and musical perception: automatically reducing redundancy in what is perceived and attenuated focus towards the variations in the stream of perceived information, which holds naturally also for the other senses. When the information is largely redundant as in a repeated musical pattern, this results in a micro-focus on tiny variations in how the human player slightly changes the pattern over time–something that especially minimal music, but also many electronic music styles effectively *base* on.

In this paper we introduce Varianish, a musical instrument concept aiming at capitalizing on the aforementioned phenomenon by enabling ad-hoc improvisation with repetitive rhythmic patterns, to which Varianish gradually adds more and more musical layers autonomously–but only if the pattern is repeated strictly. When reaching levels of multiple added layers, the musician can start experimenting with slight variations of the original pattern by removing layers and then adding other layers by accurately repeating the new pattern for some time. While the technical challenges in creating Varianish have been tackled before, the most interesting aspect about Varianish indeed emerges only when using it: the system encourages and rewards repetition of musical patterns, and leads the human player into an exploration of tiny timing differences, imperfections of human play and subsequently the experimentation with timing and rhythmic patterns–in other words, a

[*]Corresponding author. Email: m.funk@tue.nl

microscope for musical micro-structures. This might feel counter-intuitive at first sight, in a musical culture, in which variation is sought after. However, rhythmic monotonicity and depth in repetition have their own appeals and inspire even musical subcultures and communities.

Varianish as a musical instrument allows for (solo) improvisation as well as for group music play. While in the first case the added layers can be broader and more expressive in terms of harmonics, sound effects, and added voices, in the second case, added layers need to comply more with the overall intention of the piece or performance, which can be influenced by choosing appropriate instruments or effects for the Varianish layers. Interactive composition is possible

The main challenge in designing Varianish was how to create an instrument, which is able to recognize musical patterns in real-time and which can respond to the repetition of a pattern as a positive feedback loop by adding one or more musical layers. This technical challenge can be divided into two parts; the software backbone encompassing pattern finding and matching algorithms, and the implementation part using DAW (digital audio workstation) software and input hardware to create an instrument that is able to add musical layers. This requires the software to be able to cope with input given in real-time, which is a challenge in itself and should be a consideration in both the software backbone as well as the implementation part.

In the following, related work is presented, after which the conceptual design of Varianish is shown, followed by an implementation and a showcase of the final prototype. The paper concludes with a short evaluation and a discussion of future steps.

2. Related Work

Analysing the rhythmic structure of music, either from score, but more so in real-time is a well-researched topic, framed as real-time pattern recognition or beat-tracking [5, 9]. The use of such derived information as input for generative purposes is not a new concept. For example, research can be found dating back to the 1980s when pattern recognition was used in real-time music generation by using predefined accompaniment sequences and patterns [4]. Later applications focus, for instance, on beat-tracking [19], interactive composition [15, 21] and the use of Bayesian Belief networks [18] to improve the synthesis of more open and richer accompaniments. Varianish differs from these approaches in the intention and purpose of generated musical layers: while most generative applications focus on enriching and varying a musical experience, Varianish aims at positive feedback loops for repetition leading towards deeper focus on micro-timing and the actual beauty of human inaccuracies in playing rhythmic patterns.

Other research areas focus on real-time pattern recognition for interacting with a computer system [8], although this is a limited scope and applies to predefined input patterns only. Related research can also be found on onset detection algorithms [20] and transcribing music through MIDI input data, by using pattern matching [2] and through Monte Carlo methods [22], often using a static tempo, which is either pre-determined or pre-set [2, 20, 22]. A related research area is the use of pattern recognition software to identify music genre [6] and the use of pattern recognition software to identify specific musical pieces (songs) from a database [3, 7], which are usually based on artificial neural networks [23]. Repetitive musical structure are also analysed and used for music visualisation [17].

We are not aware of any other expressive musical instrument concept that focuses at leveraging pattern repetition for rhythmic micro-focus. Varianish aims at giving human players a different kind of control over the generated layers of music, and is this sense, this approach is similar to other approaches to control music and sound with rather abstract representations (see for instance, [13]).

3. Varianish

Varianish[†] is a musical instrument that will encourage the musician to invent first, and then consequently repeat a rhythmic and, given a pitched instrument, also melodic pattern. The instrument will analyse the live input and at some point – after a sufficiently stable repetition phase – gradually start adding layers to the overall sound. If the pattern is varied, layers will again disappear before coming back after the new pattern stabilized. These additional layers can be diverse, and depend very much on the type of music that is intended, for instance, drones, harmonics, noise, and other additional elements are possible. Varianish can also be used to control effect processors that shape the incoming sound in a creative way, such as sound-based effects like distortion and filtering, sample-based like stuttering and pitching, or delay-based effect like Chorus or Flanger. In the remaining sections, the core concept of Varianish is presented together with a demonstrator implementation for a drum machine with additional effects and harmonics.

[†] The name “Varianish” is a straightforward mash-up of “variation” and “to vanish”.

3.1. Technical Concept

At the technical level, Varianish processes timed events: it (1) finds patterns in a stream of events, and after "tuning" into a certain pattern; (2) checks the stream for repetitions of this pattern. The amount, consistency and accuracy of such pattern repetitions are then used to determine how additional layers can be added to the overall sound output. Varianish accepts input of rhythmic patterns of one or more (pitched) trigger events, called notes in the following, and, in the case of multiple of such notes (differing for instance in pitch), Varianish performs the pattern processing for these notes in parallel. In this way, music based on complex polyrhythmic or melodic patterns can be fed through Varianish and can be enriched by it. In the following, this brief outline of the Varianish concept shall be unfolded:

3.1.1. Pre-processing

Enrichment through Varianish starts with pre-processing: Raw timed notes generated by a human player, are usually not quantized, i.e., timing-wise noisy, which would lead to problems when looking for patterns and their repetitions [2, 20]. Therefore a first pre-processing step is needed, in which incoming events are quantized into a grid of notes that can be better processed subsequently. This quantization step removes slight variations in timing from the stream and yield events with corrected timing that is according to a global metric such as 1/4 or 1/16 of a bar.

The pre-processing step deals with timed sound signals, which can be transients in a stream of audio data [20], but which are more likely input in the form of notes such as implemented and delivered by the MIDI protocol. This MIDI data has to be converted to MIDI onset timing data, which contains the MIDI note and time at which the note had its onset, but no duration and other optional note attributes.

To compare the incoming MIDI data, the data has to be processed and quantized into a timing grid. Without quantization, calculation time would increase and data required for calculation would be much larger. For this application, a relative quantization method is chosen, i.e., the coarseness of the quantization is determined by the MIDI timing data–in contrast to quantization methods based on a set tempo or pre-determined tempo.

To relatively quantize the note onsets, the smallest inter-onset interval should be found. For this purpose, a dataset is created that contains only the onset timing data from the MIDI onset timing data of all MIDI notes (making it a list with timing events), we will call this dataset $x(x_1, x_2, \ldots, x_n)$. The dataset x is ordered in chronological order.

This data is converted to an inter-onset interval dataset, i.e., a data set containing the intervals between each consecutive note: $y\ (y_1, y_2, \ldots, y_n)$, which is calculated by:

$y_{z-1} = x_z - x_{z-1}$, z being an arbitrary number in the range of 2 to n). Due to the inaccuracy in human play we can expect an slight onset difference in polyphonic music between two notes played at the "same" time, which represents a second source for onset difference. To compensate for both kinds of onset differences we introduce the threshold o_v, which is the maximum timing difference two note onsets can have, in which they are considered having been played simultaneous, o_v ranging between 5ms and 50ms (an optimum for this value should still be found). To account for this in the data set, a new data set is created, called i $(i_1, i_2, \ldots, i_n)$. Basically, i is y with the onsets y_z removed that comply with $y_z - y_{z-1} < o_v$, z being an arbitrary number in the range of 2 to n). From data set i, the smallest interval will be selected, which will be referred to as S_i in the following.

To compensate for the player's note onset inaccuracy both in terms of tempo and beat matching, a dataset, which we shall name s_c $(s_1, s_2, \ldots, s_n)$, is made with the values from i, that range between S_i and $S_i + S_i \cdot f$, where f is a pre-defined fault tolerance constant. In the practical application of the Varianish a constant of 1.4 was used for f. With the data set of s_c, the mean is calculated, which we will name d (cf. Equation 1). To allow for dotted notes (1.5 times a normal note), the final divider D, which will be used for quantization, will be determined by Equation 2. Without dotted note compensation, D is equal to d.

$$d = \frac{\sum_{i=1}^{n} s_c}{n}$$

Equation 1

$$D(d, o_v) = \begin{cases} \frac{d}{2} if\ d > o_v \cdot 2 \\ d\ if\ d \le o_v \cdot 2 \end{cases}$$

Equation 2

The reason why D is not divided by two if it is smaller than $o_v \cdot 2$ is to prevent the divider coming into the range of o_v, which would result in erroneous quantization. D will be used to quantize the notes by selecting the earliest onset time from the onset timing data, which will be named t_s. Now the relative position (p_r) of a MIDI note (t_m) from the received MIDI data can be calculated in reference to t_s and by means of D, with a fault tolerance given by f_c (in the practical application of Varianish, a constant of 0.7 was used for f_c), given by Equation 3.

$$p_r(D,t_m,t_s,f_c) = \begin{cases} \frac{(t_m - t_s)}{D} + 1 & if \quad (t_m - t_s)\%D \geq D \cdot f_c \\ \frac{(t_m - t_s)}{D} & if \quad (t_m - t_s)\%D < D \cdot f_c \end{cases}$$

Equation 3

Converting the MIDI onset timing data set from point t_s up to an arbitrary point will give a new grid position data set which will be called P for now.
Before this data will be used in pattern finding or pattern matching, another conversion is necessary: the conversion from position data to a bit array grid. For example: we have P_n, with P_n being an arbitrary position data set for a specific note, P_n contains the data (1,5,9), so the total grid size will be 9. This will be converted to a bit array (figuratively reading from left to right) which will contain the following data: 10001000 1, as you can see having an onset on the first fifth and ninth position.

3.1.2. Pattern finding

After a first step of quantization, the processed stream of events is ready for the subsequent pattern finding step: For pattern finding we use the quantized data and compare it to itself. This is done by means of a comb-filter. This comb-filter calculates a pattern on a note sequence (bit array for one note) by note sequence basis. For the comb-filter to give a positive result the minimum requirement is that a pattern should at least repeat itself three times within the entire sequence of incoming notes and that no noise is present (the bit array should only contain the pattern repetition).

The comb-filter works by comparing the bit array with itself. This is done by comparing the beginning and the end of the bit Array to itself. For example it starts to compare the beginning and the end of the bit array to each other (1 bit); in the next iteration it compares the two bits at the beginning and the two bits at the end and so on (see figure 1).

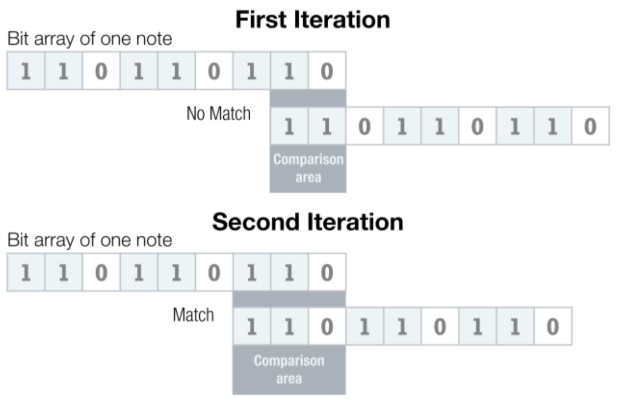

Figure 1. Example of two iterations of a comb-filter

To prevent false positives, i.e. finding patterns where the player does not actually repeat anything recognizable, the comparison process is not started at the first bit and last bit, but started at slightly less than a third of the bit array size and stops at a maximum of the bit array size -1. To find a second pattern exact matches have to be found. The reason for this is that with two matches the exact size of a pattern can be found and also to prevent false positives (for example a pattern which has an inner repetition).

After a pattern for each note sequence is found (note sequences without input are skipped) the total pattern size will be the least common multiple (LCM) between the individual pattern lengths. The individual note patterns will then be repeated to fit within the total pattern size and stored in the final pattern data set, which contains the entire repeated individual note pattern of all the notes. For example: an individual note sequence has a pattern size of 2; and another, second note sequence has an individual pattern size of 4. The first note sequence will be repeated twice and the second note sequence will be stored as is in the final pattern data set. That way, polyrhythmic patterns of different notes will be compatible in terms of sequence length.

3.1.3. Pattern matching

Pattern matching occurs when a known pattern is compared to incoming data. The incoming data is compared after it has been quantized. The data can then be compared in chunks that are equal in size with the known pattern's size. These data chunks will be compared on a note-by-note basis to their pattern counter parts by means of a basic Levenshtein comparison algorithm [10], which will output a Levenshtein distance for each individual note.

3.2. Implementation

The implementation of the project consisted of using a MIDI pad controller (a Novation Launchpad [14]) as input device, the Processing programming environment [16] for the processing of the MIDI input/output and for the pattern matching and finding. Ableton Live [1] was used to convert the MIDI to sound by means of sample triggering.

3.2.1. Input

For this project MIDI (Musical Instrument Digital Interface) [12] is chosen, as it is a widely implemented and supported protocol, which is used in musical instruments and software. MIDI input has the advantage over "normal" audio input that it directly provides events and not a continuous audio wave that first needs to be converted into discrete events, which is often processing intensive and prone to errors [20].

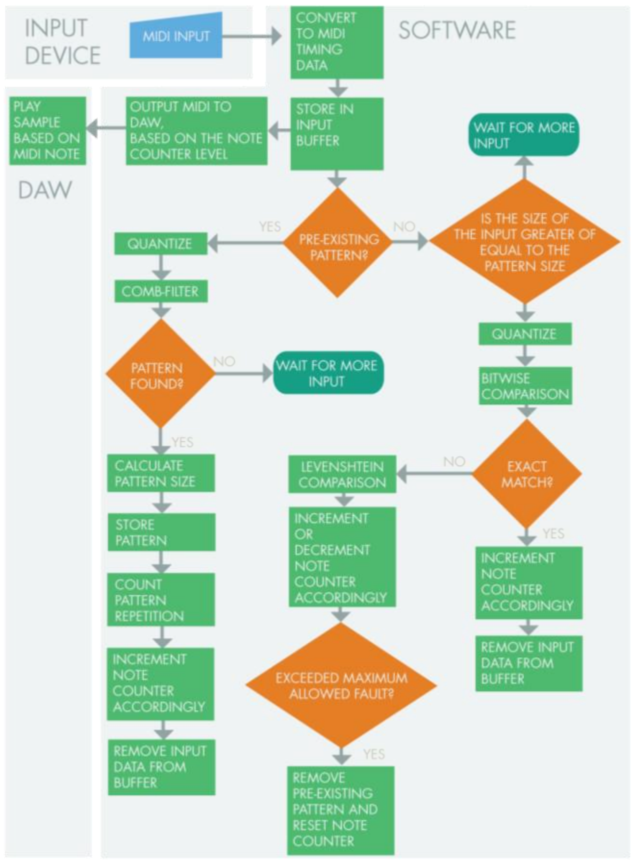
INPUT DEVICE
MIDI INPUT
CONVERT TO MIDI TIMING DATA
SOFTWARE
PLAY SAMPLE BASED ON MIDI NOTE
OUTPUT MIDI TO DAW, BASED ON THE NOTE COUNTER LEVEL
STORE IN INPUT BUFFER
WAIT FOR MORE INPUT
DAW
QUANTIZE
YES
PRE-EXISTING PATTERN?
NO
IS THE SIZE OF THE INPUT GREATER OF EQUAL TO THE PATTERN SIZE
COMB-FILTER
QUANTIZE
PATTERN FOUND?
NO
WAIT FOR MORE INPUT
BITWISE COMPARISON
YES
CALCULATE PATTERN SIZE
LEVENSHTEIN COMPARISON
NO
EXACT MATCH?
STORE PATTERN
INCREMENT OR DECREMENT NOTE COUNTER ACCORDINGLY
YES
COUNT PATTERN REPETITION
INCREMENT NOTE COUNTER ACCORDINGLY
REMOVE INPUT DATA FROM BUFFER
INCREMENT NOTE COUNTER ACCORDINGLY
EXCEEDED MAXIMUM ALLOWED FAULT?
REMOVE INPUT DATA FROM BUFFER
YES
REMOVE PRE-EXISTING PATTERN AND RESET NOTE COUNTER

Figure 2. A flow chart depicting the system-level view on the Varianish core system.

3.2.2. Software

An overview of the software is shown in Figure 2. The path from MIDI to sound is kept relatively short; Varianish is simply chained into the MIDI connection between input and DAW output, with the possibility to analyze incoming MIDI notes at real-time and, if needed, add more MIDI data, before sending everything to the DAW (digital audio workstation), which in this case is, Ableton Live. The DAW will output MIDI notes it receives from the Varianish as sounds, based on a predefined set of sounds samples. The MIDI output of the Varianish software consists of MIDI note-on events and of messages output on different MIDI channels, where each MIDI channel represents a new musical layer.

For the actual pattern processing the incoming MIDI data has to be stored in an input buffer. Depending if a pre-existing pattern is present the program will try to find a pattern or try to match the pre-existing pattern to the input data. When a pattern has to be found, the data is quantized and then comb-filtered. To account for input "noise", meaning that the beginning input might not contain any pattern, the comb-filter is run several times where each time it is run, a smaller piece of the input buffer is selected for comb-filtering. If no pattern is found, the program will wait for more input. If a pattern is found the program will calculate the pattern size and store the pattern. The amount of repetitions of the pattern is then counted and the note counter (counts the amount of pattern repetitions for each individual note) is incremented accordingly. Finally, the input data is removed from the input buffer. When a pre-existing pattern is present, the program will try to match the input data to this pre-existing pattern. First the program checks if the amount of data in the input buffer is sufficient to have the pattern present in it; if not, it will wait for more input; if so, it will quantize the data and use bitwise comparison to see if the data matches exactly. The reason to check for an exact match first is that this is much faster than doing a Levenshtein distance comparison and can thus save in calculation time. If an exact match is found the note counter is incremented accordingly and the input buffer is cleared.

If no exact match is found, the Levenshtein distance for each individual note is calculated, and depending on the distance, the note sequence counter will be incremented or decremented. When too few successful repetitions have occurred, i.e. the note counter is decremented more than a tolerance maximum since an exact match is found, the pre-existing pattern is removed and the note counter is reset.

3.2.3. Tolerance

The tolerances added in the software have been largely discussed in the theory section of this paper; a short overview of these fault tolerances are given below.

The first is onset tolerance, notes that are played at the same time always have some onset variance between them; the tolerance for this is added during the quantization process. The second is tempo variance tolerance; the user will always drift in tempo. To compensate for this a new divider is calculate each time data is quantized, meaning that the data will always be quantized in comparison to its own tempo. The third tolerance is the allowance of inaccuracies or non- pattern data (data which does not contain a pattern) in both the pattern finding stage as well as in the pattern matching stage. During the pattern finding stage the input buffer is compared to itself several times, each iteration taking a smaller piece of the input buffer (by removing the oldest inputs). During the matching stage only the newest chunk of input data with the same size as the pattern is compared exactly and by means of Levenshtein comparison. The Levenshtein comparison is able to give a relative fault distance, thus being inaccurate "friendly".

3.2.4. Processing performance

Because the software had to be able to run real-time, processing performance had to be taken into account. Some considerations have been mentioned briefly throughout this paper. A small overview of these considerations is given below.

1. It was really important that the basic comparison algorithms were fast and lightweight. To achieve this, the data is quantized into a bit array. The advantage of using a bit array is that bitwise operations can be performed; these operations are primitive and directly supported by the processor, which means they are fast.
2. For performance the choice was made to “favor” an exact match when the input data and a pre-existing pattern are compared. What is meant by this is that it first checks for an exact match, instead of doing a Levenshtein comparison first (which is also able to determine an exact match), because the Levenshtein comparison is much more processing intensive.
3. Another consideration was the maximum size of the input buffer. To keep the software real-time it can only process a limited amount of input data. Therefore, a maximum input buffer size is set. This has as a result that the maximum size of the pattern is limited.

3.2.5. Instrument

As final result for this project we aimed for an instrument/device, which was able to find and match patterns and to convert this to something musical. It had to be able to add musical layers to a player’s performance by using pattern matching and finding. This can be best compared to the way organs add more richness to their sound: organs add layers of sound by activating registers or voices resulting in a bigger soundscape. This principle was transferred to enable the instrument to add samples on samples the more a specific note was repeated within a pattern; resulting in a bigger soundscape the more a pattern is repeated.

The resulting instrument used a Novation Launchpad (a MIDI pad controller) as interface. With this controller MIDI notes were generated as input for the software. It also provided feedback to the player, trough lighting the pads . The player plays the instrument the same as a MIDI pad controller would normally be played. The goal is to repeat a pattern over and over again to create a bigger and bigger soundscape. More samples per note are added the more a player repeats a pattern. These samples are added by sending the same note on different MIDI channels the more the note is correctly repeated in a pattern the more MIDI channels are used. On each of these MIDI channels a different complementary sample is present for each note. Feedback to the player is two-fold. One is visual status indication, where the color of the pad indicates a certain level and the brightness of the pad indicates the inter-level progress an illustration of which can be found in figure 3. The level and inter-level feedback, is there to provide visual feedback about the amount of layers added on each individual note. Feedback was also given when the pattern was found by flashing all the pads once.

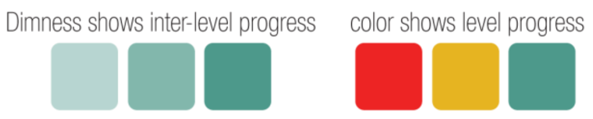
Dimness shows inter-level progress
color shows level progress

Figure 3. A graphic representation of the level progress given by the instrument.

The other is auditory feedback: adding sound layers to notes can be heard. Besides this, timed delays based on the original divider can be added to intrinsically motivate the user to stay in the original tempo.

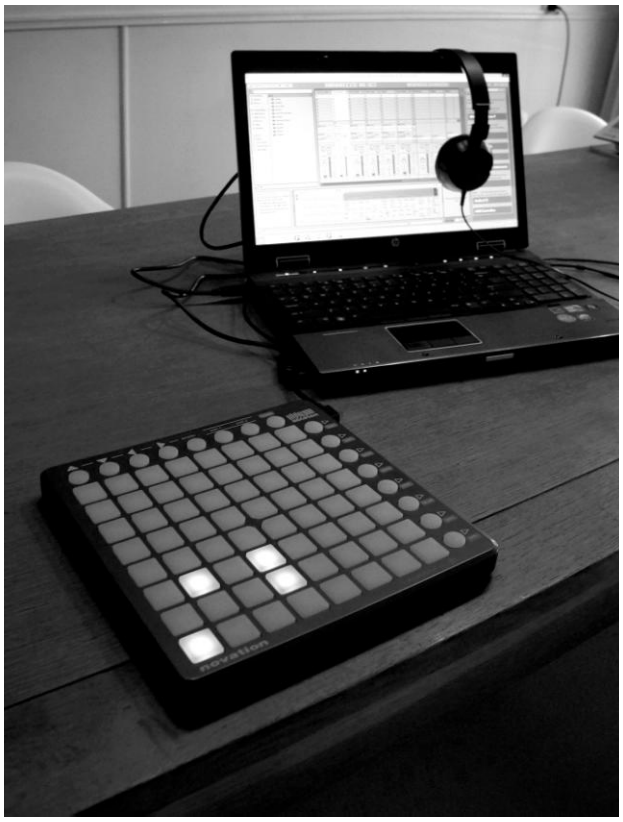

Figure 4. System overview

4. EVALUATION

The system was informally evaluated during development and in an open exhibition-like setting with musicians and non-musicians. The setup (see Figure 4) was as described, using a Novation Launchpad as the controller and Ableton Live as the sound generator. Varianish was chained in-between to capture all events from the Launchpad, analyse them, and, in the case of successful pattern repetitions, trigger additional musical layers in Live as well as control LEDs on the Launchpad to visually indicate progress.

In the following, evaluation findings are presented divided into technical and usage aspects.

4.1. Technical Evaluation

During the evaluation it became clear that a large, almost full input buffer significantly increased the calculation time and thus imposed a lag on the rest of the system (in cases when the processing of the incoming notes takes longer than the interval between notes). Giving the input buffer a maximum size reduced this problem.

It also became apparent that not only the buffer size was of importance for the calculation time, but also the data input into the buffer. The calculation time is often within the maximum allowed range, but setting a maximum buffer size is not a sufficient means to ensure that the calculation time is within an acceptable range. To counter this, optimization has to be used and a way of dynamically changing the input buffer size. The calculation time also influences the maximum pattern size that can be recognized, which has as a result that the pattern can only have a limited size and that the system is only able to process relatively small patterns, an estimate would be a pattern consisting of 12 to 30 note onsets, this estimate is rather wide and further evaluation and optimization with the system has to be done to get a more accurate estimate, which were still large enough for most musicians that tried the system.

In future versions, a dynamically changing input buffer size would be advisable, by means of monitoring the calculation time and adjusting the buffer size accordingly. This would prevent the software from exceeding the maximum allowed calculation time. It also became clear that the system had difficulty with recognizing shuffle patterns. The difficulty of picking up shuffle patterns will probably be fixed by compensating for dotted notes as discussed in the theory, which was not yet implemented fully during the evaluation. Another point was the optimization of the algorithm in favour of perfect pattern repetitions, resulting often in looking for an exact match and not finding it, after which the Levenshtein algorithm is still run. These points will be further investigated and optimized in a future version of Varianish, given more extensive input from follow-up user evaluations.

4.2. User Evaluation

During the exhibition setting participants were instructed to play with the device and repeat a pattern over and over. The players often started by exploring the initial sounds available to them and only then tried to invent and repeat a pattern. A significant amount of the participants gave up repeating a pattern after a few times or were not able to repeat even a simple pattern. For many participants it was indeed a difficult task to repeat the same pattern several times accurately. The level of focus and the occurrence of rhythmic inaccuracies have a very direct relation, however, fine motoric skills play a role as well. Reducing the complexity of the pattern has a positive effect, however, this did not naturally occur to many participants.

The visual feedback that is given when the system recognized a pattern was startling for some users, which often resulted in them stopping the pattern repetition. The participants were all able to hear layers being added to the notes they played, which meant that the auditory feedback was chosen well and that the layers of sound are clearly distinguishable. So, while the auditory feedback was clear, the visual feedback of the system (when it recognizes a pattern) should be revised in a future version.

5. CONCLUSION

This paper introduces Varianish, a musical instrument concept based on repetition of musical rhythmic patterns that result in the generation of additional sound layers to be added to the overall sound. Our findings suggest that Varianish is able to perform its core basic task: recognizing musical patterns in real-time and triggering both visual output to the player as well as generating additional musical layers. This works in real-time with interchangeable sample or sound sets resulting in the desired micro-focus experience.

An interesting aspect of the Varianish concept is that patterns can both be very simple and very complex–the instrument works the same, and thus allows in principle for similar experiences for novice and expert players alike. The evaluation, however, showed that the current implementation is not yet robust enough for beginners and better suited for slightly more advanced novice or expert players who can rely on their fine motor skills to consistently repeat a pattern.

The pattern recognition core itself has also potential to be used as a method of interaction in itself, which, to our knowledge, is a new way of interacting with a system that potentially has no musical context. An open-source release of the processing core is planned in a future release.

References

[1] Ableton Live, http://ableton.com
[2] Blostein, D., and Haken, L. Template Matching for Rhythmic Analysis of Music Keyboard Input, In Proc. of the 10th International Conference on Pattern Recognition, 1990.
[3] Chen, J.C.C., and Chen, A.L.P. Query by rhythm: an approach for song retrieval in music databases. In Proc. of the Eighth International Workshop on Research Issues In Data Engineering: Continuous-Media Databases and Applications, Orlando, Florida, 1997, 139–146.

[4] Dannenberg R. B. An On-Line Algorithm for Real-Time Accompaniment. In Proc. of the 1984 International Computer Music Conference, International Computer Music Association, 1984, 193-198.

[5] Davies, M. E. P., and Plumbley, M. D. Beat tracking with a two state model [music applications]. In Proceedings. (ICASSP '05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005. (Vol. 3, p. iii/241–iii/244 Vol. 3). IEEE, 2005.

[6] Dixon, S., Gouyon, F., and Widmer, G. Towards Characterisation of Music via Rhythmic Patterns. In Proc. of the 5th International Conference on Music Information Retrieval, 2004.

[7] Eikvil, L., and Huseby, R. B. Pattern Recognition in Music, 2002.

[8] Ghomi, E., Faure, G., Huot, S., Chapuis, O., and Beaudaouin-Lafon, M., Using Rhythmic Patterns as an Input Method. In "CHI '12: Proceedings of the SIGCHI Conference on Human Factors and Computing Systems," Austin, United States, 2012.

[9] Goto, M. An Audio-based Real-time Beat Tracking System for Music With or Without Drum-sounds. Journal of New Music Research, 30(2), 2001, 159–171.

Влади́мир И. Левенштейн (1965). "Двоичные коды с исправлением выпадений, вставок и замещений символов" [Binary codes capable of correcting deletions, insertions, and reversals]. Доклады Академий Наук СССР (in Russian) 163 (4): 845–8. Appeared in English as: Levenshtein, Vladimir I. (February 1966). "Binary codes capable of correcting deletions, insertions, and reversals". Soviet Physics Doklady 10 (8): 707–710.

[10] Juslin, P. N., and Madison, G. The Role of Timing Patterns in Recognition of Emotional Expression from Musical Performance. Music Perception, Vol. 17, No. 2, 1999, 197-221.

[11] MIDI, The MIDI Manufacturers Association, MIDI 1.0 Detailed Specification, 1995.

[12] Momeni, A., and Wessel, D. Characterizing and controlling musical material intuitively with geometric models, 2003, 54–62.

[13] Novation Launchpad, http://global.novationmusic.com/midi-controllers-digital-dj/launchpad-s

[14] Pachet, F. The Continuator: Musical Interaction With Style. Journal of New Music Research, 32(3), 2003, 333–341.

[15] Processing, http://processing.org

[16] Puzoń, B., and Kosugi, N. Extraction and visualization of the repetitive structure of music in acoustic data. In Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services - iiWAS '11 (p. 152). New York, New York, USA: ACM Press, 2011.

[17] Raphael, C. Synthesizing Musical Accompaniments With Bayesian belief networks. Journal of New Music Research, 30(1), 2001, 59–67.

[18] Robertson, A., and Plumbley, M. B-Keeper: a beat-tracker for live performance, 2007, 234–237.

[19] Schrader, J. E., and Vogten, L. L. M. Detecting and interpreting musical note onsets in polyphonic music, 2003.

[20] Spasov, M. Music Composition as an Act of Cognition: ENACTIV – interactive multi-modal composing system. Organised Sound, 16(01), 2011, 69–86.

[21] Taylan Cemgil, A., and Kappen, B., Monte Carlo Methods for Tempo Tracking and Rhythm Quantization. Journal of Artificial Intelligence Research 18, 2003, 45-81.

[22] Weyde, T., and Dalinghaus, K. Recognition of musical rhythm patterns based on a neuro-fuzzy-system. Smart Engineering System Design: Neural Networks, Fuzzy Logic, Evolutionary Programming, Data Mining and Complex Systems 11, 2001, 679-684.