ACA: Homework 1-PitchTracking

Due on 7.11 at 23:59pm

Alexander Lerch

Zhonglin Nian, Yiming Yin

Problem 1

A. Block-wise Auto-Correlation based pitch tracker

1. Implement a MATLAB function: **[xb, timeInSec] = myBlockAudio(x, blockSize, hopSize, fs)** for blocking the input audio signal and storing time-stamps (in seconds) for the blocks.
2. Implement a MATLAB function: **[r] = myCompAcf (inputVector, bIsNormalized)**that takes a block of audio and computes the ACF with optional normalization.
3. Implement a MATLAB function: **[f0] = myGetF0FromAcf(acfVector, fs)** that takes the ACF of a block of audio as input and computes the fundamental frequency f0 of that block in Hz.
4. Implement a MATLAB function: **[f0, timeInSec] = myPitchTrackAcf(x, blockSize, hopSize, fs)** that estimates the fundamental frequency f0 of the audio signal based on a block-wise autocorrelation function approach.

Implement Details

1. This function splits vector (audio) into overlapping blocks. We first need to determine how much blocks will be given **blockSize** and **hopSize**. To do so , we simply let

$$numBlocks = sampleLength/hopSize$$

In some tail cases that we need to pad zeros at the end of audio. The number of zeros padded is calculated by

$$numBlocks \times hopSize + blockSize - sampleLength$$

Note that the time stamp is defined by the starting sample in the block instead of the middle sample. To test this module, we tried some simple cases:

```
>> myBlockAudio([1;2;3;4;5;6;7;8],2,2, fs)

ans =

   1   3   5   7
   2   4   6   8

>> myBlockAudio([1;2;3;4;5;6;7;8],2,1, fs)

ans =

   1   2   3   4   5   6   7   8
   2   3   4   5   6   7   8   0

>> myBlockAudio([1;2;3;4;5;6;7;8],3,2, fs)

ans =

   1   3   5   7
   2   4   6   8
   3   5   7   0

>> myBlockAudio([1;2;3;4;5;6;7;8],3,1, fs)

ans =

   1   2   3   4   5   6   7   8
   2   3   4   5   6   7   8   0
   3   4   5   6   7   8   0   0
```

Figure 1: Simple Test of Blocking

2. This function calculate the auto correlation given an input vector. The implement is straight forward. To test this module, we compare the output of our implementation with **xcorr()** using all one vectors as input.

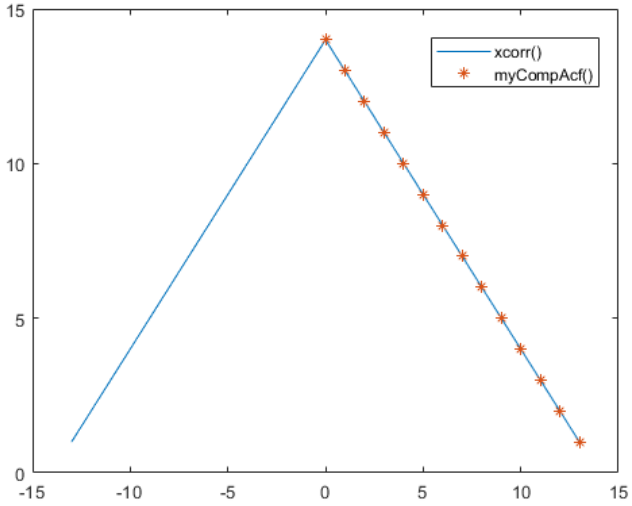

Figure 2: Simple Test of ACF

3. This function finds the fundamental frequency in ACF of one block. In solo sound source, we can use ACF to detect fundamental frequency by finding the period of repeated patterns. The intuitive idea is to find peaks in ACF and take average of distances between each peaks, which can be regarded as fundamental period. The following is a visualization of peaks picking in ACF(700th block of 01-D_AMairena.wav).

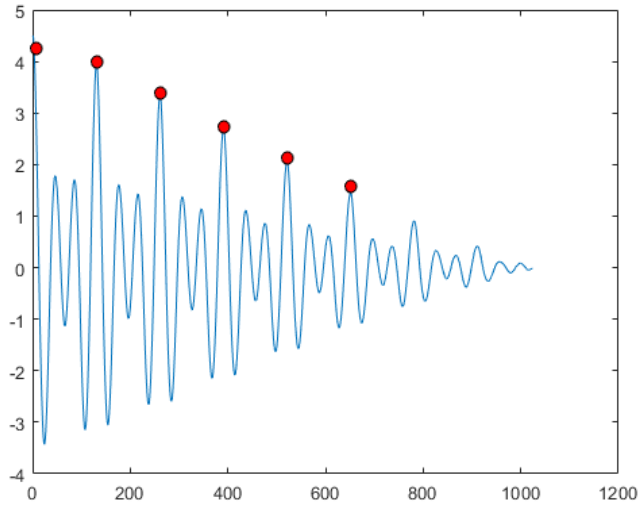

Figure 3: Peak Picking

4. This function just make use of previous functions that find fundamental frequency in each split blocks. (more evaluation in Section B)

Problem 2

B. Evaluation

1. In a separate script named assign1.m, generate a test signal (sine wave, f = 441 Hz from 0-1 sec and f = 882 Hz from 1-2 sec, fs = 44100), apply your **myPitchTrackerACF()** and plot the f0 curve (expected error $\leq 5\%$). Also, plot the absolute error per block and discuss the possible causes for the deviation.
2. Implement a MATLAB function: **[pitchInMidi] = myFreq2MidiPitch(pitchInHz)** that converts a column of pitches in Hz to pitches in MIDI
3. Implement a MATLAB function: **[errCentRms] = myEvaluation(estimation, annotation)** that calls **myFreq2MidiPitch()** to convert frequency f0 to MIDI pitch and computes the RMS of the error. Note: The evaluation should exclude the blocks with annotation == 0
4. Evaluate your **myPItchTrackerACF()** using the training set. Report the overall errCentRms of the training set. What are the potential solutions to improve their performances?

Implement Details

1. Result of evaluation

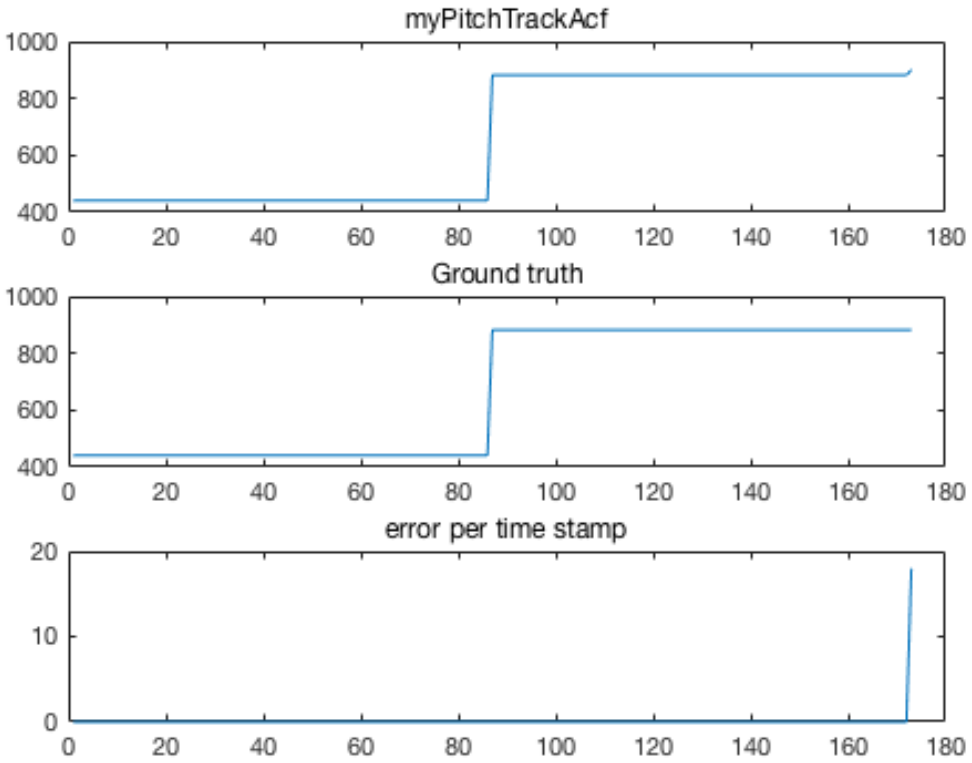

Figure 4: Pitch Tracker Error

The average error is 0.17%, but there is significant peak at end of the vector. This may due to at the end of block, we pad some zeros, making our tracker to detect higher frequency, which can be ignored. Other reasons lead to the error are blocksSize, hopSize, errors in peak picking, frequency resolution, etc.

2. Won't bother to explain this function.
3. Won't bother to explain this function.
4. Evaluate result:

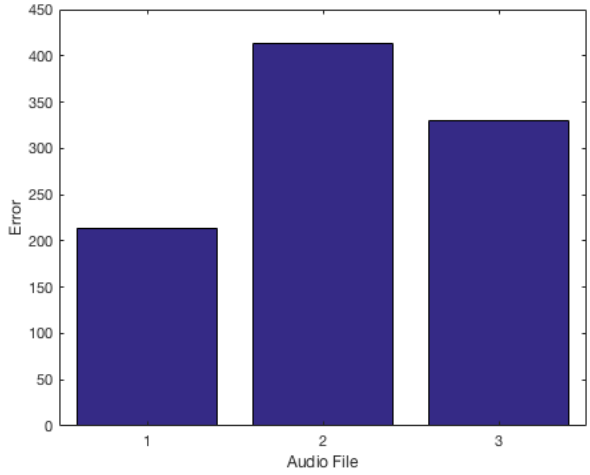

Figure 5: Error in Audio

Time stamp difference:

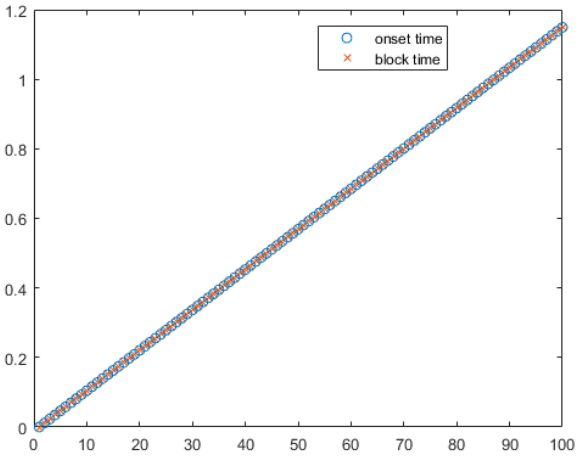

Figure 6: Time stamp compare

As we can see, there is no difference in time stamps.

Problem 3

C. (Bonus) Improving your Pitch Tracker To improve the system, we did two things:

1. Do a preprocessing to the input audio that behave as a low pass filter
2. Take several peaks when calculating the fundamental frequency per block and averaging the sample offset.

The result is :

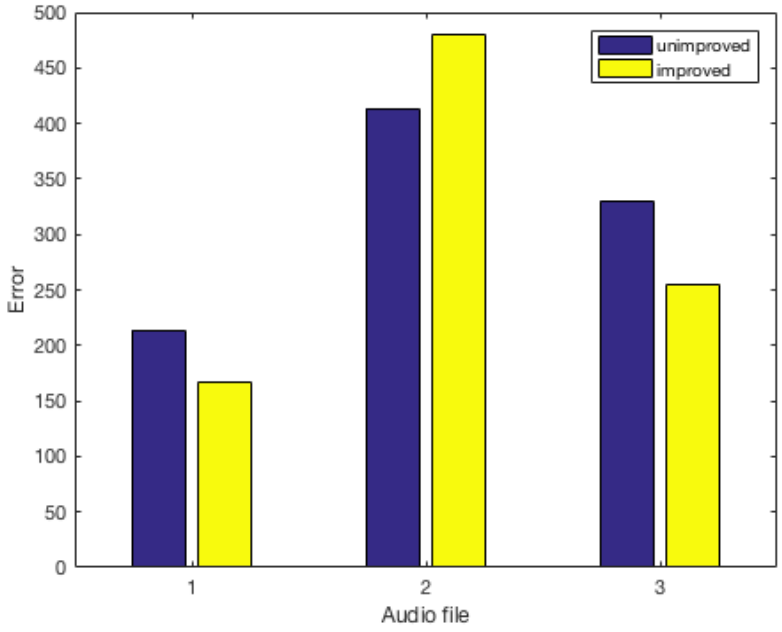

Figure 7: Improved

It's strange that for the audio 2, the error increases instead of decrease.