Audio Engineering Society

Convention Paper

Presented at the 116th Convention
2004 May 8–11 Berlin, Germany

A Methodology for Detection of Melody in Polyphonic Musical Signals

Rui Pedro Paiva[1], Teresa Mendes [2] and Amílcar Cardoso[3]

[1] CISUC (Center for Informatics and Systems of the University of Coimbra), Department of Informatics Engineering, University of Coimbra (Polo II), Pinhal de Marrocos, P3030, Coimbra, Portugal
ruipedro@dei.uc.pt

[2] CISUC (Center for Informatics and Systems of the University of Coimbra), Department of Informatics Engineering, University of Coimbra (Polo II), Pinhal de Marrocos, P3030, Coimbra, Portugal
tmendes@dei.uc.pt

[3] CISUC (Center for Informatics and Systems of the University of Coimbra), Department of Informatics Engineering, University of Coimbra (Polo II), Pinhal de Marrocos, P3030, Coimbra, Portugal
amilcar@dei.uc.pt

ABSTRACT

We present a bottom-up method for melody detection in polyphonic musical signals. Our approach is based on the assumption that the melodic line is often salient in terms of note intensity (energy). First, trajectories of the most intense harmonic groups are constructed. Next, note candidates are obtained by trajectory segmentation (in terms of frequency and energy variations). Too short, low-energy and octave-related notes are then eliminated. Finally, the melody is extracted by selecting the most important notes at each time, based on their intensity.

We tested our method with excerpts from 12 songs encompassing several genres. In the songs where the solo stands out clearly, most of the melody notes were successfully detected. However, for songs where the melody is not that salient, the algorithm performed poorly. Nevertheless, we could say that the results are encouraging.

1. INTRODUCTION

As a result of recent technological innovations, there has been a tremendous growth in the Electronic Music Distribution (EMD) industry. Factors like the widespread access to the Internet, bandwidth increasing in domestic accesses or the generalized use of compact audio formats with CD or near CD quality, such as mp3, have given a great contribution to that boom. Presently, it is expected that the number of digital music archives, as well as their dimension, grow significantly in the near future, both in terms of music database size and in number of genres covered.

However, any large music database, or, generically speaking, any multimedia database, is only really useful if users can find what they are looking for in an efficient manner. Today, whether it is the case of a digital music library, the Internet or any music database, search and retrieval is carried out mostly in a textual manner, based on categories such as author, title or genre. This approach leads to a certain number of difficulties, namely in what concerns database search in a transparent and intuitive way. Therefore, in order to overcome the limitations described, research is being conducted in an emergent and promising field called Music Information Retrieval (MIR).

Query-by-humming (QBH) [1, 4, 6] is a particularly intuitive way of searching for a musical piece, since melody humming is a very natural habit of humans. Therefore, several technologies have been developed that aim to permit such function. However, presently, this work is being carried out only in the MIDI realm, which places important usability questions. In fact, usually we look for recorded songs, which can be obtained from CDs or are stored in audio formats such as mp3. Looking for musical pieces in the MIDI format is a much easier problem, since this is a symbolic format where all the notes, as well as their timings, are already available. The main issues are, then, to extract the notes from the hummed query (a well-known monophonic pitch extraction problem,) and to match the query to the melody, usually available in the MIDI file (an information retrieval problem).

Querying "real-world" polyphonic recorded musical pieces requires that some sort of melody representation be extracted beforehand, which creates many more difficulties. Polyphonic musical signals can be converted to symbolic formats either manually or automatically. Manual conversion requires, obviously, a tremendous amount of man-work and specialized skills. On the other hand, analyzing polyphonic musical signals is a rather complex task, since we can have many different types of instruments playing at the same time, whose spectra interfere severely with each other. This fact makes it very complicated to separate the different sound sources.

Source separation is a major concern for polyphonic music analysis and automatic music transcription systems, and has no general solution yet. One way to approach this problem is to build computer models that emulate human auditory processing. The human brain processes auditory information in a way called "auditory scene analysis" [3]. As an attempt to replicate human behavior, some work has been carried out aiming to develop computational auditory scene analysis systems. The results obtained are not very accurate yet and are only acceptable for simpler or well-constrained problems. Namely, Ellis [5] tries to analyze a sound signal by means of competitive theories, where each of them proposes a combination of sounds that might have produced the resulting sound. Sound source models are used as a basis for the proposed method. Bello *et al* [2] and Martin [9] have used computational blackboard systems for simple automatic music transcription. The blackboard system is composed of a global database, where hypotheses are proposed and developed, a scheduler that determines how hypotheses are developed, and knowledge sources, corresponding to experts. Scheirer [11] proposes a model based on perceptual issues, using dynamic clustering of comodulation data. In contrast to the other systems referred, this model is designed for analysis of complex music. Klapuri [8] proposed a method for multi-pitch estimation where the musical signal is analyzed at separate frequency bands. Namely, 18 logarithmic distributed bands from 50 Hz to 6 kHz are used. Then at each band, a fundamental frequency likelihood vector is calculated. Finally, the results from each band are combined to yield global pitch likelihoods. They report results that outperform the average of ten trained musicians. Other models impose constraints in the number of instruments present or the harmonic interaction between them, as referred in [7].

Melody detection can be seen as a sub-problem of polyphonic pitch detection and source separation, where the aim to detect the main melodic line, regardless of the other sources present. This requires the detection of the dominant notes at each time, not the whole set of notes present. For instance, when we hear a pop song, we have vocals, guitar, bass, percussion and so forth. Yet, in spite of all that information, our brains still can retain the main melodic line.

Only little work has been carried out in the particular problem of melody detection in "real-world" songs. One interesting approach is the one followed by Goto [7]. The author uses a probabilistic model for the detection of melody and bass lines. The signal is first band-pass filtered and then a probability density function (*pdf*) is computed for each signal component. The *pdf*s are generated from a weighted-mixture of tone models of all possible fundamental frequencies. The more dominant a model is in the PDF, the more likely the fundamental frequency belongs to that model. The author compared the dominant frequencies detected with hand-labeled marked notes and reports an average rate of 88.4% for the melodic pitch line.

Song *et al* [14] use a different approach, based on the fact that there is no single method that is both accurate and generic. They argue that their method is more pragmatic when the final goal is QBH: instead of trying to extract the melody, they use a mid-level melody representation, which consists of a sequence of audio segments where each segment contains a set of note candidates. Then, they use a variation of dynamic programming for matching the query with the melody mid-level representation.

In this paper, we describe a multi-stage bottom-up method for melody detection, based on sinusoid tracks [12]. Our approach is laid on the assumption that the melodic line is often composed of the notes with the highest intensity (energy). In short, the method works as follows. The signal is first divided into frames where stationarity can be assumed. For each frame, we get the magnitude spectrum and find its spectral peaks. Then, we define harmonic groups, composed of harmonically related peaks, and compute the energy of each harmonic group. We create group trajectories, formed by connecting consecutive groups with similar fundamental frequencies. After trajectory creation, note candidates are obtained by trajectory segmentation and elimination. Short-duration, low-energy and octave-related notes are then eliminated. Finally, the melody is extracted by selecting the most important notes at each time, based on their intensities.

We tested our system on excerpts of 12 songs, encompassing several different genres. The obtained notes were then compared with the correct ones, previously hand-labeled. In the songs where the solo stands out clearly, most of the melody notes were successfully detected. However, for songs where the melody is not that salient, the algorithm performed poorly. Yet, we could say that the preliminary results presented are encouraging.

The following sections describe the work carried out in this paper. Section 2 describes the melody detection method. In Section 3, experimental results are presented and evaluated. Finally, in Section 4, conclusions are drawn and possible directions for future work are pointed out.

2. MELODY DETECTION METHOD

Our melody detection algorithm is composed of five modules, illustrated in Figure 1.

The first module, harmonic group detection (HGD), receives a raw polyphonic music signal and returns a set of candidate harmonic groups, composed of harmonically related spectral peaks, and their respective energies. Then, group trajectories, formed by connecting consecutive groups with similar base frequencies, are created in the harmonic group trajectory construction (HGTC) module.

The resulting trajectories are then segmented, based on frequency and energy variations, leading to an initial set of candidate notes. Since many of the obtained notes are irrelevant, short-duration, low-energy and octave-related notes are eliminated. Finally, the notes comprising the detected melody are extracted by selecting the most intense notes at each time.

For the sake of visualization simplicity, we will illustrate the method with a simple example: a monophonic saxophone riff.

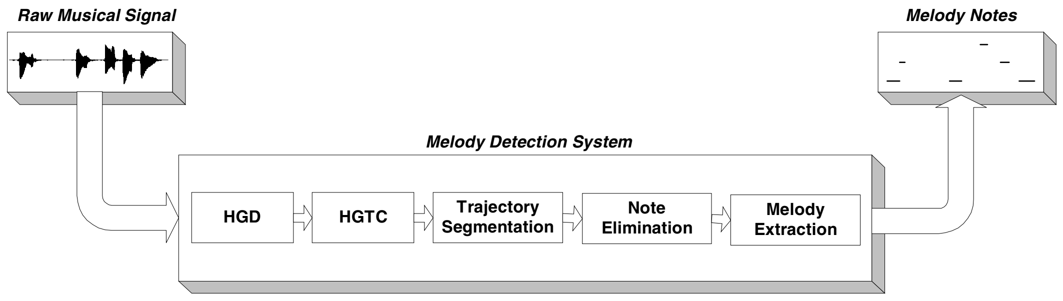

Figure 1: Melody detection system overview.

2.1. Harmonic Group Detection

In the first stage of the algorithm, the goal is to capture a set of candidate harmonic groups, which constitute the basis of possible future notes. The HGD algorithm receives as input a raw music signal (monaural, sampling frequency f_s = 22050 Hz, 16 bits quantization) and outputs a set of harmonic groups, characterized by their fundamental frequencies and energies.
Since we are interested in the spectral content of the sound wave, the signal must be represented in the frequency domain. A typical way to accomplish this task is by applying the Discrete Fourier Transform (DFT) to the temporal signal. However, the DFT is only suited for stationary signals, i.e., signals that always have the same frequency content throughout time. This calls for a windowed version of the DFT: the Short-Time Frequency Transform (STFT) [13]. Here, the main idea is to divide the signal into a set of time frames and calculate the DFT for each of those frames. We define a frame length of 20 ms, so that stationarity can be assumed. This length is also usually considered a good trade-off between temporal and frequency resolution. The corresponding number of samples per frame is N = 441. In order to allow for a smooth transition between frames, 50% frame overlap is employed.

A simple division of the signal into frames is the same as multiplying it by a sliding rectangular window. However, this window leads to significant spectral leakage. Therefore, we use a Hamming window instead, which is characterized by a good trade-off between spectral resolution and leakage [13].

In order to reduce the frequency interval, the frame data is zero-padded. Zero-padding does not improve resolution but improves single peak location accuracy, which is important for obtaining more accurate peak frequencies. Furthermore, the DFT is performed more efficiently with the Fast Fourier Transform (FFT) algorithm, which is optimized for speed when the number of samples is a power of 2. Therefore, we add the number of zeros that is necessary to obtain 4096 samples, leading to a frequency interval of 5.38 Hz, which seems adequate. In fact, the melody is usually in a mid-range frequency, above 100 Hz. Since the frequency difference between A2 (110Hz) and A2# (116.54 Hz) is above our threshold, the defined interval seems appropriate. Other authors improve peak location accuracy by spectral peak interpolation [10, 12].

After defining a windowed, zero-padded frame, its magnitude spectrum is obtained via the FFT. Additionally, we convert the spectrum to dB units, taking its logarithm. The reason for doing this is that we found experimentally that spectral peaks show up more clearly in the logarithmic magnitude spectrum.

Next, we find peaks in the magnitude spectrum, based on the assumption that the fundamental frequencies present in the signal correspond to clear peaks in the spectrum. However, unlike Serra [12], we do not find only the most prominent peaks. Instead, we look for all spectral peaks. This is motivated by the observation that, sometimes, there are important peaks whose prominence is reduced due to spectral interference from the spectra of other sources. As a consequence of looking for all peaks, their resulting number will be significantly higher. However, this is not a major concern, since, as we will refer shortly, we keep only the most important harmonic groups.

After detecting all spectral peaks, we obtain a set of candidate harmonic groups, found by grouping together harmonically related peaks. These groups are characterized by their fundamental frequencies and energies. We start by finding the highest spectral peak. As we stated before, many detected peaks are not considered. This is accomplished by defining a threshold for the minimum peak amplitude. Thus, we define the *minPeakDifference* parameter, which determines the minimum peak amplitude in comparison to the maximum amplitude peak found (see Algorithm 1). Only the peaks that keep this requirement are considered as fundamental frequency candidates. Then, for each candidate fundamental frequency, *ff*, we find all the peak frequencies that are harmonically related to it. A given peak is in harmonic relation to a candidate peak if its frequency is in the range (1):

$$\left[\frac{k \cdot ff}{r}; (k \cdot ff) \cdot r\right], \quad r = 2^{\frac{0.5}{12}} \tag{1}$$

where r is the ratio for calculating frequencies corresponding to half semi-tones variations and k stands for the k-th harmonic of the fundamental frequency *ff*.

Once we have got all the harmonic groups, we compute their respective energies by summing up the amplitudes of the peaks belonging to each group. Since we took the logarithmic magnitude of the spectrum, we convert peak magnitudes back to their original values by inverting the logarithm. Next, only the groups that have enough energy are kept. To accomplish this task, we compute the maximum group energy, *maxHGEn*, and obtain the minimum allowed group energy, *minHGEn*, using the minimum energy ratio parameter, *minEnRatio*. We then eliminate all the groups whose energies are below this threshold. Finally, the energies of the harmonic groups in all frames are normalized to the [0; 100] interval.

This algorithm is summarized in Algorithm 1. Parameter definition is presented in Table 1.

```
1. Get frame data, $x$
    1.1. Multiply frame by Hamming window
    1.2. Zero-pad accordingly
2.Get the logarithmic magnitude spectrum
    2.1. $X = 20\log_{10}[\mathrm{FFT}(x)]$
3. Detect spectral peaks
4. Get candidate harmonic groups
    4.1. Determine minimum allowed peak value
         found
         - maxPeak ← maximum peak value
         - minPeak ← maxPeak - minPeakDifference
    4.2. For all peaks
        4.2.1. If peak magnitude ≥ minPeak
            a) Get peak frequency
            b) Create harmonic group
               - Get all harmonics peaks of the base
                 peak frequency
            c) Calculate harmonic group energy
               - Group energy ← sum of all peak
                 amplitudes
    4.3. Keep only groups with enough energy
        4.3.1. Determine minimum allowed group
               energy
               - maxHGEn ← maximum harmonic
                 group energy
               - minHGEn ← maxHGEn × minEnRatio
        4.3.2. If group energy < minHGEn, eliminate
               group
    4.4. Normalize harmonic group energies to the
         [0; 100] interval
5. Return harmonic group fundamental frequencies and
   energies
```

Algorithm 1: Harmonic group detection.

The described procedure is illustrated in Figure 2. The top picture presents a 20ms temporal frame of a monophonic saxophone riff, as referred above, and the bottom one shows the corresponding logarithmic magnitude spectrum, with the candidate harmonic group peaks marked.

Parameter Name	***Parameter Value***
frame length	20 ms
FFT size	4096
frame overlap	50%
window type	Hamming
minPeakDifference	35dB
minEnRatio	0.2

Table 1: HGD parameters.

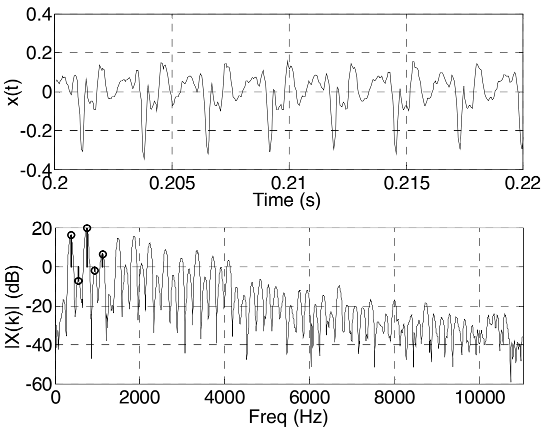

Figure 2: Illustration of the HGD algorithm.

Unlike automatic music transcription systems, this algorithm does not deal with the well known and complex, "octave problem". In fact, at this stage it is not important to analyze if a given harmonic group corresponds to a real note or appears as a ghost note, whose fundamental frequency is a harmonic of some real note, a few octaves below. Some of the ghost notes will be eliminated already at this stage based on the energy threshold for harmonic groups, whereas others will be eliminated in the following stages of the melody detection algorithm.

2.2. Trajectory Construction

The second stage of the melody detection algorithm aims at creating a set of harmonic group trajectories, formed by connecting consecutive groups with similar fundamental frequencies. The idea is to find regions of stable fundamental frequencies, which indicate the presence of musical notes. The HGTC algorithm receives as input a set of harmonic groups, characterized by their fundamental frequencies and energies, and outputs a set of group trajectories, which constitute the basis of the final melody notes.

We follow rather closely Serra's peak continuation algorithm [12]. However, our algorithm has one, yet significant, difference with the original one: instead of finding regions of stable sinusoids, we only find regions of stable *fundamental frequency candidates*, in the same way as Martins does [10]. Therefore, though the basic idea is the same, our algorithm is much lighter than Serra's. Another important point is that we first quantize frequencies to the closest MIDI note. We found experimentally that the algorithm performed better this way. One reason for this seems to come from the fact that the location of peaks oscillates somewhat due to spectral interference from other sources and, possibly, leakage effects. We found that peak continuation based on MIDI note numbers allows for a more robust trajectory build up. Furthermore, the representation of notes using MIDI numbers simplifies an eventual representation of the signal in MIDI format (e.g., for generation of a MIDI file).

The trajectory construction algorithm is described with detail in Algorithm 2.

This algorithm is based on three parameters, presented in Table 2. The first one, *maxSTDev*, represents the maximum frequency deviation in semi-tones for continuing trajectories. We assign it a value of one semi-tone, motivated by the fact that some songs comprise glissando and vibrato regions, as well as by the frequency oscillations that result from spectral interference. Therefore, in this way, all these phenomena are kept within a common trajectory, instead of being separated into a number of different trajectories, e.g., one trajectory for each note that one glissando may traverse. The drawback of allowing a larger frequency deviation is that a single trajectory can contain more than one note. This is the reason why we perform trajectory segmentation, in the next stage of the melody detection algorithm.

```
1. Quantize frequencies to the closest MIDI note
   numbers
2. Create initial trajectories
      2.1. Use the note numbers, peak frequencies and
           amplitudes of the harmonic groups in the first
           non-empty frame
3. For all frames
      3.1. Get the note numbers of all harmonic groups
           in the current frame
      3.2. For all non-finished trajectories
           3.2.1. Get all the note numbers in the current
                  frame that may continue the present
                  trajectory, i.e., note numbers that are
                  within the maxSTDev range, comparing
                  to the last note number in the current
                  trajectory
           3.2.2. Sort possible note numbers in ascending
                  distance order
           3.2.3. For all possible note numbers and while
                  no continuation found:
                  a) If the present note number does not
                     have any closer trajectory, use it to
                     continue the present trajectory (in
                     case of tie, use average note numbers)
                  b) Otherwise, try the next note number
           3.2.4. If the trajectory is continued
                  a) Update trajectory length
                  b) Add note number, peak frequency and
                     amplitude to it
           3.2.5. Otherwise
                  a) Increment the number of inactive
                     frames, numSleep
                  b) If numSleep ≥ maxSleepTime
                     Stop trajectory
      3.3. For all stopped trajectories
           3.3.1. If length < minTrajLen
                  a) Eliminate trajectory
           3.3.2. Otherwise
                  a) Mark trajectory as finished
      3.4. For all non-continued note numbers
           3.4.1. Create new trajectory, initialized with
                  the present note number, peak frequency
                  and amplitude
4. Return all finished trajectories
```

Algorithm 2: Harmonic group trajectory construction.

The second parameter, *maxSleepTime*, specifies the maximum number of frames where a trajectory can be inactive, i.e., when no continuation peaks are found. If this number is exceeded, the trajectory is stopped.

The last parameter, *minTrajLen*, controls the minimum trajectory length. Therefore, all finished trajectories that are shorter then this threshold, are eliminated.

Parameter Name	***Paramater Value***
maxSTDev	1 semi-tone
maxSleepTime	5
minTrajLen	9

Table 2: HGTC parameters.

The described procedure is illustrated in Figure 3, for our saxophone riff example. There, we can see that some of the obtained trajectories comprise glissando regions. Also, some of the trajectories include more than one note and should, therefore, be segmented.

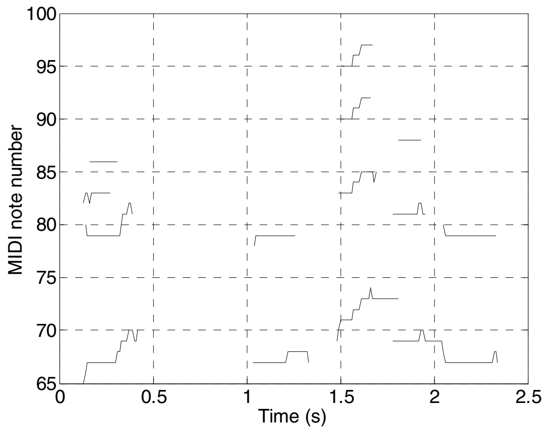

Figure 3: Illustration of the HGTC algorithm.

2.3. Trajectory Segmentation

As we mentioned previously, the trajectories that result from the HGTC algorithm may contain more than one note and, therefore, must be segmented. This is the task of the third stage of the melody detection method. The trajectory segmentation algorithm receives as input a set of harmonic group trajectories and outputs a set of segmented trajectories, i.e., note candidates.

```
1. Set initial parameters for splitting
    1.1. lastNoteNumber ← first note number in the trajectory
    1.2. currentNoteDuration ← 1
2. For all frames covered by the trajectory
    2.1. noteNumber ← current frame note number
    2.2. If noteNumber ≠ lastNoteNumber (possible note transition)
        2.2.1. Find a sequence of frames with note noteNumber
            a) numCons ← number of consecutive frames where noteNumber is kept
        2.2.2. finalNoteNumber ← note number corresponding to the frame where the sequence in 2.2.1. changed
        2.2.3. If numCons > minTrajLen and currentNoteDuration > minTrajLen
            a) split trajectory
        2.2.4. Otherwise, if finalNoteNumber ≠ lastNoteNumber and currentNoteDuration ≥ minTrajLen
            a) split trajectory
        2.2.5. Otherwise
            a) Sequence is noise or modulation → do not split
            b) Advance to the next frame
            c) Increment currentNoteDuration or reset it if finalNoteNumber ≠ lastNoteNumber (possible glissando)
            d) lastNoteNumber ← noteNumber
    2.3. If split trajectory
        2.3.1. Add current frame number to the segmentation indexes vector
        2.3.2. currentNoteDuration ← 1
    2.4. Otherwise
        2.4.1. Increment currentNoteDuration
    2.5. lastNoteNumber ← noteNumber
3. Split trajectory based on the saved segmentation indexes
4. For each sub-trajectory, i.e., note candidate
    4.1. trajectoryNoteNumber ← most frequent note in all the frames of the trajectory (in case of tie, select the one with the highest energy)
5. Return segmented note candidates
```

Algorithm 3: Trajectory frequency segmentation.

Two types of segmentation have to be conducted. The most intuitive one is frequency segmentation, where the goal is to separate all the different frequency notes that are present in the same trajectory. The other one, energy segmentation, aims at separating consecutive notes that have the same fundamental frequencies, which the HGTC algorithm may have interpreted as forming only one note. This requires segmentation based on energy minima, which mark the limits of each note.

Algorithm 3 describes the procedure carried out for frequency segmentation. The main idea is to find sufficiently long sequences of the same note number. Only then trajectories are segmented. When note transitions are found but the current note sequence is not long enough, i.e., larger than *minTrajLen* (see Table 2), the trajectory is not segmented, since it may correspond to the start of a glissando region. Furthermore, when we find short sequences delimited by the same note number, e.g., {70, 71, 71, 71, 71, 70}, these are interpreted as possible modulation regions, and so no segmentation takes place.

After frequency segmentation, the obtained candidate notes must be analyzed so as to check whether they should be further divided. In fact, there may be consecutive distinct notes at the same fundamental frequency that, erroneously, form a unique long note. In this situation, those notes must be divided. In order to accomplish this task, energy segmentation takes place, according to Algorithm 4. The main idea is to find clear energy minima that suggest the presence of more than one note. This is implemented using a recursive procedure.

The algorithm uses two parameters: *maxVpr* and *minVpd* (Table 3). The *maxVpr* parameter defines the maximum valley-peak ratio, i.e., the maximum ratio between some local minimum and local maximum under analysis. Its value is set to 0.5, i.e., any minimum must be at most half the maximum under consideration. As for the *minVpd* parameter, it represents the minimum absolute valley-peak distance and is set to 20 (recall that the energy values were normalized to the [0; 100] interval). The goal of this parameter is to prevent erroneous segmentations when local minima have very small values, e.g., minimum value is 5 and maximum value is 20.

Finally, after all notes are segmented, their onset and offset times are adjusted. For each note, we get its maximum energy value and then define the onset as the first frame were the energy rises above 10% of the maximum energy found, a value defined in the onset energy threshold parameter, *onsetEnThr*.

```
1. Find all local maxima and minima in the trajectory
   energy
2. Create empty list of frame indexes for segmentation
    2.1. segmIndexes ← { }
3. Call the recursive procedure Energy Segmentation,
   with the obtained maxima and minima
    3.1. segmIndexes ← Energy Segmentation (local
         maxima and minima, segmIndexes, minPvr,
         minPvd)
        3.1.1. The procedure returns the final list of
               frame indexes for segmentation,
               segmIndexes
4. Split trajectory based on the obtained segmentation
   frame indexes
5. Return segmented note candidates
```

```
Energy Segmentation

1. Find the global minimum
    1.1. globMin ← value of the global minimum
2. Find left and right side maxima, and respective
   distances and ratios regarding globMin
    2.1. Left side:
        2.1.1. maxL ← maximum peak to the left of
               the global minimum
        2.1.2. distL ← | globMin – maxL |
        2.1.3. ratioL ← globMin / maxL
    2.2. Right side:
        2.2.1. maxR ← maximum peak to the right of
               the global minimum
        2.2.2. distR ← | globMin – maxR |
        2.2.3. ratioR ← globMin / maxR
3. If $ratioL \leq maxVpr$ and $distL \geq minVpd$ and $ratioR \leq$
   $maxVpr$ and $distR \geq minVpd$
    3.1. Mark global minimum frame number as
         segmentation point
        3.1.1 Add it to segmIndexes list
4. Call Energy Segmentation for left and right vectors
    4.1. Divide local maxima and minima vectors into
         two sub-vectors: left side and right side of the
         minimum
    4.2. Call Energy Segmentation with the left vector
    4.3. Call Energy Segmentation with the right
         vector
```

Algorithm 4: Trajectory energy segmentation.

The procedure is the same for the offsets, i.e., the note offset corresponds to the first frame where the energy rises above 10% of the maximum energy, starting from the end.

Parameter Name	*Parameter Value*
maxVpr	0.5
minVpd	20
onsetEnThr	0.1

Table 3: Trajectory energy segmentation parameters.

Figure 4 illustrates trajectory segmentation, using the initial trajectories from the HGTC algorithm (Figure 3). The obtained notes are depicted with thick lines. We can see that glissando and modulation regions are properly dealt with, except for two notes that start approximately at time 1.5s. In fact, in those notes the final fundamental frequency (ff) has a lower number of occurrences than the initial one, because the steady part of the note (final ff) is shorter than the glissando region. This happened because the HGTC algorithm could not find any additional peaks for continuing them, as a result of their low energies compared to the most intense note. Anyhow, this is not a problem since these notes will not make part of the final melody because of their low energy levels. Furthermore, this situation only occurs in the upper harmonics of very short notes with glissando, which reduces its relevance.

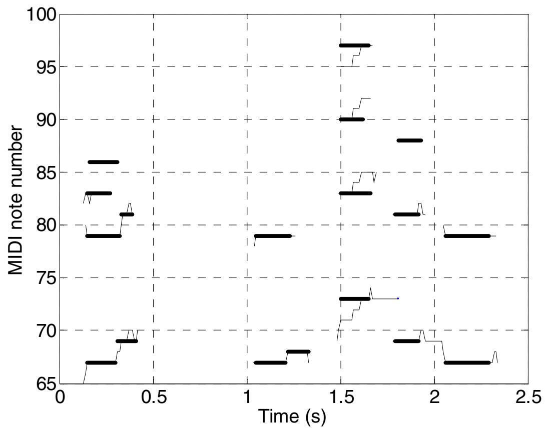

Figure 4: Illustration of the trajectory segmentation algorithm.

2.4. Note Elimination

The objective of the fourth stage of the melody detection algorithm is to delete some of the note candidates, based on their energies, durations and on the analysis of octave relations. The note elimination algorithm receives as input a set of note candidates and outputs a reduced set of notes, relevant for melody extraction.

First, low-energy notes are deleted. A note is low-energy if its average energy is below *minAvgNoteEnergy* and if the number of frames whose energy is above that threshold is not enough, i.e., below *minNumSuffEnergy*. Next, all the notes whose duration is below *minTrajLen*, i.e., which are too short, are also deleted. Finally, we look for octave relations between all notes. If two notes have approximately the same onset and offset times and are harmonically related, it is possible that the higher one is just a harmonic of the lower one. Therefore, we compare their respective energy levels in order to take a decision: if the energy of the higher note is less than half the energy of the lower note, the higher one is eliminated. The octave energy threshold is defined in the *minOctRatio* parameter.

```
1. For all note candidates
    1.1. Eliminate low-energy notes
        1.1.1. If average energy < minAvgNoteEnergy
               and number of frames above
               minAvgNoteEnergy <
               minNumSuffEnergy
               a) Delete note
    1.2. Eliminate short-duration notes
        1.2.1. If note length < minTrajLen
               a) Delete note
    1.3. Eliminate octaves
        1.3.1. Look for harmonically related notes
               with common onsets and offsets
               (tolerance for onset and offset deviation
               is maxSleepTime)
        1.3.2. If higher note energy < lower note
               energy $\times$ minOctRatio
               a) Delete note
2. Return reduced set of notes.
```

Algorithm 5: Note elimination.

This algorithm is summarized in Algorithm 5. Parameter definition is presented in Table 4. Figure 5 illustrates note elimination, based on the note candidates of Figure 4. The obtained notes are depicted with thick lines. We can see that many of the note candidates are eliminated at this point.

Parameter Name	*Parameter Value*
minAvgNoteEnergy	10
minNumSuffEnergy	5
minOctRatio	0.5

Table 4: Note elimination parameters.

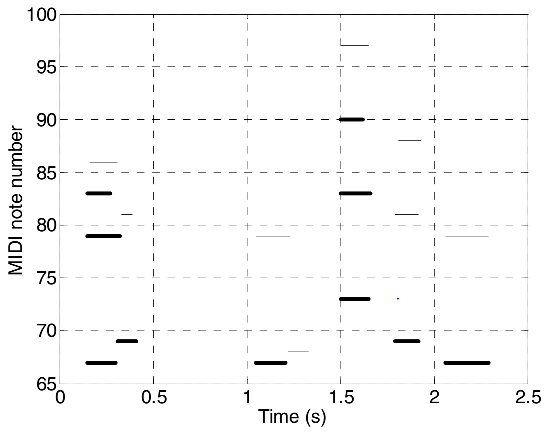

Figure 5: Illustration of the note elimination algorithm.

2.5. Melody Extraction

In the final stage of the present melody detection system, our goal is to obtain a final set of notes comprising the melody of the song under analysis. The melody extraction algorithm receives as input the set of notes returned by the note elimination algorithm and outputs the final melody notes.

This stage of the proposed melody detection system, being probably the most important one, is also the most difficult one to carry out. In fact, many aspects of auditory organization influence the perception of melody by humans, for instance in terms of the role played by the pitch, timbre and intensity content of the signal. In our approach, we do not attack the problem of source separation, as would normally be the case. Instead, we base our strategy on the assumption that the main melodic line is often salient in terms of note intensity. As of now, the algorithm for melody extraction is just a preliminary one that needs to be further worked out. Anyway, it showed promising results, as will be discussed shortly.

```
1. Segment signal based on time intersections between
   consecutive notes
2. For all segments
      2.1. Get average energy of each note in the current
           segment
      2.2. Delete low frequency notes
           2.1.2. If noteNumber < minNoteNumber
                  a) Delete note
      2.3. Keep only the numTop most intense notes
           2.3.1. Sort notes by descending average
                  energy order
           2.3.2. Keep first numTop notes
3. Delete non-dominant notes
      3.1. For all notes
           3.1.1. durationNumTop ← number of frames
                  where the current note is in the numTop
                  most intense notes
           3.1.2. durationFirst ← number of frames
                  where the current note is the most
                  intense one
           3.1.3. lenNote ← total number of frames of the
                  current note
           3.1.4. If durationNumTop / lenNote <
                  minPercDur or
                  durationFirst < minTrajLen
                  a) Delete note
4. Do not allow any simultaneous notes
      4.1. Truncate notes that end after the next note
           starts
      4.2. Delete notes included in larger duration notes
      4.3. For notes with approximate onsets and offsets,
           keep only the most intense one (tolerance for
           onset and offset deviation is maxSleepTime)
5. Return final melody notes.
```

Algorithm 6: Melody Extraction.

This algorithm starts by analyzing intersections between notes. The beginning and end of intersection regions is used to segment the signal, as illustrated in Figure 6, where s_i stands for the i-th obtained segment.

Then, for each segment, we determine the three most intense notes (set on the *numTop* parameter), based on the average energy of each note in each segment. Notes below MIDI note number 50 (143.83 Hz), a value set in the *minNoteNumber* parameter, are excluded. This procedure is motivated by the fact that the notes comprising the melody are, usually, in a middle frequency range.

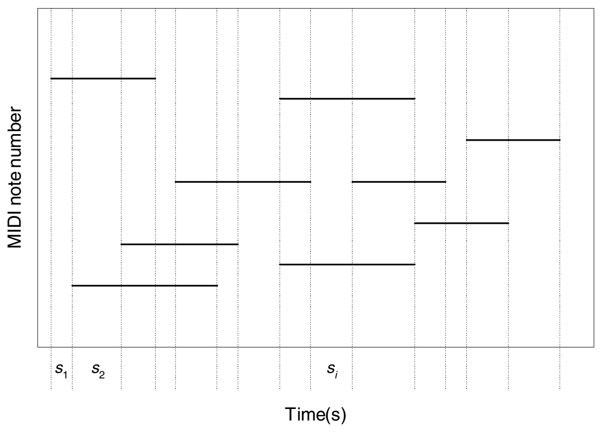

Figure 6: Segmentation based on note intersection.

Next, we eliminate all the notes that are not dominant, i.e., that are not in the three most intense notes for more than 2/3 (*minPercDur* parameter) of their total number of frames or do not have the highest energy for more than *minTrajLen* frames. Finally, we do not allow simultaneous notes. Therefore, we truncate notes that end after the next note starts, eliminate notes included in larger duration notes and, for notes with approximate onsets and offsets, keep only the most intense one.

The melody extraction algorithm is summarized in Algorithm 6. Parameter definition is presented in Table 5. Figure 7 illustrates melody extraction, based on the example in Figure 5. The final melody notes are depicted with thick lines.

Parameter Name	*Parameter Value*
MinNoteNumber	50
NumTop	3
MinPercDur	2/3

Table 5: Melody extraction parameters.

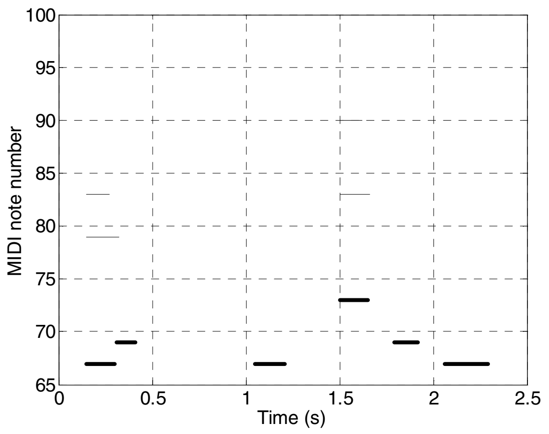

Figure 7: Illustration of the melody extraction algorithm.

3. EXPERIMENTAL RESULTS

One difficulty regarding the evaluation of MIR systems results from the absence of standard test collections and benchmark problems. Therefore, we created our own test database, having care regarding its diversity and musical content. We collected excerpts of about 6 seconds from 12 songs, encompassing several different genres. The selected songs contain a solo (either vocal or instrumental) and accompaniment parts (guitar, bass, percussion, other vocals, etc.).

The obtained results are summarized in Table 6. There, "V" stands for vocals and "I" stands for instrumental. Figure 8 shows an example of the results of the melody detection system for an excerpt of the song "Thank You", by Dido. In this example, we can see that the correct notes (thick lines) match the obtained melody notes (thin continuous lines) in most of the cases. The undetected notes are marked with circles. As can be seen, the three missing notes were present in the notes obtained after elimination (dotted lines). One of the missing notes, approximately at time 5s, corresponds to erroneous trajectory segmentation.

The detected melody notes were compared with the correct notes, previously hand-labeled. In the absence of the melody line, the system detected the dominant accompaniment part, since sound sources are not discriminated. This can be seen in Figure 8, by the thin continuous lines. This is consistent with the way humans seem to memorize melodies: a mix of solo regions with accompaniment regions, in the absence of a solo.

Song Title	*Genre*	*Solo Type*	*#Total Notes*	*#Correct Notes*	*#Correct Notes after Elimination*
Pachelbel's Kanon	Classical	I	16	10 (62.5%)	15 (93.75%)
Handel's Hallelujah	Choral	V	15	n. r.	14 (93.33%)
Enya – Only Time	Neo-Classical	V	11	8 (72.72%)	10 (90,9%)
Dido – Thank You	Pop	V	16	13 (81.25%)	16 (100%)
Ricky Martin – Private Emotion	Pop	V	10	n. r.	9 (90%)
Avril Lavigne – Complicated	Pop/Rock	V	14	n.r.	11 (78.57%)
Rua Dona Margarida	Jazz	I	19	19 (100%)	19 (100%)
Mambo Kings – Bella Maria de Mi Alma	Bolero	I	12	n. r.	9 (75%)
Compay Segundo – Chan Chan	Latin	V	10	n. r.	9 (90%)
Juan Luis Guerra – Palomita Blanca	Rumba	V	10	8 (80%)	10 (100%)
Battlefield Band – Snow on the Hills	Scottish Folk	I	26	13 (50%)	26 (100%)
Saxophone riff	(monophonic)	I	6	6 (100%)	6 (100%)

Table 6: Results of the melody detection system.

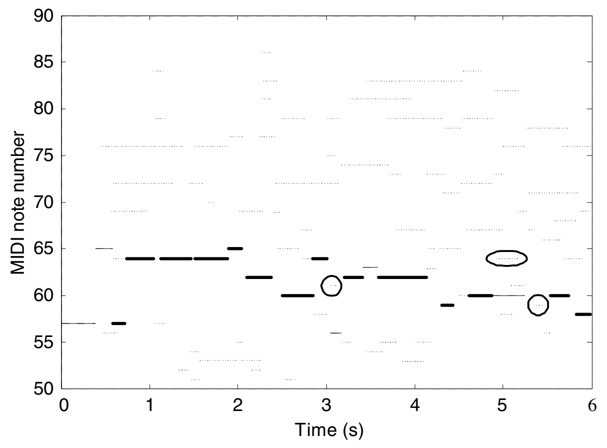

Figure 8: Detected melody for "Dido - Thank You" excerpt.

However, we decided to ignore the notes where the accompaniment part dominates, in the same way as Goto does [7]. In order to extract only the melody, we would need a means of separating notes according to their sources. The most intuitive, but complex, way to accomplish this task would be to use timbre models. Other possibilities would be to separate notes according to their frequency ranges, energy levels (since the intensity of a solo varies usually in a smooth way) or duration of notes (e.g., it is not likely that a short duration note in the middle of two long notes belongs to the same source as them).

In our test cases, we observed that some of the notes were excessively segmented and others were shorter than the original ones (fortunately, a small number of them). This resulted from noise in both the frequency and energy sequences, as well as frequency deviations in the HGTC algorithm, which lead to excessive trajectory segmentation. The noise in the energy sequences results often from spectral collision with the spectra of percussion instruments. One possible way to deal with this issue would be to smooth the frequency and energy sequences before segmentation. Another possibility would be to filter out percussion sounds from the mixture, which seems to be a challenging task. We also observed a few semi-tone deviations (once again, a small number). These errors resulted from the previous one and so should diminish after we deal with the problems coming from segmentation. We decided to ignore these small errors since our goal is to check whether a note is present or not, no matter in how many sub-notes the algorithm divides it. These errors can be reduced as was referred.

We can see that the algorithm could not find any reasonable melody in some excerpts ("#Correct Notes = n. r.: not reasonable"). However, in the cases where the melody stands clearly out of the background and percussion is not very intense, good results were achieved, which matches Goto's results [7]. There is even one jazz excerpt where all the notes were correctly detected.

In spite of the described limitations, it is interesting to note that in all of the cases tested, most of the notes comprising the melody were still present after the note elimination stage. Therefore, when QBH is a final goal, we could follow an approach similar to Song's, i.e., matching a query to the whole set of notes after note elimination, by finding a path in it [14]. This hypothesis will be evaluated in the future.

We also tested our system with a simple monophonic saxophone riff. In this, as well as other monophonic test cases not reported here, the results were very good in terms of detection of glissandos, vibratos and note onsets and offsets. Consequently, we hope our system could be used as a robust monophonic pitch detection tool.

4. CONCLUSIONS

We have presented a system for melody detection in polyphonic music signals. This is a main issue for MIR applications, such as QBH "real-world" music databases. The work conducted in this field is presently restricted to the MIDI realm, and so we guess we make an interesting contribution to the area, though our results were not very accurate and general for the time being. However, the achieved results are encouraging, since we have not exploited the full potential of our approach yet. Furthermore, to our knowledge, only Goto [7] addresses the issue of melody detection in polyphonic music, but without trying to explicitly extract notes. Also, our system is reasonably simple and light, except for the harmonic group detection module, due to the DFT analysis.

Regarding future work, we plan to further work out some of the described limitations, namely the melody extraction algorithm. Additionally, we plan to try out a different approach: evaluating Independent Component Analysis capability for source separation. The main idea would be to separate the solo and accompaniment parts and then detect the melody in the solo part.

5. ACKNOWLEDGMENTS

This work was partially supported by the Portuguese Ministry of Science and Technology (MCT), under the program PRAXIS XXI.

6. REFERENCES

[1] D. Bainbridge, C. Nevill-Manning, I. Witten, L. Smith and R. McNab, "Towards a Digital Library of Popular Music", presented at the ACM International Conference on Digital Libraries, pp. 161-169, 1999.

[2] J. P. Bello, G. Monti and M. Sandler, "Techniques for Automatic Music Transcription", presented at the First International Symposium on Music Information Retrieval, 2000.

[3] A. S. Bregman, "Auditory Scene Analysis: the Perceptual Organization of Sound", MIT Press, 1990.

[4] W. Chai, "Melody Retrieval on the Web", MSc Thesis, Massachusetts Institute of Technology, 2001.

[5] D. Ellis, "Prediction-Driven Computational Auditory Scene Analysis", PhD Thesis, Massachusetts Institute of Technology, 1996.

[6] A. Ghias, J. Logan, D. Chamberlin and B. C. Smith, "Query by Humming: Musical Information Retrieval in an Audio Database" presented at the ACM Multimedia Conference, 1995.

[7] M. Goto (2001), "A Predominant-F0 Estimation Method for CD Recordings: MAP Estimation Using EM Algorithm for Adaptive Tone Models", presented at the IEEE International Conference on Acoustics, Speech and Signal Processing, 2001.

[8] A. Klapuri, "Multipitch Estimation and Sound Separation by the Spectral Smoothness Principle", presented at the IEEE International Conference on Acoustics, Speech and Signal Processing, 2001.

[9] K. D. Martin, "Automatic Transcription of Simple Polyphonic Music: Robust Front End Processing", presented at the 3rd Joint Meeting of the Acoustical Societies of America and Japan, 1996.

[10] L. G. Martins, "PCM to MIDI Transposition", presented at the 112th Audio Engineering Convention, 2002.

[11] E. D. Scheirer, "Music-Listening Systems", PhD Thesis, Massachusetts Institute of Technology, 2000.

[12] X. Serra, "Musical Sound Modeling with Sinusoids Plus Noise", In "Musical Signal Processing", eds. C. Roads, S. Pope, A. Picialli and G. De Poli, 1997.

[13] S. Smith, "The Scientist and Engineer's Guide to Digital Signal Processing", California Technical Publishing, 1997.

[14] J. Song, S. Y. Bae and K. Yoon, "Mid-Level Music Melody Representation of Polyphonic Audio for Query-by-Humming System", presented at the International Symposium on Music Information Retrieval, 2002.