Monophonic Music Recognition

Per Weijnitz
Speech Technology 5p
per.weijnitz@gslt.hum.gu.se

5th March 2003

Abstract

This report describes an experimental monophonic music recognition system, carried out as an exercise in signal processing and recognition. It does not involve any ground-breaking new methods, but rather techniques well-known from the area of speech recognition. The method and the system are described and evaluated, and compared to other work in the field.

1 Introduction

This report describes an experiment with the goal of evaluating a hypothetical method for recognising monophonic music in wave files. A piece of music is monophonic if the maximum number of notes played at the same time is one. The hypothetical method can briefly be described as follows. Combining several basic, well known techniques used in the field of speech recognition, it should be possible to convert a number of recorded pieces of music into a suitable representation, and store in a database. Given an unknown piece of recorded music, it should then be possible to find the song most similar to it from the collection of songs in the database. There are two sub tasks involved. The first is to distinguish noise and silence from a tone, and determining a tone's pitch. The second is to perform recognition.

An experimental system was built, and a small set of testing material was collected consisting of whistling. The evaluation will show the abilities to recognise songs in the database and reject songs not in the database. The results will be compared to similar systems' results.

This kind of work belongs to the field of music information retrieval, MIR. A number of projects have been published that address monophonic music recognition, but it seems like most, if not all, have been trained on perfect data. Perfect, in the sense that it was either converted from MIDI songs[1] or taken

[1] Musical Instrument Digital Interface, a widely accepted standard for music communication

from existing databases of symbolic music representations. This work will be trained on the same kind of waveform data that will be used in the recognition process. In case the conversion from the input waveform to the symbolic representation is not perfect, this implies a decreased probability of successful recognition since there may be errors in the database.

This work involves the practical application of a number of techniques that are central in speech recognition, which makes it a task suitable for an introductory course in speech technology. Being familiar with and able to use such methods seems important in order to do further research in this field. The techniques include discrete Fourier transformation and edit distance. Additional techniques were deployed to handle various problematic issues, such as handling noisy signals and far-from-perfect music.

Given the limited time allocated to this assignment, it is necessary to set some limitations on its extent. It will only be tested with short songs or parts of songs like refrains. Primarily, it will not try to recognise parts of songs, which means it will only try to map unknown input to whole songs in the memory. It will not be a real time tool and only do batch operations. No effort will be put into optimisation. Finally, it will not involve recognition of the type of the sound source.

The next section is an introduction to existing work and to the characteristic problems involved in music recognition. The following section describes the method and the experimental test system. Finally, there is an evaluation and a discussion including future work.

2 Background

In this section I will present some related work, followed by an introduction to the terminology and how tone recognition differs from voice recognition.

The Tuneserver is a closely related project ([Prechelt and Typke, 2001]) and software system. It recognises a musical tune whistled by the user, finds it in a database, and returns its name. It is one of several systems that use a common music representation, proposed by Parsons ([Parsons, 1975]), which don't use the absolute tone values but only the direction of the melody [2]. Duration and rhythm are also ignored. The article mentions a similar project, *Query by Humming* ([Ghias et al., 1995]), which uses autocorrelation instead of spectrum analysis like the Tuneserver. The MELody inDEX, system was designed to retrieve melodies from a database on the basis of a few notes input sung into a microphone[McNab et al., 1997]. Its pitch determination algorithm finds repeating pitch periods in the waveform without performing spectrum analysis. A Pitch Determination Algorithm, PDA, is responsible for finding the correct pitch of a tone (see [Liljencrants, 1997], p16-18).

and representation.

[2] An example: C4, E4, G4, C4 would be represented by Up, Up, Up, Down.

2.1 Sinus Tone vs. Voice

Here, a tone is defined as a signal with a single fundamental pitch. I will not deal with overtones, but assume that it is possible to establish the fundamental pitch by the technique described below. A voice is generally producing a sound that is more complex than a single sinus tone. According to the source-filter theory, the voice can be perceived as having a glottal sound producing source due to vibrations of the vocal chords as the air stream from the lungs passes through the larynx. The vocal tract performs filtering and the sound radiates from the nostrils and the lips.

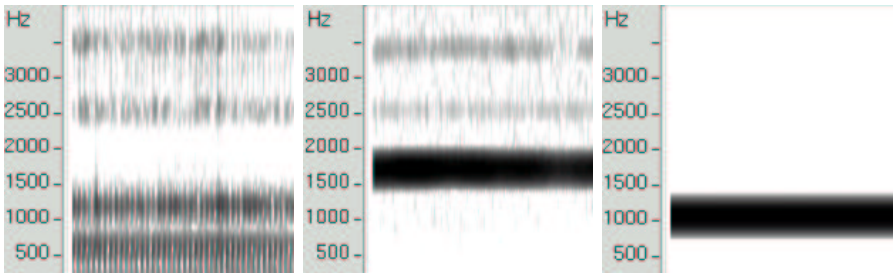

Figure 1: Spectrograms of singing, whistling and a sinus tone.

This figure shows that the sound produced by the voice can be more complex than the sound of a whistling, which in turn is more complex than a sinus tone. Although the whistle in this example is quite clean, it often happens that the tone is scrambled by the friction sound of the wind passing the lips. A computer generated sinus tone has the potential of being the cleanest of the three, as it does not contain the noise that is present in analogue sounds. The sampling quality, in terms of amplitude quantisation and sampling frequency, is crucial to the noise level and the sound quality of any sampled sound.

In speech recognition, the number of parameters needed to describe a voice is higher than merely a pitch. Apart from the pitch of the voice, additional parameters are needed to describe the filtering, formants F1-F4. Dealing with a monophonic signal, only one parameter is needed to capture pitch. For more complex processing, where more information than pure pitch is needed, more parameters are needed even for monophonic signals.

In this assignment only whistling has been used, which is relatively uncomplicated. Other instruments have characteristics that may make them less suited for the described method.

Speech recognition and music recognition share a number of difficulties. In speech, there are no clear boundaries, neither on the phone level nor on the word level. In a piece of music produced by whistling, this is often the case too. There is vibrato and continuous glides between notes. Distinguishing noise from the instrument is the same kind of problem as distinguishing noise from a voice. The noise may come from various sources, like room acoustics, other

sound sources, bandwidth limitations of the sound channel, radio static and reverberation to mention a few. Errors in music performance, like being out of tune or continuously changing key can be thought of as a kind of counter part to speech disfluencies.

2.2 Characteristic Problems

The monophonic music recognition task involves a number of problems:

1. Separating the tone from noise. The instrument itself can emit sounds that are not tones. Whistling produces a fricative sound when the air passes the lips. Background room noise and room acoustics can also contribute to the noise level. The quality of the microphone, the position of the sound source, and the quality of the analogue to digital conversion are also of importance. If spectrum analysis is used, and as long as the noise does not have a higher power at some frequency than the tone has, it will not confuse the pitch determinator. Noise can confuse the preprocessor that attempts to find the start of the signal by measuring the input wave's amplitude. If the noise is stronger than the preprocessor's threshold value, it will pass that stage as a tone.

2. Separating the tone from silence. Especially if a piece of music has a sequence of notes of the same key, it is important to be able to make out the silence in between them. Otherwise it will overlap with a piece of music that has the same note, but with a long duration. This problem can be handled by threshold values, on both the amplitude and the power at different stages in the process.

3. Classifying frequencies into distinct notes. A related problem is if the musician systematically plays a few percent off the exact frequencies, for example with a guitar that is in tune with itself, but not with a tuning fork.

4. A song in the database may have been played in one key and in one tempo, and the candidate song to be recognised may be played in another key and another tempo. This is related to the way the music is finally represented. The most direct way would be to simply store either absolute frequencies, or the tones they represent. This would however lead to problems when there is a difference in key. Parsons' code does not have this problem, since it only represents whether a note is above, the same or below the previous note. Downie and Nelson use intervals instead, which represent the relative direction and distance between every note ([Downie and Nelson, 2000]).

5. Errors in the performance of a piece of music. Just like there exist speech disfluencies, there exist errors in music performed by humans. Extra pauses, wrong/extra/missing notes etc. Improvisation is not an error, but it certainly makes things harder.

3 The Experiment

In this section I will describe the hypothesis and the experiment. The hypothesis to be evaluated is that it is possible to use a number of techniques from the speech technology domain to perform monophonic music recognition on the basis of wave files. The system was implemented as modules connected in a standard UNIX pipe, where the first module either takes its input from a microphone or a wave file. The languages C and Perl were used.

The following terms will be used in this section:

- Sampling rate. The number of samples that is used per second. The sampling rate sets a limit on tones' maximum frequency. The Nyquist frequency is the bandwidth of a sampled signal, and is equal to half the sampling rate of that signal. It is the highest frequency that can be represented unambiguously. Higher frequencies are aliased.

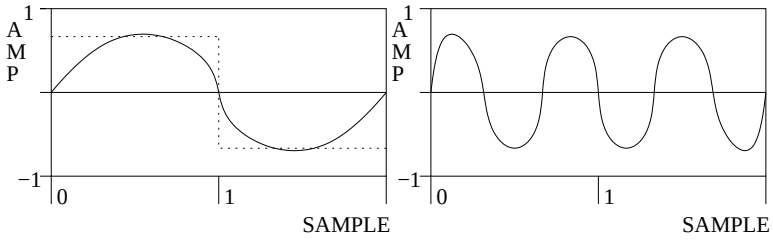

Figure 2: The curves denote the analogue wave to be sampled. The dotted line marks valid sample values. In the left case, the wave frequency is exactly half the sampling rate, and the samples will properly describe the wave. In the right case, the wave frequency is higher than half the sampling rate, which results in samples that do not reflect the tone.

- Sample blocks. The continuous stream of samples, each representing `1/(sample rate)` seconds, is divided into blocks that will be the smallest time unit the program will work with. A 0.1s sample block consists of `0.1*(sample rate)` samples etc. Tone changes within a block will be lost, unless they continue on to the following block.
- Power. The pitch determination algorithm computes a power spectrum for each sample block. The power spectrum shows the signal's energy for every frequency (see further [Liljencrants, 1997]).
- Threshold value. In this report, a threshold value is a limit between low and high values, typically used to separate noise from potentially interesting input. The threshold values in this system are expressed in percent, which means the data are normalised before the discrimination takes place.

This is an overview that describes how the process works:

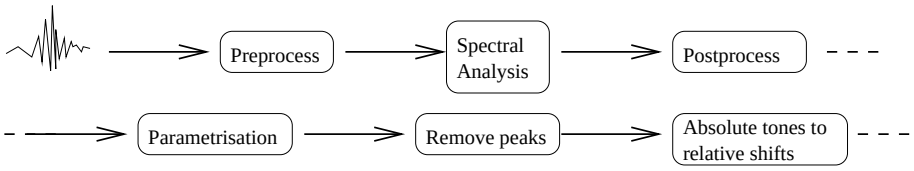

Figure 3: The base process.

This base process is the starting point for both the building of the database of known songs, and the music recognition. In both cases, an additional module is appended to the base process. Each step is described in detail below.

Preprocessing. The preprocessor takes raw samples as input and tries to determine where the music starts and stops. There can be noisy silence including clicks and static before and after the music. Noise like clicks is not removed by threshold values on amplitude or power, since it can be quite strong. The idea is to identify very short transients in the amplitude, where silence goes to a short loud sound, followed by silence. The start and end of the music is recognised by a loud, not so short sound bordering to a stretch of silence which is connected to either the start or end of the sample stream. The required degree of loudness and the minimum length are determined by threshold values. Once the preprocessor has determined what part of the stream is music, it normalises its volume before sending it on to the next module.

Spectral Analysis. The spectral analysis is part of the pitch determination algorithm. It divides the input stream into sample blocks. For each block, a fundamental pitch is determined. This is done by computing a power spectrum for each block (see further [Pickover, 1990]). A power spectrum uses a discrete Fourier transform to sum up, for each sample in the block, the energy at certain frequencies. The frequency with the highest energy is selected as the dominant pitch. It is possible but not necessary to compute the power for all discrete frequencies that the human ear can register. The frequencies used by the program are the frequencies of octave 4 to 8, twelve notes in each (`C4, C#4, D4, D#4` etc). The program also has a resolution parameter which makes it possible to divide each of these steps into smaller steps. In case the tones are around 50% off a clean note, rounding errors might occur.

Postprocessing. Before determining that there is a tone at a certain pitch in the block, all blocks' powers are normalised. This is done by determining the maximum power of the melody, and converting each power to its percent of the maximum. This is to make it easier to use threshold values that are not absolute for what is silence and what is a tone. Furthermore, the postprocessor repeats the work of the preprocessor, in order to capture any mistake that was

made there. It will determine the start and the end of the music by locating the place where initial silence is changed into tones of at least a minimal length and power.

Parametrisation. Finally, the array of powers are compared to the silence threshold, and the result is sent to the next module. For powers below the threshold, a symbol representing silence is sent, and for the others, the index (scaled according to the resolution parameter) of the note representing the dominant pitch is selected.

Peak Removal. A certain kind of noise can result in quick but large transients, looking like `43, silence, 45, 189, 41, ....` This can happen when the whistler accidently has blown into the microphone, or something has gotten in contact with the microphone. These are replaced by silence, as that results in a smaller penalty in the recognition process. There is always a risk that something is removed that was done intentionally by the musician, but it seems quite unusual with seemingly random notes several octaves above or below the surrounding notes' octaves.

Absolute Tones to Relative Shifts. In order not to be dependent on certain keys, the absolute tone numbers are changed into relative steps. This is done by, for each note, taking the direction and distance to the previous note. The first note is 0. For example, `44, 46, 42` is represented by `0 2, -4`, which is also the representation for the same note sequence transposed to other keys, like `45, 47, 43` or `32, 34, 30`. The difference between this and Parsons' code is that here the distance is included too, not just the direction.

The piece of music is now represented by a sequence of shifts, each representing the number of seconds of the size of the sample block, which can be around 0.1s. This opens for encoding both pitch and duration in the database. However, due to time limitations, this experiment only covers encoding of pitch. It would be an interesting task for the future to include duration in the representation.

Sequences of the same tone are collapsed in order to decrease the amount of data. The recognition uses an type of edit distance that does not punish differences in length of sequences of the same symbol, but having less data decreases the memory consumption and speeds up the process.

Song Database. The data is now ready to be stored, along with its originating file name for identification. The format is:

```
|''filename'' length0,tone0 length1,tone1 length2,tone2 ...

An example:
|"gubben noa" 3,0 3,0 3,0 3,4 3,-2 ...
```

Recognition. Recognition is done by computing the edit distance between the input sequence and all reference sequences in the database. This involves computing the local distances. By a small modification to the standard way of doing this, the method was adapted to data that is not merely equal or not equal, which is the case when comparing characters of strings. The notes are expressed as relative shifts, which means we rather want to capture the grade of difference than merely 0 or 1 depending on whether characters match or not. The local distance between two notes is defined as `abs(n1 - n2)`, eg `abs(3 - 3) = 0, abs(-2 - -3) = 1`. The global distance compensates for speed differences between the reference and the candidate. This has both advantages and disadvantages. The advantage is that you don't need to know the tempo in which the reference music was played, when you record music to be recognised. The disadvantage is that you loose important information like tone duration and pause duration. However, for small to medium size collections of songs, this can still work. If the songs are in a genre that are using a limited set of tones, the problem of overlaps increases, and could lead to worse performance. When the global distances have been computed for all reference songs, they are ranked by lowest distance and if the best match is below an acceptance threshold value, the system will consider it recognised as a song in the database. If the best match is worse than the acceptance threshold, it is considered to be an unknown piece of music.

A few additional programs were made to help in the process. A visualisation tool was made which draws a spectrogram with the selected pitch marked out, along with the determined beginning and end of the music. This helped in manual tuning of the parameters. A second tool using distributed computing was made for brute force tuning of parameters. As the number of tuning parameters increased, it helped in finding some successful configurations.

4 Evaluation

The hardware used was an IBM ThinkPad T23. The built-in microphone is not optimal, but I assumed it would be sufficiently accurate for this experiment. The same assumption was made for the built-in audio controller [3]. The sampler program was set to 16 bit at a rate of 16000 times/sec. The Nyquist frequency is 8kHz which is sufficient for our purposes.

The evaluation of the system will consist of training the system with 13 pieces of music, each in two recorded versions. It should preferably have been trained more rigorously, but there was no time. Then 26 candidates, all separately recorded, will be run through the system. 13 of these are songs that the system has been trained with and should recognise, and 13 are songs that the system has not been trained with. This way, it will be possible to tell the ability to recognise songs in the database, and the ability to reject songs not in the database.

[3] Intel Corp. 82801CA/CAM AC'97 Audio (rev 1)

There are obvious flaws in this evaluation. The number of songs is very low, with more songs the results would be more reliable. Preferably there should have been more than one person producing the whistled samples, a system like this should be whistler independent and potential problems associated with different persons will not be revealed. The three different versions of the songs in the database, two training versions and one evaluation version, were sampled at different times without intention of matching key and tempo.

Precision:	45%
Recall:	38%

Table 1: The ability to recognise songs in the database. The precision is calculated as *# of accepted relevant songs / # of accepted songs = 5/11* and the recall is calculated as *# of accepted relevant songs / # of relevant songs = 5/13*.

Precision:	73%
Recall:	84%

Table 2: The ability to reject songs not in the database. The precision is calculated as *# of rejected relevant songs / # rejected songs = 11/15* and the recall is calculated as *# of rejected relevant songs / # relevant songs = 11/13* (relevant means that which should be rejected).

It is somewhat difficult to compare these results to other systems' results. First, because the other systems I have found return a list of songs as an answer whereas this system selects one song only. Second, the other systems have not been trained on the same kind of input as this system. Instead of using a sound wave, other systems tend to use existing symbolic representations that are 100% correct. The Super MBOX is a system using autocorrelation for pitch determination and a dynamic time warp measure for tempo differences. It reports a top-20 success rate, meaning the correct song is in a list of the 20 best matches, of 75% based on a database containing 13000 songs and a test material of 2000 sound clips ([Jang et al., 2001]). It is not clear how that evaluation deals with songs not known to the system. The Tuneserver reports that the system returns the correct song somewhere in a top-40 list 77% of the attempts, and in 44% the best match was correct. This excludes attempts with songs not known to the system. For my system, the corresponding rate is 38%.

The system was not constructed with speed in mind, and has not been optimised. Recognition is very slow, the test run above of matching 26 songs to a database trained with 26 samples took around 13 minutes on a 1GHz, 256MB RAM computer.

5 Conclusion

The task in this experiment was to see whether a proposed method consisting of combining several basic speech recognition techniques could be a successful way of performing monophonic music recognition. An experimental system was built and tested using sampled whistling for training and recognition. The pitch determination was based on spectral analysis. The symbolic representation of songs were integer shifts representing direction and distance to the previous note. The recogniser computes the global distance, using the standard way except for the way local distance is computed. This has the advantage of not being sensitive to tempo differences, but it also means that duration is lost. This makes the music recognition somewhat primitive. A drawback with the current system is that it will not be able to match parts of songs, as any input is mapped to the entire songs in the database.

The evaluation, despite the flaws mentioned, indicate that it is possibly not too far behind the other systems. The only comparable rate found was the recall of Tuneserver, 44% and the recall of this system, 38%. This may not mean anything however, as the evaluation of the Tuneserver is much more detailed and exhaustive than this report's evaluation which only included a small number of tests. This system was constructed in a very short time as a course assignment, and it is expected to perform worse than bigger systems.

5.1 Future Work

There are many things that could be improved.

- It should be possible to include tone duration as a relative unit, simply by analysing the song after the relativisation step, and determine the existing length of tone sequences. They could then be normalised. If duration is a relative unit, it could be integrated in the distance table.
- For music that is systematically out of tune by a certain percent, the program could recognise this before parametrisation, and compensate. This could improve parametrisation, especially if the tones are around 50% out of tune.
- Other ways of doing recognition should be evaluated. HMMs and Viterbi are successfully used in speech recognition, and could work well here too.

The system and whistling data is available for download at
`http://stp.ling.uu.se/~perweij/gslt/kurs/speech/`.

References

[Downie and Nelson, 2000] Downie, S. and Nelson, M. (2000). Evaluation of a simple and effective music information retrieval method. In *Proceedings of the 23rd annual international ACM SIGIR conference on Research and development in information retrieval*, pages 73–80. ACM Press.

[Ghias et al., 1995] Ghias, A., Logan, J., Chamberlin, D., and Smith, B. C. (1995). Query by humming: musical information retrieval in an audio database. In *Proceedings of the third ACM international conference on Multimedia*, pages 231–236. ACM Press.

[Jang et al., 2001] Jang, J.-S. R., Lee, H.-R., and Chen, J.-C. (2001). Super mbox: an efficient/effective content-based music retrieval system. In *Proceedings of the ninth ACM international conference on Multimedia*, pages 636–637. ACM Press.

[Liljencrants, 1997] Liljencrants, J. (1997). Speech signal processing. Handbook of Phonetic Science.

[McNab et al., 1997] McNab, R. J., Lloyd, A., Smith, D. B., and Ian, H. (1997). D-lib magazine: The new zealand digital library melody index. World Wide Web. http://www.dlib.org/dlib/may97/meldex/05witten.html.

[Parsons, 1975] Parsons, D. (1975). The directory of tunes and musical themes.

[Pickover, 1990] Pickover, C. A. (1990). *Computers, Pattern, Chaos and Beauty*. St. Martin's press, Inc.

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