Automatic Classification of Guitar Playing Modes

Raphael Foulon(✉), Pierre Roy, and François Pachet

Sony Computer Science Laboratory, Paris, France
{raphael.foulon,pachetcsl}@gmail.com

Abstract. When they improvise, musicians typically alternate between several playing modes on their instruments. Guitarists in particular, alternate between modes such as octave playing, mixed chords and bass, chord comping, solo melodies, walking bass, etc. Robust musical interactive systems call for a precise detection of these playing modes in real-time. In this context, the accuracy of mode classification is critical because it underlies the design of the whole interaction taking place. In this paper, we present an accurate and robust playing mode classifier for guitar audio signals. Our classifier distinguishes between three modes routinely used in jazz improvisation: bass, solo melodic improvisation, and chords. Our method uses a supervised classification technique applied to a large corpus of training data, recorded with different guitars (electric, jazz, nylon-strings, electro-acoustic). We detail our method and experimental results over various data sets. We show in particular that the performance of our classifier is comparable to that of a MIDI-based classifier. We describe the application of the classifier to live interactive musical systems and discuss the limitations and possible extensions of this approach.

Keywords: Audio classification · Playing mode · Guitar · Interactive musical systems

1 Introduction

An ideal interactive musical system should allow users to play as if they were performing with fellow musicians. To achieve this, the behavior of such an ideal system must be predictable and consistent. One way to achieve this is to provide explicit controls such as manual switches, pedals, knobs, or sliding faders. However, some musical intentions cannot be given explicitly because it creates a cognitive overhead that interferes with the performance.

Musicians typically alternate between several *playing modes* when they improvise. A guitarist, for instance may alternate between chord comping, solo melodic improvisation, or walking bass. Each mode calls for a specific reaction from the other musicians. For instance, in a guitar duo, if one musician improvises a solo melody, the other will usually complement with chord comping.

In this paper, we address the problem of automatically classifying the audio input of a guitar improviser into three musical modes (melody, chords, and bass). We show that recognizing these modes automatically and in real-time opens the way to musically aware interactive applications such as VirtualBand [14] or augmented instruments.

Research developed in the MIR community has focused mainly on expressivity parameters and on the classification of playing techniques for various musical instruments. Lähdeoja et al. proposed a system that classifies in real time different playing techniques used by a guitarist [12, 15]. These techniques include up and down legatos, slides, slapped/muted notes, as well as the position of the pick on the neck with regards to the bridge. The system relies on the analysis of both the incoming audio signal and/or gesture capture data. Similar topics have been investigated with the goal of modeling *expressivity*, such as the articulation in nylon guitar [13]. Abesser et al. present a feature-based approach to classify several plucking styles of bass guitar isolated notes, and describes an automatic classifier of guitar strings for isolated notes using a feature-based approach [1] and a two-step analysis process [2]. Barbancho et al. study the retrieval of played notes and finger positions from guitar audio signals [3]. Instrumental technique classification methods have been investigated for beatboxing [17, 19] and for snare drums [20] with some success.

Stein et al. [18] describe an approach to analyze automatically audio *effects* applied to an electric guitar or bass, and Fohl et al. [6] studied the automatic classification of guitar tones. As we will see below, their objective is in some sense opposite to ours, since we aim at extracting information from the guitar signal that is precisely timbre-independent.

We propose a method to extract information about the musical content that is played, regardless of expressive parameters and of the technique used. For instance, jazz guitarists use various *playing modes*: octave playing (typical of Georges Benson or Wes Montgomery), chord comping, bass lines, mix of chords and bass (as in bossa nova). Our aim is precisely to detect these playing modes in real time for interactive applications. Each mode may be played in many different ways using different techniques and with different expressivity parameters. We do not aim at extracting information about *how* something is played, *e.g.*, staccato, legato, slapped, but rather about *what* is played, *e.g.*, melodies, octaves, single notes, bass, chords.

Playing modes would be in principle easy to analyze from a score of the performance. However, score-related symbolic data (pitches, durations) are available only from MIDI instruments. Furthermore, the accuracy of MIDI guitars is far from perfect and requires specific, expensive hardware. The accuracy of automatic score transcription from audio [10, 16] is not high enough to build robust live systems. More specifically, Hartquist [8] addresses the problem of automatic transcription of guitar recordings, but this approach is only evaluated on a pre-recorded note template library for a nylon string acoustic guitar, therefore, it is unclear whether it can cope with the dynamic variation that can occur in live recordings and with other types of guitars.

One key problem is to detect accurately and robustly polyphony from the guitar signal. The classification of monophony *versus* polyphony has been investigated [11] using the YIN pitch estimation algorithm [4] with bivariate Weibull models. This method is practical since it only requires short training sets (about two minutes of audio signal is enough) and works for many instruments with good performance (6.3 % global error rate). Most importantly, this work shows that YIN is an accurate descriptor for polyphony detection.

In this paper, we describe a mode classifier that classifies guitar audio signals into three basic playing modes described above: bass, chords and melody. Following the approach of Abesser [2], the classifier is based on a two-step analysis process (single frames then smoothing on larger windows), based on YIN-derived features, pitch and inharmonicity indicator [3, 11].

Our classifier is largely timbre-independent, *i.e.*, it performs well on the four types of guitar we tested. We describe the training data in the next section. The classifier is described in Sect. 3 and its performance is discussed in Sect. 4. Section 5 describes applications of the classifier for interactive music systems.

2 Datasets

Although all guitars exploit the sound of vibrating strings, there are different types of guitars and guitar sounds. In order to avoid biases or overfitting due to the use of a single guitar for training data, we built an audio dataset recorded with four guitars of different types:

- Godin LGX-SA solid-body guitar (*God*) – which has also a MIDI output – which output has been fed to a AER Compact 30 jazz amplifier,
- Cort LCS-1 jazz guitar (*Cort*),
- Ovation Electric Legend (model Al Di Meola) electro-acoustic guitar (*Ovat*) and
- Godin Nylon SA nylon string guitar (*Nyl*) (see Fig. 1).

For each guitar, we created a dataset that consists of the recordings of seven jazz standards: *Bluesette*, *The Days of Wine and Roses*, *LadyBird*, *Nardis*, *Ornithology*, *Solar*, and *Tune Up*. Each song has been recorded three times, one for each playing mode: *melody*, *bass*, and *chords*. The database contains therefore 84 files – 4 guitars × 7 songs × 3 modes – for a total audio duration of 1 h 39 min. The audio datasets may be downloaded at http://flow-machines.com/mode_classification_sets.

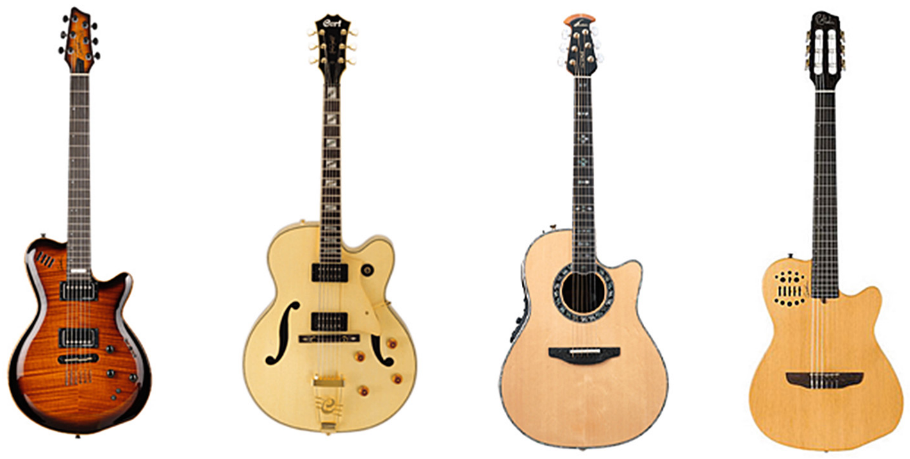

Fig. 1. The four guitars used to record the dataset. *From left to right*: pure solid-body Godin LGX, hollow-body Cort jazz archtop, Ovation electro-acoustic, and nylon-string guitar Godin Classic.

3 The Mode Classifier

Our method uses a two-phase analysis: first, short signal frames (50 ms) are classified with a supervised classification algorithm, which determines the playing mode over short-time windows, with an imperfect accuracy. Then, information obtained over the 50 ms frames is aggregated to classify a whole audio chunk. The scheme of the algorithm is shown in Fig. 3.

3.1 Feature Selection

We use a training set that consists of jazz guitar recordings corresponding to the three playing modes "bass", "melody" and "chords". The training sets described in this article are all extracted from the guitar recordings presented in Sect. 2.

We performed feature selection to determine which features are the most relevant for our problem. We used the Information Gain algorithm [7] of Weka [9], set with the lowest possible threshold (-1.8×10^{308}) to obtain a list of features ranked by information gain, and ran it on a set of 37 features divided in two sets:

(1) Basic audio features: MFCC (13), harmonic-to-noise ratio, spectral centroid, spectral flatness, spectral kurtosis, spectral decrease, spectral spread, spectral rolloff, spectral skewness, chroma (12), RMS.
(2) YIN features [8]: YIN pitch, YIN inharmonicity indicator and YIN variance.

Feature selection yields the six following features: harmonic-to-noise ratio, YIN pitch and YIN inharmonicity, spectral spread, spectral centroid and spectral kurtosis. This confirms that YIN features are indeed interesting for our task. To further reduce the feature set, we retained only the four following features:

(1) YIN pitch, which was quantized to avoid overfitting (this point is explained below), computed with an absolute threshold of 0.2 for aperiodic/total ratio,
(2) YIN inharmonicity coefficient, computed in the same manner,
(3) Harmonic-to-noise ratio (HNR) of the signal, computed with a fundamental frequency of 185 Hz (which is the lowest frequency possible with 50 ms frames, we would have to work with larger frame lengths to decrease the fundamental),
(4) Spectral spread.

3.2 Frame Selection and Training

The audio signals in the training set are normalized and then sliced into 50 ms frames, with a 75 % overlap. We chose 50 ms to ensure that each frame contains at most one musical event, even when dealing with fast tempos or virtuoso solos.

Preliminary empirical results show that, given our feature set, common statistical classifiers (SVM, decision trees, and Bayesian networks) fail to classify correctly the frames that contain transients. We remove silent frames and frames that contain transients from the training set, and train the classifier on the frames that contain the steady part of the signal.

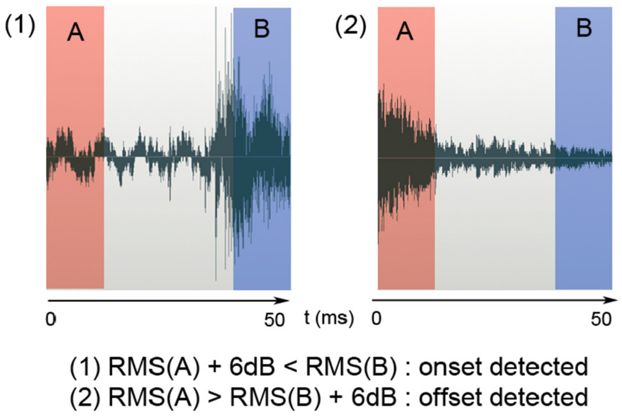
(1)
A
B
0
50
t (ms)
(2)
A
B
0
50
(1) RMS(A) + 6dB < RMS(B) : onset detected
(2) RMS(A) > RMS(B) + 6dB : offset detected

Fig. 2. Simple onset/offset detection procedure. The output of the algorithm is positive if there is a difference of 6 dB or more between the two RMS values.

To do so, we first use a noise gate with a −13 dB threshold to remove silent frames. To detect quickly transient frames, we use a simple onset/offset detection algorithm, presented in Fig. 2, which computes the difference between the RMS values of the first 10 and last 10 milliseconds of the signal, and applies a 6 dB threshold on it. More sophisticated techniques such as frequency domain-based onset detection [5] can be used, but the proposed solution is fast and works well enough for our goals.

Eventually, we extract the four features (YIN Pitch, YIN Inharmonicity Factor, Spectral Spread, and HNR) from the remaining frames (i.e., the frames that contain steady parts of the signal). We use this data to train classifiers using various machine learning algorithms: a Support Vector Machine with linear, radial and polynomial kernels, a Bayesian network and a J48 tree. The best classifier turns out to be a Bayesian network (Weka's *BayesNet* with a "GeneticSearch" algorithm with the default parameters).

3.3 Performance on Frame Classification

To evaluate the performance of the Bayesian network, we train the classifier on one song, *The Days of Wine and Roses* (the longest song of the database), taken from the Godin guitar (*God*) subset, and test it on the six other songs. When we classify the selected audio frames (discarding silent frames and transients) with our feature set and the Bayesian network, we obtain an average F-measure of 0.87. This result is not sufficient for a robust, real-time classifier. In the next section we add an aggregation, or smoothing step to our method to further improve the classifier performance, following the approach of Abesser [2].

3.4 Aggregation

In order to improve the classification performance, we aggregate the results of individual frame classification within a given time window (called thereafter chunk) and

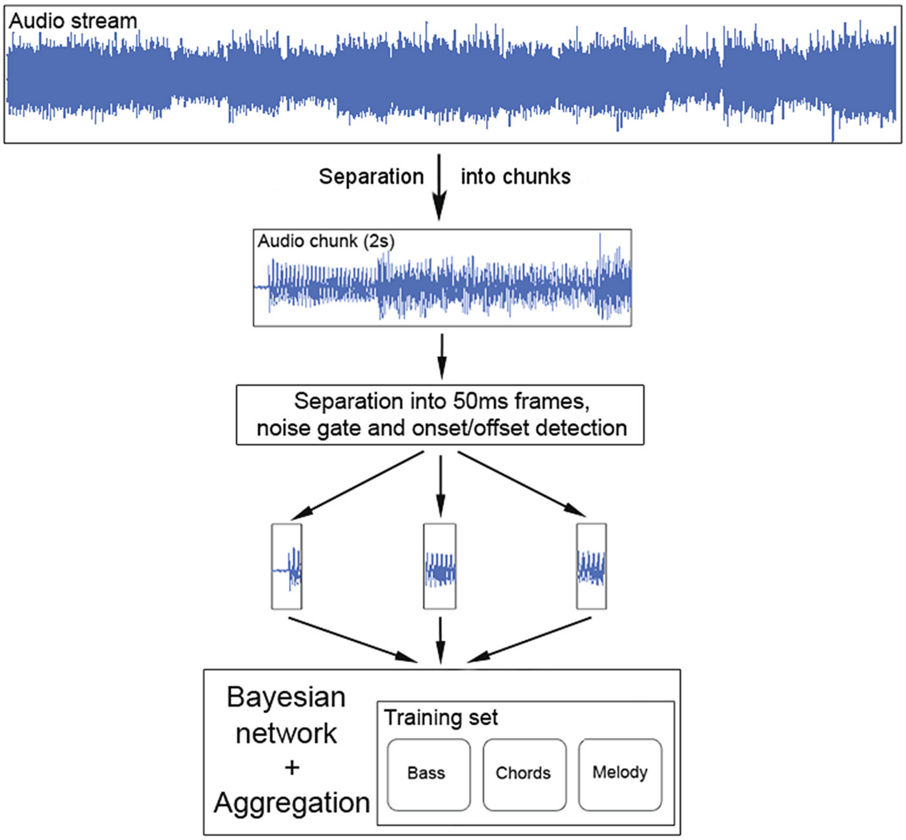
Audio stream
Separation into chunks
Audio chunk (2s)
Separation into 50ms frames, noise gate and onset/offset detection
Bayesian network + Aggregation
Training set
Bass
Chords
Melody

Fig. 3. General scheme of the classification algorithm.

apply a winner-takes-all strategy to identify the mode of the chunk. For interactive musical applications, a typical chunk size is one bar at reasonable tempos (1 s at 240 bpm, 4 s at 60 bpm). For extremely fast tempos, chunks of more than one bar should be considered to avoid performance decrease due to small numbers of frames.

4 Results

This section describes various evaluations of the classifier (including aggregation), highlighting the impact of using different guitars on classification robustness.

4.1 Evaluation on a One-Song Training Set

First, we train the classifier on one single song, "The Days of Wine and Roses", taken from the Godin guitar (*God*) subset. Then, we test it on the six other songs, for each guitar subset, with a chunk duration of 1.5 sec. (the duration of one 4/4 bar at 160 bpm). The results are displayed on Table 1.

Table 1. Classification performance obtained over six songs, for various guitar models

Guitar dataset	*God*	*Cort*	*Ovat*	*Nyl*
Mean F-measure	**0.96**	**0.941**	**0.854**	**0.839**

For the guitar subsets *Cort* and *God*, the results are slightly better than the preliminary ones obtained with the Bayesian network without the aggregation step (0.87 average F-measure). However, the classification results are poor for the *Ovat* and *Nyl* guitar subsets.

4.2 Evaluation with the Use of Larger Training Sets

To improve the performance, we increase the size of the training set: we train and evaluate the classifier with the leave-one-out procedure. Hence, each training set contains now six songs. To study the influence of the guitar type used for training and testing, we repeat this procedure for each guitar subset. The results are displayed on Table 2.

Table 2. Classification performance obtained with the leave-one-out procedure on the whole dataset. The first number is the minimum F-measure over the six tested songs, the second is the average F-measure

		Tested guitar dataset			
		God	*Cort*	*Ovat*	*Nyl*
Training guitar dataset	*God*	0.956 0.971	0.933 0.968	0.654 0.90	0.71 0.901
	Cort	0.94.3 0.963	0.974 0. 984	0.753 0.94	0.922 0.972
	Ovat	0.885 0.92	0.917 0.955	0.964 0.978	0.956 0.978
	Nyl	0.92 0.943	0.961 0.975	0.961 0.975	0.981 0.992
	Avg. F-measure	**0.949**	**0.971**	**0.948**	**0.961**

These results show that while a larger training set increases the accuracy, the classification performance depends on the guitar used for training and testing: more specifically, the pairs *God*/*Cort* and *Ovat*/*Nyl* seem to give better results when used together (one for training and the other for testing). This can be explained by the fact that the guitars used to record *Ovat* and *Nyl* subsets produce more high-frequency content than the other ones: a feature such as spectral spread is sensitive to timbre.

4.3 Evaluation with a Mixed Training Set

In order to make the classifier more independent of the guitar type, or more generally of timbral variations, we pick tracks from each of the four subsets to build a new training set. We use the recordings of *The Days of Wine and Roses* and *Ladybird* from each subset to train the classifier and test the performance on the five remaining tracks. Results are shown on Table 3.

Table 3. Classification performance obtained with the use of a mixed training set. We compare the minimal F-measures over the four guitars in order to evaluate the timbral sensitivity of the classifier.

		Tested guitar dataset				
		God	*Cort*	*Ovat*	*Nyl*	Min. F-measure
Test song	Bluesette	0.971	0.988	0.989	0.995	**0.971**
	Nardis	0.941	0.966	0.973	0.99	**0.941**
	Ornithology	0.99	0.988	0.99	0.999	**0.988**
	Solar	0.977	0.965	0.985	0.997	**0.965**
	Tune Up	0.968	0.984	0.962	0.952	**0.952**
	Min. F-measure	**0.968**	**0.978**	**0.98**	**0.987**	**0.968**

Here, we can see that the use of a mixed training set, containing two songs (or a total 31 min of audio), increases the overall performance. We evaluated the classifier with larger training sets, but larger sets do not increase classification accuracy in a significant way. This last training set will be used in the rest of this article.

4.4 Influence of the Analysis Window Length

Since the algorithm includes an aggregation step, we can assume that the accuracy of the classifier depends on the length of the analyzed audio chunks. Figure 4 displays the classification performance obtained over the five tracks which are not included in the training set, for various analysis windows. As a comparison, we added, for each guitar subset, the F-measures obtained without performing the aggregation over the 50 ms frames.

We can see that the classification improves when increasing the analysis window length, reaching a plateau at about .98.

4.5 Real-Time

Since the algorithm consists in feature extraction and simple Bayesian classification, the overall complexity of the algorithm is linear with the analyzed audio window length (other computation such as the aggregation is negligible). The average classification

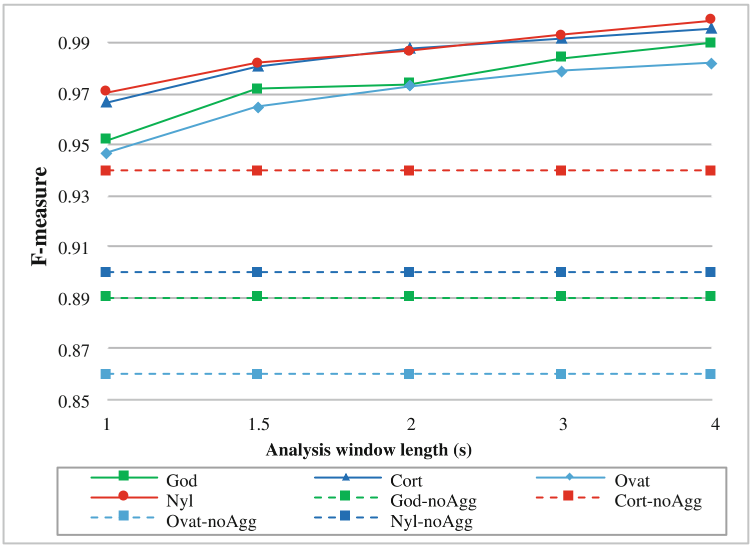

Fig. 4. Classification performance obtained with different analysis window lengths. The series followed by the mention *noAgg (dotted lines)* show the results obtained without the aggregation step.

CPU is 2 % of real-time, with a Java implementation running on an Intel i7 2.67 GHz quad-core, Windows laptop (e.g. the experienced latency obtained for the analysis of a 4/4 bar at 120 bpm is 20 ms). This clearly enables interactive musical applications on commonly available hardware.

4.6 Comparison with MIDI-Based Classification

We compared our algorithm with the performance of a MIDI-based mode classifier described in [14], using the Godin guitar subset (this guitar has a MIDI output). The MIDI-based classifier trains a Support Vector Machine classifier on 8 MIDI features, related to pitch, duration, velocity, and more advanced ones, aggregated over one bar. This classifier is been trained on the song *Bluesette*, and tested on the six other songs. In order to work with the same experimental settings, we adapted the analysis window of our audio-based algorithm on each song, to match the length of a bar. The results are displayed in Tables 4 and 5.

The results we obtain are still reasonable, but weaker than with the preceding training sets. This is due to the fact that audio analysis requires larger training sets than MIDI to reach the same performance. To illustrate this point, we increase slightly our training set and train our classifier with two songs: *Bluesette* and *The Days of Wine and Roses*. We repeat the testing procedure on the five remaining tracks. The confusion matrix is displayed on Table 6.

These results show that our method provides results which are comparable to the ones obtained with the MIDI output of the guitar. This result enables us to integrate our algorithm in actual interactive live applications, without any MIDI support.

Table 4. Classification results obtained with the MIDI-based SVM classifier

		Predicted class			
		Bass	**Chords**	**Melody**	**F-measure**
Actual class	**Bass**	**1314**	6	2	0.98
	Chords	24	**1294**	13	0.97
	Melody	1	12	**1318**	0.99

Table 5. Classification results obtained with our classifier

		Predicted class			
		Bass	**Chords**	**Melody**	**F-measure**
Actual class	**Bass**	**1118**	1	3	0.947
	Chords	85	**1017**	11	0.936
	Melody	25	31	**1065**	0.968

Table 6. Classification results obtained with our classifier, with a larger training set

		Predicted class			
		Bass	**Chords**	**Melody**	**F-measure**
Actual class	**Bass**	**811**	30	10	0.965
	Chords	0	**852**	3	0.969
	Melody	18	21	**817**	0.969

5 Interactive Applications

VirtualBand [14] is an interactive musical engine that enables one or several musicians to control virtual instruments. The virtual instruments interact with the users but also with one another in real time, thus creating possibly complex interactive performance. VirtualBand allows to build bebop-style improvisations by constraining the virtual instruments to follow a predefined harmonic sequence.

VirtualBand was used to create a *reflexive loop pedal* for guitar. This application is based on the mode classifier described here. The goal is to allow guitarists to play a trio with themselves.

Two virtual instruments are instantiated by the system: a *virtual bass player* and a *virtual chord player*. The virtual instruments are silent at the beginning of the session. When the user starts improvising on the predefined chord sequence, its musical production is analyzed by the classifier and used to feed the virtual instruments: when the user plays a bar of bass, the corresponding signal is stored in an audio database accessible to the virtual bass player. Similarly, when the user plays a bar of chords, it is stored in a database that is accessible to the virtual chord player. Each bar in the virtual instrument's databases are labeled with the harmony specified by the predefined chord sequence.

Once the virtual databases are not empty anymore, the virtual instruments start to play along with the user. They follow the "two-other-guys" principle: when the user plays melody, the two virtual instruments play, thus producing the output of a typical jazz guitar trio; when the user plays bass, the virtual chord player plays along; and when the user plays chords, the virtual bass player plays along.

VirtualBand features a real-time pitch-shifting algorithm that enables the virtual instruments to transpose the recorded audio chunks in the virtual databases. For instance, if the user played a bar of chords in C7 (the harmony specified by the chord sequence at a certain point in time), the virtual chord player will be able to play back this bar, after transposition, when the chord sequence specifies a harmony of D7. This is particularly useful if the user never played chords on a D7 harmony. This mechanism reduces the *feeding* phase, i.e., the phase during which the user is playing chords and bass lines to feed the virtual instruments.

VirtualBand also uses harmonic substitution rules enabling the virtual instruments to play back music in a context it was not recorded in. For instance, a bar of chords that was recorded in a C major context may be played back on an A minor context, using the "relative minor" substitution rule, which states that A minor and C major are somehow musically equivalent.

The combination of the transposition and substitution mechanisms allows to reduce the feeding phase to a minimum, typically a few bars for a jazz standard, thus creating lively and entertaining performance.

A detailed version of this example is given in Pachet et al. [14]. This application allows the user to control the musical process while not using physical control devices such as loop pedals, which would interfere with his creative flow. Hybrid modes, such as the "bossa nova" mode (see the next section) can be added to this setup to enable more sophisticated interactions, thanks to the addition of a new virtual instrument.

Another application of the mode classifier, implemented in the VirtualBand engine, is to automatically process the input sound according to the playing mode. Depending on the current playing mode, various audio effects are applied to the audio signal. For instance, the system can add a specific reverberation effect to monophonic melody, tube distortion to chords, and, say, apply dynamic compression and enhance the low end of the bass. The effect chain is applied with a latency that corresponds to the chunk size needed to perform mode classification, *i.e.*, about 1 s.

6 Discussion

We have shown that YIN features, that represent half of our feature set, are efficient to classify guitar playing modes: our classifier is accurate, robust to variations in guitar type, and able to cope with real-time computational constraints, thanks to a small feature set. We also showed that although the accuracy depends on the size of the analyzed audio, this classifier can be used with realistic window sizes. Three points can be improved, to further extend its applicability.

6.1 Features

The method raises issues when dealing with long chord decays (say, more than 5 sec.), when only one note keeps sounding. This case falls off the boundaries of our algorithm and feature set. One solution would be to add a robust onset detector to our algorithm, and restrict the mode computation on the first seconds that follow an onset (we did no implement such a solution).

Another limitation comes from the feature set: we work with a feature set that answers a specific problem, but it may not be efficient to distinguish efficiently yet other playing modes, such as strums or octaves. The algorithm is also somewhat specific to the guitar: the YIN inharmonicity factor may not behave the same with less harmonic instruments, such as the piano.

6.2 Hybrid Playing Modes

In our method, we perform an aggregation step because the frame classifier alone is not accurate enough. Nevertheless, it provides a good hint about the rate of chords, melody and bass, within audio chunks that contain a mixture of different playing modes. For instance, we can consider an extra "bossa nova" playing mode which consists in alternative bass/chords patterns. In order to recognize such a mode, we add an extra rule to the aggregation step of the algorithm: before applying the winner-takes-all strategy to our frames classes, we compute the weight of each class, without taking the class probabilities into account, and we express it in absolute percentage. Then, we consider the bass and chords weights: if they are both greater than, say, 20 % and lower than 80 %, then we can consider that the chunk belongs to the "Bossa nova" class. Such a rule could be also implemented in a classifier, so that the process is entirely automatic. An example of such a hybrid mode is displayed in Fig. 5.

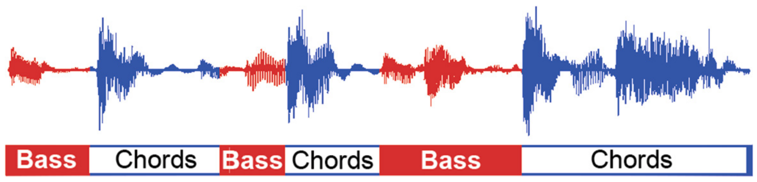

Fig. 5. Identification of bass and chord parts in a bossa nova guitar audio signal

Although the frame classifier does not provide an accurate weight for each class within a chunk, the ability to detect when the musician is performing this hybrid playing mode brings new possibilities for building interactive applications. The pattern displayed on Fig. 5 is correctly detected, however it represents only a particular case of the bossa nova playing mode, in which bass and chords do not overlap. In the (frequent) case when they overlap, the classifier performance drops sharply.

6.3 Absolute Aggregation *vs*. Time Series

In this paper, we use the simple winner-take-all strategy to aggregate the 50 ms frames over the entire analysis window. This method does not take into account the time-series nature of a musical audio signal. For instance, guitar players sometimes use low-pitched notes in their melodic improvisation, and conversely, play walking bass with high-pitched notes. With our window-based scheme, the classifier uses the YIN pitch as a strong hint to distinguish the melody from the bass. As a consequence, the user might be surprised by some results for those notes with intermediary pitches (*e.g.*, in the range C3–E3) also, since there is no high-level musical analysis of the currently played phrase. The evolution and continuity between the different features values extracted within an audio chunk could be evaluated over time, leading to a smarter way to process these frames. We assume that a classifier that exploits such knowledge would be more accurate and could efficiently identify more sophisticated playing modes such as arpeggios, muted notes strumming, and, in general, playing modes based on longer temporal patterns.

7 Conclusion

We have presented a simple and fast method to classify in real time guitar audio signals into three basic playing modes: chords, melody, and bass. Our method is timbre-independent and proves accurate with four different types of guitar.

The good performance of our classifier paves the way for interactive musical systems allowing users to improvise with virtual copies of themselves to form, e.g., a guitar trio. Future designs of the algorithm, in particular taking into account the continuity between frame analyses, will be able to distinguish more complex playing modes, such as the bossa nova bass/chord mixing, thereby enabling the application to other musical genres.

Automatic playing mode classification brings a lot of potential for designing smarter augmented instruments. Interestingly, developing subtler and subtler playing mode classifiers, from polyphony-based detection as we presented here to the identification of player-specific patterns (Montgomery's octaves, Benson's licks or Van Halen's fast harmonic arpeggios), infringes on the emerging domain of style modeling.

Acknowledgments. This research is conducted within the Flow Machines project which received funding from the European Research Council under the European Union's Seventh Framework Programme (FP/2007-2013)/ERC Grant Agreement n. 291156.

References

1. Abesser, J., Lukashevich, H.M., Schuller, G.: Feature-based extraction of plucking and expression styles of the electric bass guitar. In: Proceedings of International Conference on Acoustics, Speech and Signal Processing, Dallas, pp. 2290–2293 (2010)
2. Abesser, J.: Automatic string detection for bass guitar and electric guitar. In: Proceedings of the 9th International Symposium on Computer Music Modelling and Retrieval, London, pp. 567–582 (2012)

3. Barbancho, I., Tardon, L.J., Sammartino, S., Barbancho, A.M.: Inharmonicity-based method for the automatic generation of guitar tablature. IEEE Trans. Audio Speech Lang. Process. **20**(6), 1857–1868 (2012)
4. De Cheveigné, A., Kawahara, H.: YIN, a fundamental frequency estimator for speech and music. J. Acoust. Soc. Am. **111**(4), 1917–1931 (2002)
5. Dixon, S.: Onset detection revisited. In: Proceedings of the 9th International Conference on Digital Audio Effects, Montreal, pp. 113–137 (2006)
6. Fohl, W., Turkalj, I., Meisel, A.: A Feature relevance study for guitar tone classification. In: Proceedings of the 13th International Study for Music Information Retrieval Conference, Porto (2012)
7. Guyon, I., Elisseef, A.: An Introduction to variable and feature selection. J. Mach. Learn. Res. **3**, 1157–1182 (2003). (http://www.jmlr.org)
8. Hartquist, J.: Real-time musical analysis of polyphonic guitar audio. Ph.D. thesis, California Polytechnic State University (2012)
9. Holmes, G., Donkin, A., Witten, I.H.: Weka: a machine learning workbench. In: IEEE Proceedings of the Second Australian and New Zealand Conference on Intelligent Information Systems, pp. 357–361 (1994)
10. Klapuri, A., Davy, M.: Signal Processing Methods for Music Transcription. Springer, New York (2006)
11. Lachambre, H., André-Obrecht, R., Pinquier J.: Monophony vs polyphony: a new method based on Weibull bivariate models. In: Proceedings of the 7th International Workshop on Content-Based Multimedia Indexing, Chania, pp. 68–72 (2009)
12. Lähdeoja, O., Reboursière, L., Drugman, T., Dupont, S., Picard-Limpens, C., Riche, N.: Détection des Techniques de Jeu de la Guitare. In: Actes des Journées d'Informatique Musicale, Mons, pp. 47–53 (2012)
13. Özaslan, T.H., Guaus, E., Palacios, E., Arcos, J.L.: Attack based articulation analysis of nylon string guitar. In: Proceedings of the 7th International Symposium on Computer Music Modeling and Retrieval, Malaga, pp. 285–297 (2010)
14. Pachet, F., Roy, P., Moreira, J., d'Inverno, M.: Reflexive loopers for solo musical improvisation. In: Proceedings of International Conference on Human Factors in Computing Systems (CHI), Paris, pp. 2205–2208 (2013). (Best paper honorable mention award)
15. Reboursière, L, Lähdeoja, O., Drugman, T., Dupont, S., Picard-Limpens, C., Riche, N.: Left and right-hand guitar playing techniques detection. In: Proceedings of New Interfaces for Musical Expression, Ann Arbor, pp. 30–34 (2012)
16. Ryynänen, M.P., Klapuri, A.P.: Automatic transcription of melody, bass line, and chords in polyphonic music. Comput. Music J. **32**(3), 72–86 (2008). (MIT Press)
17. Sinyor, E., Mckay, C., Mcennis, D., Fujinaga, I.: Beatbox classification using ACE. In: Proceedings of the 6th International Conference on Music Information Retrieval, London, pp. 672–675 (2005)
18. Stein, M., Abesser, J., Dittmar, C., Schuller, G.: Automatic detection of audio effects in guitar and bass recordings. In: Proceedings of the 128th Audio Engineering Society Convention, London (2010)
19. Stowell, D., Plumbley, M.D.: Delayed Decision-making in real-time beatbox percussion classification. J. New Music Res. **39**(3), 203–213 (2010)
20. Tindale, A., Kapur, A., Tzanetakis, G., Fujinaga, I.: Retrieval of percussion gestures using timbre classification techniques. In: Proceedings of the 5th International Conference on Music Information Retrieval, Barcelona, pp. 541–544 (2004)