# Automatic Transcription of Musical Whistling and an Application to Melody Retrieval

Bruno Dias, Rodrigo Ventura, José Gaspar
Instituto Superior Técnico,
Universidade Técnica de Lisboa, Portugal
bfsd@ist.utl.pt; yoda,jag@isr.ist.utl.pt

***Abstract*—In the first part of this paper we describe an automatic system for transforming (transcribing) man-made melodic whistles into MIDI-like symbolic representations. Given the monophonic nature of whistling, our system is mainly based in pitch detection and tracking methodologies. In particular, we compare four pitch detection techniques: Temporal Autocorrelation Function, Average Magnitude Difference Function, Spectral Autocorrelation Function, and the Harmonic Product Spectrum. Results for both synthetic and real (man-made) whistling signals are presented in the paper, showing that the system can effectively do the transcription work. A comparative evaluation of the four pitch detection algorithms is also performed.**

**In the second part of the paper we present an application of the transcription system to the retrieval of musical themes. Users whistle a segment of a song as input, and as output they receive a list of music candidates, ranked by their relevance. In order to build an efficient and error tolerant retrieval system, the central component of the system, melody matching, is implemented based on dynamic programming. Three matching distances are presented to rank the retrieval results. Performance measure metrics show promising results in the retrieval system.**

***Index Terms*—Music transcription, pitch tracking, music search by melody, automatic music retrieval, music search engine, query-by-whistling.**

## I. Introduction

With the continuous advances in digital signal processing techniques, the automatic transcription from an acoustic waveform to a symbolic representation is now possible. Musical transcription of audio data is the process of taking a segment of digital audio data and extracting from it the symbolic information that can be represented, for instance, in a music score[1]. It can be viewed as reverse-engineering the "source code" of a music signal [7], that we do not expect that the average human are able to do easily.

The reason why we have chosen whistling as our input instrument is due to its universality and accessibility to anyone regardless of the musical background. The purpose of the present work is to develop an automatic score extraction system, using monophonic melodic whistles as input. The features used to identify each note are the pitch (*i.e.*, the fundamental frequency $F_0$), amplitude, the onset, and the duration.

Pitch is the perceived quality of a sound that is chiefly a function of its fundamental frequency [18]. Whereas pitch is a perceptual attribute evoked in the auditory system, the fundamental frequency ($F_0$) is the corresponding physical attribute defined for periodic or nearly periodic sounds only, and corresponds to the inverse of the period. Humans are said to be interval-sensitive (the difference between two pitches is called an interval), perceiving two different melodies that have the same pattern of intervals (melodic contours) to be equivalent, despite their absolute pitches.

To achieve the stated objective, the first two features (pitch and amplitude) are considered in our work. The strategy consists in tracking the pitch of the signal and correcting possible errors of the tracker (in particular note onset and duration) using the loudness information (rhythm).

Pitch identification, i.e. the quantization of the pitch frequency e.g. to the Musical Instruments Digital Interface (MIDI) scale, which assigns an integer to each note of the Western music scale, has been object of research for several decades. It is practically a solved problem now in the case of single instruments [15]. However, transcribing sounds such as that of the 'human instrument', including singing, humming and whistling, is a much more difficult task. Related work, transcribing singing monophonic music [8] and hummed tunes [1], were somehow successful. One still critical aspect is the detection of the pitch frequency.

Various algorithms can be used for pitch detection. In general these algorithms can be divided into two groups: those that look for frequency partial at harmonic *spectral locations* (in time domain), and those that observe *spectral intervals* between partials [7] (in frequency domain). The temporal Autocorrelation Function (tACF) and the Average Magnitude Difference Function (AMDF) that belongs to the first group, while the frequency Autocorrelation Function (fACF) and Harmonic Spectrum Product (HSP) to the second. In the following sections we test and compare these methods considering our the specific application transcribing whistling.

One potential application of the transcription system is in query-by-whistling systems (music search engines) that retrieve information of whistled songs, such as name and composer, by matching a query whistle with a database of songs.

Several aspects have to be considered in these search engines, such as efficiency (fast operation) while being tolerant to errors in the whistled data, but keeping some precision in the data representations, in order to return accurate results [11], [16]. In other words, building a query-by-whistling system implies defining appropriate melody representations and song

[1]`http://www.synthonia.com/artwhistling`

matching metrics. These subjects on music searching are explored in the second part of this paper, a prototype system is proposed and in the end the system's performance is evaluated.

The structure of this paper is the following. Section II describes the main blocks of the music transcription algorithm, followed by the detailed description of the pitch tracker methods in Sec.III. An experimental comparison of the pitch detection methods is presented in Sec.IV. Section V describes the framework of the music retrieval system, and Sec.VI details the finding and ranking of database melodies matching a query. Section VII shows music retrieval results. Finally, Sec.VIII presents conclusions and future work.

## II. TRANSCRIPTION SYSTEM OVERVIEW

The main algorithm has the task of translating sound to a musical score format. In this format all notes have a starting time, a duration and a pitch, listed sequentially in time.

Our transcription system has three main steps: preprocessing, pitch detection, and notes segmentation (see Fig 1). These steps are detailed next.

The first step of the transcription system consists in segmenting the sound signal into frames (time windows) of constant length. The signal envelope is then calculated for each frame in order to skip the pitch detection when the energy falls below an audibility threshold. When a silence moment is detected, a null pitch value is assigned to that frame.

**Pitch detection** assigns a fundamental frequency, $F_0$ to each signal frame. It is based on the computation of a similarity measure between a frame of the signal and delayed versions of that frame, and then finding the pitch at the maximal peak of the similarity. In the next section we detail four alternative methods for pitch detection, including one detector based not on time but on products of decimated spectrograms [21].

Given an array of pitch values, one for each signal frame, our system quantizes the pitch values to a MIDI-like quantized-scale, and applies non-linear filtering to remove pitch outliers. The non-linear filtering reduces the number of pitch-outliers within silence according to a minimum note duration parameter ($100ms$). Pitches lasting less than the minimum duration are discarded. Pitch-outliers, spanning less time than the minimum duration and detected within groups of samples with constant pitch values, are also corrected: the middle (outlying) group of pitches is assigned the value of their neighbors. This allows correcting some variations inside a tone. In order to separate the notes accurately, our system comprises also an outliers removal procedure for pitches between tones.

The **notes segmentation** process groups sequences of equal pitch values into notes. The constant pitch value characterizing each group defines the note height while the number of grouped pitch values defines the duration. Hence, the termination of a note is determined by the beginning of a new note or by the detection of silence. An energy based onset detector is not directly applied, but is used to fix some notes fragmentation at the non-linear filtering block. Finally, the pitch value of each note is converted to a MIDI-key, $K(F_0)$:

$$K(F_0) = round\left(12 \times \log_2\left(\frac{F_0}{440}\right)\right) + 69 \qquad (1)$$

For example, the A4 pitch ($F_0 = 440Hz$) corresponds to the MIDI-key number $K(F_0) = 69$. A total of 10 complete octaves (128 notes) are possible to represent, ranging from $8.176Hz$ to $13344Hz$.

## III. PITCH DETECTION METHODS

### *A. Temporal Autocorrelation*

Time-domain autocorrelation function (*tACF*) based algorithms are among the most popular $F_0$ estimators [7]. The technique consist in picking peaks in the autocorrelation function:

$$r_{xx}(n) = \frac{1}{N}\sum\nolimits_{K=0}^{N-n-1} x(k)x(k+n) \qquad (2)$$

Because a periodic signal will correlate strongly with itself when delayed by the fundamental period, the time offset $(n)$ corresponding to the highest peak in the autocorrelation will give the period $(\frac{1}{F_0})$ of the waveform. In practice, a similar and more efficient ($NlogN$ instead of $N^2$) function is used via the fast Fourier transform (FFT). This expression, based on the *Wiener-Khinchin* theorem [2], is given by:

$$r(\tau) = IDFT(|DFT(x(n))|^2) \qquad (3)$$

Another advantage of this method is its efficiency in identifying hidden periodicities, e.g. in situations with a weak fundamental. Its superior robustness to noise is another quality. Presenting a desired logarithmic resolution[2] even with a lower FFT order and window size, excellent results are possible. However, this effectiveness at mid to low frequencies could introduce errors at high fundamental frequencies, in which the range of possible fundamental frequencies is limited. The calculation of $F_0$ is performed directly from a shift of samples, and then a lower sampling rate implies a lower resolution in pitch. Another shortcoming of this method, besides this sensitiveness to the sampling rate, is its tendency to halving the correct $F_0$ in harmonic sounds. This happens because the periodicity of that signals provokes a periodicity in the *ACF*, with peaks at integer multiples of the period.

### *B. Average Magnitude Difference Function*

Average Magnitude Difference Function (*AMDF*) looks for the difference of a signal with a time lag of itself, rather than the product.

$$\psi(\tau) = \frac{1}{N}\sum_{n=0}^{N-1}|x(n) - x(n+\tau)| \qquad (4)$$

In opposition to *tACF*, there will be valleys at maximum similarity instead of peaks. Excluding the first null at time zero, the smallest minimum will correspond to the fundamental period ($T_0$). As in *tACF*, several candidates to $T_0$ are discarded, corresponding to multiples of $T_0$.

[2]Notice that in occidental representation of music the two consecutive semitones are separated by a $2^{\frac{1}{12}}$ ratio, copying the logarithmic sensibility of the human auditory system.

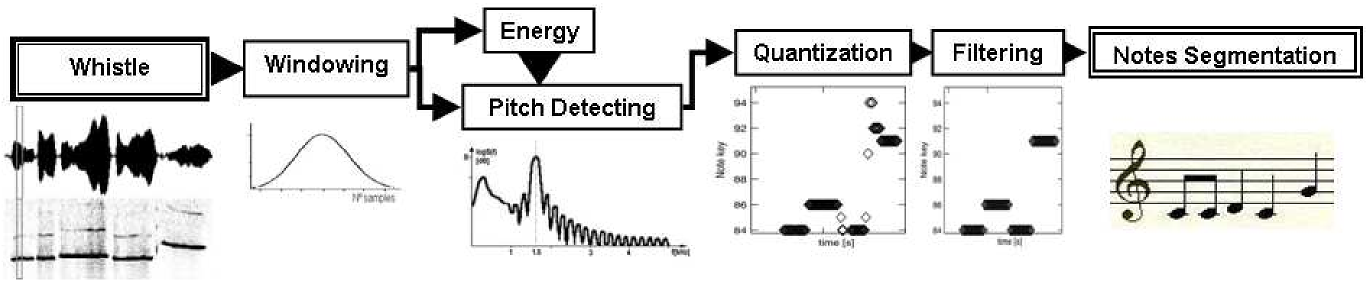


Fig. 1: Transcription system.

### C. Spectrum Autocorrelation

The frequency-domain autocorrelation function (*fACF*), as the *time-domain ACF*, has been used with success over the years in some $F_0$ estimators. But in opposition with the *tACF*, the *spectral ACF* is a spectral-interval type $F_0$ estimator, observing frequency locations between partials. The basic principle is based on the observation that harmonic sounds possess a periodic magnitude spectrum. Thus, any two spectral components with a frequency interval $m$, multiplied by the sampling rate and the inverse of the FFT order ($mF_s/K$) is a $F_0$ candidate, as shown in Eq.5, where $t$ represents the time delay:

$$\tilde{r}(m) = \frac{2}{K}\sum_{k=0}^{K/2-m-1}|X(k)||X(k+m)| \qquad (5)$$

The maximum value of the autocorrelation correspond to the period of the signal waveform, discarding the obvious maximum value at $m = 0$ (that corresponds to the energy of the signal). Another local maximums appear at integer multiples of $F_0$, because the periodicity of the magnitude spectrum at multiples values of $F_0$ rate, and so, some erroneous doubled values of $F_0$ could appear. Another handicap results from its constant frequency resolution. This problem is more severe in low frequencies, where resolution could be not enough. That contradiction with the human perception of music, which is logarithmic, will damage the final detection. Thus, bigger FFT orders[3] are needed to increase the range of possible frequencies at low frequencies, which lead to a unnecessary wide range of possible frequencies at high frequencies, and a rise in computation time.

### D. Harmonic Product Spectrum

The Harmonic Product Spectrum *HPS* pitch-detection algorithm [14], measures the maximum coincidence for harmonics, for each spectral frame $X(\omega)$,

$$Y(\omega) = \prod_{d=1}^{D}|X(\omega r)| \qquad \hat{Y} = \max_{\omega_i}\{Y(\omega_i)\} \qquad (6)$$

where $D$ is the number of harmonics to be included in calculations, and $\omega_i$ are the range of candidates to $F_0$. The value of $\omega_i$ corresponding to the maximum value of the resulting periodic correlation array, $Y(\omega)$, will support the $F_0$. The idea follows from the fact that a musical input signal presents a spectrum consisting of a series of peaks, corresponding to fundamental frequency with harmonic components at integer multiples of the fundamental frequency. Hence when the spectrum is downsampled (compressed a number of times), and compared with the original spectrum, the strongest harmonic peaks line up. The first peak in the original spectrum coincides with the second peak in the spectrum compressed by a factor of two, which coincides with the third peak in the spectrum compressed by a factor of three. Hence, when the various spectrums are multiplied together, the result will form clear peak at the fundamental frequency. Figure 2 demonstrates the HPS algorithm graphically for a windowed signal waveform, where an FFT is executed to the window *(left)*, and several downsampled versions are made *(center)*. The product of the downsampled signals, with a most likely pitch for the analysis window very clear, is shown on the far right of the figure.

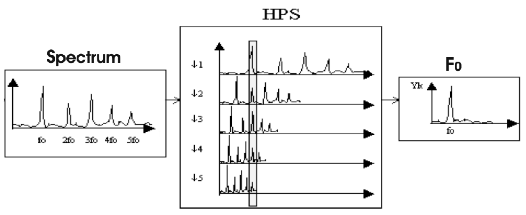


Fig. 2: Overview of the HPS algorithm

The method presents some nice features: it is computationally inexpensive, it is reasonable resistant to additive and multiplicative noise, and it is adjustable to different kind of inputs. However, its resolution is only as good as the length of the FFT used to calculate the spectrum. If a short and fast FFT is performed, a limitation in the number of discrete frequencies occurs. So, and as the *fACF*, in order to gain a higher resolution in the output, a larger FFT order is necessary[4].

## IV. RESULTS FROM TRANSCRIPTION

The data sets used to test the transcription method comprised synthetical and man-made whistle sounds. The synthetical sound allows evaluating the influence of the various parameters on the accuracy of the transcription, without having to consider the usually large errors introduced by the performance of a human whistling.

Figure 3(a), shows the *Happy-Birthday* score used to create a *MIDI* file and subsequent synthetic whistle sound, using

[3]Notice that increasing the FFT order the temporal resolution is degraded.

[4]This requires more time and decreases the temporal resolution.

a common PC software synthesizer. The figure shows also the translations by the proposed system using each of the four pitch detection methods detailed in Sec.III. As expected, the transcription of the synthetical whistle did not produce significant errors: all the pitches correspond exactly to the played notes; only small time misalignments occur.

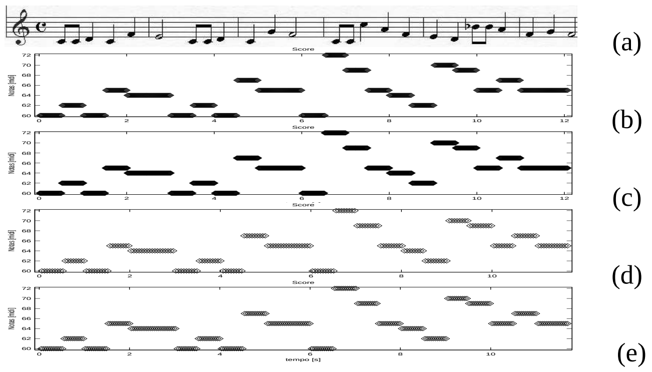


Fig. 3: Artificial whistling generated from (a) and its transcription based on temporal *ACF*, *AMDF*, *spectral ACF*, and *HPS* (b,c,d,e).

The accuracy of the translation methodology was evaluated considering various transcription-parameters: minimum and maximum detectable frequencies, minimum note duration, window type, and window length.

The minimum frequency threshold is an important factor in the performance of temporal techniques: setting it too high implies losing low pitches, while if set too low implies processing very large samples. We chose 440$Hz$ as a trade-off between the risk of not detecting low frequency notes and fast computation. The minimum note duration and maximum detectable frequencies, were set to 100$ms$ and 4410$Hz$ respectively. These allows for the detection of notes larger than 100$ms$ (10 notes *per* second, or one semiquaver in 150 B.P.M. $\approx$ *Vivace*), with pitches between an $A4$ and a $C8$, while covering the range of a typical whistling (see Sec.I).

The window type in our experiments did not influenced significantly the results. Hence, in this paper we document just the experiments done with the *Hanning* window. The windows size affects significantly however the results. Table I shows the ranges of window sizes for which there are no transcription errors. Using large windows prevents detecting fast notes, while using small windows implies processing a larger number of frames (better time resolution), but with a reduced frequency resolution.

In our experiments the *tACF* and the *HPS*, with 1024 and 2048 window sizes respectively, yielded the smallest alignment errors between the transcribed sound and the *MIDI* file format (ground truth) generated from the original score. The *tACF* was the fastest method, even in the case of the largest window size (note that this method actually works faster as the window size grows). In a 12 seconds signal, the *tACF* takes about 1 second of processing time, as compared to the 4 seconds taken by the *AMDF*.

The tests with man-made whistle sounds allow evaluating the effectiveness of the proposed solution. As noted in the introduction, the inaccuracy of whistling, as for instance with non-steady pitches, or with missing, inserting or translating notes, imply transcription errors. The extent to which these difficulties are resolved shows the robustness of the developed general algorithm in the detection and amend unintentional pitch variations in the whistling process.

TABLE I: Tolerance to the choice of the frame window size, for all the pitch tracking techniques, for a 44.1$kHz$ sample rate. *Dec* indicates the temporal decimation order (when used).

| Method | Dec | Tolerance | Dec | Tolerance |
|---|---|---|---|---|
| tACF | No | [512-8192] | 5x | [128-1024] |
| **AMDF** | **No** | **[256-8196]** | 5x | [64-1024] |
| HPS | No | [2048-8192] | 5x | [512-2048] |
| fACF | No | [None] | 5x | [512-1024] |

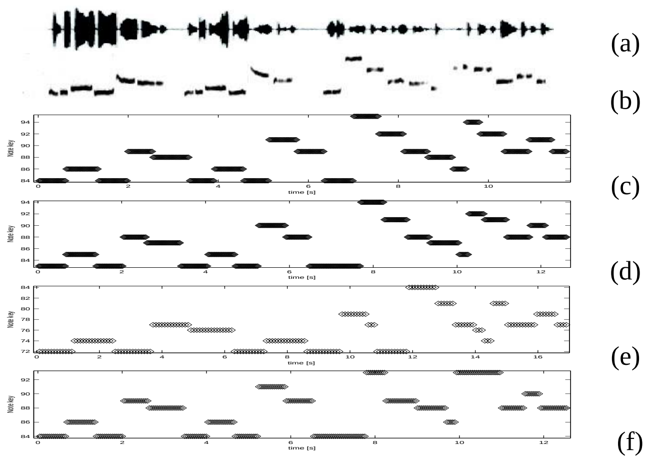


Fig. 4: Transcription of a human whistle sound (a) with spectrogram (b). Results obtained with *tACF*, *AMDF*, *fACF*, and *HPS* (c,d,e,f).

Figure 4 shows the amplitude envelope and the spectrogram of a *Happy-Birthday* whistling, together with the resulting transcription for each pitch tracking method described in Sec.III. The three best results for each technique are presented in Tab.II, as well as the insertion, deletion, and substitution errors (as a percentage of the total number of notes-symbols).

In our experiments, the *HPS* method, despite making an almost correct transcription, showed less performance than the other methods. In particular was necessary to pre-filter the man-made whistle sound, with a 99$th$-order FIR bandpass filter tuned for typical whistling sounds, as otherwise the estimated pitch would be oscillating between a base range of values and the same range translated one octave upwards. This was expected as *HPS* is known to require signals with significant harmonic components (partials/overtones), which do not happen in whistle sounds, as noted in Sec.I. This characteristic of the whistling timbre, that reduces the efficiency of spectral-interval type $F_0$ estimators as the *HPS*, actually improves the performance of spectral-location type $F_0$ estimators as the *tACF* and the *AMDF*. The existence of weak $F_0$ multiples (overtones) have the positive effect of reducing octave errors.

Comparing the *tACF* method with the *fACF*, the former is more robust, accurate and faster, even in the case where the *fACF* decimated the input signal. Temporal methods are both accurate, but the use of the FFT in the *tACF* algorithm makes it faster. Overall, the best transcription performance was obtained using *tACF* with a window of 512 samples. No tracking errors occurred in a calculation time of 15% of the input signal time.

TABLE II: Errors in the detected notes. Insertions error rate $e_i$, deletions error rate $e_d$, and substitutions error rate $e_s$. Methods tested with the specified window sizes (three best results in each method). The † means a previous decimation of the signal ($2x$ to a sample rate of $8kH$). Bold shows the best window size within each method.

| Method | window | $e_i$ | $e_d$ | $e_s$ | $e_T$ |
|---|---|---|---|---|---|
| tACF | 128 | 0% | 0% | 0% | 0% |
| | 256 | 0% | 0% | 4% | 4% |
| | **512** | **0%** | **0%** | **0%** | **0%** |
| AMDF | 256 | 4% | 0% | 0% | 4% |
| | **512** | **0%** | **0%** | **0%** | **0%** |
| | 1024 | 0% | 0% | 4% | 4% |
| fACF | 1024 | 4% | 4% | 0% | 8% |
| | 512 | 4% | 0% | 4% | 8% |
| | **512†** | **4%** | **0%** | **0%** | **4%** |
| HPS | 1024 | 0% | 8% | 4% | 12% |
| | 256† | 0% | 8% | 4% | 12% |
| | 128† | 0% | 8% | 4% | 12% |

## V. MUSIC SEARCH ENGINE

A potential application of the musical transcription system described above is the search and retrieval of musical themes, given a whistle sketching the melody. Users whistle a segment of a song as input, and as output they receive a list of music candidates, ranked by their relevance.

To accomplish this task, three modules were implemented: a melody encoder (1), a database themes extractor (2), and a matching algorithm (3). Figure 5 illustrates the architecture of the music retrieval system, employing these three modules. The melody encoder module (1) extracts features from the MIDI database and the detected notes (previously converted to the MIDI format), coding the notes into a simple and efficient representation (without losing important information about the melody, required to the matching process). This coding, which is critical to the performance of the system, will be described below. The database themes extractor (2) extracts the (monophonic) melody line from each database music (polyphonic), and the matching algotithm (3) sorts the candidate songs from the database, using similarity functions between the query and all the MIDI files, selecting and ranking the most alike songs.

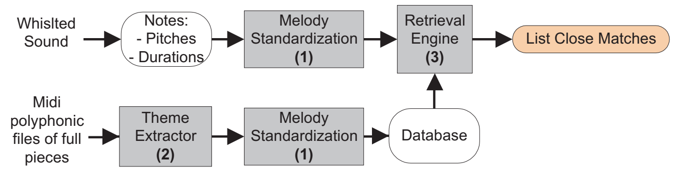

Whislted Sound
Notes: - Pitches - Durations
Melody Standardization (1)
Retrieval Engine (3)
List Close Matches
Midi polyphonic files of full pieces
Theme Extractor (2)
Melody Standardization (1)
Database

Fig. 5: Overview of the music search and retrieval system.

### A. *Melody representations and encoding*

The representations of melody lines, for the purpose of theme search, is crucial for the performance of the system. Such a representation should be flexible to a different the base pitch (transposition), to be slighly out of tune, and rhythmic inaccuracies, while retaining the fundamental distinguishing features of each theme. Pitch and rhythm information are surely the musical components that most identify a melody [12]. Suiting to whistling habits implies finding generic, e.g. speaker independent, representations.

A melody holds its identity to a human judgment under various transformations. Two pieces of music seem alike even with different keys, interpolation or omission of some notes, modified pitch intervals, replacement of notes by appropriate chords, and addition or deletion of parts [3]. In addition, a timbre, sonority, spatial localization and some times rhythm variations does not alter the melody recognition.

The encoding system should encode the melody considering these proprieties, and it should also be robust to errors, such as whistling errors, being flexible to absolute and relatives pitch, time and rhythm divergences, transcription errors, like pitch quantization or notes segmentation, MIDI database quality, as well as to be rubost to melody line extraction inaccuracies.

A pitch and time error-tolerant approach is based on Parson's experiments [4], where the representation is based entirely on a sequence of pitch intervals termed "melodic contour". Parsons showed that a simple encoding of tunes that ignores most of the information in the musical signal can still provide enough information for distinguishing between a large number of tunes. This representation, designated *DUR*-representation, classifies each pair of successive notes as either "D" (down), "U" (up), and "R" (repeated) denoting whether its pitch quality is lower, higher or the same as the previous one[5].

Although this encoding perform well when used with musical unskilled people or with people not very familiar with the song, it needs a long *DUR*-string to correctly discriminate a melody in a large database.

When people recall a melody from memory, they are able to present information more precise than just interval direction [10]. This way, an extension to the previous representation was considered, by classifying intervals into five extended types, a *DdRuU*-representation, where a "D" means a significant pitch decrease (two semitones threshold), "d" a smaller decrease, "R" same pitch, "U" significant increase, and "u" small increase (by one or two semitones) [19].

A third, even more discriminative, encoding was also considered: take the relative intervals into account, where thirteen changes of pitches (to the prior note) are considered: unison, minor/major second, until a perfect octave. All intervals higher than an octave are assigned into a new 14*th* symbol. This is the most precise representation we consider here for pitch encoding. We do not consider the use of absolute pitches as a most discriminate stage because it is much easier for humans to discriminate or reproduce the relative pitch variations than absolute pitches from perception [9].

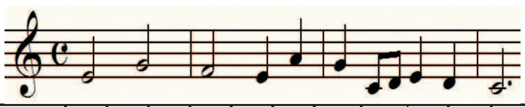
In addition to pitch, rhythm information can be important for identifying a melody, although humans are more error-

[5]Notice that rhythm is completely ignored, and the first note is used just as a reference point.

prone to notes durations, than to pitch. In our algorithm we used two different kinds of time representation. The first one is a "time contour", similar to "pitch contour", described by the duration between notes: an "R" is used for a repetition of previous duration, an "L" for a longer duration, and an "S" for a shorter one. This approach is useful for the majority of common whistlers that do not respect duration of notes, but a more precise representation for better whistlers was also implemented. Here, time invariance is achieved by dividing the duration of the actual note by the previous one. To prevent an error increase after large jumps in note duration (e.g. semibreve to a semiquaver), a logarithm function was used, as shown in equation (7).

$$d(i) = round\left(10\log_{10}\left(\frac{d_i}{d_{i-1}}\right)\right) \qquad (7)$$

In both cases, an Inter Onset Interval (IOI) technique was used, where durations are the intervals between two onset times (except where a breath is detected) of successive notes, because they are more consistent and easily detectable that the actual sound duration (humans tend to undervalue the notes termination). In our program we used an hierarchical representation, where the simpler pitch contour is initially used, and then more precise representations with time information are used, where all silences are discarded (whistling records have inevitable breathing breaks that cannot be distinguished from a pause). Examples of coded melodies are shown in Fig.6.



| Midi (chromatic scale) | 64 | 67 | 65 | 64 | 69 | 67 | 60 | 62 | 64 | 62 | 60 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Time(beats) | 2 | 2 | 2 | 1 | 1 | 1 | 0.5 | 0.5 | 1 | 1 | 3 |
| 3 types Contour | * | U | D | D | U | D | D | U | U | D | D |
| 5 types Contour | * | U | d | d | U | d | D | u | u | d | d |
| Pitch Intervals | * | +3 | -2 | -1 | +5 | -2 | -7 | +2 | +2 | -2 | -2 |
| Time Contour | * | E | E | S | E | E | S | E | L | E | L |
| Relative Duration | * | 0 | 0 | -3 | 0 | 0 | -3 | 0 | +3 | 0 | +5 |

Fig. 6: Example of various notes encoding types, for the beginning of Portuguese National Anthem.

## B. Extracting Melody representations from MIDI files

In order to search matching melodies, it is mandatory to constitute a database in a musical format, and retrieve the notes comprising the melody of each song.

Extracting a melody line from MIDI files is more efficient than extracting it from a sound recording (e.g., MP3). The MIDI format is similar to a score, and because they can be easily found on the Internet, and because they contain much of the abstract information necessary for music similarity computation, we decided to create a large database with MIDI files (where we intend to have a large amount of songs, to give a more correct answer to the query).

MIDI files consists of multiple tracks, each representing a separate instrument that could have several simultaneous notes (chords) and that can be played simultaneously. The perceived melody, however, is rarely played by a single instrument, moving from several channels, and major themes may occur anywhere in a piece. To automatically extract the notes that are perceived as the melody line, and discard the remainder is a non-trivial task, however it is indispensable for the purpose of music matching by similarity.

We used a set of heuristics to extract the melody line. In the first step we use each possible metadata to exclude all impossible channels. For example, deleting channel 10, usually fixed to percussion, and deleting all tracks with a short length. This way, part of non-melodic noise are discarded. Uitdenbogerd *et al.* [20] conducted some experiments where they concluded that, instead of using entropy information, or just selecting the highest notes from the most important instrument, combining all tracks and keeping the highest note from all simultaneous note events is the process that keeps more of the melody identity.

In our algorithm, we decided to combine all musical information into one stream of events, using the Uitdenbogerd idea but also considering fast notes events. Whenever a note starts, it chooses the highest note from all that start at the same time. Notes lengths are then truncated until we have a monophonic result (because many overlapping notes result from the previous step, as a note that is sustained in one track will cover the start of other note), and so, new notes from the original channel could appear in the monophonic melody result. In the end, we filter out notes shorter than a minimum threshold, as well as grave notes, *i.e.* notes having a pitch in the 10% of the lower pitches and whose pitch interval to the previous note is higher than a perfect octave. This way, we try to catch the top notes of all the events that occur in a piece.

We listened to several original songs, as well as to the monophonic result, and although we could notice melodies differences, the major part of the melody line is perceived as the same. Thus, we were able to constitute a large monophonic database for query, being aware of the errors inherent to the process. This files are then encoded by the corresponding module (1), and a text file with melody information about all database musics are stored.

An example of the process, extracting the melody of the Wagner's "Bridal Chorus", is shown in Fig.7.

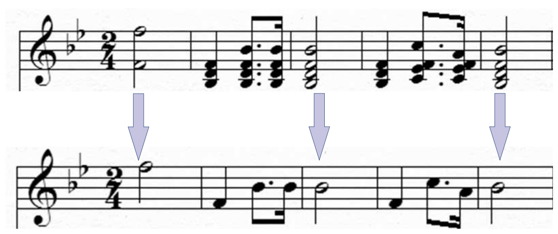


Fig. 7: A segment from Bridal Chorus of Wagner(a) and the resulting monophonic "melody" extracted from our system(b)

## C. Finding Matching Melodies

Finding candidate melodies from the database that match the whistled sound involves aligning the candidates and the whistle, computing distances, selecting possible matchings, and finally ranking the matching candidates. This is detailed in the next section.

## VI. Matching Melodies

As described in Sec.V our melody representations are in essence strings, and thus matching melodies for database retrieval is in our work equivalent to matching strings. In other words, melody matching can use directly algorithms from the very mature research area of string matching [6].

There are however still two important aspects to care with: the whistled sounds have great variability as they are produced by humans, and melody databases are usually large in order to be reasonably complete. Consequently, string matching has to be simultaneously fast and error tolerant. As suggested by [16], we need *approximate pattern* matching algorithms which can be implemented efficiently as with dynamic programming.

### A. Levenshtein Distance and Dynamic Programming

Viewing melodies as strings, allows defining a similarity of melodies as a sum of costs associated with the editing operations that must be performed to make the strings identical. The minimum sum of costs is called **Levenshtein** or **edit** distance [13].

Considering three basic editing operations, namely note insertion, deletion and replacement, then the Levenshtein distance can be computed using on a *dynamic programming (DP)* approach [6].

Given a query melody string $X = [x_1, x_2, \ldots, x_m]$ and a database melody string $Y = [y_1, y_2, \ldots, y_n]$, the dissimilarity between the strings, $D(X,Y)$, can be computed from the recursion in $1 \leq i \leq m$ and $1 \leq j \leq n$:

$$d_{i,j} = \min \begin{cases} d_{i-1,j} + \omega(x_i, \phi) & (deletion) \\ d_{i,j-1} + \omega(\phi, y_j) & (insertion) \\ d_{i-1,j-1} + \omega(x_i, y_j) & (replacement) \end{cases} \quad (8)$$

where $\omega(x_i, \phi)$, $\omega(\phi, y_i)$ and $\omega(x_i, y_i)$ represent costs associated with the insertion of $x_i$, the deletion of $y_j$ and the substitution of $x_i$ by $y_j$, respectively, and finally:

$$D(X,Y) = min\{d_{i,j} : i = m, 1 \leq j \leq n\}. \quad (9)$$

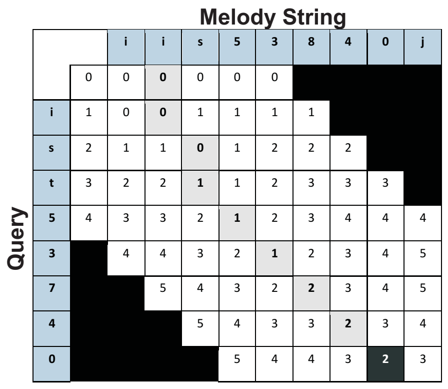
The initial conditions to find the longest common subsequence are:

$$\begin{cases} d_{i,0} = 0, \; i \geq 0 \\ d_{0,j} = d_{0,j-1} + \omega(\phi, y_j), \; j \geq 0 \end{cases}$$

where the prefix and suffix, i.e. the edges of a string that does not belong to the minimal path, are discarded with zero cost from the aligning process.

Figure 8 shows an example where DP is used to match a melody segment $[i,i,s,5,3,8,4,0,j]$ with the query $[i,s,t,5,3,7,4,0]$. The minimal cost is the lowest value found in the last row (in this case is 2), meaning that 2 differences exist between the strings (the deletion a *t* and a substitution of a *7* by an *8*). The optimal matching pattern is shown in bold and it can be discovered in a bottom-up process, tracing the path from $d_{n,j}$ to the first column that minimize the cost.

Because there is no fixed start and ending points, we cannot restrict computation to a specific window, and the time complexity is $O(mn)$. However, we can skip the calculation of some points, assuming that people do not skip many notes, insert many notes. Thus, the "true" matches will fit along or near a diagonal of the matrix $d_{i,j}$. Just these points are calculated, as illustrated in Fig.8.



Melody String

Query

| | | i | i | s | 5 | 3 | 8 | 4 | 0 | j |
|---|---|---|---|---|---|---|---|---|---|---|
| | 0 | 0 | **0** | 0 | 0 | 0 | | | | |
| i | 1 | 0 | **0** | 1 | 1 | 1 | 1 | | | |
| s | 2 | 1 | 1 | **0** | 1 | 2 | 2 | 2 | | |
| t | 3 | 2 | 2 | **1** | 1 | 2 | 3 | 3 | 3 | |
| 5 | 4 | 3 | 3 | 2 | **1** | 2 | 3 | 4 | 4 | 4 |
| 3 | | 4 | 4 | 3 | 2 | **1** | 2 | 3 | 4 | 5 |
| 7 | | | 5 | 4 | 3 | 2 | **2** | 3 | 4 | 5 |
| 4 | | | | 5 | 4 | 3 | 3 | **2** | 3 | 4 |
| 0 | | | | | 5 | 4 | 4 | 3 | **2** | 3 |

Fig. 8: The dynamic programming algorithm search and the resulting alignment pattern (gray; minimum $d_{n,j}$ in black). The algorithm is windowed, i.e. $d_{i,j}$ is not computed in too improbable matching points (black areas).

### B. Hierarchical Matching, Finding Candidates

Knowing that pitch contours in a *DdRUu* format are fast to compare but comprise a low distinctiveness nature (as compared to the pitch intervals and duration-ratios) we decided to equip our search engine with a two-steps, hierarchical, matching method. The hierarchical method selects in a first step the set of probable matching melodies based just on the pitch contour. In the second step, it does a more discriminative ranking based on both pitch and time intervals. The general idea of the hierarchical matching process is illustrated in Fig.9, and detailed in the following.

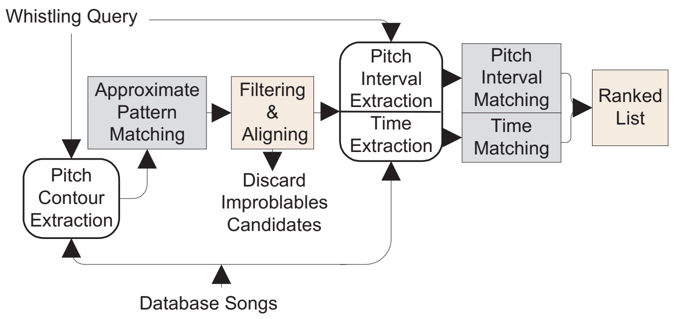


Fig. 9: Hierarchical ranking algorithm

Approximate pattern matching and DP are used to align the *DdRuU*-pitch contours between the query and the candidates from the musical segments. If the distance (*k*-differences) is larger than a threshold, we discard this unqualified piece. Otherwise, the matched path between the pairs will be recorded for

[6]This application of DP is similar to the Minimal Edit Distance and to the Dynamic Time Warping algorithms typically applied in speaker dependent speech recognition http://www.dcs.shef.ac.uk/~stu/com326/

further processing. Here, the threshold is the sum of insertion, deletion and replacements. However, we always guarantee a minimum number of candidates to pass to the next round. So, this step constitutes a filter to discard improbable candidates, and to reduce future calculations.

Experiments shown that a local cost function, using the edit distance, is not the most correct in musical applications. Hence, we decided to consider different classes of replacement, deletion, and insertion operations, attributing specific costs to each one of them (in Eq.8). Adapting [17] to our type of data (whistling) we follow four rules:

- Deletions "'D"' and insertions "'I"' have the same cost, in order to maintain a symmetric distance function;
- "Repeat"-"Up/Up a lot" and "Repeat"-"Down/Down a lot" replacements, have the same cost to avoid a distance function sensitive to melodic inversion;
- Any "D,I,S" followed or preceded by a "R" has a smaller cost than the others, due to the difficulty from the recognition systems to determine the correct number of "R's";
- The replacement cost is larger than an insertion or deletion, but smaller than their sum, as the replacement can always be performed by a deletion plus an insertion.

To speed up the matrix calculation, we skip the distance calculation where the sum of accumulated errors are over a defined threshold, assigning an $\infty$ cost (see Fig.8).

### C. Hierarchical Matching, Ranking Candidates

Given a set of melodies possibly matching a query whistle, the main objective now is to compute more precisely the similarities to the query and than ranking the candidates. We consider three alternative comparison functions: (i) Levenshtein distance $D(X,Y)$, (ii) correlation based dissimilarity function $D_2(X,Y)$, and (iii) euclidean distance $D_3(X,Y)$.

While in the previous section only (rough) pitch differences were considered to compute the **Levenshtein distance** now we consider both the pitch and time differences. These two aspects are linearly combined in a single cost function:

$$\omega(x_i,y_j) = \lambda \cdot \omega_p(p(x_i),p(y_j)) + (1-\lambda)\cdot \omega_d(d(x_i),d(y_j))$$

where $x_i$ and $y_j$ contain both pitch and time differences, $p(.)$ and $d(.)$ select pitch or duration information, $\omega_p(p(x_i),p(y_i))$ and $\omega_d(d(x_i),d(y_i))$ weight the pitch and time differences, and $\omega$ combines the two costs to insert into the Dynamic Programming recursion Eq.(8) though the weight $\lambda$ which balances pitch vs time differences information (to be used also in the next equations). The distance is again $D(X,Y)$ as in Eq.9.

In order to be faster, the **correlation function** is implemented on aligned queries and candidates. Suppose the aligned pitch interval lists are[7] $x_i$ and $y_i$, and the aligned rhythm arrays are $a_i$ and $b_i$, where $i = 1,\ldots,N$, and $N$ is the length of the aligned sequences then the pitch interval similarity is measured by:

[7]Note that the alignment of features allows dropping the index $j$, i.e. $j \equiv i$.

$$\omega_p = \frac{\sum_{i=1}^{N}(x_i-\bar{x})*(y_i-\bar{y})}{\left(\sum_{i=1}^{N}(x_i-\bar{x})^2\right)^{1/2}\left(\sum_{i=1}^{N}(y_i-\bar{y})^2\right)^{1/2}} \tag{10}$$

and rhythm similarity is calculated by [11]:

$$\omega_d = \prod_{i=1}^{N}\left(1-\beta\cdot\frac{|a_i-b_i|}{\max(a_i,b_i)}\right) \tag{11}$$

where $\bar{x}$ and $\bar{y}$ are the averages of $x_i$ and $y_i$, $\beta$ is a weight. Finally, the dissimilarity function is

$$D_2(X,Y) = 1-[\lambda\omega_p + (1-\lambda)\omega_d]\,. \tag{12}$$

The third comparison function, the **Euclidean distance**, is also implemented with the aligned strings from step 1:

$$D_3(X,Y) = \sum_{i=1}^{N}(x_i-y_i)^2 + \lambda.(a_i-b_i)^2. \tag{13}$$

After listening over several whistling sounds, we observed that people have more rhythm variations, while playing tones more carefully. We also notice some irregularities in long notes, where the energy sometimes decays abruptly in the middle of a note. This way, we decide to attribute a larger weight to $\omega_p$ than $\omega_d$ (0.7 to 0.9).

Notice that both correlation and Euclidean distance function calculates the deviation between two vectors, but a higher similarity in normalized correlation function ($[-1,1]$) corresponds to a greater value, but in Euclidean distance a smaller result means a great similarity (0 signify equality).

The final action of this hierarchical matching method is the presentation of ranked sequences of similar music to the whistle, where we attempt to maximize the number of correct matches considering the usual whistling behavior from humans.

## VII. MELODY SEARCHING RESULTS

The effectiveness of the re-encoding methods along with similarity functions used are here analysed. Several combinations of representations and matching functions were tested in order to discover the maximum performance the system can carry out, and to select parameters (which provide better results to the system) to use in the final application, the music search engine.

A total of six hundreds MIDI songs were collected into our database, with a wide variety of genres.

It was asked to some colleagues and relatives, with different musical skills, to record some specific whistles (in a total of forty songs) to enrich our set of testing sounds. Twelve available participants sent us a total of one hundred different whistles, in a total of 33:52 minutes (20,5 seconds per whistle). All queries were stored and identified by the song title and user name, to the purpose of a future performance measurement. Three different database were constituted, with songs from the collected corpus (encoded as MIDI pieces), with six hundreds, two hundreds and forty songs. Therefor, the system's dependency to the database size can be perfectly deduced.

A characteristic confusion matrix [8] of our music search engine is exhibited in Figure 10, in which a diagonal line is clearly identified, meaning a reduced but existing confusion. Each column of the matrix represents one of the 100 whistles used in the experiments, whereas each row represents each of the 40 MIDI songs. A binary classification is attributed to each MIDI songs, *1* if it is similar to the given whistle (±5%), or *0* otherwise.

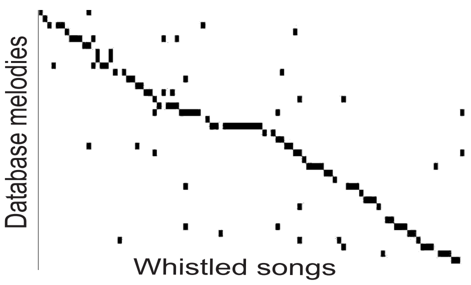


Fig. 10: Confusion matrix

## A. Evaluating the Search Engine Performance

Each test scenario and parameter combination is evaluated using three performance measures. Let $rank_i$ denote the ranking of the correct answer in the retrieved melodies in the $i$th query among $N$ queries. The first measure is the *Top(X) hit rate*, which computes the percentage of successful queries

$$Top(X) = \#\{rank_i : rank_i \leq X\}/N \tag{14}$$

with $X$ typically equal to 3 or 10. The second measure is the *Mean Reciprocal Rank (MRR)* [9], defined as

$$MRR = \sum_{i=1}^{N} (rank_i)^{-1}/N \tag{15}$$

gives a hint of how often the target reaches one of the first rank. A good MRR value, highly dependent on database size, on 200 song would be above 0.2 [5]. Finally we define also *Mean of Accuracy (MoA)* as

$$MoA = \frac{1}{N}\sum_{i=1}^{N}\left(\frac{N - rank\_i}{N-1}\right) \tag{16}$$

which shows the average rank at which the target was found for each query. Values above $2/3$ represent a good accuracy.

## B. Experiments and results

Table III shows the music search engine performance for various combinations of parameters. Each row represents one set of database queries considering the parameters shown on the first five columns. The next four columns show performance measurements using the metrics detailed in the previous section: Top-X hit rates (with $X = 1$ and $X = 3$), the Mean Reciprocal Rank (MRR) and the Mean of Accuracy (MoA) [10].

The matching techniques tested included the explained matching algorithms (section VI), and simplified versions (to extrapolate about the improvements). It was checked in the first stage the use of a pitch melodic contour, using the edit distance and different degrees of contour granulations, with three or five levels. It was also tested the use of a pitch interval matching, using a second edit distance function, the autocorrelation function or the euclidean function. Its efficiency was tested alone and simultaneously with pitch melodic contour and/or time information. The time re-encoding tested comprises the time contours and the logarithm duration ratio (Eq. 7). Edit distance, as well as Eq. 11 and Eq. 13 were used (apart) as time similarity measures.

Different weights were associated with each one of the three representations (melodic contour, intervals and time), ranging from 0% (no contribution to the final rank) to 100% (used isolated to rank the songs). Those results were acquired knowing in advance the identification of the tested whistles and making a comparison between the achieved results and the correct ones (ground truth).

Different conclusions can be taken from the experiments. When it concerns about the similarity functions, matching melodies, the use of dynamic programming (modified Levenshtein distance), due to its error-flexibility it can achieve an enhanced performance (better matching) than the correlation or euclidean functions.

Due to the high error rate of the input queries collected from the less musically-skilled participants, the simple use of pitch melodic contours matching brings on better results than the application of pitch intervals matching, in a short database. When the length of database increases, both representations decrease their performances. However, in that circumstances, and because intervals are a more discriminant representation than contours, the correct song is identified less confusedly by the former representation.

The efficiency of a single representation based on three vs five melodic contour levels was examined. The difference between them is large enough to reject the simple $DUR$ representation (the use of the 5 levels re-encoding, e.g., decrease the $Top1$ rate in 26% and $Top3$ in 12%). It can be assumed a three level re-encoding (even with badly whistled melodies) do not reach the accuracy level to be incorporated into music engine, and the approximate pattern matching should use a five levels contour.

About the discriminating level applied in the time re-encoding it was inferred that the use of a more precise representation, using a logarithmic duration ratio between consecutive notes, is not a better solution than using a simple duration contour, evidencing the difficulty non-professional musicians have in reproducing exactly the rhythm of a song.

[8] A confusion matrix is a visualization tool used to represent errors, assigning classes to observed patterns (matching matrix).

[9] Performance measure as used in MIREX 2007, http://www.music-ir.org/mirex2007

[10] The *MRR* and *MoA* formulas have been adapted to the case of having more than one whistle per melody, e.g. 100 queries to 40 database melodies in the first four experiments. In order to enforce $MRR = MoA = 0$ in the worst situation (total inefficacy), and 1 in a perfect situation, the returned values were shifted by the worst case, and then multiplied by the inverse of the subtraction between 1 and the worst case. Hence, *MRR* and *MoA* are in the range of $[0, 1]$

|  | 1st Step | 2nd step: Pitch | 2nd step: | Time | Results |  |  |  |
|---|---|---|---|---|---|---|---|---|
| **Songs** | **Contours** | **Intervals** | **LSR** | **Ratio** | **Top1** | **Top3** | **MOA** | **MRR** |
| 40 | 100 | - | - | - | 81% | 91% | 98,36% | 86,19% |
| 40 | 50 | - | 50 | - | 82% | 91% | 98,31% | 86,99% |
| 40 | 33 | 33 | 33 | - | 81% | 89% | 98,05% | 85,73% |
| 40 | 33 | 33 | - | 33 | 80% | 89% | 97,97% | 85,02% |
| 200 | 33 | 33 | 33 | - | 77% | 88% | 98,13% | 83,66% |
| 600 | 33 | 33 | 33 | - | 61% | 78% | 96,21% | 71,83% |

TABLE III: Query-by-whistling performance. The performance of the musical retrieval was tested using 100 whistles (of 40 different melodies) with a database size of 40, 200 and 600 songs.

As expected, pitch representations portrays a melody more accurately than time representations. Anyway, their cooperatively use reaches a higher discriminant and precise level.

In the short database, the best performance was obtained using the edit distance matching pitch and durations contours ($MRR = 87\%$ and $Top3 = 91\%$), both encodings with 50% contribution to the final listing. The use of pitch intervals data is helpful when the number of songs stored in the database is enlarged. With 200 songs, and sharing 33% of the total cost between pitch contour, time contour, and interval matching), it can be achieved $MRR = 84\%$ and $TOP3 = 88\%$Searching the whistles in a 600 songs database, $MRR$ decreases down to 71.5% and $TOP3$ to 78%.

Concluding, in a real application of our music search engine, with a large database of songs, the modified edit distance is the most effective similarity function, matching the query and the candidates twofold, with three distinctive re-encodings: five levels pitch contours (used in the first stage to discard and align the candidates strings, pitch intervals and the time contours.

## VIII. CONCLUSIONS

In this paper we proposed a transcription system of musical-whistling to a MIDI-like representation. In particular, we have compared four options for pitch detection, a central component of the system. Our experiments showed that the system performs successfully the transcription, and the pitch detection methods proved to be functional in a wide range of the parameters.

As an application for the successful whistling transcription system we proposed a music search engine, which uses the score produced by the transcriber (query) and matches it with a local database of melodies.

Performance measure metrics showed encouraging results of the retrieval system, with the implementation of melody matching based on dynamic programming and using both pitch and time data.

As future work, the currently fixed melody representations can be made more flexible in order to improve the retrieval process. The automatic tuning of the representations will be therefore an open issue.

## REFERENCES

[1] T. Brøndsted, S. Augustensen, B. Fisker, C. Hansen, J. Klitgaard, L. Nielsen, and T. Rasmussen. A system for recognition of hummed tunes. In *Proceedings of the COST G-6 Conference on Digital Audio Effects(DAFX-01)*, pages 6–8, Limerick, Ireland, Dec 2001.

[2] L. Cohen. The generalization of the wiener-khinchin theorem. *IEEE T-ASSP*, 3:1577–1580, 1998.

[3] J.S. Downie. Melodic information processing and its development. In *The Psychology of Music, chapter 13*, pages 413–429. Academic Press, Inc., 1982.

[4] D.Parsons. *The Directory of Tunes and Musical Themes*. Spencer Brown, 1975.

[5] Alexander Duda, Andreas Nürnberger, and Sebastian Stober. In *TOWARDS QUERY BY SINGING/HUMMING ON AUDIO DATABASES*, 2007.

[6] Ning Hu and Roger B. Dannenberg. A comparison of melodic database retrieval techniques using sung queries. In *JCDL 02: Proceedings of the 2nd ACM/IEEE-CS joint conference on Digital libraries*, pages 301–307, New York, NY, USA, 2002. ACM.

[7] A. Klapuri. Automatic music transcription as we know today. *Journal of New Music Research*, 33(3):269–282, 2004.

[8] W. Kuhn. A real-time pitch recognition algorithm for music applications. *Computer Music Journal*, 14(3):60–71, 1990.

[9] D.J. Levitin. Absolute memory for music pitch: Evidence from the production of learned melodies. *In Perception and Psychophysics 56 4*, pages 414–423, 1994.

[10] A Lindsay. Using contour as a mid-level representation of melody. Master of science in media arts and sciences thesis, M.I.T. Media Lab, 1997.

[11] Lie Lu, Hong You, and HJ Zhang. A new approach to query by humming in music retrieval.

[12] Rodger J. McNab, Lloyd A. Smith, Ian H. Witten, Clare L. Henderson, and Sally Jo Cunningham. Towards the digital music library: tune retrieval from acoustic input. In *DL 96: Proceedings of the first ACM international conference on Digital libraries*, pages 11–18, New York, NY, USA, 1996. ACM.

[13] Gonzalo Navarro. A guided tour to approximate string matching. *ACM Computing Surveys*, 33:2001, 2001.

[14] M. Noll. Pitch determination of human speech by the harmonic product spectrum, the harmonic sum spectrum, and a maximum likelihood estimate. *In Proceedings of the Symposium on Computer Processing Communications*, pages 779–797, 1969.

[15] M. Plumbley, S. Abdallah, J. Bello, M. Davies, G. Monti, and M. Sandler. Automatic music transcription and audio source separation. *Cybernetics and Systems: An International Journal*, 33(6):603–627, 2002.

[16] Emanuele Pollastri. Melody-retrieval based on pitch-tracking and string-matching methods.

[17] L. Prechelt and R. Typke. An interface for melody input. *ACM Transactions on Computer-Human Interaction*, 8(2):133–149, June 2001.

[18] D. Randel. *The new Harvard dictionary of music*. Belknap Press, Cambridge, MA, 1st edition, 1986.

[19] S. Rho and E. Hwang. Fmf: Query adaptive melody retrieval system. In *The Journal of Systems and Software 79*, pages 43–56, 2006.

[20] A. Uitdenbogerd and J. Zobel. Melodic matching techniques for large music database. In *Proc. of ACM Multimedia*, pages 57–66, 1999.

[21] Rodrigo Ventura, José Gaspar, and Bruno Dias. Automatic transcription of musical-whistling: comparing pitch detection methods jetc08. In *Quartas Jornadas de Engenharia de Electrónica e Telecomunicações e de Computadores*, 2008.