Real-Time Transcription of Guitar Music

Michael Groble
mg2467

***Abstract*—MIDI guitar synthesizers currently exist to transform the sound coming from the vibrating guitar strings to MIDI events. These achieve high accuracy and real-time performance through dedicated hardware. In particular, special pickups are used to generate a single signal for each string and special processing units convert the signals to MIDI events. This paper explores the ability to achieve real-time transcription using a standard pickup and a desktop computer instead of the specialized hardware. While we do not achieve accuracy comparable to specialized hardware, we do achieve frame-level accuracy in the range of published results with real-time speed.**

I. Introduction

The Music Information Retrieval Evaluation eXchange (MIREX) [3] includes an evaluation task called Multiple Fundamental Frequency Estimation & Tracking task. The goal of this task is to be able to process an audio signal containing multiple simultaneous notes to extract a transcription of those notes including pitch contours. Guitar to MIDI transcription can be thought of as an instance of this type of problem. In traditional systems, such guitars employ special pickups which generate a separate signal for each string. The signal processing in these cases is simplified since there is only one note playing on one string at any instance, but it is still difficult since results need to be generated in real-time and need to account for variations in how or where notes are picked.

This paper explores the ability to perform guitar to MIDI transcription using a standard pickup, meaning a monophonic audio stream consisting of all notes played simultaneously across all the strings of the guitar. This task is very similar to the Multiple Fundamental Frequency Estimation & Tracking task, but has some specific characteristics of its own. Simplifying characteristics are that the audio signal is coming from a single instrument (in other words a single timbre) and that a guitar has a limited pitch range relative to other instruments. Also, for purposes of this paper, string bending is not considered which means the desired output is MIDI pitch values, not arbitrary frequency values. It also differs in that the MIREX task does not consider real-time performance.

The MIREX task is divided into two parts, a frame-by-frame one which scores the predicted frequencies vs. actual frequencies for each individual 10 msec segment of audio and a note contour one which scores note onsets, offsets and pitches between the predicted and actual values. This project did not progress far enough to predict note onsets and offsets so the results are limited to frame-by-frame analysis.

The MIREX 2007 results contain references to many different types of algorithms currently in use for this task. The approach most similar in spirit to this paper is [4] which treats the task as a classification one. Also similar is the method introduced by [2] to iteratively determine simultaneous notes. This paper introduces novel ways of training pitch classification models, computing feature vectors and scoring models against individual audio sample frames. These novelties produce an algorithm which achieves frame-level precision and recall numbers over 60% while maintaining real-time performance, requiring less than 34 msec to process 1 second of audio.

II. Corpus

The task of estimating notes at the frame level, and a full transcription, are both rather complicated with many different performance measures which may be of interest. In order to separate the complexity inherent in computing such measures for real musical passages, two separate corpora are used in this analysis. The first is a non-musical corpus of notes with known polyphony. This corpus provides a simple baseline for evaluating a wide range of processing approaches. The second is a corpus of songs used to evaluate the algorithm in the full complexity of a real setting.

A. Random Note Corpus

Logic Studio is a software package for audio composition and performance available on Mac OS X. It includes a sampling synthesizer called EXS24 and a number of sampled instruments. The random note corpus consists of audio clips generated from ten of the sampled guitars: “Acoustic Harmonics”, “Classical Acoustic Guitar”, “Clean Electric Guitar”, “Eighties Electric”, “Spanish Flamenco Guitar”, “Muted Electric Guitar”, “Roundback Acoustic Guitar”, “Steel String Acoustic”, “Sunburst Electric”, and “Twangy Electric”.

The sampled instruments contain multiple samples for each note at different playing volumes. Across all ten guitars and all volume levels, each note has 63 different samples. Two-second audio clips were synthesized for each of these 63 groups and for seven different tuning offsets (-15 cents, -10 cents, -5 cents, perfect tuning, +5 cents, +10 cents, +15 cents). These clips are 16 bit PCM at 44.1 kHz sampling rate. These clips were generated for 50 different notes, from B0 to C5 (this is in Logic notation, not MIDI notation, the low E string, for example, is E1 in Logic, or MIDI pitch 40). Finally, 2048 sample frames were extracted randomly within each of 10 different offset ranges into the two-second clip. The result is a corpus of 220,500 frames, each 46.44 msec long spanning a wide range of guitar types, note volumes, note tunings and note offsets.

In addition to the single note case, these samples were combined to generate 2-note and 3-note samples. In each case, the individual samples were combined with others from the same group (meaning same guitar and volume level), the

Table I
POLYPHONY DISTRIBUTION

Polyphony	Frames	Note-Frames
0	3,257	0
1	15,606	15,606
2	36,992	73,984
3	33,056	99,168
4	30,109	120,436
5	5,478	27,390
6	716	4,296
7	108	756
8	61	488
9	3	27
10	1	10
all	125,387	342,161

same offset range and the same tuning. The pitch values were chosen randomly and resulted in over 100,000 frames for each polyphony value.

B. Musical Corpus

Unlike [4], where a mechanically-driven piano was used to generate audio samples from MIDI files, there are not readily-available systems to mechanically play a real guitar from a MIDI file. Completely accurate transcription of real-audio recordings is very time-consuming to generate. Consequently, the musical audio samples were generated with the sampling synthesizers as above. The musical corpus consists of 45 sound files (16 bit PCM at 44.1 kHz as the prior corpus) synthesized from MIDI files containing fingerstyle guitar arrangements. Three of the files were captured from someone playing a guitar synthesizer, while the others are hand-sequenced. The captured files exhibit much more variation in timing and note velocities than the sequenced ones do. Table VIII summarizes the 45 files and indicates the sampled instrument used in each case. Table I describes the distribution of frames by the number of simultaneous notes in the frame.

These numbers are considering notes which sound at least 20% of the frame duration as notes in the frame. A note is considered to be sounding based solely on the note on events within the MIDI file. In other words, if the MIDI duration is 1 second, the note will count as sounding for roughly 22 consecutive frames, even if the volume of the sound dies to an inaudible level due to the decay of the sound over that 1 second.

Note also that unison intervals, two notes sounding the same pitch, are possible on the guitar, but are not considered in this analysis. If multiple notes ever simultaneously sound the same pitch, they are counted as a single note.

The MIDI files were found on the web and were modified slightly for this analysis. First, all pitch bends were removed. Second, the sampling synthesizer definitions treat certain velocity ranges as special commands, for instance to play a harmonic version of the note, to trill a note, or to include string noises after the note. The velocities were modified so that only the normal note played, as would be the case if the MIDI file were played through one of the non-sampling synthesizers.

III. PERFORMANCE METRICS

The performance metrics used to evaluate the random note corpus are very straight forward. In each case, a frame consists of a predefined number of notes. The predicted notes are compared to the true notes and any non-matching notes are counted as substitutions (sub_f). Again, since the random notes may include unison intervals, the actual metric compares substitution errors between distinct pitches. The substitution error is defined as the sum of substitutions over all frames divided by the sum of polyphony (distinct pitches, $poly_f$) in each frame.

$$E_{sub} = \frac{\sum_f sub_f}{\sum_f poly_f} = \frac{sub}{poly}$$

For the musical corpus, the algorithm needs to determine the number of notes rather than being given the true polyphony. In addition to substitutions, there is now the possibility of insertions (ins_f) and deletions (del_f). Insertions count the number of notes hypothesized by the algorithm greater than the true number of notes while deletions count the number of true notes greater than the number hypothesized. The MIREX task refers to these respectively as "false alarms" and "misses" and also references the quantities Acc and E_{tot} defined below.

$$\begin{aligned} E_{ins} &= \frac{\sum_f ins_f}{\sum_f poly_f} = \frac{ins}{poly} = E_{fa} \\ E_{del} &= \frac{\sum_f del_f}{\sum_f poly_f} = \frac{del}{poly} = E_{miss} \\ Acc &= \frac{match}{match + ins + del} \\ E_{tot} &= \frac{sub + \max(ins, del)}{poly} \end{aligned}$$

where $match_f$ is the number hypothesized notes that are correct in a given frame and $poly_f$ is the number of true distinct pitches in a frame. Finally, recall R and precision P can be defined in these terms by the following

$$\begin{aligned} R &= \frac{match}{match + sub + del} = \frac{match}{poly} \\ P &= \frac{match}{match + sub + ins} \end{aligned}$$

IV. FEATURES

The feature set was determined from analysis of E_{sub} on the single-note frames from the random note corpus. The features are based on the magnitudes of the short-term Fourier transform over the 2048 sample frames. The frame size was chosen to be 2048 based on a desire to have temporal accuracy no worse than the resulting 46 msec. Various shorter windows sizes were chosen to determine to see if the full frame would be required for the desired accuracy. Hann windows of 2048, 1024, 512 and 256 samples long were used. The 1024 sample window caused a 5-time increase in E_{sub} over the 2048 sample window so the feature calculations use the entire 2048 sample frame.

The next analysis was to determine how many frequency bins would be required to achieve the best results. The fewer number of features means the faster real-time processing, but potentially poorer performance. On the single-note data, performance leveled off above 3 kHz, for safety, I chose features up to about 4 kHz which means the magnitudes of the first 180 frequency bins from the STFT.

V. Note Prediction

As in [4], note prediction is treated as a classification problem, instead of a problem of estimating the fundamental frequencies. This limits the resulting algorithm to be applicable to only those performances with fixed pitches (i.e. no string bends). To do so, each pitch has a model associated with it. Frames are scored against these models to determine which pitches are present in the frame. In the analysis for both corpora, the model used to predict a frame from a particular guitar is generated from all other guitars excluding the one used to generate the frame. For the random note corpus, this results in a form of n-fold testing where the samples from one guitar are tested against a model learned from the samples of all other guitars.

[4] used an SVM to train a classifier. On a subset of the single-note frames SVMs gave a slightly better classification error, but the training process was too slow to be useful with 50 classes and over 200,000 samples. Scaled averaging gave slightly worse classification performance, but much better speed. With scaled averaging, the model for a particular pitch is generated by scaling each individual feature vector at that pitch to have a unit height, averaging the unit height vectors together, and normalizing the resulting average vector to have unit length. More formally, the model vector m_p for a specific pitch p is computed from samples x_i with pitch labels y_i as follows

$$\begin{aligned} a_p &= \sum_{j:y_j=p} \frac{x_j}{\|x_j\|_\infty} \\ m_p &= \frac{a_p}{\|a_p\|} \end{aligned}$$

A. Single Note

In the single note case, the problem is simply that of determining which pitch model is closest, or scores best compared to the frame. One approach from information retrieval is to use cosine similarity, the dot product of two unit-length feature vectors. This approach provides reasonable results on the single-note samples of the random note corpus, $E_{sub} = 2.2\%$. Slightly better results are obtained by summing the absolute difference between the unit-length feature vectors, resulting in $E_{sub} = 1.8\%$. Specifically, the predicted pitch p_{est} for a feature vector x is the one satisfying

$$p_{est} = \arg\min_p \left\| \frac{x}{\|x\|} - m_p \right\|_1 \quad (1)$$

Quite a number of other feature normalizations, and scoring functions were evaluated (including only comparing bins or ranges of the feature vector based on the pitch corresponding to m_p as in [1]), but none performed better than this.

B. Multiple Note

For multiple notes, the note prediction problem becomes more difficult. With the random note corpus, the number of notes was known before hand so the algorithm had a fixed number it is trying to predict. [2] describes an iterative method, where the single best scoring note is determined from the feature vector, then the feature vector is modified to "remove" the effects of that pitch. The remainder is then rescored until the desired number of notes have been predicted. With the non-linear scoring metric in (1), a slightly different approach is required. In particular, a base profile $base^i$ at iteration i stores the feature vector representing the pitches which have already been detected for the current frame. Pitches for iteration i are then scored as follows with the initial base set to a zero vector $base^0 = 0$.

$$\begin{aligned} score_p^i &= \left\| \frac{x}{\|x\|} - \frac{m_p + base^{i-1}}{\|m_p + base^{i-1}\|} \right\|_1 \\ p_{est}^i &= \arg\min_p score_p^i \\ base^i &= base^{i-1} + m_{p_{est}^i} \end{aligned}$$

Additionally, each iteration does not need to include models of just single pitches. As will be shown in Section VI, analysis from the random note corpus shows that all of the true notes show up at relatively shallow depth within the ranked score list from the very first iteration. The ranked list from the scoring therefore provides information for choosing multiple pitch combinations at an iteration, rather than just choosing a single pitch at each iteration. This has the benefit of reducing the number of scorings that need to be made in total.

In *combined* scoring, each iteration starts by computing a ranked list of pitch scores $score^i$. Note combinations are generated by combining highly ranked pitches and scoring the combination. If the best combination scores better than the best single-pitch score, the combined pitches are added to the estimated pitches and the corresponding combined feature vector added to the base.

Finally, for multiple note estimation, an algorithm needs to determine how many pitches to hypothesize. The approach described in this paper uses a sentinel pitch determined from observation of the empirical results. In particular, the algorithm often selects the highest pitch model for the note at the boundary between the correct pitches and incorrect ones. The intuition is that this pitch, being the highest, has the most number of individual feature vector elements close to 0. It has the fewest, most spread out peaks of all the pitches. Choosing this as a cutoff is effectively the same as choosing the cutoff to be the case when the remainder of the frame is closer to the zero vector than it is to all other pitches in the model. The hypothesis that the zero vector would perform this way has yet to be validated. For the results presented below, the iteration is terminated at the point where the algorithm determines the best scoring model is the one corresponding to the highest pitch value.

Table II
RANDOM NOTES, SINGLE RANKING

notes	E_{sub}	mean rank		
		note 1	note 2	note 3
1	1.8%	1.043	-	-
2	23.3%	1.161	5.079	-
3	41.8%	1.394	5.462	10.769

Table III
RANDOM NOTES, RE-RANKED

notes	E_{sub}		
	none	iterative	exhaustive
2	23.3%	11.9%	7.6%
3	41.8%	22.3%	13.5%

VI. RESULTS

A. Random Note Corpus

Table II shows the result for the random note corpus when the hypothesized notes are taken to be the top-n notes from a single scoring as in 1. The E_{sub} column shows the substitution error and the mean rank columns shows the rank averaged over all test samples required to find the notes matching the true ones. The mean ranks show that, even with multiple notes, the top-ranked note is frequently a true note in the frame.

Table III shows the results on this corpus for two different iterative approaches in addition to the baseline of a single ranking. In the iterative column, the error is shown for the case where a single note is predicted at each iteration and that note is taken as the top ranked one. The exhaustive column shows the case where the notes are ranked once and then combinations of notes from the ranked list are scored to find the best match. In the 2 note case, all combinations of note pairs between the top 3 notes and the top 10 notes are scored to determine the best pair. In the 3 note case, all triples between the top 3, top 10 and top 20 were tried. These combined notes were generated by adding the single-note model vectors and re-normalizing to a unit vector.

The results show that the iterative method cuts the error in half compared to the single ranking while the exhaustive method cuts the error to a third of the single ranking.

B. Musical Corpus

For the musical corpus, two different approaches are reported. The first case is the iterative method where one note is predicted at each iteration. The second case is a combined approach where at each iteration, all notes are scored. Pairs are generated from the ranked list, again using depths of top 3 and top 10, and the pairs are scored. In this case, in addition to the single-note models, training data was used to generate note-pair models. Feature vectors for note combinations are not the same as simply adding the feature vectors together. The pair data provides a more accurate profile than the individual ones, although does not accommodate different note volume levels as the single iterative case can.

The results are tabulated separately for frames based on the amount of polyphony within them (again, defined as notes sounding for more than 20% of the frame duration). Table IV shows the single iterative results while Table V shows the combined iterative results. As can be seen in the summary, the main difference between these two approaches is that the single iterative method provides better precision, making extremely few insertion errors, while the combined approach provides better recall at the expense of worse precision. The recall numbers of the combined approach really show benefit in the frames with high polyphony while the precision numbers of the combined approach show the biggest detriment in the frames with low polyphony. Insertion errors are larger for the combined approach in these low polyphony frames also.

Table IV
MUSICAL NOTES, ITERATIVE

Poly	P	R	Acc	E_{tot}	E_{sub}	E_{ins}	E_{del}
1	0.85	0.88	0.78	0.15	0.01	0.14	0.11
2	0.83	0.73	0.72	0.27	0.06	0.08	0.21
3	0.71	0.52	0.61	0.48	0.19	0.03	0.30
4	0.64	0.41	0.52	0.59	0.23	0.01	0.37
5	0.56	0.32	0.42	0.68	0.24	0.01	0.43
6	0.51	0.26	0.34	0.74	0.25	0.00	0.49
7	0.46	0.21	0.28	0.79	0.25	0.00	0.54
8	0.48	0.23	0.30	0.77	0.24	0.00	0.53
9	0.64	0.26	0.30	0.74	0.15	0.00	0.59
10	0.75	0.30	0.33	0.70	0.10	0.00	0.60
all	0.71	0.52	0.60	0.48	0.17	0.04	0.31

Table V
MUSICAL NOTES, COMBINED

Poly	P	R	Acc	E_{tot}	E_{sub}	E_{ins}	E_{del}
1	0.75	0.88	0.69	0.29	0.01	0.28	0.11
2	0.76	0.79	0.68	0.25	0.05	0.20	0.16
3	0.64	0.61	0.62	0.39	0.18	0.16	0.21
4	0.60	0.52	0.64	0.48	0.27	0.08	0.22
5	0.54	0.43	0.58	0.57	0.31	0.06	0.26
6	0.47	0.35	0.58	0.65	0.40	0.00	0.26
7	0.39	0.27	0.46	0.73	0.42	0.00	0.31
8	0.43	0.26	0.41	0.74	0.35	0.00	0.39
9	0.69	0.41	0.50	0.59	0.19	0.00	0.41
10	0.50	0.30	0.43	0.70	0.30	0.00	0.40
all	0.65	0.61	0.64	0.39	0.19	0.14	0.20

The results show quite a wide range of values across the different songs. Table VI shows some individual song details from the combined case. The table is ranked by decreasing value of precision. The group at the top shows the 5 highest precision songs, the group on the bottom shows the 5 lowest precision songs and the group in the middle shows the three songs which have MIDI files captured from someone really playing a guitar. These captured performances happened to fall in the middle of the pack.

As the table show, the results vary over a large range, from greater than 80% precision in the best case to under 40% in the worst. The last columns in the table attempt to explain some of the differences. The g column describes whether the song was synthesized using either an acoustic guitar (A) or an electric one (E). The $poly_{avg}$ column describes the polyphony level averaged across all frames in the song and the dur_{avg} column describes the note duration in seconds averaged over every note in the song.

While a clear-cut explanation is not evident, some generalities appear. In particular, the songs with lower polyphony,

Table VII
RAGPICKIN ROBUSTNESS

Test	P	R	Acc	E_{tot}	E_{sub}	E_{ins}	E_{del}
Plain	0.57	0.47	0.55	0.53	0.25	0.10	0.28
Detuned	0.56	0.47	0.56	0.53	0.26	0.11	0.26
Piano	0.59	0.63	0.60	0.44	0.20	0.24	0.17
Brass	0.47	0.65	0.57	0.74	0.29	0.45	0.06
Vibes	0.26	0.08	0.09	0.92	0.17	0.07	0.75

shorter notes and played with electric guitars (and hence have notes with longer sustain) do better than songs with higher polyphony, longer notes and played with shorter sustaining acoustic guitars. The captured songs being in the middle of the pack potentially means that the high variation in note timing and velocity are not significant performance factors for the algorithm, however this sample size is possibly too small to be of statistical significance.

C. Robustness Analysis

As a final experiment, the ragpickin song was used for model robustness testing. This song had a MIDI file captured from someone playing a guitar. Further, it had each separate string as a separate track in the file. The first experiment was detuning the strings as they would be on a real guitar. The specific detuning used in the experiment was: low E, -5 cents; A, +1 cents; D, -6 cents; G, +12 cents; B, -4 cents; high E, +8 cents.

The other model robustness experiment was to synthesize the song using other instruments. In particular using a piano (the "Steinway Grand Piano" instrument in the EXS24 sampling synthesizer), brass ("Trumpets" for the high notes and "Tuba Solo" for the low), and vibraphone ("Vibraphone"). These songs were then estimated using the same guitar model used in the default case for this song (the Clean Electric Guitar model, meaning the model from all other guitars excluding this one).

Table VII show the results. There is virtually no difference between the plain and detuned versions. Somewhat surprising, the piano and brass versions of the song performed better than the guitar version of the song, even though the model used was the corresponding guitar model. Listening to the audio samples, the piano has more sustain and the brass significantly more sustain than the guitar. Both show much improved recall over the guitar baseline which is potentially due to this increased sustain. Precision improves slightly for the piano, but decreases for brass. This potentially is explained by the difference in harmonic profile between the string instruments and the wind ones. Finally, the audio of the vibraphone version clearly indicates the different harmonic profile of the hammered notes as compared to the guitar. The results show very clearly the inadequacy of the guitar model to predict vibraphone notes.

D. Real-Time Performance

These results were generated by a stand alone application which reads in the audio file, the model files and writes out the predicted notes on a frame-by-frame basis. The single-note model is learned in time that is linear in the number of samples and has size that is linear in the number of pitches. The single note model was less than 40 KB in size (binary, uncompressed) while the pair-note model was less than 1 MB in size.

The tests were run on a 2.8 GHz dual Quad-Core Xeon Mac Pro with a single-threaded application. Including the overhead of reading all the music and model files and writing the prediction outputs, the single iterative case took 109 seconds to process the roughly 97 minutes of audio while the combined case took less than 198 seconds. This translates to an average of 0.9 msec for the iterative case and 1.6 msec for the combined case to process each 46.44 msec frame of audio.

The results show the resulting algorithm is completely adequate for real-time use, and the nature of the algorithm supports parallelization to further reduce latency if needed.

VII. CONCLUSION

From the results so far, specialized guitar to MIDI transcription hardware does not have anything to fear from standard pickup approaches. The isolation of a single note per audio channel is a significant advantage. What is promising is that the results of this paper show it is possible to obtain reasonable results at real-time processing speeds.

The results also show how much variation exists in performance of these systems to given song segments. The MIREX 2007 results show a similar wide variation even in the best performing algorithm. It is encouraging to see that while the best speed results from MIREX is 271 seconds of processing time for 840 seconds of audio, the algorithms described in this paper would process that same amount of audio in less than 30 seconds (although this is not apples-to-apples comparison since the computer specs differ).

It would be very interesting to see how this approach performs on the MIREX corpus. Since the performance on the the musical corpus in this paper improves when the audio is generated from pianos, even when the model is learned from a guitar, it may be that the performance similarly improves on the MIREX corpus. Alternately, the specifics of the songs cause quite a variability in the results and it may be that there are fundamentally "harder" segments in the MIREX corpus. Finally, the MIREX corpus truth values are determined by hand annotation of when particular note amplitudes decay lower than a threshold. The corpus used in this paper does no such compensation for decay and assumes (incorrectly at times) that a note is always at a detectable amplitude over the entire notated duration from the MIDI file. This provides another potential for better performance on the MIREX corpus. Finally, another uncertainty comes from the fact that the MIREX site does also not describe how notes which last a fragment of a frame are counted. [4] provides a corpus, but unfortunately the audio samples are not at the 44 kHz sampling rate required by the current implementation. Application of the approach in this paper to the MIREX corpus would potentially require training different models than just the existing guitar ones and would require creating pitch models over a wider range of pitches than has been done for the guitar analysis.

Table VI
MUSICAL NOTES, COMBINED, SONG DETAIL

Song	P	R	Acc	E_{tot}	E_{sub}	E_{ins}	E_{del}	g	$poly_{avg}$	dur_{avg}
cavatina	0.82	0.52	0.52	0.48	0.06	0.06	0.42	E	2.32	0.55
blakerg1	0.81	0.81	0.78	0.19	0.07	0.12	0.12	E	2.36	0.33
sweet_geogia_brown	0.79	0.85	0.78	0.22	0.06	0.16	0.08	E	2.59	0.44
watermelonman	0.79	0.84	0.79	0.23	0.08	0.14	0.08	E	2.40	0.63
two_for_the_road	0.78	0.78	0.78	0.22	0.11	0.11	0.11	E	3.24	0.81
lazybird	0.59	0.46	0.52	0.54	0.22	0.11	0.32	E	2.49	0.47
ragpickin	0.57	0.47	0.55	0.53	0.25	0.10	0.28	E	2.95	0.46
porkpie	0.56	0.54	0.59	0.46	0.25	0.17	0.20	E	2.21	0.38
oh_lady_be_good	0.46	0.44	0.53	0.56	0.34	0.16	0.22	A	3.02	0.57
blueingreen	0.44	0.40	0.63	0.60	0.44	0.07	0.16	E	3.76	1.15
on_the_sunny_side	0.43	0.35	0.47	0.65	0.36	0.11	0.29	A	3.22	0.63
atico	0.43	0.42	0.50	0.58	0.36	0.20	0.22	A	2.29	0.40
roundmidnight	0.37	0.26	0.36	0.74	0.36	0.08	0.38	A	2.88	1.01

REFERENCES

[1] Anssi Klapuri. Multiple fundamental frequency estimation by summing harmonic amplitudes. In *7th International Conference on Music Information Retrieval*, pages 216–221, Victoria, Canada, 2006.

[2] A.P. Klapuri. Multiple fundamental frequency estimation based on harmonicity and spectral smoothness. *Speech and Audio Processing, IEEE Transactions on*, 11(6):804–816, Nov. 2003.

[3] MIREX. Music Information Retrieval Evaluation eXchange, 2007. http://www.music-ir.org/mirex/2007.

[4] Graham E. Poliner and Daniel P. W. Ellis. A discriminative model for polyphonic piano transcription. *EURASIP J. Appl. Signal Process.*, 2007(1):154–154, 2007.

Table VIII
MUSICAL CORPUS DETAILS

name	frames	note-frames	type	guitar	avg polyphony
aint_misbehavin	1700	4600	sequenced	Classical Acoustic Guitar	2.83
amazinggrace	1806	4929	sequenced	Classical Acoustic Guitar	2.79
angeleye	2260	6559	sequenced	Classical Acoustic Guitar	2.96
atico	4142	9115	sequenced	Classical Acoustic Guitar	2.29
berniestune	1192	3061	sequenced	Classical Acoustic Guitar	2.64
blackorph	4162	12903	sequenced	Classical Acoustic Guitar	3.16
blakerg1	2517	5673	sequenced	Clean Electric Guitar	2.36
blueingreen	990	3652	sequenced	Clean Electric Guitar	3.76
californiadreamin	2431	7507	sequenced	Clean Electric Guitar	3.15
caravan	2396	7150	sequenced	Clean Electric Guitar	3.07
cavatina	4031	9031	sequenced	Clean Electric Guitar	2.32
clairdelune	2253	7208	sequenced	Clean Electric Guitar	3.26
desafinado	2799	10076	sequenced	Eighties Electric	3.71
dolphindance	3415	12544	sequenced	Eighties Electric	3.71
doxy	1604	4110	sequenced	Eighties Electric	2.69
dreamof	2618	7292	sequenced	Eighties Electric	2.87
eleanorigby	1962	6352	sequenced	Eighties Electric	3.33
girlfromipanema	2153	6572	sequenced	Spanish Flamenco Guitar	3.11
gpspells	3664	9591	sequenced	Spanish Flamenco Guitar	2.74
highmoon	1065	3143	sequenced	Spanish Flamenco Guitar	3.05
jordu	1808	4037	sequenced	Spanish Flamenco Guitar	2.31
lazybird	4725	11327	captured	Clean Electric Guitar	2.49
longwindingroad	2872	7904	sequenced	Spanish Flamenco Guitar	2.83
meditation	2792	9380	sequenced	Roundback Acoustic Guitar	3.44
missionimpossible	1384	3280	sequenced	Roundback Acoustic Guitar	2.47
moonlight	2962	9539	sequenced	Roundback Acoustic Guitar	3.27
mrjimi	1396	2730	sequenced	Roundback Acoustic Guitar	2.00
nardis	2467	7831	sequenced	Roundback Acoustic Guitar	3.21
oh_lady_be_good	3607	10623	sequenced	Steel String Acoustic	3.02
on_the_sunny_side	2196	6701	sequenced	Steel String Acoustic	3.22
peacheri	5311	13735	sequenced	Steel String Acoustic	2.68
petitefl	2587	7895	sequenced	Steel String Acoustic	3.11
porkpie	5529	11693	captured	Clean Electric Guitar	2.21
ragpickin	2368	6686	captured	Clean Electric Guitar	2.95
roundmidnight	2864	8096	sequenced	Steel String Acoustic	2.88
sails	3158	6845	sequenced	Sunburst Electric	2.28
someone2	2142	7058	sequenced	Sunburst Electric	3.44
sunfslow	6163	12834	sequenced	Sunburst Electric	2.19
sweet_geogia_brown	2515	6240	sequenced	Sunburst Electric	2.59
tarrega_capricho_arabe	6078	13874	sequenced	Sunburst Electric	2.34
teafortwo	1490	4049	sequenced	Twangy Electric	2.78
two_for_the_road	4126	12903	sequenced	Twangy Electric	3.24
watermelonman	1073	2515	sequenced	Twangy Electric	2.40
wave	2309	7128	sequenced	Twangy Electric	3.18
youarethesunshine	2305	6190	sequenced	Twangy Electric	2.78