Pitch Recognition through Template Matching

Salim Perchy
Master in Engineering
Computer Science

Pontifica Universidad Javeriana
Cali, Colombia
Email:ysperchy@javerianacali.edu.co

Abstract

In this article the authors describe and document, along with tests and results, a methodology to perform automatic pitch recognition in real-life music audio signals. The methodology uses a matching criterion of the signal against a number of templates in order to best approximante the pitch in a given time, and although the methodology is fitted to work with monophonic music it can be easily extended into more elaborate sources of tonal sounds(i.e. chords.). We believe the methodology outputs good results and is straightforward enough to warrant the case study presented here.

INTRODUCTION

Pitch recognition is a task consisting of accurately stating the tone(or tones) being performed in a musical piece without the score information. Even to the trained ear, this task is sometimes demanding and difficult. Automatic pitch recognition was among the first problems laid by the *Computer Music* field to be solved on computers[3]. A whole sub-field called *Music Information Retrieval*(or MIR for short), still infant, deals with these type of problems with music signals.

There are a variety of automatic pitch recognition solutions with good results already in existence, a good source of the best methods is in [6] and a thorough survey can be found here [5]. The solution presented here offers a simple view on the problem without delving into complex probabilistic models of which a beginner into audio processing may find difficult to follow.

This papers is presented as follows; section I gives a formal definition and insight into the automatic pitch recognition problem, section II describes in detail the method used here to solve the problem, also the signal processing details and graphical results(in terms of frequency components) applied to the templates used are shown here. Section III describes some of the limitations of the stated method, section IV shows three cases of musical signals and analizes their ouput when using our method of pitch recognition and finally section V offers some concluding remarks on the matter at hand.

I. PROBLEM OVERVIEW

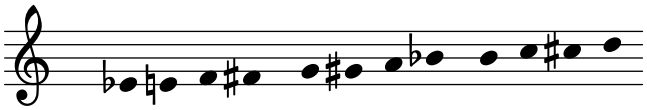
As said before, automatic pitch detection is the task of automatically detecting the note pitches being executed in a musical audio signal. The problem is inherently deterministic if one considers tonal music with ideal performers(never incurring in mistakes) and with ideal instruments(never deforming or untuning through time). Figure 1 intuitively shows the data flow of the task; a signal is feed into the pitch recognition program and the output is the exact pitches contained in the signal.

Figure 1: Automatic Pitch Recognition Problem Flow

We can formally define the problem, let the audio signal be A_s and a sequence of notes of length m be $[n_1, n_2, ..., n_m]$, therefore:

$$pitch(A_s, t_0, \Delta_t) = [n_1, n_2, ..., m_m,]$$

Where $pitch$ is a total non-injective function with the co-domain defined as the set of tonal notes $S_{notes} =$



The output of this function corresponds with the time interval Δ_t initiating in time t_0 in the audio signal A_s. The general description of the problem is:

$$pitch(A_s, 0, t_f) = [n_1, n_2, ..., n_p]$$

where there are exactly p notes in the signal A_s of length t_f.

However this is a simpification of the problem because we only take into account monophonic tonal music, that is, one note at a time and the whole note language ouput is limited to the twelve notes of tonal scales(c, c♯, d, d♯, etc.)[4]. In respect to noise present in real audio signals we do take a general denoising technique sufficient for our purposes, we shall explain this later.

We can take a step further and identify the height of the note(the relative octave of the pitch), in musical notation this is identified by how high it is in the stave or by an accompanying number(i.e. A4, C♭3) indicating how many octaves over the lowest audible frequency of that particular note.

For this we have to enrich our co-domain of the function $pitch$ with duplicates inditacing the specific octaves of each pitch in the twelve tonal notes. This accounts for approximately 8 octaves(8 times 12) in conventional music[8].

Finally, a desirable feature is to have the duration of the notes, musicians manage this feature with names specifing relative durations(whole notes, quarter notes, crotchets...) as shown in figure 2.

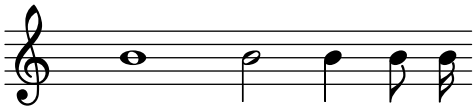


Figure 2: Relative note duration

In terms of signal processing it is preferable to have absolute time and this translates into knowing the exact time a note starts and ends. To achieve this we need again to extend the co-domain of the function $pitch$ with an output representing a silence or a *rest* as musicians formally call it, figure 3 shows a rest in musical notation.

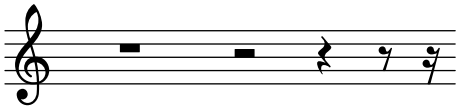


Figure 3: Musical Silence(Rests)

II. Template Matching Method

The method proposed here is to perform pitch detection in an uncompress signal using note templates, these are also uncompressed signals with a length of approximately 1 second. There is no restriction in the length of the target signal to detect its pitches.

To test it we will pitch-detect three distinct signals using six note templates(including a rest), the results on these three signals will be shown and discussed in section IV. The note templates used for testing are:

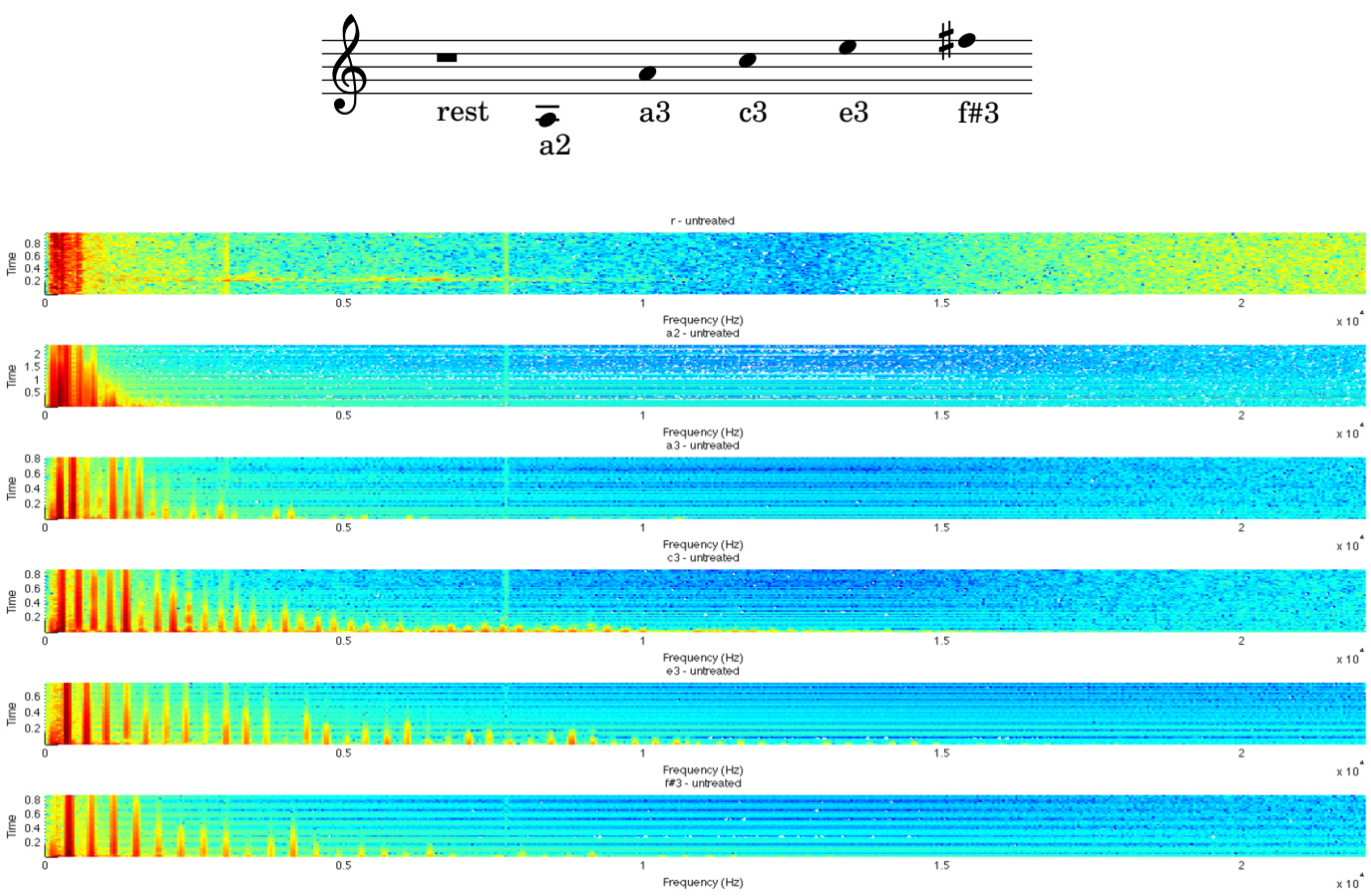


Figure 4: Untreated Templates Spectrum

Their frequency behavior through time is shown in figure 4.

The overall method is:

1) Prepare Signals(Target and Templates)
   a) Denoise Signals
   b) Remove Unwanted Frequencies
2) Store STFT's of signals
3) Match portions of the target signal to each template
4) Translate results to note names and absolute time

First, the templates and target audio signals are prepared for proper use in a matching criterion(i.e. converting the signals to a usefull presentation of their information), this is achieved by two stages.

A. Denosing

To denoise the signal we program a denosing stage applying 2 passes of a moving average filter of length 5, these parameters were chosen in accord to the previous experiments and results done in class where optimum length and number of passes of using a moving average filter where empirically found for denosing a musical audio signal(refero to lab3 [7]).

The spectrums of the templates after denosing are shown in 5.

B. Frequency Removing

Another stage necessary for preparing the signals is to use a band-pass filter for passing the frequency range where the usefull information is contained. In this case, we are only interested in musical note frequencies, in tonal music this is defined as

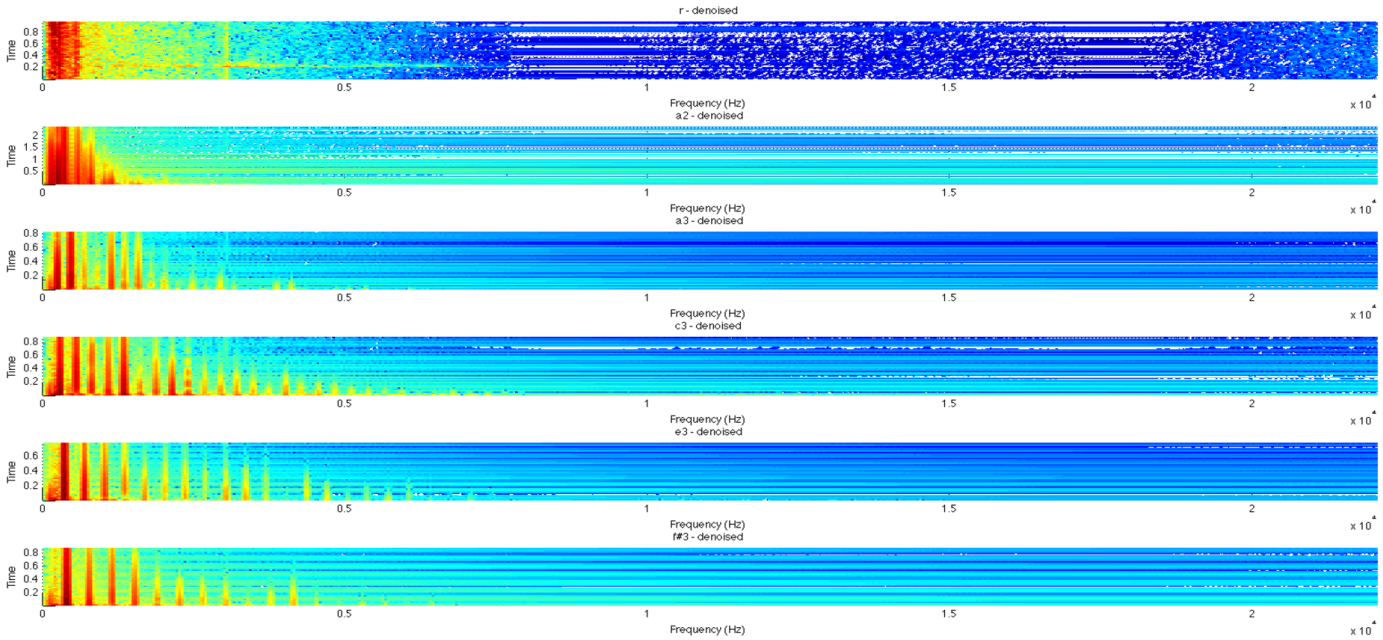


Figure 5: Denoised Templates Spectrum

approximately below 5000Hz(see [8]). We performed this stage using a band-pass filter of order 30 designed with *frequency sampling*(again, according to the best results for audio signals empirically found in [7]. The frequency-time results of the templates are plotted in figure 6.

C. STFT and Matching

Having prepared the signals we proceed to make a short time fast furier transform of all the signals(including the target signal) and store them for future processing. The STFT's are made with windows of 1024 samples with 64 samples overlapping on

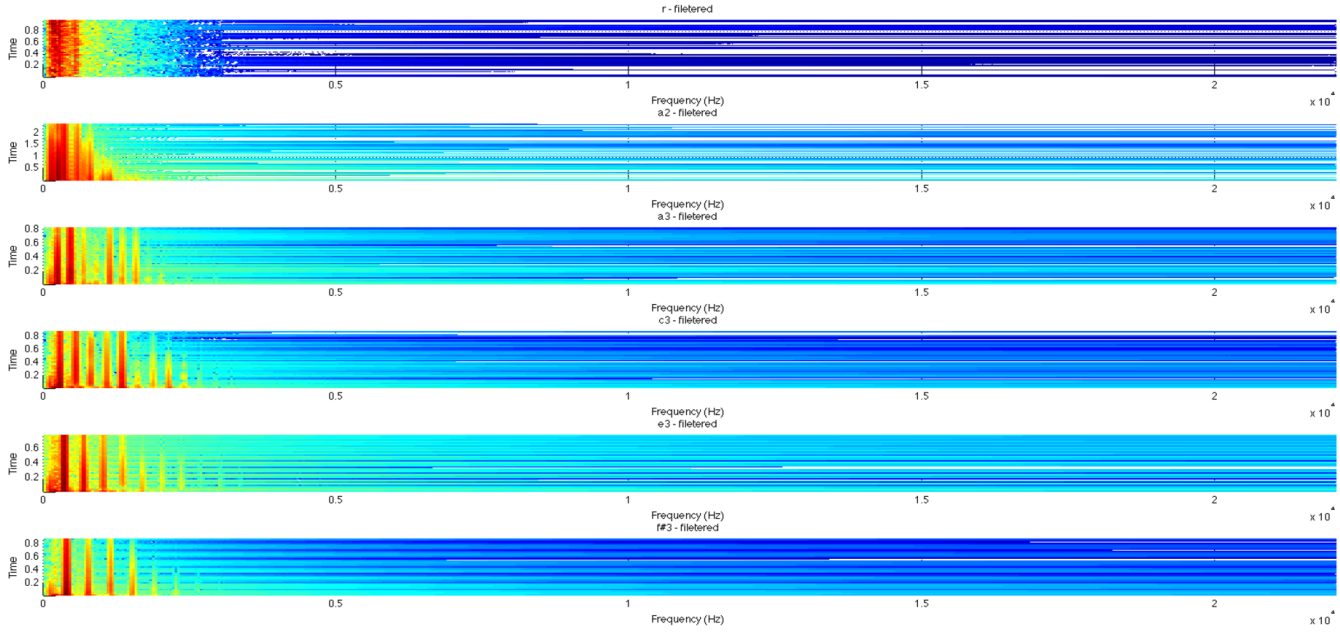


Figure 6: Filtered Templates Spectrum

each side of the window, these values are sufficient to not distort the audio content[1].

After doing the short time fast fourier transform, the target signal is splitted in time intervals, we can proceed to match each of these time intervals with the most similar template giving us a good estimate of the note being played at that moment.

This last step gives us a progression in terms of templates of the target signal, naturally we have to flatten this progression(compress repeated sequences, estimate the closest time of an event) as to best represent the needed information(pitches and duration).

An implementation in Matlab® of this method is presented in the appendix section V.

III. Drawbacks

The reader may have already noted that the templates given to the algorithm have to span the whole lenguage of the musical signal, otherwise an unknown event will be forced to match the closest(in terms of frequency) of the templates.

The matching criterion used is the *sum of squared differences*, this criterion is mathematically expressed as:

$$\mathbf{S} = \sum_{i=0}^{\text{window length}} (f^i_{signal} - f^i_{template})^2$$

There are other matching criterions more ellaborated and robust that could be used here[2].

1) Innacuracies: When matching parts of the target signal with the templates, there may exist some innacuracies inbetween a sequence of correct estimations of the pitch. This is more common in the decaying part of any note, as the power of the signal faints rapidly.

These innacuracies happen in time intervals very short, for example consider the next unflatten progression of recognized events:

```
c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 c3 a3 r r r r r r r r r r r r r r r r r r r.
```

It is clear that the signal contains a `c3` note and then silence, but there is a wrongly matched `a3`. If the number of samples of each segment is 1024, then this *supposed* `a3` note would have to be played by the musician in 0.02 seconds(at a sampling rate of 44100Hz), which is impossible even for the most virtuous performer, and even so it is imperceptible to the human ear.

To solve this situation we have to provide the program with a rejection time interval(in seconds), for which if any event detected is of any length less than this interval it will be absorbed(replaced) by the preceding event.

The Matlab® code to manage and eliminate this innacuracies is presented in the appendix section V

IV. Results

This section presents the results obtained when performing automatic pitch recognition on three audio signals with the method presented on the last section.

All signals(including templates) are musical notes played with an acoustic guitar recorded through a laptop microphone, their sampling rate is 44100Hz and their bit depth is 16 bits.

In all signals we execute the program as this:

```
pitch_recog( target_signal, {signal_templates}, {signal_names}, reject_interval );
```

A. Signal 1

This audio signal is composed of the next note sequence:

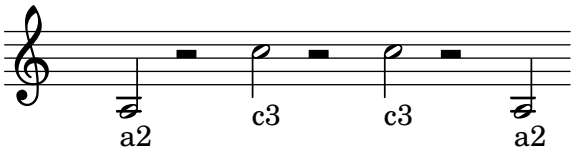


Figure 7: Musical sequence of first signal

Figure 8 shows the preparation for template matching of this signal(untrated, denoised and filtered). Executing the command:

```
  pitch_recog('signal1.wav',
              {'r.wav';'a2.wav';'a3.wav';'c3.wav';'e3.wav';'fsharp3.wav'},
              {'r';'a2';'a3';'c3';'e3';'f#3'}, 0.1);
```

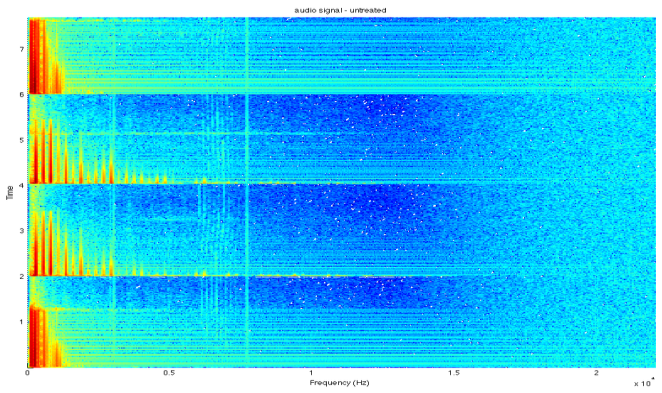
Yields these results:

```
Times =

         0    1.2800    2.0317    2.8444    4.0635    4.9575    6.0140    7.6190
Notes =

        'a2'     'r'      'c3'      'r'       'c3'      'r'       'a2'      'r'
```

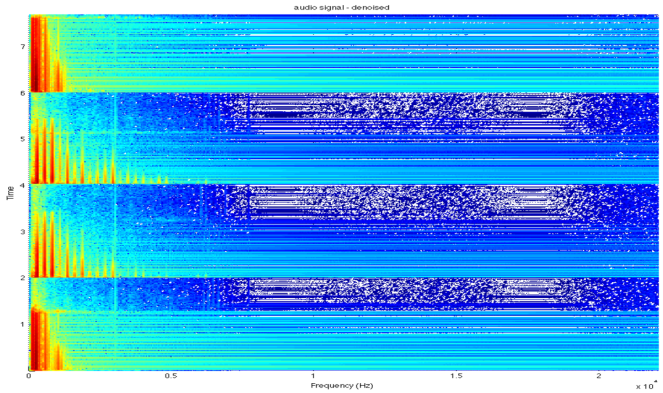
The statistics for the output are in table I. In this test we have a perfect pitch recognition.

Events	Missed Events	Wrong Events	Correct Events	Accuracy
8	0	0	8	100%

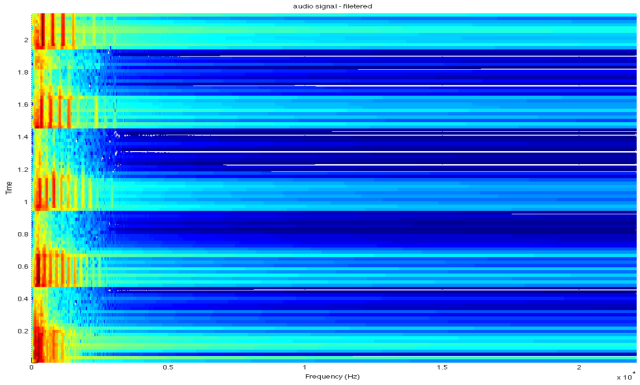
Table I: Pitch recognition statistics for the first signal



(a) Untreated



(b) Denoised



(c) Filtered

Figure 8: Preparation of first test signal

B. Signal 2

The second audio signal for testing is composed of the next note sequence:

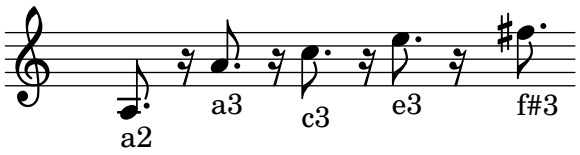


Figure 9: Musical sequence of second signal

Note that in this case the pace of the sequence is faster(notes are shorter in length), this can give us insight into the robustness of the method with respect to time changes. Figure 10 shows the preparation for template matching of this signal(untrated, denoised and filtered). Executing the command:

```
  pitch_recog('signal2.wav',
              {'r.wav';'a2.wav';'a3.wav';'c3.wav';'e3.wav';'fsharp3.wav'},
              {'r';'a2';'a3';'c3';'e3';'f#3'}, 0.2);
```

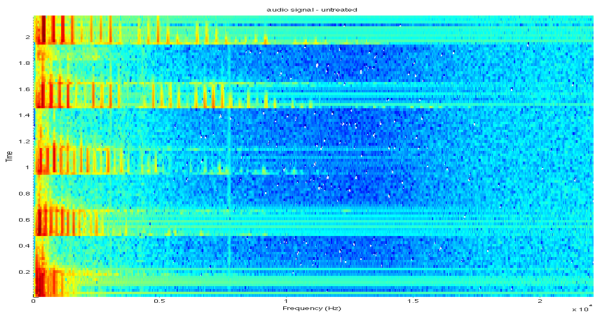
Yields these results:

```
Times =

         0    0.4673    0.6705    0.9346    1.1378    1.4629    1.6457    1.9505
Notes =

        'a2'    'a3'      'r'      'c3'       'r'       'e3'      'r'      'f#3'
```

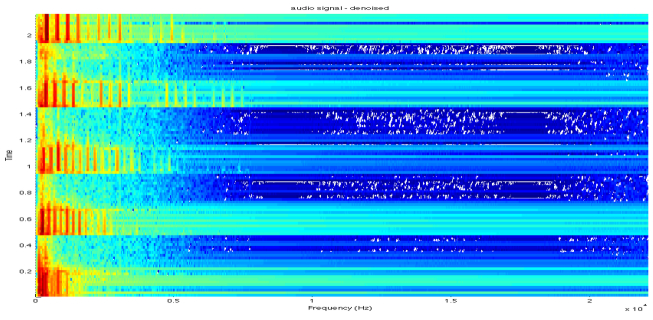
Note that in this test case we used a bigger rejection time interval(0.2s). The statistics for the output are in table II showing a 89% of accuracy.

Events	Missed Events	Wrong Events	Correct Events	Accuracy
9	1	0	8	89%

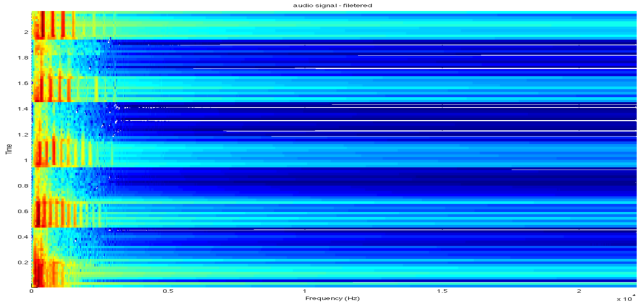
Table II: Pitch recognition statistics for the second signal



(a) Untreated



(b) Denoised



(c) Filtered

Figure 10: Preparation of second test signal

C. Signal 3

The third and final audio signal for testing is composed of the next note sequence:

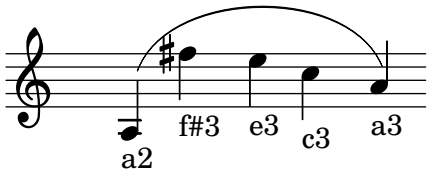


Figure 11: Musical sequence of third signal

In this signal we would like to point out two things; there are no rests and the notes are not stopped after being played(they fade naturally, this is what the arc over them means). This naturally produces an overall polyphonic sound and it is intended this way to test the tolerance of the method proposed to polyphonic signals.

Figure 12 shows the preparation for template matching of this signal(untrated, denoised and filtered). Executing the command:

```
  pitch_recog('signal3.wav',
              {'r.wav';'a2.wav';'a3.wav';'c3.wav';'e3.wav';'fsharp3.wav'},
              {'r';'a2';'a3';'c3';'e3';'f#3'}, 0.1);
```

Yields these results:

```
Times =

          0.0406    0.8330    1.4019    1.5238    1.7270    2.2146    2.8851    2.9257    3.3930
Notes =

         'a2'    'f#3'    'a2'    'e3'    'a2'    'c3'    'a3'    'a2'    'r'
```

The statistics for the output are in table III showing a 40% of accuracy. Note that all the pitches in the actual sequence are actually recognized, but between each one there is a `a2` inbetween, this is becuase the first `a2` was let sound thoughout all the musical sequence.

Events	Missed Events	Wrong Events	Correct Events	Accuracy
5	0	3	5	40%

Table III: Pitch recognition statistics for the third signal

We can improve our accuracy by executing the command with a larger reject time interval:

```
  pitch_recog('signal3.wav',
              {'r.wav';'a2.wav';'a3.wav';'c3.wav';'e3.wav';'fsharp3.wav'},
              {'r';'a2';'a3';'c3';'e3';'f#3'}, 0.2);
```

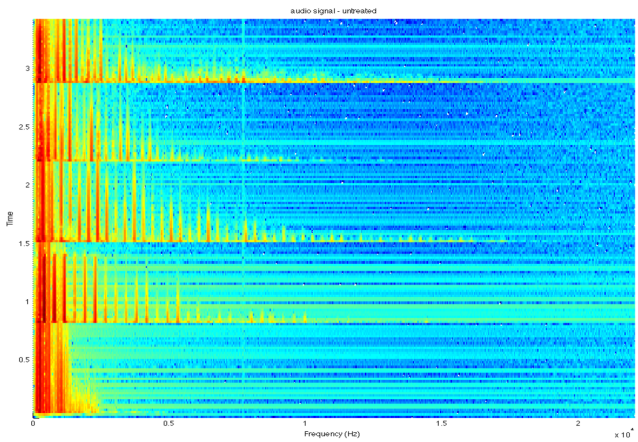
Yielding these results:

```
Times =

          0.0406    0.8330    1.5238    1.7270    2.2146    2.8851    2.9257    3.3930
Notes =

         'a2'    'f#3'    'e3'    'a2'    'c3'    'a3'    'a2'    'r'
```

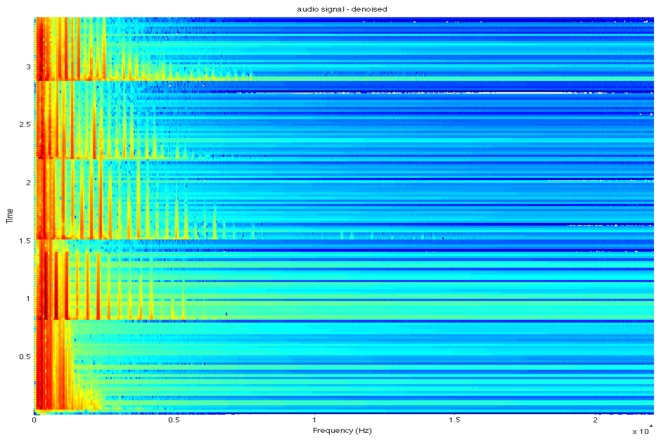
Table IV shows the statistics for these trial. We now have an accuracy of 60%.

Events	Missed Events	Wrong Events	Correct Events	Accuracy
5	0	2	5	60%

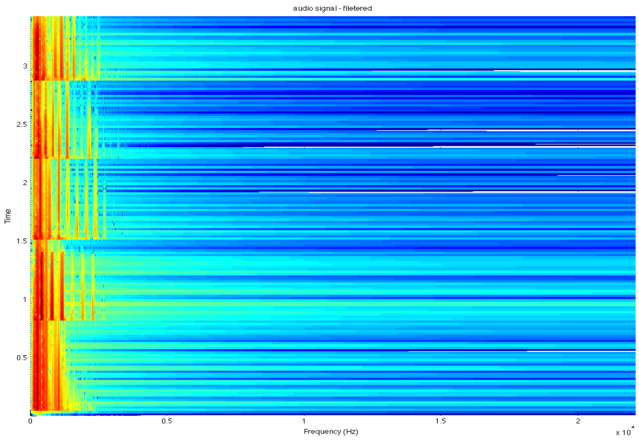
Table IV: Pitch recognition statistics for the third signal



(a) Untreated



(b) Denoised



(c) Filtered

Figure 12: Preparation of third test signal

V. Concluding Remarks

We proposed a simple method of automatic pitch recognition by matching parts of an audio signal with note templates, these signals are prepared for analysis(denoised and filtered from unwanted frequencies) and then a comparison from each time interval in the target signal is matched to a corresponding note.

We can conclude that:

- Pitch recognition through template matching behaves well in monophonic music signals providing that the user inputs all possible notes in the templates.
- In polyphonic music signals there are detected events that correspond to resonances of past notes, nonetheless this is innacuarte and a more robust method is needed.
- The frequency information through time is sufficient to analyze and detect pitches provided that they are monophonic.
- Denosing a signal and rejecting unwanted frequencies(above musical notes) is a proper step before analyzing pitch data in these signals.
- The matching criterion can be changed in the method in order to adjust to more complex scenarios and/or better complexity.

APPENDIX

```
% Rest(silence) has to be the first template
function [ notes ] = pitch_recog( signal, templates, names, delta )
    % Parameters
    window  = 1024;
    overlap = 128;
    rejectf = 1500;

     % Audio signal and templates
    [audio,freq,nbits] = wavread(signal);
    for i=1:size(templates,1);
        taudio{i} = wavread(templates{i});
    end

    % Denoise signal and templates
    daudio = denoise( audio, 5, 2 );
    for i=1:length(taudio);
        dtemplates{i} = denoise(taudio{i}, 5, 2);
    end

    % Reject unwanted frequencies(no good for pitch recognition)
    faudio = defreq(daudio, 30, rejectf, freq);
    for i=1:length(dtemplates);
        ftemplates{i} = defreq(dtemplates{i}, 30, rejectf, freq);
    end;

    % STFT's
    stftaudio = spectrogram(faudio, window, overlap, window, freq);
    for i=1:length(ftemplates);
        stfttemplates{i} = spectrogram(ftemplates{i}, window, overlap, window, freq);
    end

    % Pitch matching
    indexes = pitch_index( stftaudio, stfttemplates );

    % Correct frequencies changes under delta
    indexes = correct_delta(indexes, window, overlap, freq, delta);

    % Pitch name and time determination
    notes = pitch_det(indexes, window, overlap, freq, names);
end
```

Listing 1: General program to perform pitch recognition

```
function [ out ] = correct_delta( indexes, window, overlap, fs, delta )
    ti = 0.0;
    tf = 0.0;
    tc = 0.0;
    pi = 1;
    pf = 1;
    ln = indexes(1);
    for i=1:length(indexes);
        if indexes(pi) ~= indexes(i);
            tf = ((i-1)*window - (i-1)*overlap)/fs;
            if tf - ti < delta;
                indexes(pi:i-1) = ln;
                ti = tc;
                pi = i; %%%
            else
                tc = ti;
                ln = indexes(i-1);
                ti = tf;
                pi = i;
            end
    end
    out = indexes;
end
```

Listing 2: Program to correct frequency variations ocurring in less than a delta interval

REFERENCES

[1] Charles Dodge, Thomas a. Jerse *Computer Music* Thomson Learning, Second Edition, 1997.

[2] Ken Steiglitz *A Digital Signal Processing Premier* Addison Wesley Publishing Company, 1996.

[3] Iannis Xenakis, *Formalized Music* Pendragon Press, Revised Edition, 1991.

[4] Guerino Mazzola, *La Vérité du Beau Dans la Musique* Delatour France Editions, 2007.

[5] David Gerhard, *Pitch Extraction and Fundamental Frequency: History and Current Techniques* Department of Computer Science - University of Regina, November 2003.

[6] Patricio de la Cuadra, Aaron Master and Craig Sapp, *Efficient Pitch Detection Techniques for Interactive Music* Internation Computer Music Conference, 2001.

[7] Salim Perchy *Lab 3 - DSP Course* Pontifica Unviersidad Javeriana, 2012.

[8] The University of New South Wales, *Note Frequencies*. School of Physics - Faculty of Science, http://phys.unsw.edu.au/jw/graphics/notes.GIF Retrieved on 2012.