Code description

The goal

The following code demonstrates the data processing done in the article. The code is written in Matlab. In particular, the most computation intensive and unusual part of data processing is shown - synchronization of data records (i.e. acceleration and sound signals on backpack, as well as sound on wall microphone). The most efficient way to align logger records is to synchronize them with the clock on a stationary computer (reference clock) that generates the discrete events that are transmitted to the loggers via infrared (IR) pulses (environmental noise can affect IR transmission, but the computer record of all timing events is error-free). This code performs the alignment of one logger record with the IR pulse train stored on the computer (if no computer or IR transmission is available, one can use the same code with minimal modifications to align records from several loggers to the record of one selected logger). The code has been tested in Matlab versions 2011b, 2013a. It also works in version 2010b, but with disabled GPU processing (very slow).

What does the code do?

The code takes two files as input. One of them is recorded with the stationary computer, its file extension is LVD. The second file is recorded with a logger and has extension DAT. The LVD file contains usually two channels – the wall microphone and the synchronization channel. If audio playback is used in the session, it also contains the playback record. The DAT file contains acceleration data, backpack sound data and the digital synchronization record.

The main script "Batch" produces several combined LVD files (output files) that all have three data channels. Each numbered file contains one separate vocalization fragment (fragments without vocalizations are omitted to save memory and time). The output files can be viewed using a very simple viewer script "SimpleViewer": Calling SimpleViewer(x), where x is the file number will display spectrograms of all three channels and indicators of timer points when vocalization detections took place (and IR pulses were emitted). In addition, Batch saves the generated figures in EMF format in the same directory where the original data is stored.

How to test the code?

1. Unpack the provided ZIP archive in a directory, it will produce several Matlab scripts and functions. All of them except Batch and SimpleViewer are daughter functions that will be called indirectly when you run Batch. There will also be three subdirectories. The directory "+fb_abstract" contains abstract classes definitions used by "fb_plugin_detect_song_harmonics2A.m" function. The directory "Data_original" contains two short data files provided for testing purposes. These files contain short records of bird "g2k8" (session without noise). The directory "Settings" contains the stationary computer settings file. This settings file is matched to the format of output files. In its text version (.txt) you can

see channel definitions. The settings files are not used by the code and are provided exclusively for reference.

2. Run Batch.m. If it is executed successfully, the novel directory "Data" is generated with subfolder containing the output data files. Note that data alignment needs some time. If you have CUDA-compatible GPU, the job will be finished in about a minute. But if you do not have a CUDA GPU, all computations will be done on the main CPU and it can take more than 25 minutes even for this small test dataset. The default GPU to run the GPU code has number 2 (Param.MyGPU = 2). You can specify GPU number in the 9th string of Batch.m. For instance, if you want to run it on a CPU you should set Param.MyGPU = 0 (you also should select this option for Matlab versions older than 2011a that only have rudimentary GPU support). Be default, if only a single GPU is present, Batch.m will run on GPU 1. And if no CUDA-compatible GPU is present, it will run on the CPU, one core (slow).

3. To see the output run SimpleViewer. If you run it without input argument, it will load the fist LVD file from, the Data\... directory. To see the second file run SimpleViewer(2) and so on. Note that relative paths are used in Batch and SimpleViewer (thus, you should always call them from their expected locations; you can redefine paths editing the beginning of the scripts).

What parameters can be changed in the code?

In **Batch.m** the most interesting parameters are:

```
Param.SongSelection = [1]; %channel number: [1 2 3 4]
      %1 – wall microphone, 2 - backpack microphone,
      %3 - Accelerometer, 4 - full record

Param.scanrateDesired = [19200]; %can be [19200 32000]

Param.SongDetectionThreshold = [2 4/3/2 0.5/4];
      % one threshold for each channel

Param.BuffersAbovePositiveAll = [5 20];
      %Parameter for song detection, in units of 4 ms steps,
      %default values, good for calls

Param.Threshold = 250;
      %Parameter for selection of non-empty frames for
      %synchronization, in points,
      %one pulse - 800uS or 15.36 points, every 4 ms

Param.Pre = 2; %include two seconds before detection

Param.Post = 1; %include one second after detection
```

The first parameter SongSelection determines what channel will be used for vocalization detection. As there was no noise in the example session, we selected 1 (wall microphone). You can select several

channels simultaneously. In this case several subdirectories will be generated in Data directory (one per channel). If "full record" is chosen, no data selection will be done

The second parameter `scanrateDesired` specifies sampling rate of the output file. Only two fixed rates are possible - 19.2 and 32 kHz. The first is the internal sampling rate of the backpack, and the second - the default sampling rate of the computer running the external clock.

The third parameter `SongDetectionThreshold` determines threshold for song detection for wall microphone, head microphone and accelerometer respectively. They depend on background noise and useful signal amplitudes.

The fourth parameter `BuffersAbovePositiveAll` determines how many consecutive 4-ms samples (5) should be above threshold in a reference window of specified width (20) for song detection to take place. These parameters are good for single call detection. If one is interested exclusively in songs, parameters can be larger, for instance [80 150].

The fifth parameter `Threshold` determines the minimal number of non-zero points (each 800-us detection pulse = 15.36 points on average) in the synchronization frame (usually 20 s width) to be used for record alignment. Frames with fewer synchronization pulses are not used.

The last two parameters are periods before the first detection and after the last detection that will be included in the vocalization file.

In SynchronizeNI_BP.m the most interesting parameters are:

```
Param.SynchronizationLag = 10*Param.scanrateMax;

Param.SynchronizationFrame = 20*Param.scanrateMax;

Param.MaxShiftAllowedEnd = round(0.004*Param.scanrateMax);
```

The first parameter is the step size of moving synchronization frames (10 seconds). The second parameter is the width of each frame (20 seconds, i.e., their overlap is 50%). The third parameter sets the search range of time shifts (+/-4 ms). Such a small range is good for testing purposes. For 2.5-h records it is usually +/-3 seconds.

Data format

DAT file (generated by backpack)

Data frames of 6-byte length are stored with frequency 19.2 kHz. Each frame contains data from 4 analog channels (11-bit record each) and one digital channel (1 bit). The structure of the frame is the following:

Byte 1: <Ch#0 Lo 8 bit>
Byte 2: <1> <Ch#0 Hi 3 bit><IR 1bit><Ch#1 Hi 3 bit>

Byte 3: <Ch#1 Lo 8 bit>
Byte 4: <Ch#2 Lo 8 bit>
Byte 5: <0> <Ch#2 Hi 3 bit><0><Ch#3 Hi 3 bit>
Byte 6: <Ch#3 Lo 8 bit>

The format of 11-bit record is unsigned integer, i.e. zero input voltage (relatively to reference potential) has value 1024. Most values lie in the range 512 to 1536 (10-bit range). The value is the difference between the 10-bit (microphone/accelerometer) potential after the amplification cascade and a 10-bit reference potential (1V relative to digital ground). In total, 11-bits are needed to cover this difference. The channel assignments are:

Ch#0 - microphone amplified 101 times (Ku=101),
Ch#1 - microphone Ku=11,
Ch#2 - accelerometer Ku=101,
Ch#3 - accelerometer Ku=11.

LVD files (generated either by the host computer of the wall microphone or by Batch.m)

The LVD file header consists of 4 double floating point values (8 bytes each), big-endian byte ordering. After the header follows the sequence of frames consisting of N channels*int16 data values (sample rate 32 kHz). If the LVD file is generated by Batch.m, the sampling frequency can be either 19.2 or 32 kHz. The structure of the header is the following:

value 1 - Sampling rate = 32000 or 19200 (Hz),
value 2 - Number of channels = 2 (3 with playback) or 7 (8 with playback),
value 3 - Codes date-time of the file start,
value 4 - Input range = 5 (means +/-5 Volts).

The date-time can be decoded to the standard date-time format using the following two Matlab operations:

```
FileStartTimeSt = num2str(par(3),'%017.3f'); %get string from double

FileStartTime = datenum(FileStartTimeSt,'yyyymmddHHMMSS.FFF'); %get usual
      date-time double from string
```

Channel assignments in host-computer generated LVD files:

Ch#0 - wall microphone,
Ch#1 -vocalization events detected in real time by the host computer and used for synchronization.
Additionally, if playback is used: Ch#2 - playback channel.

Channel assignment in Batch.m generated LVD:

Ch#0 - wall microphone,
Ch#1 - backpack microphone,
Ch#2 - accelerometer,

Ch#3 - vocalization detection events stored by the host computer (used for synchronization),
CH#4 – IR events detected by the logger (should be similar to Ch#3),
Ch#5 - detected vocalizations on backpack microphone channel (generated by Batch.m),
Ch#6 – detected vocalizations on accelerometer channel (generated by Batch.m)
Additionally, if sounds were played back during the experiment, they are stored in Ch#1 - playback output. In this case, the backpack microphone and all following channels are shifted by +1.