Assignment 2 - Song Recognition

Yann Debain
debain@kth.se

Harsha Holenarasipura Narasanna
harshahn@kth.se

April 29, 2018

1 Introduction

Goal is to design a pattern recognition system capable of distinguishing between a few brief snippets of whistled or hummed melodies. In this task 2, we present a robust feature engineering mechanism. We are given a set of features for the different songs and we post-process them to extract the key features for melody recognition.

The code is enclosed into the package **files/GetMusicFeatures**, you will find the design of our feature extractor and different tests on it.

2 Design of the features extractor

The requirements for the features are :

— They should allow distinguishing between different melodies.

— They should also allow distinguishing between note sequences with the same pitch track, but where note or pause durations differ.

— They should be insensitive to transposition.

— In quiet segments, the pitch track is unreliable and may be influenced by background noises in your recordings.

— They should not be particularly sensitive if the same melody is played at a different volume. (optional)

— They should not be overly sensitive to brief episodes where the estimated pitch jumps an octave. (optional)

Because the hearing is sensible on a logarithm scale, we will work with the logarithm of the features.

2.1 Feature profile of the given melodies

The first features are given with the function *GetMusicFeatures.m*. The features are pitch, correlation and intensity. Each of the features is estimated depending on a frame-length. For music, a typical frame-length is $\sim 20ms$. For the project, we used a frame-length of $30ms$.

Feature profiles are shown below in figure 1 and figure 2.

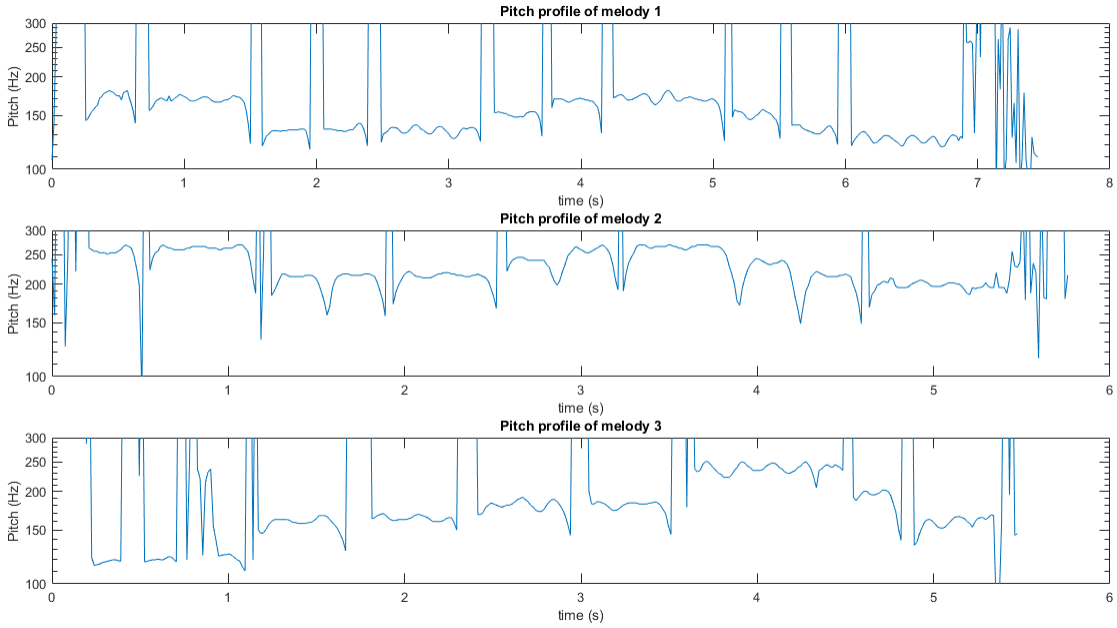

FIGURE 1 – Pitch estimated for the different melodies

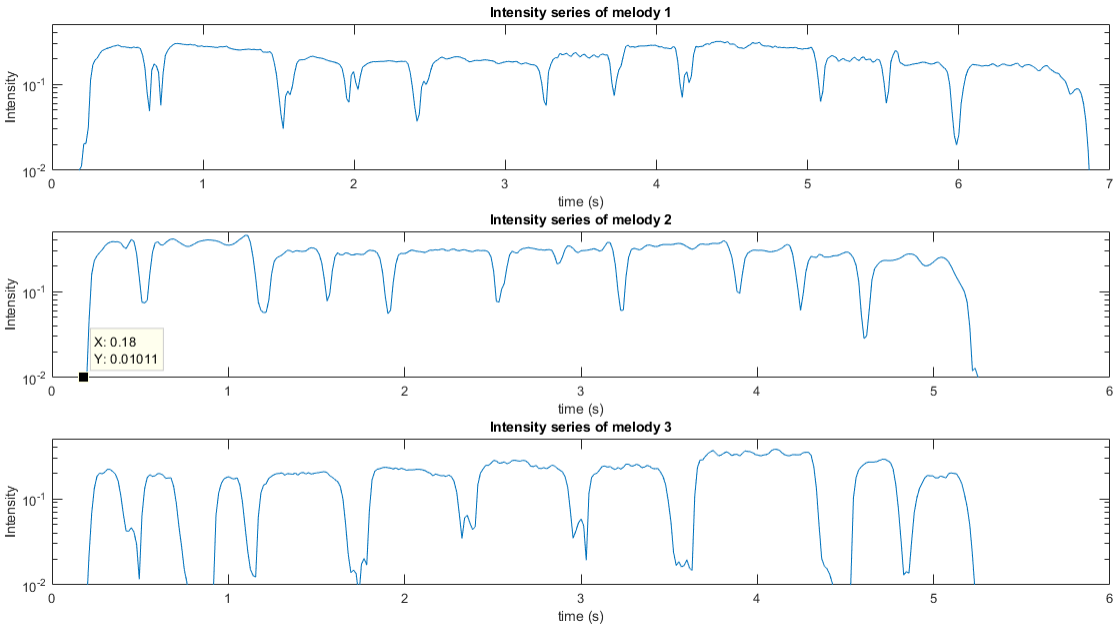

FIGURE 2 – Intensity estimated for the different melodies

2.2 Remove non-harmonic components

One of the important features in a melody is the pitch. If we look at the pitch estimated given by the function *GetMusicFeatures.m* we can see that we have to distinguish between harmonic components (that represent the melody) and non-harmonic components (that represent noise and silence).

By observing the pitch, correlation and intensity estimate from previous figures we can state :

— There is a high correlation in harmonic segments and a low correlation in non-harmonic segments.

— There is high intensity in harmonic segments and low intensity in non-harmonic segments.

— Silences are located in high frequencies segments.

To be sure to remove the non-harmonic components (noise and silence) we will use arbitrary thresholds on the pitch, correlation and intensity estimate based on our observations.

The threshold on the correlation will remove the noise (we assume it is white) while the thresholds on the intensity and the pitch will remove the silence (silence has high-frequency pitch and low intensity if it is not too much influenced by background noise).

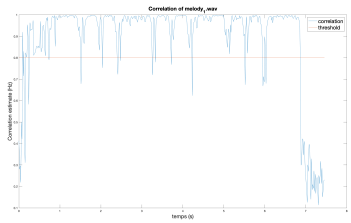

FIGURE 3 – correlation estimate and the threshold for $melody_1$

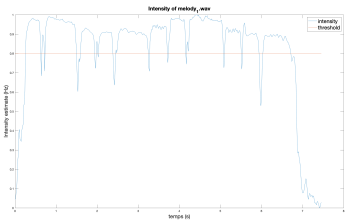

FIGURE 4 – intensity estimate and the threshold for $melody_1$

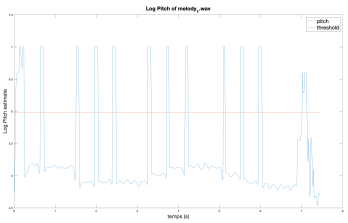

FIGURE 5 – pitch estimate and the threshold for $melody_1$

Here the thresholds are set arbitrarily :

— threshold on correlation thR = 0.8

— threshold on correlation thI = mean(I) (after normalization)

— threshold on pitch thP = mean(P) + std(P)

By using all the different thresholds we can remove the non-harmonic components. We can see the results on the pitch track in FIG.6.

2.3 Add noise in pitch

We know that noise can provide information to the HMM. As we want the output distributions of the HMM to be a Gaussian Mixture Model, we add a simple Gaussian noise to the non-harmonic state. The mean of the Gaussian noise is chosen at the minimum pitch frequency in the harmonic components in order to reduce the number of Gaussian distributions in the Gaussian Mixture Model. The variance of the Gaussian noise is arbitrarily fixed but set in order to create instantaneous change between segments.

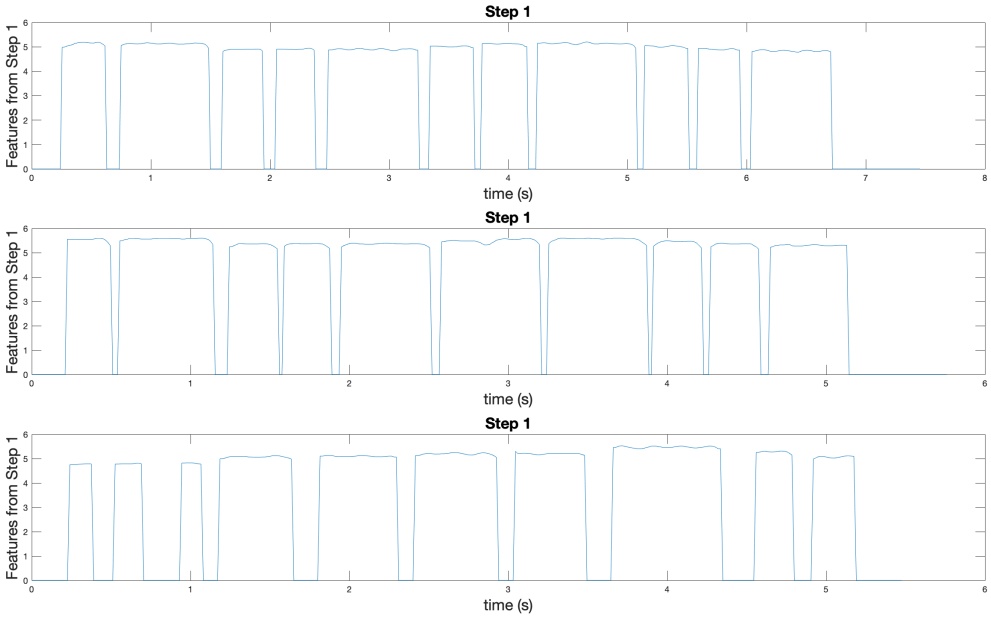

FIGURE 6 – Features with non-harmonic component

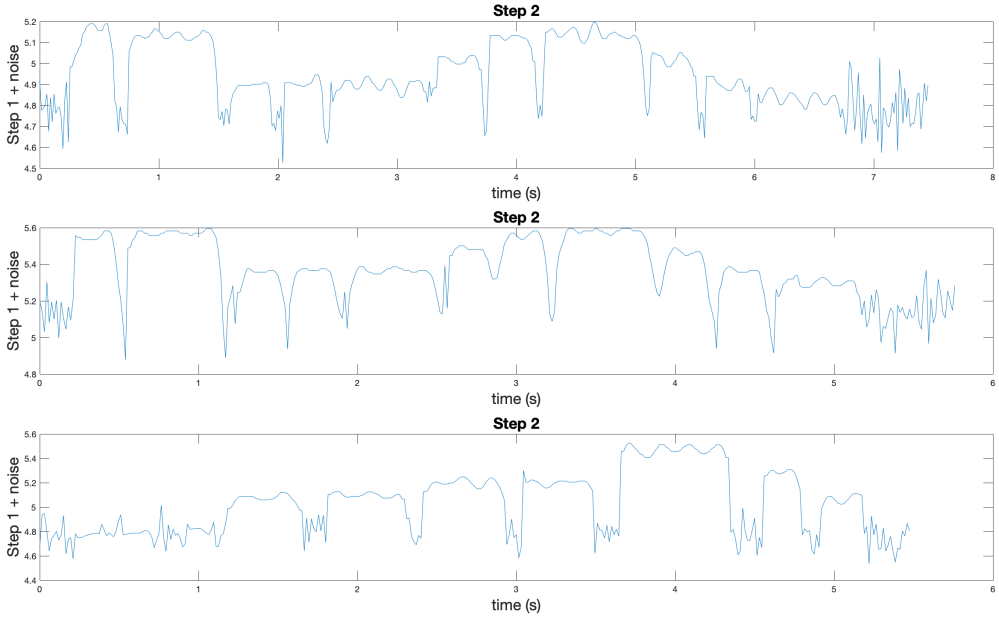

FIGURE 7 – Added noise to features

2.4 Semitones transformation

The formula to convert pitch to semitones is : $s = 12log_2(\frac{f}{f_0})$ where f is the pitch frequency of our features and f_0 is the fundamental.

We consider that the minimum pitch frequency is the fundamental. The transformation into semitones make the features transposition independent.
Suppose the song is transposed by a factor of α, then $s_t = 12log_2(\frac{\alpha f}{\alpha f_0}) = 12log_2(\frac{f}{f_0}) = s$

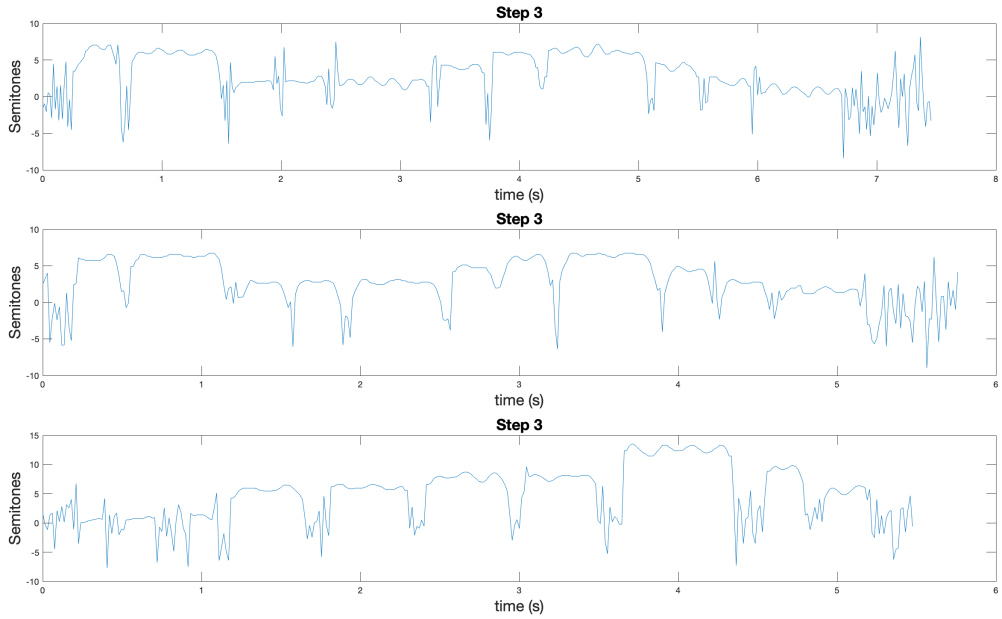

FIGURE 8 – Semitones transformation of the features

3 Test of the feature extractor

In our case, $melody_1$ and $melody_2$ possess exact underlying information. In either samples, we performed post-processing and results are shown in figure 9. We observe from our features that we can distinguish different melodies : features from $melody_1$ and $melody_2$ are alike even if the notes and pause duration differ while the features from $melody_3$ are different.

We can notice that the background noise does not have a big influence on our features and we will see in the next section that the features are transposition independent.

By looking at the range of the semitones we can determine the number of Gaussian distributions in the Gaussian Mixture Model necessary for the HMM.

In order to model the 3 features, we will use the highest number of Gaussian distributions (in the case that we need less Gaussian distributions, the weight will be set to 0). From the feature extractor, the highest range for the features is 12 semitones, so we will use a 12 Gaussian Mixture Model distribution.

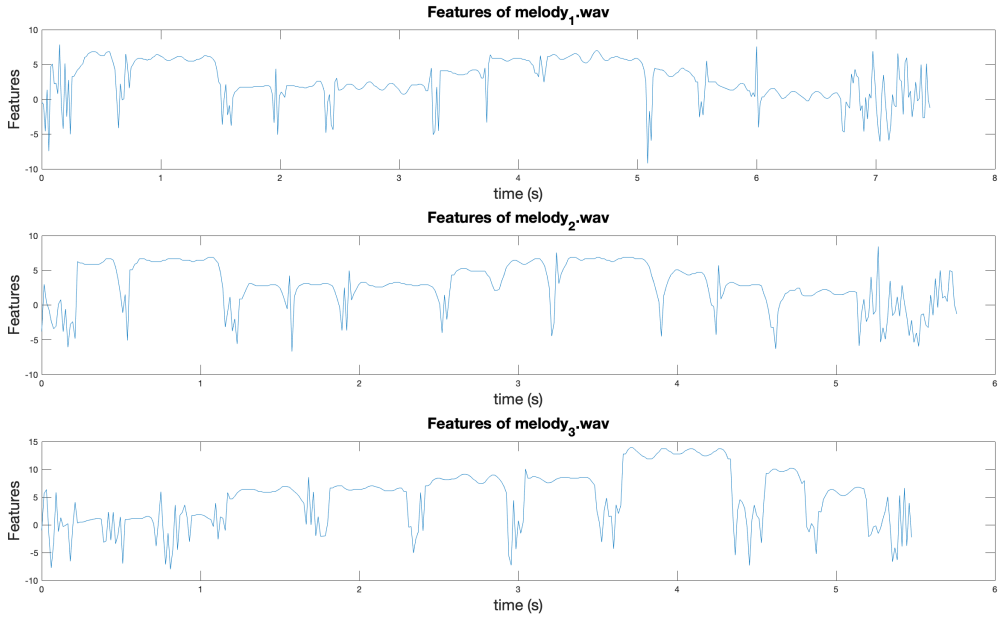

FIGURE 9 – Test feature extractor

3.1 Test on the transposition

To verify that our features are transposition independent, we multiply the pitch feature track returned by *GetMusicFeatures* by 1.5, and use this pitch track in our feature extractor.

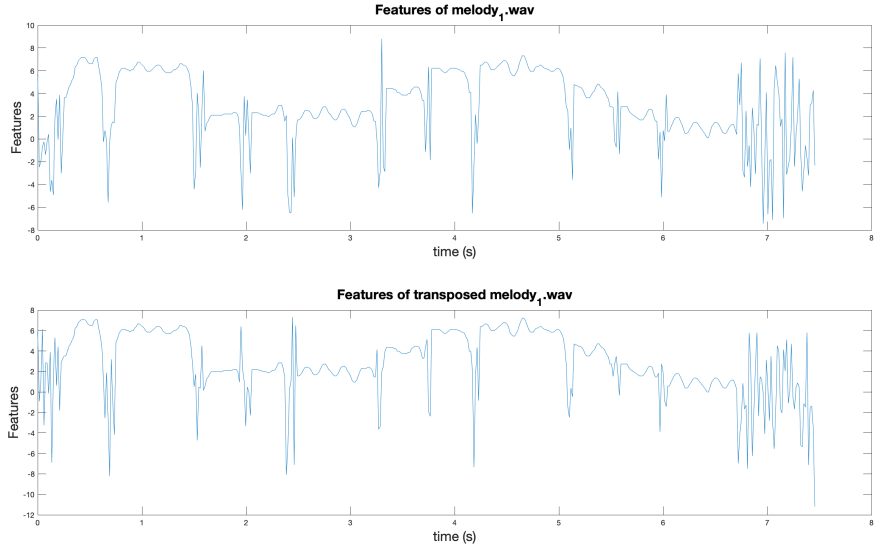

FIGURE 10 – Test on the song with transposition on $melody_1$

We can see from the figure above that we have the figure as the reference (top figure). The only difference is on the Gaussian distribution on the non-harmonic state (because it is set randomly).

3.2 Test on the volume (optional)

For the test on the volume, we multiplied the sound by 10.

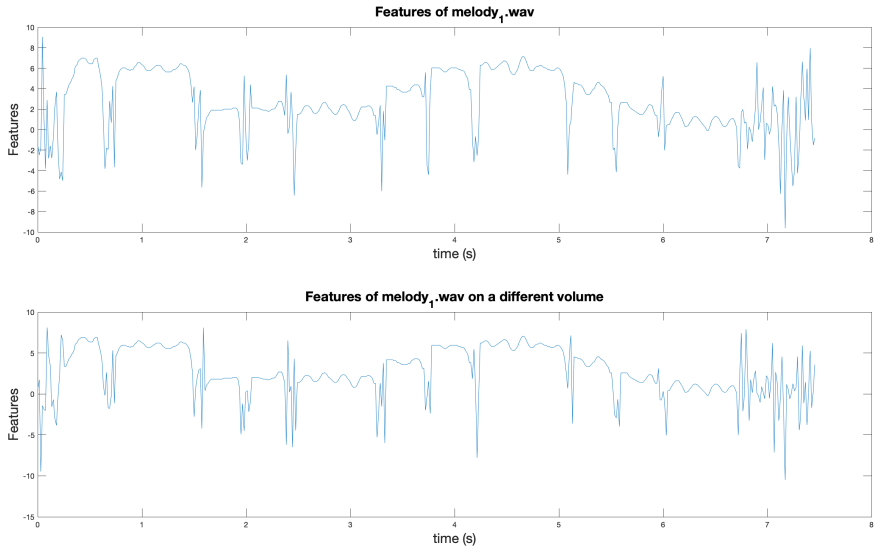

FIGURE 11 – Test on the song with a different volume (x 10) on $melody_1$

We can see from the figure above that our feature extractor gives good results on changing the volume in a fixed manner.

The same test has been done by turning the volume down (we multiplied the sound by 0.1) and gave the same results.

We have also test varying volume changing, from the tests we have observed that our feature extractor is resistant to little variation in the volume (a variation of $\sim$ 5dB during all the duration of the melody). Our feature extractor still faces some issues with large volume variation.

3.3 Test on the pitch halving/doubling (optional)

To test the resistant to pitch halving/doubling we multiply by two or by a half a certain number of pitch frequencies (the pitch frequencies are chosen randomly).

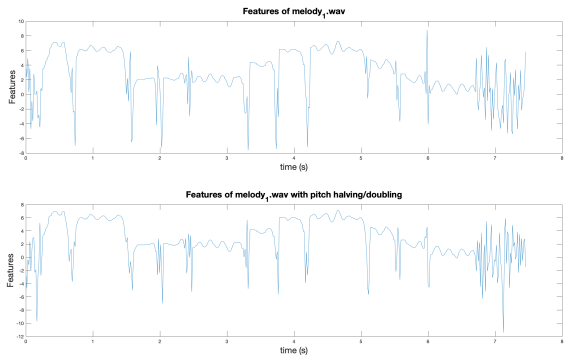

FIGURE 12 – Test on pitch halving/doubling on $melody_1$ with 10 bad pitch estimates

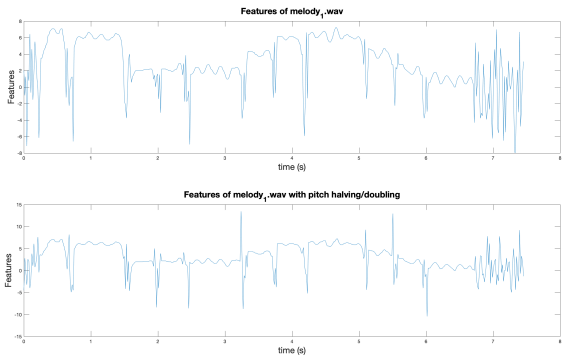

FIGURE 13 – Test on pitch halving/doubling on $melody_1$ with 20 bad pitch estimates

To take care of the pitch halving/doubling we apply a median filter on the pitch feature to eliminate these bad pitch estimates : the median filter smooths the pitch feature and removes these "artefacts".

We can see from the figure above that our feature extractor gives good results. However, the median filter gives good results but it has its limitation, we only use a third order median filter to not extremely modify the pitch track but for some frequencies halving/doubling a higher order would be necessary.

4 Limitation of the feature extractor

The main limitation of our feature extractor is the different thresholds that have been set arbitrarily. These thresholds work with the given melodies but might not work on different melodies. For a better feature extractor, we should have automatic thresholds that adapt themselves to the features from *GetMusicFeatures*.

As all our feature extractor is based on those thresholds, the definition of them is very important. A bad separation between harmonic and non-harmonic components will drive to bad features or might remove important information.

With big varying volume issues still, arise. As some of our thresholds are based on the intensity estimate, some harmonic components are removed and give bad features at the end. Further, if there is a lot of pitch halving/doubling, the median filter would not be sufficient and we need to find another way to cope with it.

With all these limitations, the feature extractor can be tricked. One can generate two melodies with same variations in the intensity and/or the pitch or in the opposite way, the feature extractor can generate features based on the thresholds and make two different melodies have the same features.

5 Conclusion

As a conclusion, we have seen how to extract important features in a melody in order to distinguish between different melodies.

Through observations and computations, we have verified that our feature extractor is transposition independent. We have upgraded our feature extractor to be independent of pitch halving/doubling (with restrictive conditions) and made robust against volume factor.

6 Annexes - Code

6.1 Extract the harmonic components

```
function newPitch = removeNoiseAndSilent(frIsequence)

    P = log(medfilt1(frIsequence(1,:)));
    R = frIsequence(2,:);
    I = log(frIsequence(3,:));

    lwb = 0;

    % Normalize
    In = (I - min(I))/(max(I) - min(I));

    % Threshold
    thI = mean(In);
    thR = 0.8;
    thP = mean(P) + std(P);


    newPitch = P;

    newPitch(In < thI) = lwb;
    newPitch(R < thR) = lwb;
    newPitch(P > thP) = lwb;

end
```

6.2 Add Gaussian noise

```
function newPitch = addNoise(pitch, stdnoise)
    newPitch = pitch;

    th = mean(pitch) - std(pitch);
    harmonics = pitch(pitch > th);
    f0 = min(harmonics);
    lwb = min(pitch);

    virtualNoise = f0 + stdnoise*randn(1, length(pitch == lwb));
    newPitch(pitch == lwb) = virtualNoise(pitch == lwb);
end
```

6.3 Transform into semitones

```
function semitone = transformSemitone(pitch)
    th = mean(pitch) - std(pitch);
    harmonics = pitch(pitch > th);
    f0 = min(harmonics);
    semitone = 12*log2(pitch/f0);
end
```

6.4 The whole post-process

```
function feature = PostProcess(features)
    newPitch = removeNoiseAndSilent(features);
    newPitch = addNoise(newPitch, 0.2);
    newPitchHz = exp(newPitch);
    feature = transformSemitone(newPitchHz);
end
```