A user configurable voice based controller for Midi synthesizer

A Prototype for Real-Time Performance.

Ahmed Nagi

MASTER THESIS UPF / 2010

UNIVERSITAT POMPEU FABRA

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my supervisor Hendrik Purwins, who never hesitated to give plenty of time to hear my clouded thoughts about the topics in this research and other matters alike. If it weren't for this guidance and support, this research would have never been completed.

I would like to thank Dr. Xavier Serra, whose hard work, has resulted in the existence and growth of the Music Technology Group, which in turn provided plenty of interesting researches that motivated and inspired this research.

I would also like to thank all my friends which I have met, and with whom we had memorable experiences: learning from one another, supporting each other during tough times, and laughing together during great time.

In particular, I would like to thank Laia Cagigal, Panos Papiotios (a.k.a Papitos) whose professional voice were used for testing purpose; and Felix Tutzer who helped me a great deal getting started with the Max/Msp environment.

Last but not least, I would like to thank my parents, whose ample love and support, have allowed me to pursue my interest and curiosity.

Abstract

Through the research we undertook for this master thesis, we developed a prototype, which we believe is the first step, towards building a voice-based controller for a midi synthesizer to be used in real-time. A key component of this research was the evaluation of the different onsets detection algorithms, and their applicability to the prototype developed herein. Another focus of the research was the smoothing of the fundamental frequency estimates, produced by the YIN algorithm. As a result of the research undertaken, we have found that an onset detection method based on the estimated note changes was more robust to background noise. Our prototype is developed in Max/Msp environment, which renders our prototype open to changes by its user, and make it useful with most software midi synthesizer.

Index or summary [Arial 16 points]

List of Figures

List of Tables

1. INTRODUCTION

Music is one of the oldest art form. It has is been enjoyed by the majority of mankind since the dawn of ages; played and performed by smaller number of people (artist), and composed and created by an even smaller group relative to the number of listener. There has been a lot of musical instrument used by artist to play music, and they varied depending on several categories: the style of music, range of frequency they produced, to name a couple among many. The learning process of most traditional instruments (e.g. bass, guitar) is time consuming, and becomes more difficult for adults to learn. This hinders the ability of an adult to express himself (herself) musically.

The advances in audio and digital signal processing have among other things, improved the recording and the electronic music creation process. It's no longer necessary to go to professional studio to obtain high quality recording, nor is it necessary to spend a lot of money on musical instrument, because most of them have been sampled with high accuracy and synthesized using software running of personal computing. Computer and musical interfaces has been an active area of research. One of its aims is to allow the user to interact with all the various parameters the software instrument, and express him/herself more freely.

The human voice is a rich sound generator, which is used in various communication methods: in speech, in entertainment (by singing an original song), and/or in imitation (by humming, or singing an cover song). It's the primary tool of expression for professional singer, and by amateur musician who use it as intermediary step when composing music using a Digital Audio Workstation, such as Ableton live[1]. As a result, there have been numerous attempt to use the human voice as input for the interactive tools developed to aid musicians, and non-musicians alike, to express themselves musically.

Some of the tools developed commercially in the past, which fall into the categories of audio-Midi converters are: Epinoisis' Digital Ear[2], e-Xpressor's MIDI Voicer 2.0[3] and WIDI[4]. However, as found in (Clarisse, Martens, Lesaffre, & Baets, 2002), most of these system fared poorly, when evaluated against system using non-commercial libraries for the task of melody extraction. For an in-depth discussion of potential merits and problems encountered when using the singing voice as input for sound synthesizer, we refers the reader to (Janer, 2008).

As a result, and motivated by our passion for music, yet our inability to play any musical instrument, we decided to pursue the topic of this thesis as a master research project, and develop a prototype that can be used, in real time, for live performance as well as a compositional tool.

Looking at the technical aspect, and the heart, of any system can use the human voice as an input signal to drive a midi synthesizer, we can identify two main tasks: an onset detection (i.e. the beginning of a note), and a pitch tracking algorithms. For the later

[1] http://www.ableton.com/
[2] http://www.digital-ear.com
[3] http://www.e-xpressor.com
[4] http://www.widisoft.com

task, we will be using the YIN algorithm (De Cheveigné & Kawahara, 2002) which is known for it's robustness, good results, and in turn have been used as a benchmark for other pitch tracking algorithms such as the one recently developed in (Ben Messaoud, Bouzid, & Ellouze, 2010). Moreover, given that an implementation of the YIN algorithm is publically available for Matlab[5] and for Max/MSP[6] - part of the FTM library[7], (Schnell & Schwarz, 2005), we thought it would ideal for our usage in the prototype described herein.

As for the onset detection algorithms, this topic has been an active area of research, which is reflected by fact that there has been a competition for the "best" algorithm at the MIREX conference. Given our application's need, we will comment on some on methods used only monophonic signal, and not for the polyphonic case.

In (J. P. Bello et al., 2005) the author gives a thorough review of the different algorithms published until then. These algorithm falls into two main classes: a class of methods based on the signal features, and another class of methods based on probabilities models. From the first class on model, we note the method described in (Schloss, 1985) which use the signal amplitude envelope to detect the onsets; the work presented in (Masri, 1996) which utilize the high frequency content of the signal; the method described in (Duxbury, Bello, Davies, & Sandler, 2003) which uses the phase information found in the spectrum of the signal, and finally the approach described in (Bello, Duxbury, Davies, & Sandler, 2004) which combine the energy and the phase information for onset detections. These methods will be described in greater details in the next section,

Methods belonging to the class based on probability models are based on the assumption that the signal can be described by some probability model. As implied, the success of the methods belonging to this class depends on the closeness of fit between the assumed model and the input signal. As the authors in (J. Bello et al., 2005) noted, the above mentioned class can be fited in this class under certain assumptions for probability models, and hence this class offers a broader framework to describe the various onset detection algorithms.

Since the publication of (Bello et al., 2005; Brossier, 2006), the main area of research have shifted toward to the modeling of onsets for polyphonic signal. From the fewer publication dealing with the monophonic case, we note the work done in (Toh, Zhang, & Wang, 2008) in which the authors combine the signal spectral features, along with a machine learning classification technique, Gaussian Mixture Model, to detects onsets for solo singing voice.

Since most the method described in (Bello et al., 2005; Brossier, 2006), are available in the Aubio library, a publicaly available, we will take the method described in those reference as a starting point to develop our prototype. Given the added complexity of the method described in (Toh, Zhang, & Wang, 2008), and the relatively small incremental benefit, we will defer the testing and implementation of their approach for a later time.

[5] Matlab is a numerical computing environment developed by MathWorks
[6] Max/Msp is a visual programming language and environment developed Cycling '74.
[7] A Max/Msp library developed at IRCAM, and available at http://ftm.ircam.fr/index.php/Download

We chose to develop our prototype in Max/Msp since it is design to be an interactive enviroment for musician and artist alike. It's used by researchers and academic to develope and test their algorithm as it's done in (Malt & Jourdan, 2008; Puckette, Apel, & Zicarelli, 1998) to name a couple of examples.

For reason that we will be clear by section 4, we chose to rely soley on the different algorithm offered by Max/Msp, along with the FTM library and CNMAT package for Max/Msp, both of which are publically available. The rest of the document is organized as follows. In section 2, we will review some of the theory used for our prototype. In section 3, we discuss our evaluation of the different onset detection approach tested for use in our prototype. In section 4, we discuss the implementation of our prototype and different post-processing techniques that we applied to improve the accuracy of the discussed algorithm. In section 5, we discuss the results of our implementations, and we conclude in section 6.

2. THEORY

When using the human voice as an input control signal, in the DSP domain, the first step is usually sampling the voice signal. Let *x*(*n*), be the sampled signal, where the paramters, *n* = 1,2,3 …, denote the index of sample. Given the fast sampling rate found in most audio card, e.g. 44,100 samples/second, as it often done in most DSP algorithms, we process a chunk of samples at any given time. Let *N* be the length of this chunk, commonly refered to as frame or window size. As we step through the different frames, of size *N*, we usually proceed in step of size, *H*, commonly referred to as hop size. In the next section, we will look at some of the possible way of calculating features of the input signal which inturn will be used for our onset detection algorithm

3.1 Common features of input signal used for Onset Detection

One of the primitive feature of the input signal which can be extracted, is the average energy over a frame of length *N*.

$$E = \frac{1}{N}\sum_{i=0}^{N} x_i^2$$

Another variant of this estimate uses the sum of the absolute value of the signal, instead of the sum of squares, however, both measure are used as energy estimates, over the frame of length *N*. As we estimate the energy for each processed frame, one can construct a time series of this energy estimate, the energy envelop, which in turn can be used to detect onsets. A more elaborate and descriptive features, of the input signal, can be based on the spectral analysis, which we will describe in the following subsection

A) Short-Time Fourier Transform and Spectral features.

The Fourier transform is a mathematical operation that transforms one complex-valued function of a real variable into another. That is to say that Fourier Transform maps a time series (eg audio signal) into the series of frequencies (their amplitudes and phases) that composed the time series. Since we want to use this transform on a sampled signal, we need to introduce and make use of the its discrete version, or Discrete Fourier Transform (DFT), given by:

$$X(k) = \sum_{i=0}^{N-1} x(i) \cdot e^{-j2\pi i/N}$$

where *j* is $\sqrt{-1}$. In the above, and subsequence formula, we are omitting the dependence of *X*(*k*), on the frame number, for clarity purpose.

One of the problem with using the DFT as describe in the above formula, is called spectral leakage, which occurs when the signal is not periodic in the interval of length *N*. To reduce the effect of such problem, the input signal is often multiplied by a window function. When a window function is applied, the above formula becomes:

$$X(k) = \sum_{i=0}^{N-1} w(i) \cdot x(i) \cdot e^{-j2\pi i/N}$$

where the window function, *w*(*i*), often evaluate to 0 at the boundaries. Common window functions used for signal analysis are the Hanning and the Hamming windows, just to name a couple.

Another potential problem when using of the assumption of the fourier analysis, is that the signal analyzed has to be stationary, meaning that the mean and average and the variance of the signal doesn't vary over time. In order to satisfy this limitation, the analysis window is often shorten, the value of *N*, to satisfy this constraint. Common value of *N* used in practice are, 256, 512, 1024, 2048 (powers of 2 for computational purpose, using the FFT method, which is an efficient way of computing the DFT).

Once we have computed the spectrum of a given frame, *l*, we can proceed to compute differents signal features. One such feature, is the high frequency content (HFC), of a given frame. To calculate this estimate, we use the following formula:

$$HFC(l) = \sum_{k=0}^{N-1} k \cdot |X(k)|$$

where *k* referes to frequency index, and |*X*(*k*)| denotes the magnitude of the complex number *X*(*k*). Similary to the energy envelope, this new HFC time series can be used to determine the onset time.

Another potential time series to be used, is the "distance" between successive frame of the spectrum, treating the value of the spectrum as points in an *N*-dimensional space. If we were to use the L_2 norm to compute the metric, then we will be using the following formula to compute this time series:

$$SD(l) = \sum_{k=0}^{N-1} (|X(l,k)| - |X(l-1,k)|)^2$$

where |*X*(*l*,*k*)| is the magnitude of the spectrum of frequency bin *k*, computed at frame *l*, and *SD*(*l*), is the estimate of the spectral difference, at time frame *l*. One modification to the above formula, as done in (Duxbury, Sandler, & Davies, 2002), is to rectify the fun before computing the norm. In such case, the above formula would become:

$$\tilde{SD}(l) = \sum_{k=0}^{N-1} [H(|X(l,k)| - |X(l-1,k)|)]^2$$

where *H*(*x*) = max(0, *x*). Such modification, the authors argue, is intended to emphazie onsets rathers than offset, in the detection stage.

A fourth onset detection that uses both the magnitude of the spectrum and it's phase, known as the complex domain method, computed using the following formula (Dixon, 2006):

$$CD(l) = \sum_{k=0}^{N-1} |X(l,k) - X_T(l,k)|$$

where,

$$X_T(l,k) = |X(l-1,k)|\, e^{\psi(l-1,k)+\psi'(l-1,k)}$$

and $\psi(l,k)$ is the phase of the spectrum for frequency bin *k*, at frame *l* such as:

$$X(l,k) = \ |X(l,k)|\, e^{\psi(l,k)}$$

The quantity $\psi'(l,k)$ is the phase derivative computed as $\psi'(l,k) = \psi(l,k) - \psi(l-1,k)$.

Once any of the above onset detection function has been computed, the next step would be to apply define and apply a threshold for deciding on the onsets. As discussed in (Bello et al., 2005; Brossier, 2006), there are two approaches for defining a threshold: a fixed threshold, and an adpative threshold.

Fixed threshold methods defines onsets as peaks where an onset function, any of the time series discribed above, exceeds a threshold δ. Although this approach can be succesful for signal with little dynamics, this wouldn't be applicable in the general case (for instance when the singing voice is used as input). For this reason, an apadtive threshold is usually required, where the threshold value δ, changes over time -the frame time, *l*, to be more precise. In (Brossier, 2006; Brossier, Bello, & Plumbley, 2004) the authors propose to use to the following adpative threshold function:

$$\delta_t(l) = \lambda \cdot median(D[l_m]) + \alpha \langle D[l_m] \rangle$$

with l_m belonding to the interval [*m*- *a*, *m*+ *b*] where $D[l_m]$ contains the values of the onset function (time serie), *a* frame, before the frame *l*, and *b* frame after it. The quantity $\langle D[l_m] \rangle$ is the moving average over the interval [*m*- *a*, *m*+ *b*], α and λ are two positive number. This adpative threshold method the authors argues can be used for computing onsets in real-time.

The use of the median in the adpative threshold is motivated by the fact, that unlike the moving average, the median filters, is robust to outliers – an idea which we will make use of when smoothing the output of the Yin algorithm described in the next subsection. Moreover, the authors argues that the use of the moving average term in the adpative threshold, attempts to replicate the effects of the normalisation and DC-removal processes, which is possible if the onset detection would be done offline.

This approach to model the threshold over time, is the one implemented in two of three libraries (Aubio and Essentia), which we will use to evaluate the different onset detection algorithm as described in the next section.

In the next subsection, we will describe the YIN algorithm, which we will use to estimate the fundamental frequency of the input signal.

B) The YIN algorithm

The estimation of a pitch of musical signal can be done in time domain, or in the frequency domain. The Yin algorithm is an example of time domain estimates; and is composed of 4 major steps. The first one is the main idea behind the algorithm, and the remaining ideas, can be considered as a modification to make the algorithm more robust.

As a first step, consider a perfectly periodic signal (e.g. a sine wave at a certain frequency). If we were to sample it, and observe the sequence of number, we would notice that the difference between the first sample, and each subsequent sample would approach to zero for the *first* time, at a distance of half the wavelength of the signal. The frequency and the wavelength of the signal are related, by the fact that their product is the speed of sound. Hence in order to find the frequency, one can exploit the above observation, and that is the main idea behind the first step in the Yin algorithm.

In the second step of the algorithm, one define the difference function

$$d_t(\tau) = \sum_{j=1}^{W} (x_j - x_{j+\tau})^2$$

and tries to find the minimum of this function, and *W* is the length of the window. For an analytical signal, the above equation has infinite solution corresponding to multiple of the fundamental frequency. In estimating the fundamental frequency of a sampled signal, one would find the smallest value of this set. (in practice, the set of solutions are bounded by the sampling frequency).

The subsequent steps of the algorithm are modification to the above idea to deal with not perfectly periodic signal.

In the third step of the algorithm, we notice that the above equation has minimum when τ is 0. To correct for this problem, and for situation when the first formant (f1) produce a series of dip lower than that of the correct period, they author of the paper introduce a new function

$$d_t''(\tau) = \begin{Bmatrix} 1 & \text{if } \tau = 0 \\ d_t(\tau) \Big/ \left[(1/\tau) \sum_{j=1}^{\tau} d_t(j) \right] & \text{otherwise} \end{Bmatrix}$$

The above function starts at 1, for τ = 0, then increases for small values of τ, and then dips below 1 the first time the distanced, is smaller than the average value that preceded it. Using this new function *d*', the algorithm would favor smaller value for τ, given two candidate with same distance *d*.

In the following step of the algorithm, the authors set an absolute threshold in order to avoid situation when the higher order dips are returned in the minimization of the function *d*'. This is the situation when the algorithm might detect a sub-harmonic instead of the fundamental frequency. Given this threshold value, the modified algorithm searches for the smallest value of τ that gives a minimum value of *d*' smaller than the threshold value. If none is found, the algorithm returns a global minimum. This threshold can be interpreted as the proportion of the aperiodic power tolerated within a periodic signal.

To further improve the algorithm, they perform a parabolic interpolation around each local minimum of *d*'. This step is correct for the possible offset, if the period to be estimated is not a multiple of the sampling period.

In situations where the signal analysized is non-stationary, the value of *d*' tends to be large; and in its last step, the YIN algorithm look in the vicinity of each analysis point for a better estimate.

When using YIN algorithm, we make use of the estimated value of *d*', which approximates the amount of aperiodicty in the signal, along with the estimated fundamental frequency f0.

3. LIBRARIES COMPARAISON

3.1 Ground Truth

In order to evaluate the applicability of the different method implemented in different libraries (Aubio, Essentia, and MIR toolbox), we recorded 2 users with different singing skills: an amateur singer (*El recordings*), and a professional singer (*Ella recordings*). The recording setting was semi-clean to resemble the typical use of the prototype application: the recording was done in UPF studio, but presence of back ground singing voice from one user can be heard in the recording of the other user. We proceeded by carefully marking each of the onset found in these recordings. The annotation was done using Sonic Visualizer software[8], and the files are included in the reference folder. The following table list the number of onset found in each of the recordings:

File Name	Number of Onsets in the files
ElRecording1.wav	69
ElRecording2.wav	128
EllaRecording1.wav	70
EllaRecording2.wav	124

Table 1: Number of ground truth onsets events

3.2 Evaluation Method

To evaluate the performance of each library, we computed 3 various statistics: Precision, Recall and the F-measure. Often used in the field of information retrieval (Baeza-Yates & Ribeiro-Neto, 1999), these measures can be understood in our context as follow:

- Precision: is the number of onset (from the ground truth) "correctly" estimated by the library divided by the total number of onsets returns by the library
- Recall: is the number of onset (from the ground truth) "correctly" estimated by the library, divided by the total number of onsets (in the ground truth)
- F-measure: is a statistics that combine precision and recall. While there are several way to define this measure, the one we used is computed by the following formula:

$$F = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

We define the notion of correct estimate of an onset, if the estimated onset time is within 25 milliseconds on with side of the time of the onset in the ground truth. We will use these statistics throughout the remaining of this section for comparison purpose.

[8] http://www.sonicvisualiser.org/

3.3 Compraison Results:

A) Aubio Library

Aubio is a tool designed for the extraction of annotations from audio signals. Its features include segmenting a sound file before each of its attacks, performing pitch detection, tapping the beat and producing midi streams from live audio (Brossier, 2003). One of the functions in this library is *aubioonsets* - a command line tool that was used for offline onset extraction. This function has 2 main parameters, which we focused on:

- The threshold value, which set the threshold value for the onset peak picking.
- Onset mode, which the onset detection function the library uses.

For the threshold values, we varied the value from 0.01 to 0.999, in steps of 0.001 (a total of 990 points) to determine the effect of the threshold value on the statistics described above.
We have also tried the following detection methods (onset mode): high frequency content (HFC), complex domain, energy, and spectral difference.

The other parameters that are used by the Aubio library are: window size of 1024 samples, and hop size of 512 samples.

The following two figures show the relations between the threshold values and the various statistics:

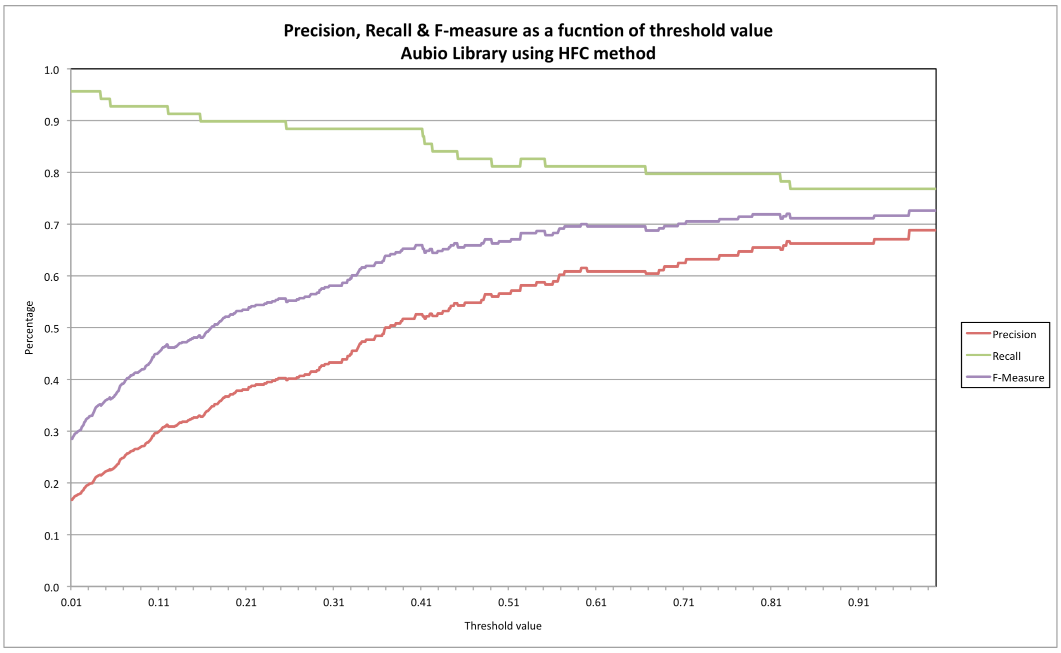


Figure 1: Effect of threshold on the Precision, Recall, F-measure for ElRecording1, using the HFC method

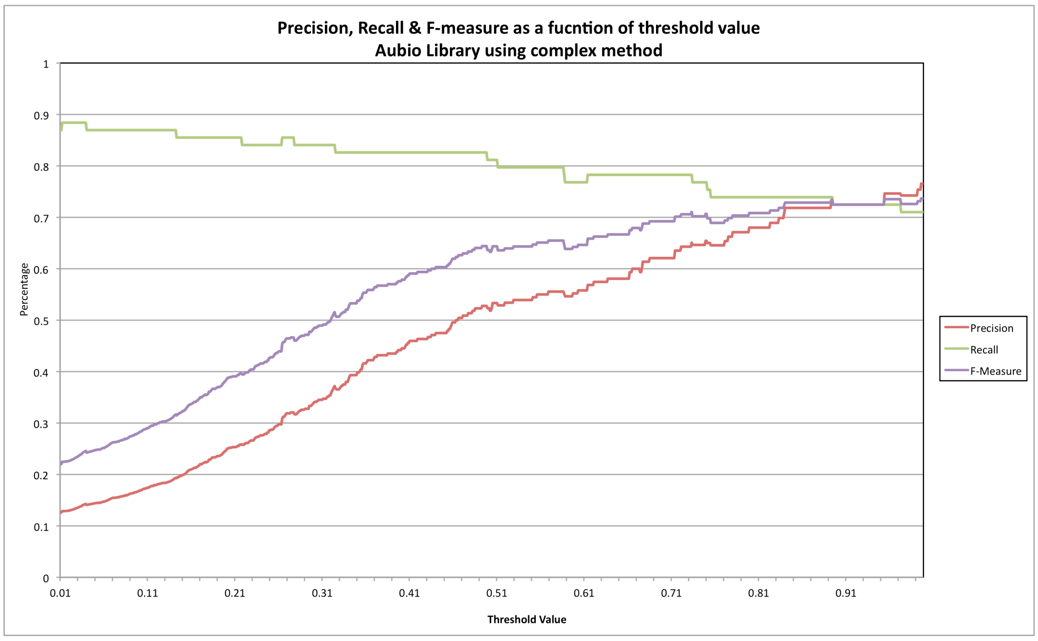


Figure 2: Effect of threshold on the Precision, Recall and F-measure for ElRecording1 using the complex domain method

We can see from the above two graphs that as we increase the threshold value for detecting onsets, the precision function increases (since the library returns less results), and the recall function decreases for the same reasons.

In the following tables, we list the maximum value of these statistics achieved by the using different onset detection function

		File Name	
Onset Function	Data	**ElRecording1.wav**	**Threshold value**
complexdomain	Max of Precision	0.766	0.996, 0.997, 0.998
	Max of Recall	0.884	[0.01 - 0.03]
	Max of F-Measure	**0.737**	0.996, 0.997, 0.998
energy	Max of Precision	0.329	
	Max of Recall	0.783	
	Max of F-Measure	0.458	
hfc	Max of Precision	0.688	
	Max of Recall	0.957	
	Max of F-Measure	**0.726**	
specdiff	Max of Precision	0.491	
	Max of Recall	0.899	
	Max of F-Measure	0.594	

Table 2: Comparison of the various onset detection function for "El Recording 1"

		File Name	
Onset Function	Data	**ElRecording2.wav**	**Threshold value**
complexdomain	Max of Precision	0.862	0.996, 0.997, 0.998
	Max of Recall	0.984	[0.01 - 0.22]
	Max of F-Measure	**0.916**	0.996, 0.997, 0.998
energy	Max of Precision	0.374	
	Max of Recall	0.891	
	Max of F-Measure	0.527	
hfc	Max of Precision	0.801	
	Max of Recall	0.984	
	Max of F-Measure	**0.880**	
specdiff	Max of Precision	0.642	
	Max of Recall	0.984	
	Max of F-Measure	0.773	

Table 3: Comparison of the various onset detection function for "El Recording 2"

		File Name	
Onset Function	Data	**EllaRecording1.wav**	**Threshold value**
complexdomain	Max of Precision	0.690	
	Max of Recall	0.929	
	Max of F-Measure	**0.662**	
energy	Max of Precision	0.365	
	Max of Recall	0.843	
	Max of F-Measure	0.473	
hfc	Max of Precision	0.672	[0.946 -0.95]
	Max of Recall	0.900	0.01, 0.011
	Max of F-Measure	**0.676**	[0.807 - 0.817]
specdiff	Max of Precision	0.449	
	Max of Recall	0.900	
	Max of F-Measure	0.531	

Table 4: Comparison of the different onset detection function for “Ella Recording 1”

		File Name	
Onset Function	Data	**EllaRecording2.wav**	**Threshold value**
complexdomain	Max of Precision	0.803	0.955, 0.956
	Max of Recall	0.960	[0.047 - 0.062]
	Max of F-Measure	**0.814**	0.952, 0.953
energy	Max of Precision	0.376	
	Max of Recall	0.935	
	Max of F-Measure	0.526	
hfc	Max of Precision	0.803	
	Max of Recall	0.984	
	Max of F-Measure	**0.813**	
specdiff	Max of Precision	0.528	
	Max of Recall	0.984	
	Max of F-Measure	0.646	

Table 5: Comparison of the different onset detection function for "Ella Recording 2"

As we can see from the above tables, the complex domain method and the high frequency methods gives comparable results, and outperform the energy based and the spectral difference method for this recording.

Since the best recall statistics, can be obtained trivially, but setting the threshold to 0 (i.e. detecting every possible onsets), and similarly, the optimal precision can be achieved by setting a very large threshold value, we will focus our attention on the F-measure in determining the optimal threshold.

We also note that the same range of threshold values (0.95 – 0.999 using the complex method) results in optimal detection for the onsets time for all detection. This implies that given the same recording condition, we can apply the same level of threshold regardless of the singing ability of the user.

Finally, in order to explain why the same method (the complex method for instance), with the same threshold value resulted in different value of the statistics (74% vs 92% for the F-measure, for “El Recording 1”), we went back to recording and found that in “El Recording 1” and “Ella Recording 1”, the background noise (the singing of the other user) was relatively louder than in the other recordings. This confirms that even though the same threshold value range could be optimal in different noise condition, the overall performance of the onset detection algorithm will vary accordingly.

B) Essentia Library

Essentia is a python based library that is developed at the Music Technology Group (MTG) that contains various DSP algorithms used for research purpose at UPF. Among those algorithms are two onset detection functions, which are based on (Bello et al., 2005; Brossier, Bello, & Plumbley, 2004).

The two onset detection functions implemented in this library are the high-frequency method, and the complex method. One of the features implemented in this library is the ability to mix the two different onset detections to form a new one. The other default parameters used in this library are: window size of 1024, hop size of 512, a silence threshold of 0.019, and a threshold parameter of 0.1[9].

Given that the results using the Aubio library favored the two methods available in Essentia, we decided to vary the mixing parameter, *p,* and see the impact on the various statistics.

The following table shows the effect of the mixing parameter on the various statistics.

El Recording 1

	max value	Mixing Parameters (HFC)
Precision	0.623	1
Recall	0.696	0.79
F measure	0.658	1

Ella Recording 1

	Max value	Mixing Paramters (HFC)
Precision	0.525	0.88
Recall	0.600	0.88
F	0.560	0.88

El Recording 2

	max value	mixing parameters (HFC)
Precision	0.911	0.98
Recall	0.961	0.98
F measure	0.935	0.98

Ella Recording 2

	max value	Mixing Parameters (HFC)
Precision	0.638	1
Recall	0.492	0.93
F measure	0.552	0.98

Table 6: Mixing parameter's effect on the Precision, Recall, F-measure (Essentia)

Few points are worth noting about the above table. First the mixing parameter of 1, also produced an maximum value for the recall measure in the "El recording". Similarly, a mixing parameter of 0.96 and 0.98 also produced maximum value for all statistics of the "Ella recording 1". Given the above the data, we can deduce, that given the default parameters, given above, if we were to use Essentia library to estimate onsets time, we would use the high frequency method rather than the complex method. We will come back to this finding in the next section, when considering the implementation of our prototype. Finally, it's worth noting, than the values of the statistics produced using Essentia were not as good as the one using Aubio, since the effect of the threshold parameters has a bigger impact on the results than the mixing parameters.

[9] The exact parameters values for the silence threshold and onset thresholds are: 0.019999999553, and 0.10000000149 respectively

C) MIR toolbox

The MIRtoolbox is an integrated set of functions written in Matlab, dedicated to the extraction of musical features from audio files (Lartillot & Toiviainen, 2007). Its design is based on a modular framework in which the different algorithms are decomposed into stages so the user can build his/her algorithm using a bottom-up approach. As a result, there are various way of building an onset detection algorithm, and so we focused our attention on the readily available methods utilizing the energy based envelope, and the spectral difference approach.

We varied the parameters of the "*mironsets*" function available in the MIRtoolbox. The user manual defines this paramters as: "a local maximum will be considered as a peak if its distance with the previous and successive local minima (if any) is higher than the contrast parameter" (Lartillot, 2010). This parameter has a similar effect to the threshold parameters in the above Aubio and Essentia libraries.
In total, 100 different contrast value of the contrast were inputted into the *mironsets* function. These points were linear distributed in the log space: the minimum value of contrast was 0.0001, and the maximum value of 0.3. The following table summarizes our results using the MIRtoolbox.

		Precision	**Contrast Parameters**	**Recall**	**Contrast Parameters**	**F-measure**	**Contrast Parameters**
ElRecording1	1024 WSize - 512Hop	0.06527	0.00011	0.4058	0.0001	0.1125	0.00011
ElRecording1	1024-512 Smoothed 10	0.01449	0.00043	0.0145	0.0001	0.0145	0.00043
ElRecording1	1024-512 Smoothed 5	0.02500	0.00133	0.0290	0.0001	0.0268	0.00133
ElRecording1	Default Envelope	0.02273	0.00199	0.0290	0.0001	0.0255	0.00199
ElRecording1	**Spectral Flux**	**0.50000**	**0.30000**	**0.6522**	**0.0001**	**0.4636**	**0.14488**
ElRecording2	1024 WSize - 512Hop	0.05082	0.00012	0.2422	0.0001	0.0840	0.00012
ElRecording2	1024-512 Smoothed 10	0.00000	0.00010	0.0000	0.0001	0.0000	0.00010
ElRecording2	1024-512 Smoothed 5	0.00000	0.00010	0.0000	0.0001	0.0000	0.00010
ElRecording2	Default Envelope	0.00474	0.00011	0.0078	0.0001	0.0059	0.00011
ElRecording2	**Spectral Flux**	**0.72277**	**0.30000**	**0.8047**	**0.0001**	**0.7072**	**0.17032**
EllaRecording1	1024 WSize - 512Hop	0.16667	0.21709	0.2714	0.0001	0.1126	0.00275
EllaRecording1	1024-512 Smoothed 10	0.14286	0.13363	0.0429	0.0001	0.0387	0.00010
EllaRecording1	1024-512 Smoothed 5	0.18182	0.12325	0.1286	0.0001	0.0994	0.00234
EllaRecording1	Default Envelope	0.16667	0.21709	0.0571	0.0001	0.0412	0.00070
EllaRecording1	**Spectral Flux**	**0.63636**	**0.30000**	**0.7857**	**0.0001**	**0.6087**	**0.04307**
EllaRecording2	1024 WSize - 512Hop	0.14362	0.00855	0.3952	0.0001	0.1923	0.00113
EllaRecording2	1024-512 Smoothed 10	0.10526	0.00089	0.0806	0.0001	0.0913	0.00089
EllaRecording2	1024-512 Smoothed 5	0.22034	0.00170	0.2339	0.0001	0.2149	0.00170
EllaRecording2	Default Envelope	0.14765	0.00096	0.1774	0.0001	0.1612	0.00096
EllaRecording2	**Spectral Flux**	**0.88889**	**0.30000**	**0.7823**	**0.0001**	**0.7040**	**0.08918**

Table 7: Summary of Onset Detection using MIRtoolbox

The defaults parameters for window size used in the *mironsets* are: window size of 0.1 sec (4410 samples, if sampling rate is 44.1Khz), and a hop size of 0.1 of window size (441 samples). These values were varied too, in order match the window size/hop size of Aubio/Essentia. We have tried to apply a autoregressive smoothing of different order (5 and10) to the energy envelope however the results were not encouraging as it can be seen in the strange/poor results obtained for "El Recording 2". The above table shows that the spectral flux method (which is similar to the complex method – expect it doesn't use the phase information of the spectrum), outperforms the energy-based method.

4. MAX/MSP IMPLEMENTATION

4.1 Onset Detection Algorithm

The analysis done in the previous section was not intended to be exhaustive comparisons of the libraries used (as we used defaults parameters for many parameters). However, the purpose of it was to evaluate which onset detection methods works better for out type of recording, and which library should we use, if any, in our implementation of the prototype.

From the Aubio library analysis, we determined that both high frequency method, as well complex method for onset detection produced similar results. We also determined that the same level of threshold is applicable to users with different singing ability (El vs. Ella), but that the overall results depended on the background noise in the recordings.

When we evaluated the MIRtoolbox, our initial intention was to use Matlab enviroment for the analysis, and send the results using Open Source Control message to Max MSP. This approach would have limited the usage of our prototype to the offline usage (i.e. use existing audio recording to drive the our midi soft instrument). Moreover, given the quality of the results, using the readily available method in the toolbox, (see the reported statistics in the above tables); we decided against using this library.

Finally, given that the high frequency content requires less computations, and was found to perform better than the complex detection function approach for real-time setting (P. Brossier, J. P. Bello, & M. D. Plumbley, 2004), we decide to follow this approach in the our prototype implementation. This onset detection method was also favored given the results using Essentia library shown in the previous section. Since the high frequency content method is realtively simple to code, we decided to implement it directly in Max/Msp using the FTM library (Bevilacqua, Müller, & Schnell, 2005; Schnell & Schwarz, 2005; Schnell, Borghesi, Schwarz, Bevilacqua, & Muller, 2005) and keep all the computation in one application for efficiency purpose.

4.2 High Frequency Content implementation

In the following figures we illustrate our max implementation of the high frequency content method in Max/Msp. The default analysis window size used is 512 samples (corresponding to roughly to 11 milliseconds using a sampling rate of 44.1Khz), and the hop size is 256 samples. These values, which can be changed by the user, were chosen, since we found experimentally that the shortest percussive sound imitated by mouth could last as short as 10 milliseconds. Also, since we don't care much about the frequency resolution – this relatively short frame length will give us a better time resolutions, as we compute the Fourier transform to calculate the HFC.

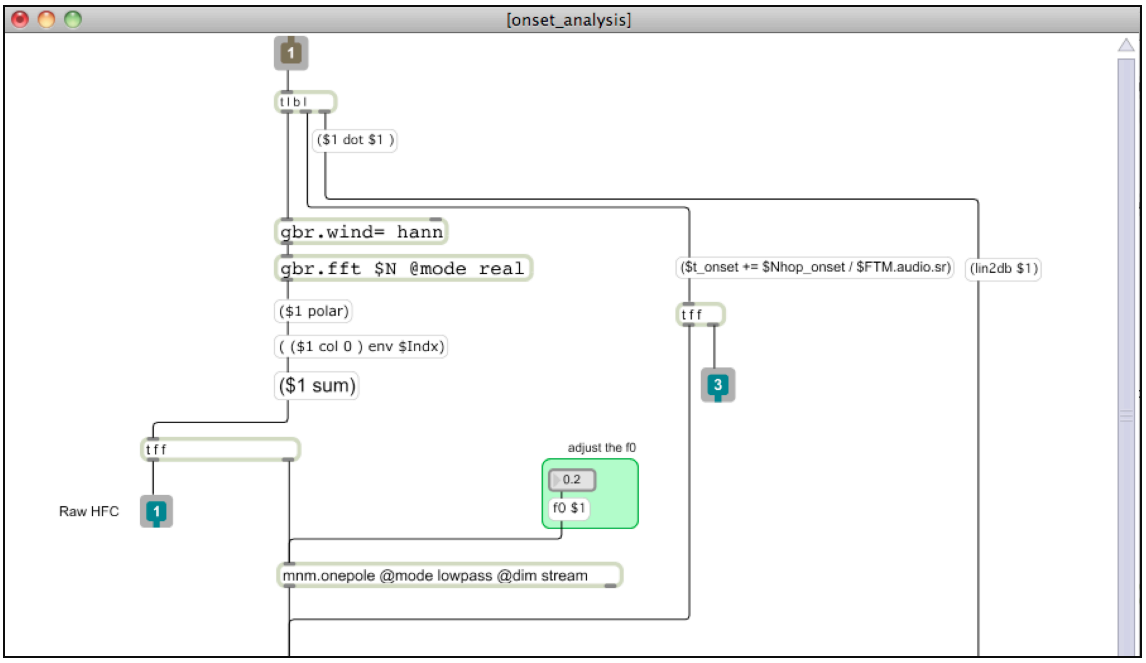


Figure 3: Computation and pre-processing of High Frequency Content

In order to reduce the spectral leakage and to distinguish the peaks of the magnitude spectrum, which will results in better estimate of the HFC, we apply a window function to the signal. We chose the Hanning window since it has a small main lobe width (4 bins), and the height of its highest side lobe is sufficiently small (-43Db).

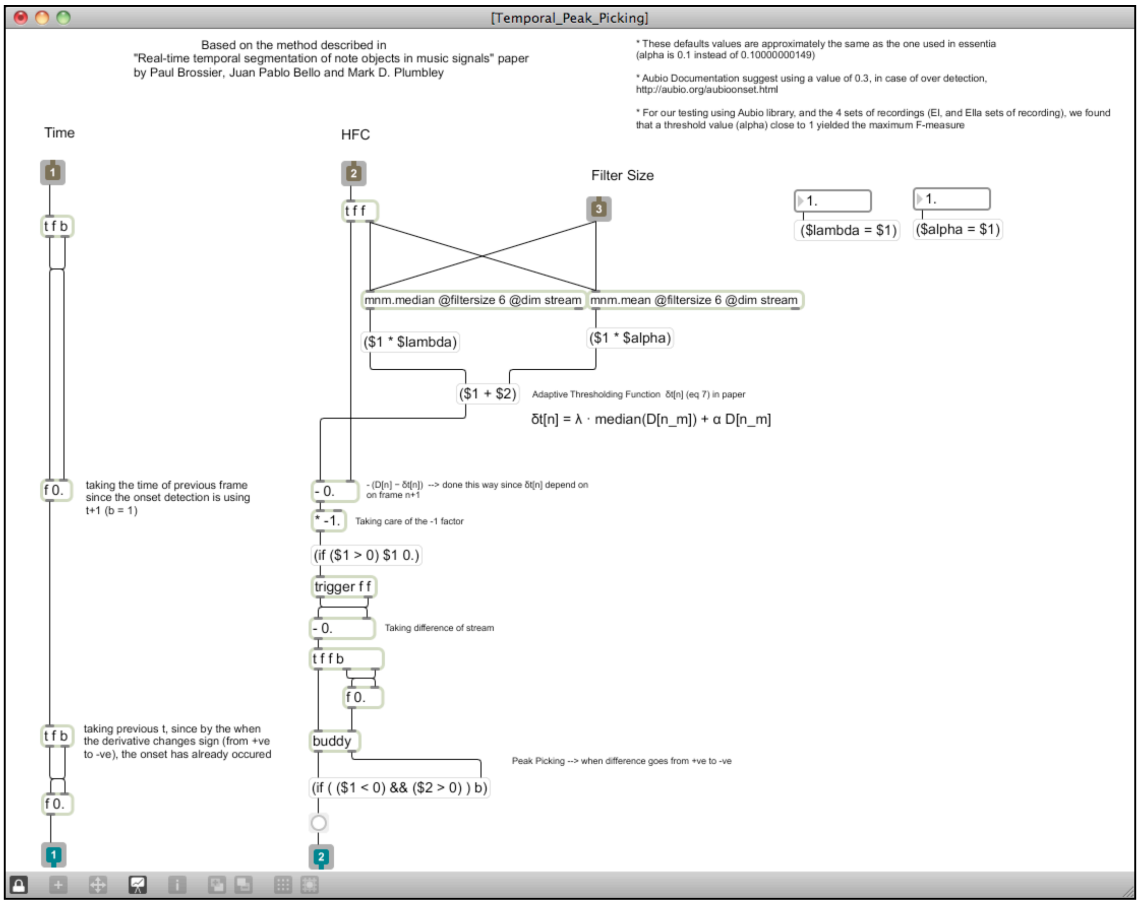
In order to smooth the HFC curve to better estimate the onset times, we apply a low pass filter, as recommended in (Brossier, 2006). The IIR filter we used is a first order difference equation, which is equivalent to Brown's Simple Exponential Smoothing (Natrella, 2010; Nau, 2005). For this type of filter, we used the following equation

$$f0 = 2/(N+1)$$

relating the cutoff frequency of the filter (normalized by the Nyquist frequency), to the length of the moving window, N, used in the exponential moving average. The exponential moving average is similar to the simple moving average, however, it weight the more recent observation more heavily than distant one. The term exponential stem from the fact that the weight decays in an exponential fashion.

For a defaults value of f0 = 0.2, we get a value of $N = 9$ frames. This corresponds to roughly 50 milliseconds using a hop size of 256 samples. The user can change the value of the normalized cutoff frequency, shown in the green box.

After computing and smoothing the HFC values, we proceed into temporal peak picking to estimate potential onsets as described in the theory section. Given that the potential onset at frame time t, depends on the value of the HFC at frame time t +1, we will demonstrate, in the next figure, how we implemented this using max object and it's built in logic of right to left ordering of messages



Based on the method described in
"Real-time temporal segmentation of note objects in music signals" paper
by Paul Brossier, Juan Pablo Bello and Mark D. Plumbley

* These defaults values are approximately the same as the one used in essentia
(alpha is 0.1 instead of 0.10000000149)

* Aubio Documentation suggest using a value of 0.3, in case of over detection,
http://aubio.org/aubioonset.html

* For our testing using Aubio library, and the 4 sets of recordings (El, and Ella sets of recording), we found
that a threshold value (alpha) close to 1 yielded the maximum F-measure

Adaptive Thresholding Function δt[n] (eq 7) in paper

δt[n] = λ · median(D[n_m]) + α D[n_m]

- (D[n] − δt[n]) --> done this way since δt[n] depend on
on frame n+1

Taking care of the -1 factor

Taking difference of stream

taking the time of previous frame
since the onset detection is using
t+1 (b = 1)

taking previous t, since by the when
the derivative changes sign (from +ve
to -ve), the onset has already occured

Peak Picking --> when difference goes from +ve to -ve

Figure 4: Real-time Temporal Peak Picking for Onset Detection

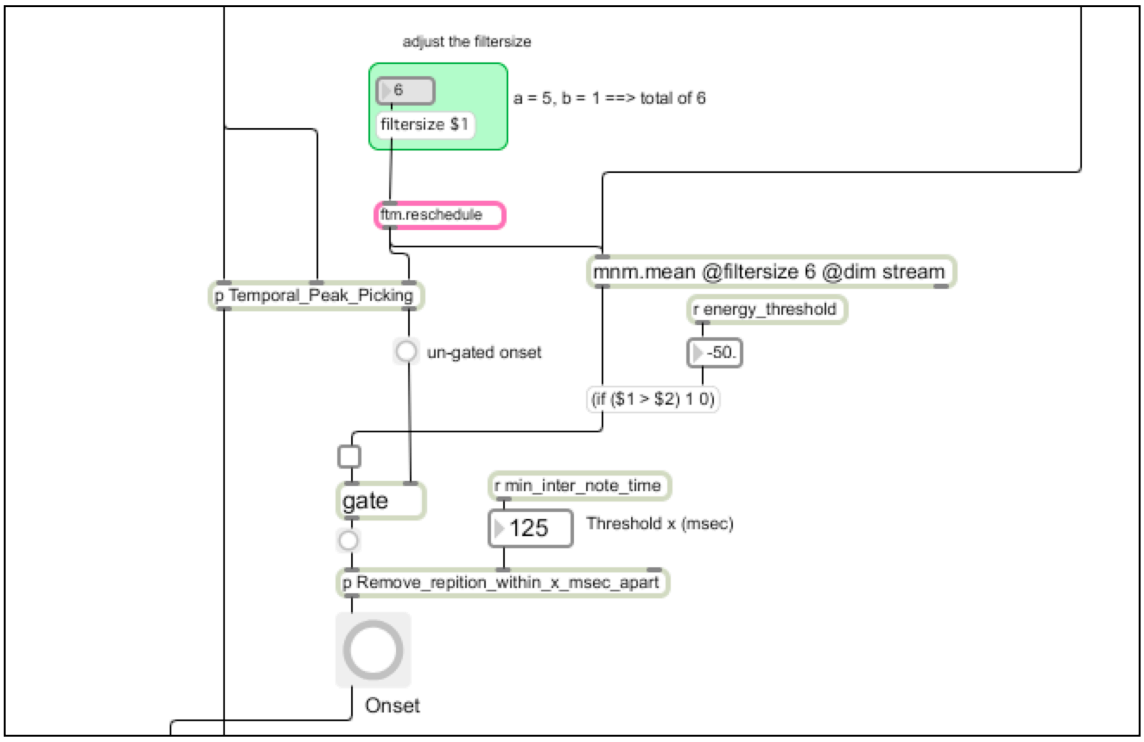
As stated in (P. Brossier, J. P. Bello, & M. D. Plumbley, 2004), the length of the window for which we compute the moving median and the moving average is 6 (a = 5 delays frames, and b = 1 future frame). Again the size of this window can be changed through the third inlet – however, the using the above implementation, restrict the value of *b* to 1. On the right hand side of the patch, we delay the time value of the frame by 1 frame, to accommodate for forward looking of the algorithm.

Once we computed the difference between the smoothed HFC and the adaptive threshold values, we apply a half-wave rectification to zeros out the resulting function when the HFC estimate is below the threshold. This step was found to improve the detection of onsets.

We then proceeded by searching for peak of this new rectified curve. In order to achieve this goal, we compute first order difference to approximate the derivative of this function. Once the sign of the derivative estimates changes from positive to negatives, that triggers the onset. Again, since by that time, we are one time frame too late, we delay the frame time by one hop-size, as it’s done in the lower left part of the patch.

As following step, as illustrated in the next figure, we apply a gate to eliminate any onsets, when the energy is below a threshold set by the user. In order to be consistent with the window size over which we computed the adaptive threshold detection function, we compute a running average of the energy, where the window length equals 6 frames.

Figure 5: Silence Gate and the removal of onsets which are less than x milliseconds apart



Finally, as a last step in the post-processing of the onset detection function, we utilize the sub-patch "*Remove_repition_within_x_msec_apart*" to remove, as the patch name suggest, any onsets which are *x* milliseconds apart. The value of *x*, as we will see below, is user define in term of note value (16th note as default), and relative to a beats per minutes (BPM), which are both set in the main driving patch.

4.3 YIN Algorithm: Smoothing and a different Onset Detection.

Now that we have implemented an algorithm for onset detection (i.e. to determine when the note/sound occurs in the recording), the next step in implementing a voice-based controller for a midi soft-instrument, is to extract the pitch (if there is a stable one) from the voice input. Given that the window size used for the onset detection was sufficiently small (512 samples ~ 11 milliseconds) for good time resolution, we had to analyze in parallels the input voice, with a different window and hop size.

Given that the typical frequency human voice can range from 100 to 1000 Hz, the longest wavelength we can expect to find in any recording would correspond to that of the frequency of 100 Hz. However, to err on the side of caution, we considered a minimum frequency of 70 Hz. Given a sampling rate of 44.1Khz, this correspond to roughly 1 cycle every 630 samples. Moreover, as a rule of thumb, in order to have a good estimate of the fundamental frequency, the size of the window analyze be least 3 times the wavelength of the target frequency. Therefore, we chose a default window size of 2048 sample, and a hop size of 1024 (roughly 21 milliseconds) in order to capture any variation in the pitch.

To estimates the fundamental frequency of the voice, we make use of the YIN algorithm, explained in the Section 2, and available part of the FTM library. In the following figures, we illustrate our usage of the algorithm, and how we smoothed the curve to get rid of the infamous "octave" error.

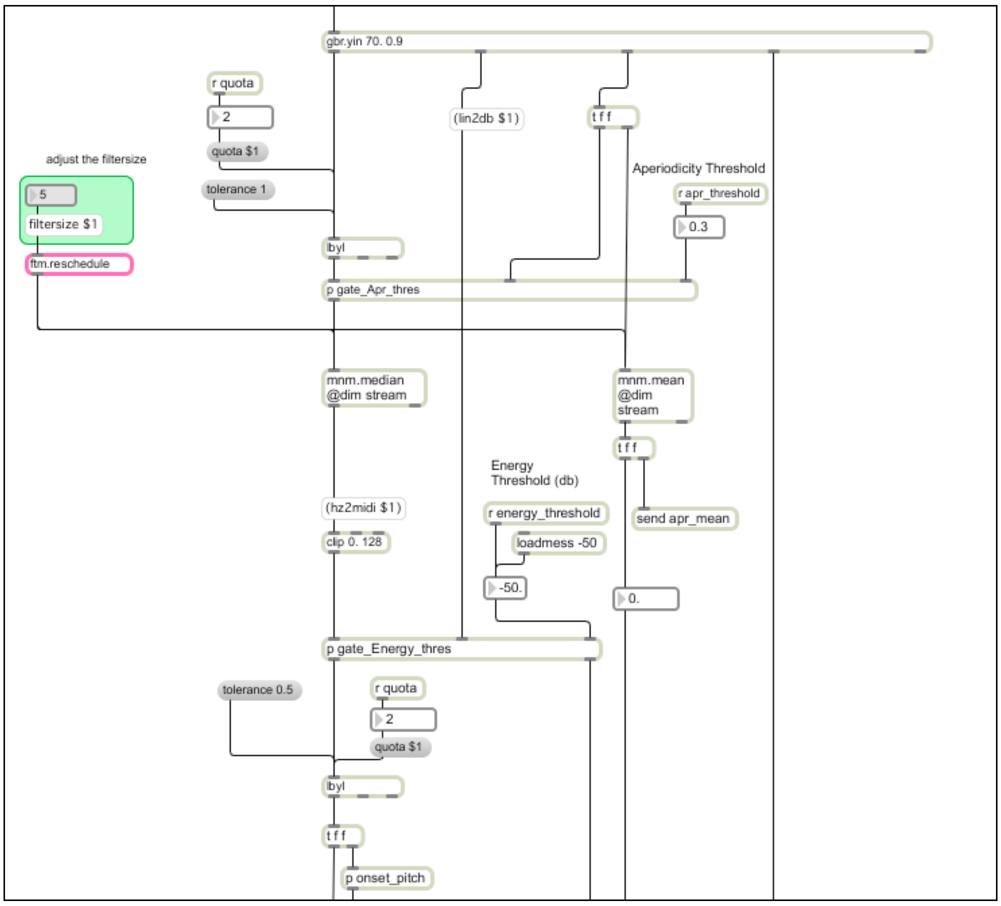


Figure 6: Smoothing of the output of the YIN algorithm and a different Onset Detection

The parameters given for the YIN algorithm are 70 Hz as a minimum frequency to be estimated, and a threshold value of 0.9 for aperiodicity. As explained in (De Cheveigné & Kawahara, 2002), the first parameter has a much bigger impact on the quality of the results, than the second one. As a results we chose a relatively relaxed value for the aperiodicity threshold[10], considering that we do not know a priori the quality of the input voice, and choose to post-process the results of the YIN estimates, using the reported aperiodicity (as recommended by the author of the referenced paper).

As in can be seen in the above graphs, once we obtain the output of the YIN algorithm, we post-process the output in several ways in order to smooth the output.

First we run the output through the look before you leap (*lbyl*) object part of the CNMAT package for Max/MSP (Wright, 2003). This object "echo an input stream of numbers to the output, but doesn't report large changes (controlled by the tolerance parameters) unless the input stays at that value for a number of input samples (controlled by the quota).

Next, we process the output based on the estimated level of aperiodicity. In our experiments, we found that the type of error that the YIN algorithm encountered were "too high" errors. This is expected since the range of the singing voice, is much closer to the lower limit of the algorithm than its upper limit - which is one quarter of the sampling rate. As a result, whenever, the value of the aperiodicity was lower than a specified threshold, we replaced the fundamental frequency estimates with 0.

Next, we applied a median filters to smooth the curve, and replaced our 0 values, with the median over a window. The default size of the window was chosen to be 5 (roughly 115 milliseconds, using a hop size of 1024). This value was chosen since we found that the "too high" error didn't persist for more than one or two frames, and a larger window would resulted in our prototype to be less responsive to frequency change.

After converting our processed YIN value to midi value, we post-processed the values based on the energy level. If the energy for the corresponding frame was below a user specified threshold, the midi value was set to 0, silencing any notes that might have been produced undesirably by background noise.

Finally, we pass the processed fundamental frequency estimates through another (*lbyl*) smoother before passing the final output to our midi synthesizer and our alternate implementation of the onset detection function. While all these different post processing step might seems like an over-kill, we found them to be necessary to produce "good results", and as we will discuss later, the parameter of the smoothing functions were chosen carefully in order not to produce too much latency.

Before we move on and explain the main driver of our prototype, we will comment briefly on our alternate onset detection method. As simple and rudimentary as it seems, we consider an onset whenever the processed midi value corresponding to the f0 estimate, changes by more than 1 midi value. This method was found to be robust, produced better results than the HFC, as we will discuss later.

[10] The maximum value for the aperiodicity is 1

4.4 Main User Interface

In this section, we will look at the main driver for the prototype we developed. We will explain how the user can use the prototype and change some of the default parameters for the onset detection and the post-processing of the YIN output to fit his/her need.

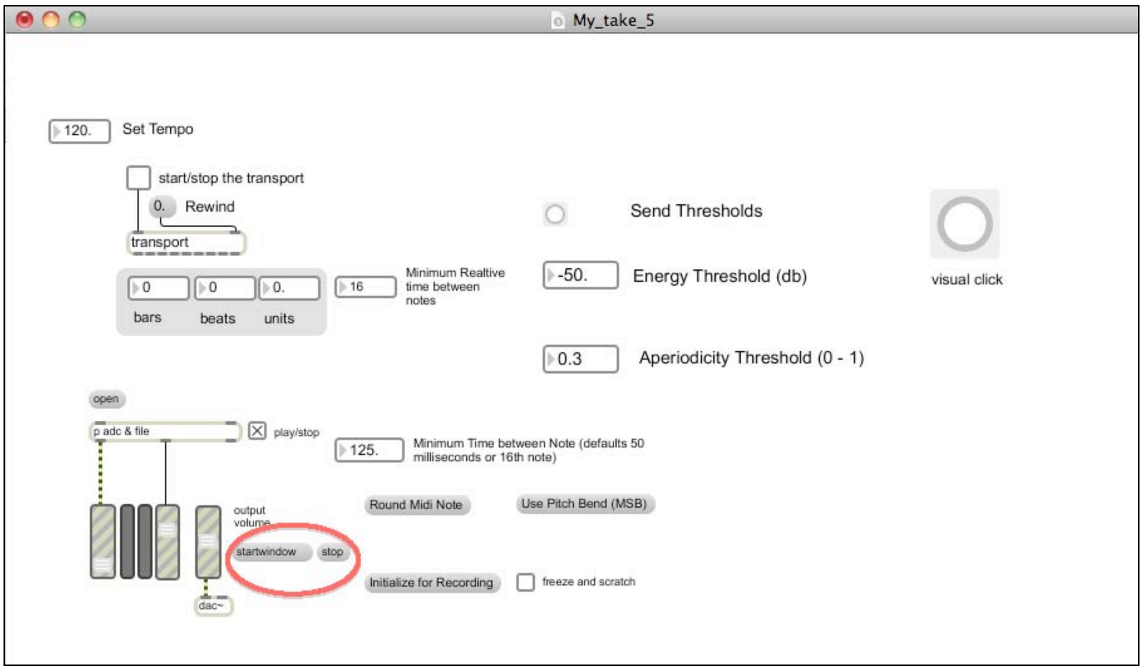


Figure 7: The Main driver of the prototype

First in order to turn the audio engine on/off we press on the “*startwindow*” button (“stop” button) respectively, located in the red circle. The volume fader to the left is to control the microphone input level, and the middle one is to control the volume level of a pre-recorded sound file (to be driving the prototype)

On the top left, the user can set the desired tempo he/she wishes to sing at. While this tempo doesn’t restrict the user in any way in interacting with the prototype, its value along “*Minimum Relative Time between notes*” determines the various parameters used in the onset detection and the post-processing of the YIN output. For example, a value of 16, for the later parameter, refers to 16th note, which at 120 BPM, translates to 125 milliseconds. This is the minimum time between onsets, which is the input in the last stage of post-processing of the onsets detection algorithm.

Similarly, for the post processing of the YIN output, a value of 125 milliseconds, using a hop size of 1024 samples, translates roughly to 5.3 frames. Since we are using two (*lbyl*) objects, we decided to the equally split the value, 5.3, and use the integer part to set the value of the quota. That is to say, a new note will have to be stable for roughly 32nd note’s duration before either of the (*lbyl*) objects considers it a note change.

Furthermore, the widow size used in the moving median was set to 5 following a similar logic: this post-processing stage looks at note interval, slightly under 16th note to estimate the median of the estimated fundamental frequency. All the computation for these above-mentioned parameters, are done automatically by the prototype, once the user changes the BPM and the “*Minimum Relative Time between notes*” from the defaults values.

Also located in the middle part of the screen, are number box so that the user can set the energy threshold level to his/her liking depending on the background noise available in the environment where the prototype will be used. In the discussion we will comment on why we chose to allow the user to control of these parameters manually, in an interactive way, rather than being calculated by the prototype.

Moreover, a little below the area where the user set the threshold parameters, are two buttons that determines whether to round the midi note to the nearest integer, or whether to use a midi pitch bend message to reflect the continuous nature of the YIN output. In case the user is controlling a midi instrument that sounds like a violin, we envisioning the user will utilize the midi-pitch bend of the feature to reflect the continuous range of note that he/she will want to imitate. Where, in the case of controlling a midi instrument that sounds like a piano, for example, we imagine the user would want to round the midi estimate to the nearest integer, so that that resulting sounds doesn't sound out of tunes. It's also worth noting that the Hertz to midi conversion is done using a tuning of 440 Hz.

Finally, from the upper left corner, the user can start, stop, and rewind a metronome, which in turn on a visual clock that will flash every quarter note. The aim of this is to aid the user to stay on beat (should he chose to, of course).

4.5 The Play and Record

This sub-patch controls few yet essential part of the prototype (such as the output/routing of the midi message), and so simple figure should be sufficient to explain all the inner workings.

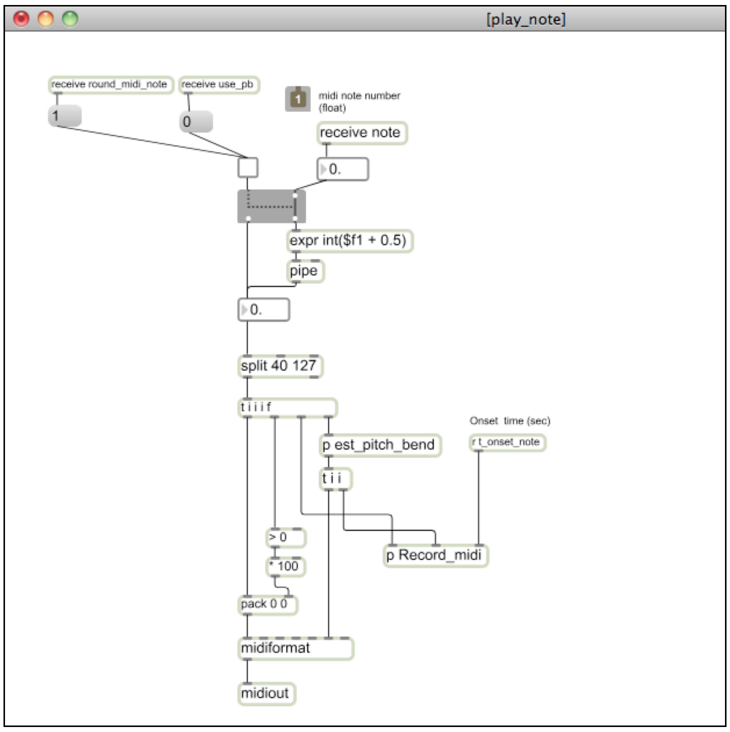


Figure 8: The Play and Record part of the patch

As it can be seen on the top, when the user pushes the button to round the midi notes, the signal is routed, and the appropriate calculation takes place in this sub-patch. Also as it can be seen, all outgoing midi notes are played at a constant velocity of 100. We will come back to this limitation in the discussion section and how to over-come it in future developement of the prototype. For those not familiar with Max/MSP, in order for the user to route the midi signal, he/she have to double click on the *midiout* object, and set the desired. Midi output port.

Finally, in the "*Record_midi_note*" sub-patch, the user will find a transcription of the detected and played midi notes, along with the estimated onsets time. This allows the user to output the midi files, so that he/she can use them in the favorite sequencer such as Ableton Live or Logic. Based on the forum[11] of the voice band application for i-phone we found this feature to be highly requested, and so we decided to offer it part of our prototype. However, if the user wishes to do so, he/she will have to press the "*initialize for recording*" button located in the center of the main driver patch.

[11] http://www.drumagog.com/phpbb2/viewforum.php?f=13

5. DISCUSSION

5.1 Libraries Evaluation and Implementation Approach.

As mentioned briefly above, the intention of the evaluation was not to determine which library outperforms the others in the generic case, but rather which onset detection library, and which approach would be best suited for the development and the architecture of our prototype. As a result of our evaluations, we decided not to use any pre-existing library, and opted to implement our onsets detection method based on the HFC. This decision was taken mainly to keep all the necessary computations and the interface in one application, hence reducing any potential overhead that might results from the different application communicating and synchronizing with one another.

When we start the implementation of the onset detection method, we considered using the library "Zsa.Descriptors" (Malt & Jourdan, 2008) available for Max/Msp and followed the approaches detailed in (Malt & Jourdan, 2009) for onset detection using low level. However, we found such an approach not promising, since it was difficult to synchronize the output of this library with the outputs of the different part of our prototype patch.

We decided to use the FTM library, because it provided an efficient framework that allowed us to compute onsets detection function, use the YIN algorithm, and the possibility to extract lower level feature will be an integral part in the future development of the prototype. We also found the Matlab-style handling of data – in matrix form, facilitated the debugging and the testing of the prototype.

5.2 Onset Detection Functions.

We have implemented two types of onsets detection: one based on the HFC, and the other based on changes in midi-note estimates

We have found that our implementation of the HFC didn't perform as expected! When testing the algorithm in different noise environment, we found that our implementation returns too many false positive, and false negatives results, to be used practically in an environment where the noise varies.

To explain this poor result, we believe that the background noise, found in the recordings used in the evaluation stage (section 3), was controlled: the singer stood a constant/stable distance apart to stand in front his/her microphone, which resulted in a relatively constant level of background noise. However, in a non-professional environment, where the noise level changes, finding a good level of threshold values is difficult. As we have seen in the evaluation section, the results of the algorithm are sensitive to the threshold level. We have tried different pre-processing and post processing techniques in order to make the algorithm useful for our implementation, but our effort were fruitless.

For testing the prototype, we relied on the built in microphone found on laptop computer (Apple Mac book pro). We hypothesize that one way of making this HFC based onset algorithm practical, is by using specialized – cardioids or a shotgun, microphone, where the sound is picked up only from certain direction. Since the intention of our research project was to implement a software tools, and we didn't want to depend on any hardware specification. For this reason, we didn't pursue the question of testing the performance of the implemented algorithm's dependence on the hardware used.

To render our prototype useful, at least when using it to control the midi note of a synthesizer, we have implemented, as discussed in the previous section, an onset detection algorithm which relies on changes in the midi notes (we will discuss the limitation of this approach in the next section). As the midi notes are estimated using stable pitches (the stability is ensured by our post-processing of the YIN frequency output), this approach for onset detection was found to be more robust to variation in the background noise. This onset detection method is used to drive the midi synthesizer.

Having implemented two onset detection methods, our first thought was two use them both simultaneously, and build a logic in our prototype which will allow to decide on which method to be used based on the type of sound input. That is to say, since the HFC is known to be good for percussive sounds, and since we have implemented an onset detection for tonal sounds, our basic logic implemented in our prototype would do the following: If the average value of the aperiodicity was above a given threshold, the input sound would trigger a pre-recorded drum sample, and if the average threshold was below a threshold level, the midi note would be triggered.

While, initially, this seemed like a good idea, our basic implementation, based on this simple logic, did not produce the desired results, even in an environment where there was no background noise. We believe that there are two reasons behind this failure. First, when imitating a percussive sound, the aperiodicity estimate of input sound varies a lot from one trial to the next, therefore it hard to come up with a good threshold level. Second, when imitating a percussive sound, and the pre-recorded sample is not triggered, sometime, the midi event would be triggered due to the presence of pitch in the input sound. We will comment on a better implementation to achieve this feature, in the future work subsection.

5.3 Latency testing of the prototype.

Again, as mentioned we have achieved good results using changes in the midi notes value as an onset detection algorithm, and after extensive smoothing of the YIN's output, we were able to control a midi-driven synthesizer. In order to test our prototype more rigorously, we recorded an excerpt of a singing voice, using the built in microphone, in an outdoor environment. We labeled the onsets manually, and use it as a ground truth.

Using the defaults value for the thresholds (-50db for background noise, aperiodicity threshold of 0.3, and 125 millisecond minimum time between onsets). We measure the time of the synthesized midi note, and compared it to ground truth to estimate the latency of the developed prototype. The following table summarizes the finding of our testing:

Ground Truth (sec)	Played Time (sec)	Difference (sec)	Difference (milliseconds)	Note sung
0.52930	0.73343	0.20413	204.13	Do
1.37755	1.56935	0.19180	191.80	Mi
1.99184	2.28917	0.29733	297.33	Sol
2.89959	3.07864	0.17905	179.05	Do
3.49932	3.82168	0.32236	322.36	Sol
4.26231	4.40218	0.13987	139.87	Mi
4.93569	5.07556	0.13987	139.87	Do
7.17533	7.35112	0.17579	175.79	Re
7.92127	8.18703	0.26576	265.76	Fa
8.74898	8.90685	0.15787	157.87	La
9.45601	9.62667	0.17066	170.66	Re
10.05197	10.23040	0.17843	178.43	La
10.62884	10.90380	0.27496	274.96	Fa
11.39875	11.53070	0.13195	131.95	Re
13.28762	13.41150	0.12388	123.88	Mi
13.94068	14.20100	0.26032	260.32	Sol
14.65034	14.92080	0.27046	270.46	Si
15.41846	15.64060	0.22214	222.14	Mi
16.01596	16.26760	0.25164	251.64	Si
16.65454	16.94100	0.28646	286.46	Sol
17.42939	17.56790	0.13851	138.51	Mi
19.26095	19.56480	0.30385	303.85	Fa
20.09252	20.26140	0.16888	168.88	La
20.76590	20.91160	0.14570	145.70	Do
21.35075	21.63140	0.28065	280.65	Fa
22.08780	22.25830	0.17050	170.50	Do
22.71202	22.86200	0.14998	149.98	La
23.27236	23.53540	0.26304	263.04	Fa

Table 8: Latency Results of The prototype

In order to assess the quality of these numbers, we first estimate the minimum latency imposed by our choice of defaults parameters set in the program. Given the window size is 2048 samples, this would contribute to roughly 46 milliseconds of the expected latency. Also, given that we estimated pitch has to be stable for a length of time, before being processed by the midi synthesizer, this introduce an added latency. This latency, as explained earlier is controlled by the "*quota*" parameter of the (*lbyl*) object used. For the default values used, this is equivalent to 4 frames of latency, which translates to roughly 93 milliseconds. Hence the minimum latency we can possibly expect is about 139 milliseconds. This number provides a benchmark against which we can compare the value in the 4th column in the above table. The elements in "red", are those notes where the latency is at least 100 milliseconds above the benchmark. However, up a closer look at the data, we notice that for all these notes sung has an explosive sound at the beginning of the notes (Fa, Si, Sol). We also noticed that there are a couple of entries, marked in blue, where the latency is below the benchmark. For these events, we went back to the audio recording, and notice the presence a pitch noise before the beginning of the notes. This was validated using Sonic Visualizer software, and the analyzed audio file is included in the reference folder.

While its ideal to bring the latency to 0, this goal is not acheivable; so instead one should try to bring the latency to a level acceptable by the user. Our intial user testing, were positive, however more testing needs to be done in order with wider user base, and with more sung notes in order to support our initial findings.

5.4 Limitations and Future Work

As mentioned before, when a user imitates a precussive sound, (such as a kick sound), the resulting sounds often contains a slight pitch. As a result, discriminating precussive from tonal sounds, sung by the user, based on aperiodicity estimates alone is not sufficient.

One approach to solve this problem, would be to let the user imitate several times, sounds corresponding to various precussive sample he/she would want to triggers with their voice. We will use the initial user input as training set, and use a k-nearest-neighbor approach for classification. Along with aperiodicity threshold, we will use the distance returned by KNN algorithm to determine the appropriate sounds to be triggered (the intended sample or the midi synthesiser). Initially, we will start with this approach, since the KNN algorithm is already available part of the FTM library which we used through-out in our prototype. Should the previous approach not yield good results, we will adopt the same approach taken when developing the VST plug-ins BillaBoop[12] (Hazan, 2005). In this alternative approach, the author used low-level features of the input signal (such as the spectral centroid), along with decision tree model (C4.5) to discriminate between percussive sounds.

Another limitation of our prototype is the inability to play micro-intonation such as vibrator or glissando. This limitation is due to the fact that we impose the following restriction in our prototype: the YIN fundamental frequency estimates has to be stable for an *x* number of frames before allowing any changes in the estimates, coupled with the fact, that a new onset is detected with the change in the midi note value is larger than 1.

As mentioned before, one of the main error of the YIN algorithm, and other algorithm that estimate the fundamental frequency, is the infamous “octave” errors. Given that the output of the YIN algorithm is a continuous value, one needs to smooth this output to make it use of it, as a control signal for a synthesizer. If we had models for the shapes of the pitch contour curves, and its variation, we would be able to use these models to smooth out any noise or error in the estimated pitch contour. To find such good models, is not an easy task, and we imagine it to be the focus of a much larger research project, possibly a PhD. One possible approach to achieve this goal would be to gather a large collection of monophonic recordings containing a wide range singing style, and use sparse coding techniques to model the variation, and the micro-intonations found in the pitch contour curves. Given the complexity of this research project, we put this down on the list of priority for the next development stage of our prototype.

Finally, in our evaluation for the onsets detection algorithm, we had dismissed the use of the energy-based method, in favor for the HFC and the complex method. As a result, we didn’t consider the “*bonk*” object, which it is an energy-based approach (Puckette, Apel, & Zicarelli, 1998), as an alternative to detect onsets. However, the “bonk”, object looks at the energy levels in different frequency band, and not in the entire range, as the above tested energy-based method do, we plan to test it’s applicability to in our prototype, in the next development phase.

[12] http://www.dtic.upf.edu/~ahazan/BillaBoop/index.html

6. CONCLUSION

Throughout this project, we researched and prototype what we beleive to be the first steps towards building a voice-based controller for a midi based synthesiser. As mentioned in the introduction, we had focused on the onsets detection algorithm and the smoothing of the pitch estimate as a first step of extracting information from the singing voice to control the parameters of a midi synthesier.

Using the current version of the prototype, a user can sing into a microphone, and our program would translate in real-time the sung note, into midi notes which are then passed to the user favorite midi synthesiser. Also, using our prototype, the user can transcribed the midi notes sung and export the midi files for later use in his/her sequencer of choice. Such a features makes our prototype usefull not just for real-time performance, but also a tool which will help the process of electronic composition.

Regarding the feature of adaptability of the parameters to the users and the enviroment, we decided on exposing the important paramters to the user and allow him/her to set/interact with them in an intitutive way. The reason behind our decision was that we want to focus on the basic tasks (onsets detection, and pitch estimate), in this current version of the prototype, and at the same time make our prototype transparent for the user to interact with. Similary, after realizing the difficulties of setting the threshold parameters for the HFC onset detection algorithm due to the changing level of background noise, we decided that it would be a difficult task to predict this noise variation in a robust way (in order to make the prototype adaptive to the varying noise level). Such prediction would have to rely on source seperation techniques (to isolate the signal from the background noise), or modeling onsets in polyphonic signal, as discussed in (Röbel, 2006), both of which were beyond the current scope of this project. Hence we prefered to design our prototype in an interactive way instead of using an adaptive approach, as we beleive that interactivity is a key component when making music. However, it's worth noting that as we overcome some of the limitations of the current version of the prototype, and implement techniques to deferientiate between precussive sound, as described in (Hazan, 2005), our system would have an adaptive feature since the user will have to use his/her voice for the training examples.

The second version of our voice based controller would be to have the ability to extract the timber informations, such as breathiness and breathiness, from the input voice signal use these estimates as control signal for the midi synthesiser. Such timber informations would be based on low level features, to calculate a mid-or higher level descriptors of the input sounds. We beleive that the enviroment which we choose to develop our prototype would be ideal for that, since most of the low level features, detailed in (Peeters, 2004) can be easily computed, and FTM library.

Once we develope features mentioned above and work on the list of items mentioned in the discussion section, we plan to post our prototype online, and ask for feedbacks from a wider user base. Those feedback will be incorported in future version of our prototype application which we hope will take place in the near future.

BIBLIOGRAPHY

Baeza-Yates, R., & Ribeiro-Neto, B. (1999). *Modern Information Retrieval. Modern Information Retrieval* (p. 513). Addison Wesley.

Bello, J. P., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., Sandler, M. B., et al. (2005). A tutorial on onset detection in music signals. *Ieee Transactions On Speech And Audio Processing*, *13*(5), 1035-1047. INSTITUTE OF ELECTRICAL AND ELECTRONICS ENGINEERS. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1495485.

Bello, J. P., Duxbury, C., Davies, M., & Sandler, M. (2004). On the use of phase and energy for musical onset detection in the complex domain. *IEEE Signal Processing Letters*, *11*(6), 553-556. Retrieved from http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=1300607.

Bello, J., Daudet, L., Abdallah, S., Duxbury, C., Davies, M., Sandler, M., et al. (2005). A Tutorial on Onset Detection in Music Signals. *Speech and Audio Processing, IEEE Transactions on*, *13*(5), 1035-1047. doi: 10.1109/TSA.2005.851998.

Ben Messaoud, M. A., Bouzid, A., & Ellouze, N. (2010). Autocorrelation of the Speech Multi-Scale Product for Voicing Decision and Pitch Estimation. *Cognitive Computation*, (3). doi: 10.1007/s12559-010-9048-1.

Bevilacqua, F., Müller, R., & Schnell, N. (2005). MnM: a Max/MSP mapping toolbox. In *Proceedings of the 2005 conference on New interfaces for musical expression* (pp. 85-88). Vancouver: University of British Columbia.

Brossier, P. (2003). Aubio, a library for audio labelling. Retrieved from http://aubio.piem.org.

Brossier, P. (2006). Automatic annotation of musical audio for interactive applications. *Centre for Digital Music, Queen Mary University of London*. Retrieved from http://aubio.org/phd/thesis/brossier06thesis.pdf.

Brossier, P., Bello, J. P., & Plumbley, M. D. (2004). Real-time temporal segmentation of note objects in music signals. In *Proceedings of the ICMC*. Retrieved from http://homepages.nyu.edu/~jb2843/Publications_files/Brossier-ICMC-2004.pdf.

Brossier, P., Bello, J., & Plumbley, M. (2004). Fast labelling of notes in music signals. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR 2004)*. Citeseer. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.100.8588&rep=rep1&type=pdf.

Clarisse, L., Martens, J., Lesaffre, M., & Baets, B. D. (2002). An auditory model based transcriber of singing sequences. In *Proceedings of 3rd*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.10.5174&rep=rep1&type=pdf

De Cheveigné, A., & Kawahara, H. (2002). YIN, a fundamental frequency estimator for speech and music. *The Journal of the Acoustical Society of America*, *111*(4), 1917. doi: 10.1121/1.1458024.

Dixon, S. (2006). ONSET DETECTION REVISITED. In *Proceedings of the 9th International Conference on* (pp. 133-137). Retrieved from http://138.37.35.209/people/simond/pub/2006/dafx.pdf.

Duxbury, C., Bello, J. P., Davies, M., & Sandler, M. (2003). A combined phase and amplitude based approach to onset detection for audio segmentation. In *WIAMIS* (p. 275–280). World Scientific Publishing Co. Pte. Ltd. Retrieved from http://eproceedings.worldscinet.com/9789812704337/9789812704337_0050.html.

Duxbury, C., Sandler, M., & Davies, M. (2002). A hybrid approach to musical note onset detection. *Proc Digital Audio Effects ConfDAFX*, 33-38. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13.451&rep=rep1&type=pdf.

Hazan, A. (2005). PERFORMING EXPRESSIVE RHYTHMS WITH BILLABOOP VOICE-DRIVEN DRUM GENERATOR. *Citeseer*, 20-23. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.74.1276&rep=rep1&type=pdf

Janer, J. (2008). Singing-driven Interfaces for Sound Synthesizers. Universitat Pompeu Fabra. Retrieved from http://www.iua.upf.edu/~jjaner/phd/defensa.pdf.

Lartillot, O. (2010). *MIRtoolbox 1.2.5 User's Manual* (p. 60). Retrieved from https://www.jyu.fi/hum/laitokset/musiikki/en/research/coe/materials/mirtoolbox.

Lartillot, O., & Toiviainen, P. (2007). MIR in Matlab (II): A toolbox for musical feature extraction from audio. *International Conference on Music*, (Ii). Retrieved from http://ismir2007.ismir.net/proceedings/ISMIR2007_p127_lartillot.pdf.

Malt, M., & Jourdan, E. (2008). Zsa. Descriptors: a library for real-time descriptors analysis. In *5th Sound and Music Computing Conference,*. Berlin. Retrieved from http://smcnetwork.org/files/proceedings/2008/session7_number3_paper45.pdf.

Malt, M., & Jourdan, E. (2009). Real-Time Uses of Low Level Sound Descriptors as Event Detection Functions Using theMax/MSP Zsa.Descriptors Library. In *SBCM 2009 – 12th Brazilian Symposium on Computer Music*. Brazil. Retrieved from http://www.e--j.com/?page_id=90.

Masri, P. (1996). Computer Music Modeling of Sound for Transformation and Synthesis of Musical Signals.

Natrella, M. (2010). *NIST/SEMATECH e-Handbook of Statistical Methods*. (C. Croarkin, P. Tobias, J. J. Filliben, B. Hembree, W. Guthrie, L. Trutna, et al.). NIST/SEMATECH. Retrieved from http://www.itl.nist.gov/div898/handbook/.

Nau, R. F. (2005). DECISION 411 - Forecasting [Lectures Notes]. Retrieved from http://www.duke.edu/~rnau/411avg.htm.

Peeters, G. (2004). A large set of audio features for sound description (similarity and classification) in the CUIDADO project. *CUIDADO IST Project Report*, 1–25. Retrieved from http://www.citeulike.org/user/ajylha/article/1562527.

Puckette, M., Apel, T., & Zicarelli, D. (1998). Real-time audio analysis tools for Pd and MSP. In *Proceedings of the International Computer Music Conference* (p. 109–112). Retrieved from http://en.scientificcommons.org/42924999.

Röbel, A. (2006). Onset Detection in Polyphonic Signals by means of Transient Peak Classification. In *MIREX – Onset Detection*. Retrieved from www.music-ir.org/evaluation/mirex-results/articles/onset/roebel.pdf .

Schloss, W. A. (1985). On the Automatic Transcription of Percussive Music -- From Acoustic Signal to High-Level Analysis.

Schnell, N., & Schwarz, D. (2005). Gabor, Multi-Representation Real-Time Analysis/Synthesis. *DAFx*, 1-5.

Schnell, N., Borghesi, R., Schwarz, D., Bevilacqua, F., & Muller, R. (2005). FTM - Complex data structures for Max. In *Proceedings of the 2005 International Computer Music Conference* (pp. 9-12).

Toh, C. C., Zhang, B., & Wang, Y. (2008). Multiple-Feature Fusion Based Onset Detection for Solo Singing Voice. In *Proceedings of the International Conference on Music Information Retrieval* (pp. 515-520).

Wright, M. (2003). *lbyl part of CNMAT Max/MSP/Jitter package*. University of Berkely (Music Departement). Retrieved from http://cnmat.berkeley.edu/downloads.