Fraunhofer
IDMT

IDMT Real Time Pitch Detection Library Documentation

21.05.2015

Fraunhofer IDMT
Hanna Lukashevich, lkh@idmt.fraunhofer.de
Sascha Grollmisch, goh@idmt.fraunhofer.de
Jakob Abeßer, abr@idmt.fraunhofer.de

Contents

1 Introduction

The Pitch Detection library by Fraunhofer IDMT analyzes audio input in realtime for detecting currently played/sung pitches. The library provides two processing modes for detecting monophonic input as well as polyphonic input using reference notes. The underlying algorithms are suited for all types of musical instruments as well as sung vocals.
The monophonic version returns the predominant fundamental frequency in Hz, e.g. 234.41 Hz. The polyphonic version requires additional information about the notes that are likely played. The algorithm searches for played notes within a range of two octaves above and below. Hence, the number of found notes and reference notes can differ. For each detected note, the fundamental frequency of the closest MIDI pitch will be returned, e.g. 440 Hz for the note A4. This means that the polyphonic pitch detection will not detect small tuning variations from the targeted note pitch (for instance due to vibrato). However, this functionality is provided by the monophonic pitch detection algorithm.

1.1 Supported Platforms

The library can be used on

- Windows (Windows 7 or higher),
- Mac OS X (10.6 64 bit or higher),
- Linux (CentOS 6 64Bit or higher),
- Android (armeabi-v7a), and
- iOS.

2 General Usage

The (block-wise) audio samples are passed into the library and analysed. The returned vector consists all detected fundamental frequencies of the notes currently played. The minimum buffer size of processable audio samples is 256. For larger buffer size values, the detected fundamental frequencies are avaraged so a buffer size of 256 is recommended for best transcription performance.

2.1 Package Content

This package contains the library for different plattforms including the header and lib files. It also contains sample files that demonstrate the usage for all supported platforms.

2.2 Usage under Windows / Linux / OSX / iOS

Include `PitchDetectionIDMT.h` in your project. Under Windows / iOS, link the static parts of the library. Under Windows / Linux / OSX, copy the dynamic library (`dll` or `dylib`) to your binary folder. The iOS library is compiled to be used in the simulator and on the device.

2.3 Interface Description under Windows / Linux / OSX / iOS

All methods return 0 on success. When different values are returned, check `getLastErrorMessage()` for the detailed error message. The class has to be initialized with the sample rate that should be 44100 Hz by calling `createPitchDetectionIDMT(44100)`. Hand over float-converted sample values (value range from -1 to 1) to the library via `processSamples()`.
The first parameter is the pointer to the data (first sample), the second the number of samples, the third will receive the pointer to the resulting frequencies and the fourth parameters returns the number of frequencies found (always one for monophonic detection). When using polyphonic mode, set the reference frequencies via `setReferenceFrequencies()`.
For switching between the two modes, call `setPitchDetectionIDMTVersion()` with `MONOPHONIC` or `POLYPHONIC` as parameter values. Finally, the allocated memory from the pitch detection can be freed by calling `deletePitchDetectionIDMT()`.

2.4 Usage under Android

Copy `armeabi-v7a` to libs folder in your Android project and the `de` folder to sources. The library only runs on the device itself not in the simulator!

2.5 Interface Description under Android

Create pitch detection instance via `createPitchDetect(int sampleRate)` and release memory when finished by calling `deletePitchDetect()`. Process audio samples in short format (`AudioFormat.ENCODING_PCM_16BIT`) to algorithm by calling `processSampleBuff`. The returned array contains all detected fundamental frequencies for all input samples. It contains one fundamental frequency value for the monophonic pitch detection. For switching to the polyphonic mode, call `setPitchDetectVersion()` with `MONOPHONIC_DETECT` or `POLYPHONIC_DETECT`. When using the polyphonic mode, set the reference fundamental frequency values via `setReferenceFrequencies()` by handing over an array with the expected fundamentla frequency values.

2.6 Samples

2.6.1 Windows/Linux/OSX

The sample folder contains the file `main.cpp` that shows how to hand over samples into the pitch detection library. The sample code contains a generated sinosoid signal and the results from monophonic and polyphonic detection are printed to the commandline. For easy compilation, the file `CMakeLists.txt` is included. It allows to create `make`, `Visual Studio`, and `XCode` files.
Under Linux and OSX, go to the sample folder and execute `cmake .` in the commandline which will crate make files. Then, compile via `make PitchDetectionSample`. An executable with same name will be created.
Under Linux, add the current directory to `LD_LIBRARY_PATH` so `libpitch_detection_idmt_shared.so` can be found.
Under Windows, run CMake with desired generator (Visual Studio, NMake, etc.) and compile `PitchDetectionSample`. The `pitch_detection_idmt_shared.dll` will be automatically copied to the executable folder. For further information, see www.cmake.org.

2.6.2 iOS

`PitchDetectIOs` demonstrates how to access audio samples from the microphone, convert them to float values, and pass them into the pitch detection for displaying the currently played frequency.

2.6.3 Android

`MainActivity.java` shows how to access audio samples from the microphone and hand them over to the pitch detection for displaying the currently played frequency.