A Manual of Audapter

Version 2.1.012

Shanqing Cai

scai@bu.edu or shanqing.cai@gmail.com

Speech Laboratory,
Department of Speech, Language and Hearing Sciences,
Sargent College of Health and Rehabilitation Sciences,
Boston University

February 2014

1. Overview: What is Audapter?[1]

Audapter, previously known as TransShiftMex, is a software package for configurable real-time manipulation of acoustic parameters of speech that runs on general-purpose computers. It is designed for research on auditory-motor interactions in speech production, but may also be of use for certain speech signal processing applications. The current version of Audapter supports manipulation (i.e., perturbation) of the following acoustic parameters:

1) Formant frequencies (F1 and F2), in both static and time-varying ways
2) Fundamental frequency (F0, or pitch)
3) Local timing, through time-warping
4) Local intensity
5) Global time delay (delayed auditory feedback)
6) Global intensity

These types of perturbation types can be automatically gated in on specific preselected parts of a given utterance, through a set of heuristic rules for online status tracking (OST, see Sect. 7). Certain combinations of perturbation types can be delivered simultaneously or sequentially, in the same speech utterance.

As a package, Audapter includes both the core algorithms for real-time speech signal processing and MATLAB wrap-arounds supporting psychophysical experiments. The real-time signal processing algorithms are coded in C++ and implemented as a MEX interface for MATLAB. A set of MATLAB scripts and programs are available for calling Audapter and utilizing it in various types of auditory feedback perturbation (AFP) experiments. These include graphical user interfaces for stimulus presentation, experimental workflow control, data preprocessing, as well as basic analyses of the data. Although Audapter has been thoroughly used and tested only on Windows PCs (both 32- and 64-bit), it should be portable to other platforms, including Mac.

Audapter has been developed at the Speech Communication Group, Research Laboratory of Electronics (RLE), Massachusetts Institute of Technology (MIT) as well as the Speech Laboratory of Boston University. Marc Boucek (Boucek 2007) and Satrajit Ghosh originated the MEX C++ project. This code was partly based on algorithms written on DSP platforms by Virgilio Villacorta and Kevin J. Riley in earlier AFP experiments. Since the year 2007, Shanqing Cai, the author of this document, made extensive modifications to Audapter and added many new functions. Cai is currently the primary maintainer of this software package.

This manual serves as a general guide to the usage of Audapter. If you are interested only in using Audapter to perform relatively simple psychophysical or MRI AFP experiments (e.g., simple formant or pitch perturbation), you can proceed directly to Section 9, 10 and 11. However, if you would like to gain a thorough understanding of the

[1] List of abbreviations: AF – Auditory feedback; AFP – Auditory feedback perturbation; ASR – Automatic speech recognizer; DAF – Delayed auditory feedback; F0 – fundamental frequency; F1- 1st formant frequency; F2 – 2nd formant frequency; FF – Forgetting factor; LP – Linear prediction; OST – Online status tracking; PCF – Perturbation configuration; P.I. – Principal investigator; RMS – Root mean square.

MATLAB interface for Audapter and potentially design your own AFP paradigms, the other sections should give you useful guidance.

1.1. How to cite Audapter?

To cite the Audapter as a software package, use the following references:

Cai S, Boucek M, Ghosh SS, Guenther FH, Perkell JS. (2008). A system for online dynamic perturbation of formant frequencies and results from perturbation of the Mandarin triphthong /iau/. In *Proceedings of the 8th Intl. Seminar on Speech Production, Strasbourg, France*, Dec. 8 - 12, 2008. pp. 65-68.

Tourville JA, Cai S, Guenther FH (2013) Exploring auditory-motor interactions in normal and disordered speech. *Proceedings of Meeting on Acoustics*. 9:060180. Presented at the *165th Meeting of the Acoustical Society of America*, Montreal, Quebec, Canada, June 2 – June 7, 2013.

Published experimental studies based on Audapter from the MIT Speech Communication Group and Boston University Speech Lab include:

Cai S, Beal DS, Ghosh SS, Guenther FH, Perkell JS. (In press). Impaired timing adjustments in response to time-varying auditory perturbation during connected speech production in persons who stutter. Brain Lang.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2010). Adaptive auditory feedback control of the production of the formant trajectories in the Mandarin triphthong /iau/ and its patterns of generalization. J. Acoust. Soc. Am. 128(4):2033-2048.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2011). Focal manipulations of formant trajectories reveal a role of auditory feedback in the online control of both within-syllable and between-syllable speech timing. J. Neurosci. 31(45):16483-16490.

Cai S, Beal DS, Ghosh SS, Tiede MK, Guenther FH, Perkell JS. (2012). Weak responses to auditory feedback perturbation during articulation in persons who stutter: Evidence for abnormal auditory-motor transformation. PLoS ONE. 7(7):e41830.

When appropriate, the above references can be cited as well.

2. Getting Started - Running Offline Demos

The Audapter package comes with a set of demo scripts that show you the basic capacity of the software as well as serve as examples for programming your own Audapter applications.

Details on how to obtain the code can be found in Appendix 1. If you opt to build the core MEX program of Audapter from scratch, instructions on how to do this in Microsoft Visual C++ can be found in Appendix 2. Alternatively, you can download the compiled binary from the author's website. In addition to setting up the MATLAB and MEX code of Audapter, you will need to install at least one ASIO sound card driver on your computer, in order to ensure the successful execution of the Audapter MEX program in MATLAB. This is necessary even if you plan to use Audapter only in the offline mode. Installing the ASIO driver does not require that you have the actual sound card hardware. You can download MOTU's universal audio installer for free at: http://www.motu.com/download. Alternatively, you can try ASIO4ALL: http://www.asio4all.com/.

To set up the environment properly, you need to add path to Shanqing Cai's MATLAB toolkit *commonmcode*, by entering in MATLAB a command such as the following:

```
addpath c:/speechres/commonmcode;
```

The drive letter and directory may vary depending on where you've put the code repositories. Then, use the cds script in the *commonmcode* toolkit to set up the paths and environment automatically:

```
cds('audapter_matlab');
```

You can check that the path to the Audapter MEX program has been set up correctly by entering command:

```
which Audapter;
```

If the output says "Audapter not found", you may need to set the path to the Audapter MEX file manually, e.g., by using the MATLAB command addpath.

2.1. Offline Demos

These offline demos can be run in MATLAB without an ASIO-compatible sound card attached, because these demo scripts utilize the offline processing option of Audapter, i.e., the "*runFrame*" option (see Sect. 3).

2.1.1. Offline Demo 1: Formant perturbation

The command for bringing up this demo is:

```
test_audapter('formant', '--play');
```

This command brings up two windows in MATLAB, each showing a spectrogram. The first window shows the spectrogram of the input signal, which is the English phrase "test a pepper" uttered by an adult male speaker. Overlaid in the

spectrogram are the F1 and F2 tracks calculated by Audapter during the supra-threshold intervals, as well as a black curve showing the OST status (multiplied by 500 for visualization purpose). The second figure shows the spectrogram of the output, i.e., perturbed, speech signal. The F1 and F2 during the word "a" and the first syllable of the word "pepper" are altered. The new formant values are shown by the green curves in this figure. This demo program also plays the input and output signals, due to the inclusion of --play in the input argument. The consequence of this joint downward F1 and upward F2 is that the word "pepper" sounds more similar to the word "paper".

This simple demo demonstrates three aspects of Audapter's capacity: 1) formant tracking, 2) formant AFP and 3) OST for tracking the progress of the sentence and delivering the perturbation at specific part of a multisyllabic utterance.

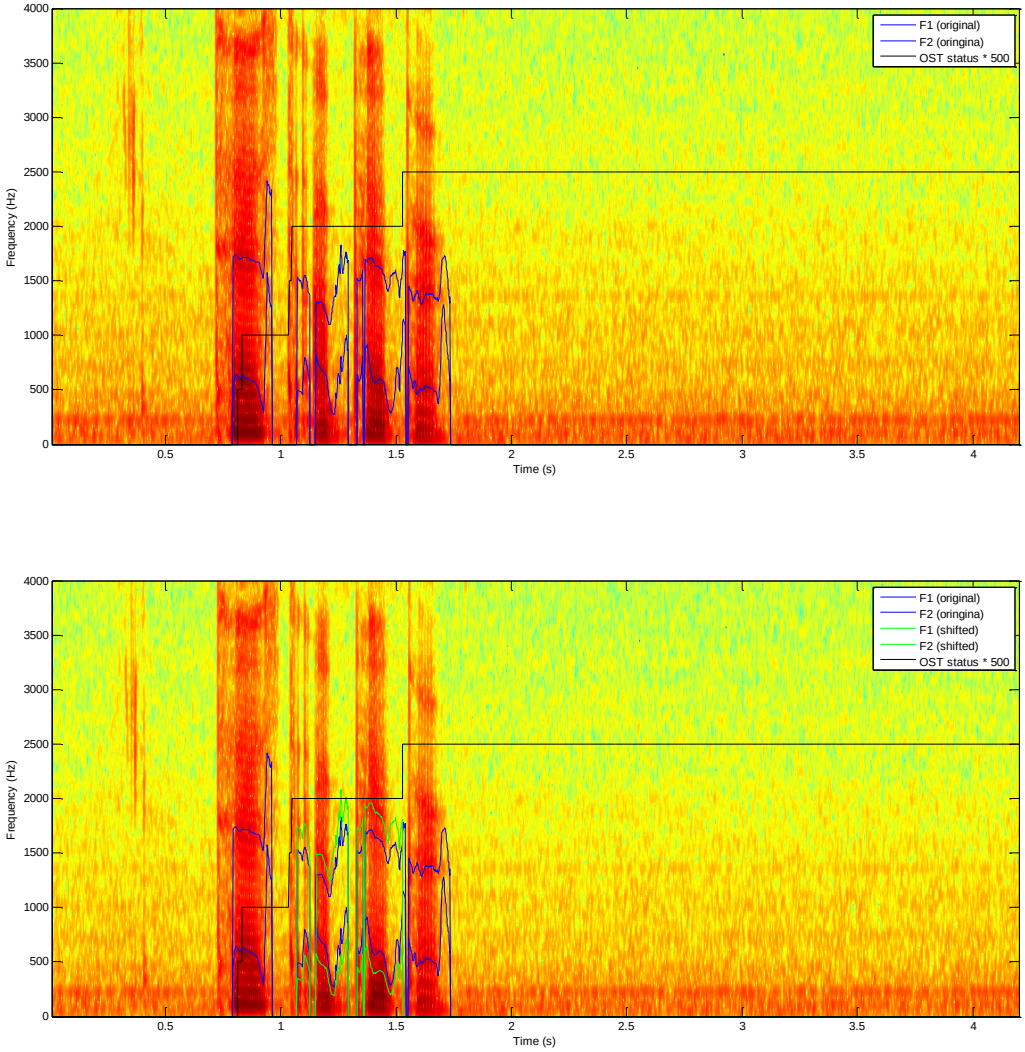

Figure 1. Graphical output of the demo command: test_audapter('formant', '--play');

2.1.2. Offline Demo 2: F0 perturbation

To see the demo of Audapter's F0 (pitch) perturbation capacity, use the following command:

```
test_audapter('pitch', '--play');
```

The graphical output of this command is similar in format to the formant-perturbation demo. This example is based on the same recording as in Demo 1. However, unlike the formant perturbation example, the fundamental frequency (F0) is shifted up during the word "a" and the first syllable of the word "pepper".

2.1.3. Offline Demo 3: Time warping

The following command brings up an example of time-warping perturbation:

```
test_audapter('timeWarp', '--play');
```

Comparing the two spectrograms, you can see change in the timing of various parts of the utterance. Specifically, two time-warping events were included in this example. The first event lengthens the duration of the [s] sound in the word "test" and delays the onset of the final [t] sound. It also delays the onset of the word "a". This warping event ends at approximately the beginning of the first syllable in "pepper". The second warping event starts during the silent interval before the onset of the second [p] sound in "pepper". It lengthens this silent interval and thereby delays the onset of the noise release in the following [p] sound. As can be seen in this example, multiple time-warping events can be included in the same utterance.

2.1.4. Offline Demo 4: Dynamic perturbation of a Standard Chinese triphthong

In this demo, we show how Audapter can impose a time-varying F1 perturbation during a triphthong [iau] in Standard Chinese (Mandarin). To run this demo, enter command:

```
audapterDemo_triphthong('--play');
```

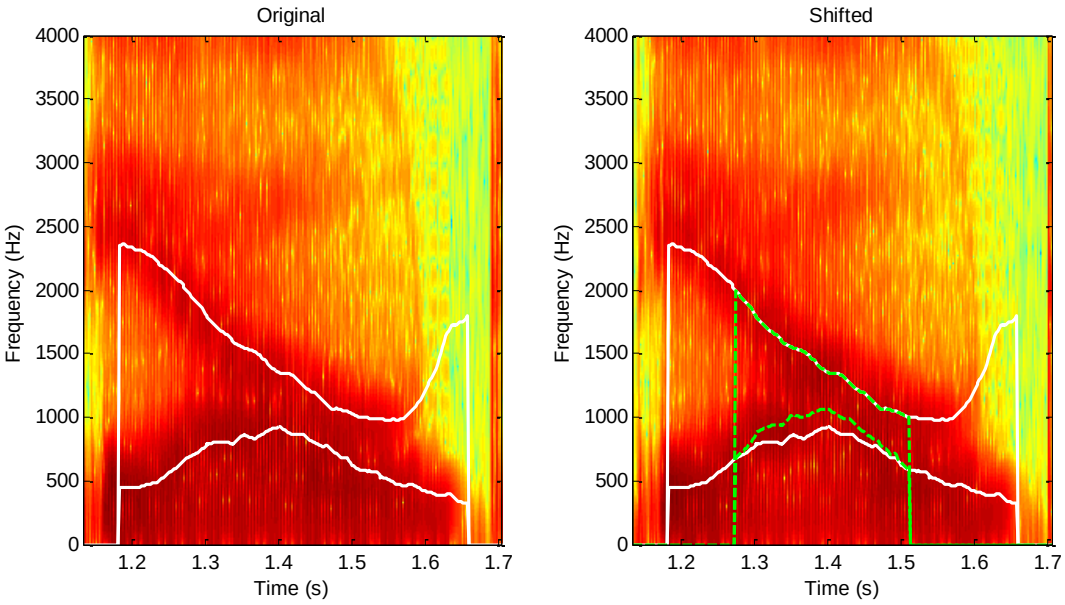

Figure 2. Graphical output of the demo command: audapterDemo_triphthong('--play'). It shows the time-varying F1 perturbation during the Standard Chinese triphthong [iau].

The "--play" option leads to playback of the input and output waveforms. The script also shows a figure with two subplots, the left of which shows the input (unperturbed) spectrogram and the right of which illustrates the perturbed output. The white curves overlaid on these spectrograms are the original F1 and F2 values in the female speaker's production, while the dashed green curves are the perturbed F1 and F2 values. The green F2 curve overlaps with the solid white F2 curve because the perturbation was limited to F1 in this example.

This type of time-varying perturbation is what is used in Cai et al. (2010). It is achieved by supplying proper values to these following Audapter parameters that support this type of OST-independent, time-varying formant manipulation: *f2Min*, *f2Max*, *f1Min*, *f1Max*, *LBk*, *LBb*, *pertF2*, *pertAmp* and *pertPhi*. Together, these nine parameters forms a structure called the *perturbation field* in the plane of F1 and F2. In this perturbation field, the amount of perturbations to F1 and F2 can vary as a function of the unperturbed value of F2. This also forms the basis for the "spatial" perturbation type used in Cai et al. (2011). Section 5 contains further details about the design and usage of the perturbation field. Detailed descriptions of the nine Audapter parameters can also be found in Section 4.

2.2. Online demos

The following demos show the real-time speech perturbation capacity of Audapter. They use the real-time processing options ("start" and "stop"; see Section 3). In order to run them successfully, you need to have an ASIO-compatible audio interface attached to the computer. A type of audio interface that has been used and tested

extensively by the author is the MOTU MicroBook. The settings in the demo scripts assume that this is the attached audio interface. If other types (e.g., MOTU UltraLite) are used, the settings will need to be adjusted accordingly, mostly in the MATLAB script *getAudapterDefaultParams.m*. These demo scripts assume that the audio interface has the following settings: sampling rate = 48000 Hz; buffer length (frame size) = 96, i.e., settings that work for MOTU MicroBook.

2.2.1. Online Demo 1. Persistent formant shift

To run this demo, enter the following command in MATLAB:

```
audapterDemo_online('persistentFormantShift', gender);
```

wherein gender is the gender of the user ('male', or 'female'). The formant perturbation in this demo consisted downward F1 shifts and upward F2 shifts, which make the vowels sound "fronter" and higher in the AF.

In this demo you can use the command line option 'fb' to select different feedback modes. Detailed description of the parameter fb is in Table 1. The default feedback mode is fb=1, i.e., speech-only feedback mode. It can also be activated with the command line:

```
audapterDemo_online('persistentFormantShift', gender, 'fb',
                                1);
```

If you select the feedback mode 0 with:

```
audapterDemo_online('persistentFormantShift', gender, 'fb',
                                0);
```

the auditory feedback (AF) is muted, however, Audapter still processes the formant tracking and shifting and store them in the output data.

To show the noise-masking feedback mode, use fb value 2.

```
audapterDemo_online('persistentFormantShift', gender, 'fb',
                                2);
```

To see the mixed speech and noise feedback mode, use fb value 3:

```
audapterDemo_online('persistentFormantShift', gender, 'fb',
                                3);
```

fb value 4 activates the so-called speech-modulated noise feedback mode:

```
audapterDemo_online('persistentFormantShift', gender, 'fb',
                                4);
```

In this mode, the auditory feedback is the noise waveform, modulated by the short-time intensity envelope of the speech waveform. Given that the level of the noise is carefully

chosen, this mode is similar to the noise-only feedback mode in that it blocks the auditory feedback, however, it can also have the advantage of eliciting less Lombard effects, which may be desirable for certain purposes such as studying the control of speech movements and speech motor learning in the absence of AF.

In the demo script, you can see that under fb values 2 to 4, where a noise waveform is required, Audapter loads the waveform from a .wav file "mtbabble48k".

2.2.2. Online Demo 2. Persistent pitch shift

This demo shows the basic pitch shifting capacity of Audapter. It shifts the pitch up by two semitones throughout the trial, without any OST tracking. To run this demo, do:

```
audapterDemo_online('persistentPitchShift');
```

2.2.3. Online Demo 3. Two fixed-delay, fixed-duration short pitch shifts in one utterance

Demo command line:

```
audapterDemo_online('twoShortPitchShifts');
```

This demo includes two short pitch shifts following the automatically detected voicing onset, each of which lasts 200 ms. The second shift begins 300 ms after the end of the first shift. This type of perturbation is similar to the pitch-shift stimuli that have been used extensively at Chuck Larson's lab at Northwestern University (e.g., Larson et al. 2008). This timing control is achieved through the OST file in ../example_data/two_blips.ost. Studying this OST file and the associated PCF file (../example_data/two_pitch_shifts.pcf) can give you an idea of how to use the OST and PCF capacities to perform this type of fixed-duration, fixed-delay perturbations.

2.2.4. Online Demo 4. Focal formant shift

To run this demo, enter command:

```
audapterDemo_online('focalFormantShift', gender);
```

In this demo, the user is expected to utter the nonsensical utterance "I said pap again", in order to see the effect. Like the example in Sect. 2.2.1, this demo involves perturbations to formant frequencies, F1 and f2. However, unlike in the previous example, Audapter tracks the progress of the sentence and triggers the perturbation during a specific syllable of it. The word "pap" undergoes a upward F1 shift and downward F2 shift, which renders the word similar-sounding to the word "pop" (in American English, see Fig. 3 below).

This online tracking and focal perturbation is achieved through the OST and PCF files "../example_data/focal_fmt_pert.ost" and "../example_data/focal_fmt_pert.pcf". The OST file is a good example for those users who want to learn how to use Audapter to track a complex sentence consisting of multiple words and various types of consonants.

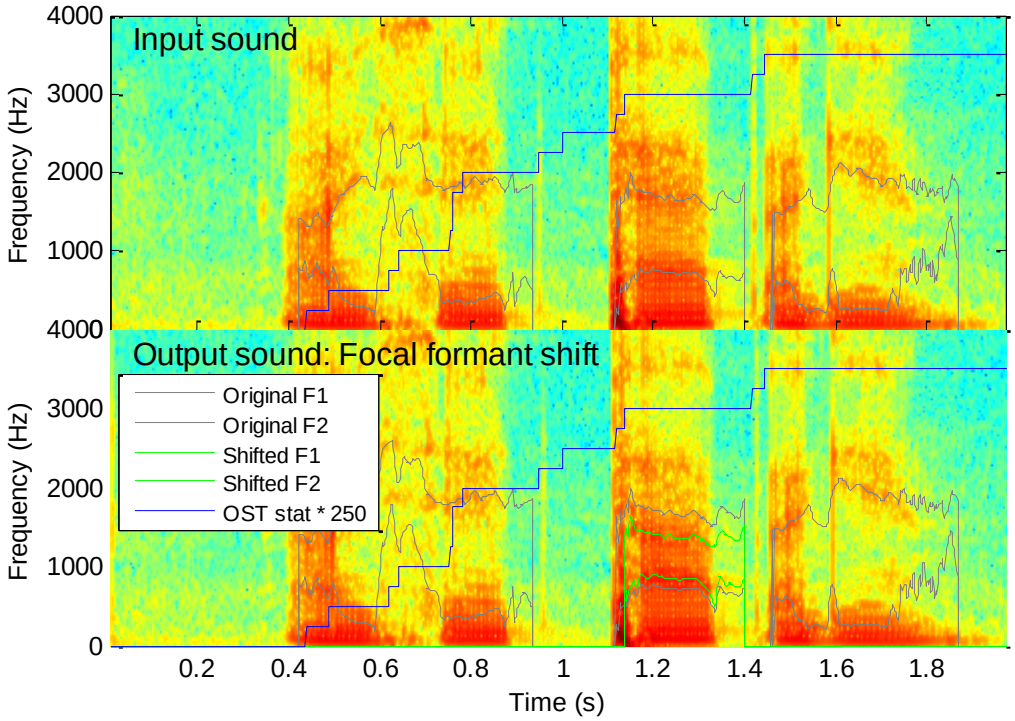

Figure 3. Example output of the demo on focal formant shift, tested on an adult male speaker, with the command line: audapterDemo_online('focalFormantShift', 'male'). The user uttered "I said pap again". The Audapter tracks the progress of the words in this utterance in real time (see the blue curve showing the OST stat numbers). Notice the focused F1 and F2 perturbation during the word "pap".

This is an appropriate place to point out that the parameters in the OST file is largely *ad hoc*. The set of parameters are specific to the microphone gain and the speaker idiosyncrasy. There is no guarantee that those OST parameters will work out of the box for all speakers. If you want to run focal perturbation experiments on multiple subjects, you need to write programs for determining the optimal OST parameter values for each subject, possibly with the help of force alignment results from an automatic speech recognizer (ASR). Here, the optimality criterion is, of course, minimal error rate in the online detection of the focus word (i.e., the word to receive AFP). You can study the code in the author's "rhythm-fmri" repository as an example of how to do this. This "rhythm-fmri" project relied on the Julius ASR engine. The code can be found at GitHub location: https://github.com/shanqing-cai/rhythm-fmri. The key script for calculating subject-

specific OST parameters is at: https://github.com/shanqing-cai/rhythm-fmri/blob/master/mcode/get_ost_pcf_params_rhy.m.

2.2.5. Online Demo 5. Time warping

Command line:

```
audapterDemo_online('timeWarp');
```

In this demo, Audapter waits for the onset of voicing, as detected by the INTENSITY_RISE_HOLD mode of OST and a hard-coded intensity threshold of 0.02. Then it waits for another 100 ms before initiating a time warping event. This time warping event is specified in the PCF file ../example_data/time_warp_demo.pcf. The effect of this demo time warping will be the most salient if you can utter fast-changing sounds with abrupt onsets, such as "puh puh puh…"

2.2.6. Online Demo 6. Globally delayed auditory feedback, pitch and gain perturbation and multi-voice feedback

To bring up the global delayed, pitch- and gain-shifted multi-voice feedback demo, use the following command:

```
audapterDemo_online('globalDAF_multiVoice');
```

This demo activates a feedback that consists of two voices and lasts for 3 seconds. One of the feedback voices has a global delay of 100 ms and a gain factor of 1.0; the other one has a longer delay (200 ms) and lower gain (0.33). In addition, the first voice contains a two-semitone downward pitch shift and the second one contains a two-semitone upward pitch shift. This example shows how Audapter can be used to simulate choral reading effects for applications such as fluency enhancement for persons who stutter (Kalinowski et al. 1993).

In the demo script, you can see lines that nullify the OST and PCF configurations in Audapter before starting the multi-voice feedback:

```
Audapter('ost', '', 0);

Audapter('pcf', '', 0);
```

This is necessary here, as existing OST and PCF configurations will set parameter nFB automatically to zero and thereby disable the multi-voice feedback mode automatically.

2.2.7. Online Demo 7. Continuous sine wave generation

To run this demo, enter command:

```
audapterDemo_online('playTone');
```

You will hear four notes (A, B, C#, A) played in a sequence. Even though this function may seem very similar to the tone sequence generation function (Sect. 2.2.9), there is an important difference. Notice that the last tone will play continuously and keep going, until the user hits Enter to trigger the Audapter('stop') command. The tone sequence generator is not capable of producing continuous tones. Another difference is in the initial phases of the individual tones. You may be able to hear the discontinuities ("clicks") in the sound produced by this example. This is because the tones produced under the "playTone" mode of Audapter do not have on/off ramps. The tone sequence generator, described in Section 2.2.9, however, is capable of imposing ramps on the tones.

2.2.8. Online Demo 8. Waveform playback

This demo can be brought up by the command:

```
audapterDemo_online('playWave');
```

You will hear an utterance being played from the output channel of the audio interface. This option is based on the "playWave" mode of Audapter. The waveform for playback is supplied to Audapter with the following syntax (see the script):

```
Audapter('setParam', 'datapb', sigInRS);
```

sigInRS must have the same sampling frequency as the audio interface's hardware sampling frequency before downsampling. In addition, its length must not exceed the maximum playback sample count maxPBSize, which can be obtained through the command:

```
maxPBSize = Audapter('getMaxPBLen');
```

2.2.9. Online Demo 9. Tone sequence generation

Use the following command to open this demo:

```
audapterDemo_online('playWave')
```

In this demo, you can hear four tones with varying frequencies, durations and amplitudes played from the output channels of the audio interface. Unlike he tones played through the "playWave" option of Audapter, these tones have smooth onset and offset ramps, so

that the entire sound does not contain audible discontinuities. This example demonstrates how to use the parameters *tsgNTones*, *tsgToneFreq*, *tsgToneAmp*, *tsgToneRamp* and *tsgInt* to set up a tone sequence.

3. Basic Command Line Usage of Audapter

Audapter is a C++ MEX program that can be called from MATLAB. If you enter the command:

```
Audapter;
```

that is, without any input arguments in MATLAB, you will see the help message in the MATLAB console. The help message lists the types of commands that you can send to Audapter. Each command can be called in two different ways, either as a command number (e.g.,1, 2) or as the corresponding character-based command name (e.g., start, stop).

To list the currently attached ASIO audio interfaces (i.e., sound cards), use:

```
Audapter('info');
```

To start a real-time audio processing trial, do:

```
Audapter('start');
```

To stop a trial, do:

```
Audapter('stop');
```

Note: to let Audapter work properly during the real-time processing trial, you need to have its parameters set properly, and optionally, have the online status tracking (OST) and perturbation configuration (PCF) files loaded properly. See the following sections (Sections 4, 7 and 8) on details of these settings. Among the parameters of Audapter, the most basic ones for ensuring the crash-free functioning include

1) downsampling factor (parameter *downFact*)
2) sampling rate (parameter *srate*),
3) frame length (parameter *frameLen*)

It is important to note that *srate* and *frameLen* should be the actual hardware sampling rate and buffer length **divided by *downFact***. The downsampling factor, typically set to 3 or 4, is for reducing the computational load on CPUs for ensuring real-time processing. For example, if your audio interface has a sampling rate of 48000 Hz and a buffer length of 96, given that you've specified a downFact of 3, the values of *srate* and *frameLen* you should use in Audapter are 16000 and 32, respectively.

To set a parameter of Audapter, use the setParam option:

```
Audapter('setParam', paramName, paramVal, bVerb);
```

In this command syntax, the second and third input arguments are the name of the parameter and the value you with set it to, respectively. The optional, fourth argument is a Boolean (0/1) variable that indicates whether the verbose mode is selected. If you do not include this argument, the verbose mode is set by default. For example, the following command

```
Audapter('setParam', 'bCepsLift', 0, 0);
```

sets the parameter *bCepsLift* (see Sect. 4) to 0 (i.e., false) under the silent (i.e., non-verbose) mode.

Oftentimes, for debugging and data analysis, you may want to run Audapter on pre-recorded speech sounds, in an offline fashion. The option "runFrame" allows you to do that. In fact, the offline demos you have seen rely on this option. In this offline mode, you supply Audapter with signal frames (i.e., buffers) of speech sound at a time, with the following syntax:

```
Audapter('runFrame', signalFrame);
```

wherein signalFrame is a speech signal vector whose length matches *frameLen* * *downFact*. In other words, signalFrame is a frame of audio **before** downsampling. You can call Audapter in this way repeatedly, but with different signalFrames, to simulate the consecutive buffers that come in during an online, real-time trial. The test_audapter script for the offline demos contains examples of how this is achieved.

The option "getData" of Audapter allows the user to extract audio and associated data from the last trial. This applies to either real-time trials triggered by options "start" and "stop" and offline trials triggered by option "runFrame":

```
[sig, dat] = Audapter('getData');
```

In the output, sig is a N×2 matrix, in which N is the number of samples in the last online or offline trial after the downsampling. The first column is the input signal; the second one is the output (potentially perturbed) signal. The second output "dat" is a N×M matrix containing various data derived from the audio input, such as calculated formant frequencies, linear prediction (LP) coefficients, short-time RMS intensity values, OST status numbers, etc. Each of the M columns is a different type of derived data. The matrix is not annotated and is not meant to be used directly by the user. Instead, there is a MATLAB script that wraps around the "getData" option of Audapter and generates much more readable data. It can be called in the following way (see the demo script: test_audapter.m, for an example):

```
data = AudapterIO('getData');
```

The output data includes both the input / output signals and the derived data. Section 6 contains a detailed description of all the fields of the output "data".

The "reset" option in Audapter allows the user to reset the status of the temporary data fields in Audapter, so as to prepare for the next incoming trial. It can be called as:

```
Audapter('reset');
```

which is equivalent to the calling the "reset" option in the AudapterIO wrap-around:

```
AudapterIO('reset');
```

This resetting does not alter the parameter values. Instead, it sets memory fields that hold past audio signals, past formant values, etc., as well as the status of the OST tracker, to zero or other proper initial values, so that a new trial can start without any lingering influence from the previous trials. This resetting action should be performed prior to the onset of any new utterance in online and offline processing. The code in test_audapter.m demo script shows that.

The "ost" and "pcf" options allows the loading of OST and PCF into Audapter, respectively, for specifying the details of online word tracking rules and perturbation to be delivered during the utterance. Details on how to use these options can be found in Sections 7 and 8.

The options of Audapter listed above are for speech signal processing. There are a number of other options in Audapter that support signal generation and playback functions that might be useful during psychophysical experiments, as listed below.

The "playTone" option lets Audapter generate a continuous sine wave, of which the frequency, amplitude and initial phase angle can be specified in parameters *wgFreq*, *wgAmp* and *wgTime*, respectively. See the demo in Sect. 2.2.7 for an example of how to use this option.

Apart from generating a continuous sine wave, the user can also load an existing waveform of which the sampling rate equals srate*downFact and it back by using the "playWave" option. See the demo in Sect. 2.2.8 for an example of how to use the playWave option.

In addition, Audapter can also generate a sequence of short tone blips of adjustable durations, frequencies, amplitudes, onset/offset ramps and inter-tone intervals, through the "playToneSeq" option. Audapter can also write the waveform of the generated tone sequence to a .wav file through the "writeToneSeq" option. See the demo in Sect. 2.2.9 for further details and an example of using these options.

The last, but the not least important, command-line option of Audapter covered is the "deviceName" option. It is used to select an audio interface to use. It should be especially useful when you have multiple ASIO-compatible sound cards attached to your computer. When Audapter starts a real-time operation, such as "start", "playTone" or "playWave", it searches for the sound card with name matching the value of the pre-set deviceName. If it fails to find such a device, it will report error and stop. This option can be called with the following syntax example.

```
Audapter('deviceName', 'MOTU MicroBook');
```

"MOTU MicroBook" is the default value of deviceName. If you use a different sound card, you'll have to set it properly yourself.

4. Adjustable Parameters of Audapter

Table 1 provides a list of all configurable parameters of Audapter. The values of these parameters can be set with the "setParam" option of Audapter. (see Sect. 3).

Table 1. Adjustable parameters of Audapter

Parameter Name	Parameter Type	Description	Default value[2]
Part 1. Basic audio interface settings			
downFact	int	Downsampling factor. The downsampling is for reducing the computational load for meeting the real-time constraint.	3
srate	int	Sampling rate (Hz), after downsampling	16000[3]
frameLen	int	Frame length in number of samples, after downsampling. This value should be an integer power of two.	32
nDelay	int	Processing delay in number of frames. The delay is due to the way in which Audapter forms an internal processing window: an internal window consists of (2 * nDelay - 1) input frames. During formant perturbation, the value of *nDelay* determines the feedback latency of the formant shifter. Note that if other types of perturbation, such as pitch shifting and time warping, is involved, the feedback latency may depend on other phase-vocoder-related settings.	7
nWin	int	Number of windows per frame. Each incoming frame is divided into *nWin* windows.	1
fb	int	Feedback mode. 0: mute (play no sound) 1: normal (speech only) 2: noise only 3: speech + noise. The level of the noise is controlled by parameter *fb3Gain*. 4: speech-modulated noise. The level of the modulated noise is controlled by parameter *fb4Gain*; the smoothness of the intensity	1

[2] These default values of downFact, srate and frameLen are the parameter values that Audapter automatically take after construction. It can be found in the constructor of Audapter (Audapter::Audapter) in the C++ source code.
[3] 48000 Hz downsampled by a factor of 3. Note that this value is set for MOTU MicroBook. For other audio interfaces, other values may need to be used.

		envelope is controlled by parameter *rmsFF_fb*. Note: these options work only under real-time processing mode of Audapter, invoked through Audapter(1) or Audapter('start'). See Sect. 2.2.1 for demo of these modes of feedback.	
fb3Gain	double	Gain applied to the noise waveform for fb mode 3 (speech + noise)	0.0
fb4Gain	double	Gain applied to the modulated noise waveform for fb mode 4.	1.0
rmsFF_fb	double array (1×4)	First element: the forgetting factor at the onset of voicing. Second element: the forgetting factor when the voicing has stabilized. Third and fourth elements: reserved. Should be set to zero. Both the first and second values should be between 0.0 and 1.0. Greater values correspond to greater smoothing. Audapter will gradually shift the value of the forgetting factor from the first to the second element during the onset of the voicing and shift the value of the forgetting factor from the second to the first element during the offset of the voicing.	[0.85, 0.85, 0.0, 0.0]
stereoMode	Int	Two-channel audio output mode. This applies only to Audapter's real-time processing mode. 0: Audio signal in left channel only; right channel muted 1: Identical audio signals in left and right channels 2: Audio signal in left channel; simulated TTL pulses for indicating pitch perturbation intervals in right channel.	1
scale	double	Output scaling factor. This can be used as a global (i.e., time-invariant) gain control.	1.0
Part 2. Basic signal processing and intensity calculations			
preemp	double	Pre-emphasis factor	0.98
rmsThr	double	Short-time RMS threshold. This threshold is used to determine during which input frames the formants are tracked and shifted. See rows "data.rms" and "data.fmts" in Table 3 for further details.	0.02. Note that this default value is by no means generalizable. It is selected more or less arbitrarily. The proper value of rmsThr depends on many factors such as microphone gain, speaker volume, identity of the vowel, etc.
rmsRatio	double	Threshold for short-time ratio between the smoothed unfiltered intensity value and the smoothed high-passed intensity value. Together with rmsThr, this parameter is involved in	1.3

		vowel detection for determining when formants are tracked and shifted. See rows "data.rms" and "data.fmts" in Table 3 for further details.	
rmsFF	double	Forgetting factor (FF) for smoothing of short-time RMS intensity (rms_o) to obtain the smoothed intensity (rms_s).	0.95
trialLen	double	Length of the trial (unit: s). If this parameter is set to a positive value, the feedback of Audapter will continue for that amount of time and goes mute. But even when the AF goes mute, the signal and data recorders will continue functioning until *Audapter stop* is called.	0.0
rampLen	double	Length of the onset / offset ramps imposed on the auditory feedback (unit: s). These are linear, multiplicative ramps that can prevent the unpleasant sound discontinuities ("clicks") at the beginning and end of a trial. If the value is 0.0 (by default), then ramp is imposed.	0.0
Part 3. Formant tracking and shifting settings			
nLPC	int	Order of linear prediction (LPC). The number of LP coefficients will be nLPC + 1.	15. For LP formant tracking to work properly, this value needs to be adjusted according to the sampling rate and the vocal-tract length of the speaker. Under 16000 Hz sampling rate (following downsampling), values 15 and 17 are recommended for female and male adult speakers, respectively. See mcode/ getAudapterDefaultParams.
nFmts	int	Number of formants to be shifted.	2
nTracks	int	Number of tracked formants. The 1st to the nTracks-th formants will be tracked.	4
avgLen	int	Length of the formant-frequency smoothing window (in number of frames). To disable smoothing of formant frequencies, use avgLen = 1.	10. Ideally, the smoothing window width should be approximately equal to the pitch cycle.
cepsWinWidth	int	Low-pass cepstral liftering window size	Depends on the F0 of the speaker.
aFact	double	α factor of the penalty function used in formant tracking. It is the weight on the bandwidth criterion. The formant tracking algorithm is based on Xia and Espy-Wilson (2000).	1
bFact	double	β factor of the penalty function used in formant tracking. It is the weight on the a priori knowledge of the formant frequencies.	0.8
gFact	double	γ factor of the penalty function used in formant tracking. It is the weight on the temporal smoothness criterion.	1

fn1	double	Prior value of F1 (Hz), used by the formant tracking algorithm.	633
fn2	double	Prior value of F2 (Hz), used by the formant tracking algorithm.	1333
bGainAdapt	Boolean	A flag indicating whether gain adaptation is to be used (see Boucek 2007)	1
bTrack	Boolean	A flag indicating whether the formant frequencies are tracked. Normally, it should be set to 1.	1
bDetect	Boolean	A flag indicating whether Audapter is to detect the time interval of a vowel. It should be set to 1 whenever *bShift* is set to 1.	0
bWeight	Boolean	A flag indicating whether Audapter will smooth the formant frequencies with an RMS-based weighted averaging.	1
bCepsLift	Boolean	A flag indicating whether Audapter will do the low-pass cepstral liftering. Note: *cepsWinWidth* needs to be set properly in order for the cepstral liftering to work.	1
bShift	Boolean	Activation for formant frequency shifting	1
bRatioShift	Boolean	A flag indicating whether the data in *pertAmp* are absolute (0) or relative (1) amount of formant shifting.	0
bMelShift	Boolean	A flag indicating whether the perturbation field is defined in Hz (0) or in Mel (1).	1
minVowelLen	int	Minimum allowed vowel duration (in number of frames). This is a somewhat obsolete parameter. It was used during prior single CVC syllable vowel formant perturbation experiments for preventing premature termination of perturbations. This capacity should have largely been superseded by OST (Section 7).	60
f2Min	double	Lower boundary of the perturbation field (Section 5)	0.0
f2Max	double	Upper boundary of the perturbation field	0.0
f1Min	double	Left boundary of the perturbation field	0.0
f1Max	double	Right boundary of the perturbation field	0.0
LBk	double	Slope of the tilted left boundary of the perturbation field	0.0
LBb	double	Intercept of the tilted right boundary of the perturbation field	0.0
pertF2	1×257 double array	The independent variable of the perturbation vector field (unit: mel or Hz, dependent on *bmelshift*). See Section 5.	All 0.0
pertAmp	1×257 double array	The 1st dependent variable of the perturbation field: amplitude of the vectors. When *bratioshift* = 0, *pertAmp* specifies the absolute amout of formant shifting (in either Hz or mel, depending on *bmelshift*). When *bratioshift* = 1, *pertAmp* specifies the relative amount of formant shifting. See Section	All 0.0

		1.2 (Section 5).	
pertPhi	1×257 double array	The 2nd dependent variable of the perturbation field: orientation angle of the vectors (radians).	All 0.0
Part 4. Sine wave (pure tone) generation and waveform playback			
wgFreq	double	Sine-wave generator frequency in Hz	1000
wgAmp	double	Sine-wave generator frequency (wav amp)	0.1
wgTime	double	Sine-wave generator initial time, used to set the initial phase.	0
dataPB	double, 1×L array	Arbitrary sound waveform for playback. The sampling rate of the playback is srate * downFact. Under the default setting, this sampling rate is 16000 × 3 = 48000 Hz. Therefore Audapter can playback 4.8 seconds of sound. L must less than or equal to the maximum allowable playback length in # of samples, which can be obtained from Audapter by using syntax: `Audapter('getMaxPBLen')`	zeros(1, Audapter('getMaxPBLen'))
trialLen	double	Length of the trial in sec. "triallen" seconds past the onset of the trial, the playback gain is set to zero.	2.5
rampLen	double	Length of the onset and offset linear ramps in sec.	0.05
Part 5. Tone sequence generator			
tsgNTones	Int	The total number of tones in the sequence. The upper limit is 64. See Sect. 2.2.9 for example code of using the tone sequence generator.	0
tsgToneFreq	Double array (1×tsgNTones)	The frequencies of all tones in the sequence (Hz)	[]
tsgToneDur	Double array (1×tsgNTones)	The durations of all tones in the sequence (s)	[]
tsgToneAmp	Double array (1×tsgNTones)	The peak amplitude of the tones in the sequence	[]
tsgToneRamp	Double array (1×tsgNTones)	The length of the onset / offset ramps of the tones in the sequence (s)	[]
tsgInt	Double array (1×tsgNTones)	The inter-onset intervals between the tones in the sequence.	[]
Part 6. Global delayed auditory feedback and multi-voice feedback			
nFB	Int	Number of feedback voice (≤4). Note: multi-voice feedback mode (nFB > 1) is overridden by PCF files. Therefore, in order to use the multi-voice feedback, you need to nullify the OST and PCF files first. See Sect. 2.2.6 for an example of how to configure the multi-voice feedback mode under nFB > 1.	1
delayFrames	Int array (1×nFB)	Amount of delay, in number of input frames. The duration of a frame can be calculated as frameLen / srate.	[0]
gain	Int array	Intensity gain factors in individual feedback voices. Value 1.0 corresponds to no intensity	[1.0]

	(1×nFB)	shift. Note that these gains are applied to the individual feedback voices before the final summed feedback is scaled by the parameter *scale*.	
pitchShiftRatio	Double array (1×nFB)	Pitch shifting ratios in individual feedback voices. Value 1.0 corresponds to no pitch shift. Values > 1.0 correspond to upward pitch shift.	[1.0]
mute	Boolean array (1×nFB)	Mute flags for the individual feedback voices.	[false]
bPitchShift	Boolean	An activation flag for the phase vocoder. It should be set to 1 (true) whenever pitch shifting and/or time shifting is involved. The name of this parameter is admittedly confusing, which is a legacy problem.	false
Part 7. Miscellaneous			
bBypassFmt	Boolean	A flag indicating whether Audapter will skip the formant tracking and shifting algorithms. It can be set to 1 (true) in applications that do not require formant tracking and require only pitch shifting, time warping and/or global delay and intensity manipulations to reduce latency and computational load.	false

5. The Formant Perturbation Field

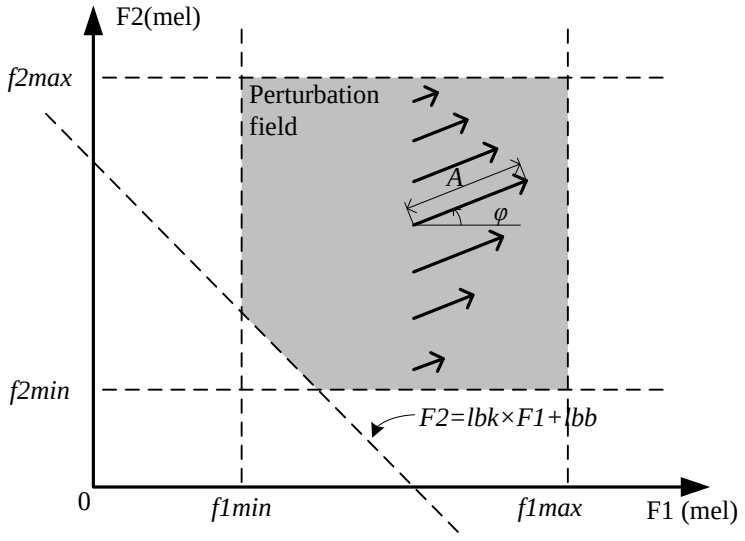

Figure 4. A schematic drawing of the perturbation field. The dashed lines show the boundaries of the perturbation field. The arrows show the perturbation vectors. The shaded region is the perturbation field. A and θ are the magnitude and angle of the vector, which are both functions of F2.

The perturbation field is a region in the F1-F2 plane wherein F1 and F2 can be perturbed in a F2-dependent way. This mode of formant perturbation is currently incompatible with the OST and PCF files, described in Sections 7 and 8. In particular, OST and PCF take precedence over the perturbation field. If you have OST and PCF files loaded into Audapter, the software will use the information in those configuration files for determine the amount of F1 and F2 perturbation. Therefore, to prevent OST and PCF configurations from overriding the perturbation field, OST and PCF settings need to be nullified prior to the onset of a trial that utilizes the perturbation field. This can be achieved through the commands:

```
Audapter('ost', '', 0);

Audapter('pcf', '', 0);
```

As shown schematically shown in Fig. 4, the location of the field is defined by five boundaries specified by six of Audapter's adjustable parameters (Table 1):

$$F1 \geq f1Min; \quad (1)$$

$$F1 \leq f1Max; \quad (2)$$

$$F2 \geq f2min; \quad (3)$$

$$F2 \leq f2max; \quad (4)$$

$$F2 \geq LBk \times F1 + LBb, \text{ if } LBk \leq 0; \text{ or } F2 \leq LBk \times F1 + LBb, \text{ if } LBk > 0; \quad (5)$$

The units of *f1Min*, *f1Max*, *f2Min*, *f2Max*, *LBb* and *LBk* are either Hz or mel depends on another parameter, *bMelShift*. If *bMelShift* = 1, their units are mel; when *bmelshift* = 0; their units are Hz. In addition, the short-time intensity and spectrum need to satisfy certain conditions for the formant perturbation to happen. The "data.fmts" row in Table 1 contains the definition of these conditions.

Detection of a vowel and shifting its formant frequencies is contingent on simultaneous satisfaction of the intensity/spectrum condition and Equations (1) – (5) and. The boundary defined by Equation (5) is in general a tilted line (see Fig. 4), and may seem a little bit peculiar. It was added because it was found to improve triphthong detection reliability in the Standard Chinese triphthong perturbation study. This boundary can be disabled by setting both *LBb* and *LBk* to zero. Similarly, if your project is concerned with only a fixed amount perturbation to a steady-state vowel in an isolated CVC syllable, you may wish not to use the boundaries *f1Min*, *f1Max*, *f2Min*, and *f2Max*, and rely only on the RMS intensity criteria in Table 3. This can be achieved by simply setting *f1Min* and *f2Min* to 0 and *f1Max* and *f2Max* to sufficiently large values (e.g., 5000).

The perturbation field is a vector field (arrows in Fig. 4). Each vector specifies how much F1 and F2 will be perturbed, respectively. Each vector is defined by a magnitude A and an angle φ, which corresponds to *pertAmp* and *pertPhi* in the list of adjustable parameters (Table 1). Both A and φ are functions of F2. *pertAmp* can be either

an absolute amount of formant shift or a relative ratio for formant shift, depending on whether *bRatioShift* is set to 0 or 1. The angle *pertPhi* has a unit of radians and starts from the positive horizontal axis and increases in the counterclockwise direction, in a fashion analogous to the complex plane. For example, if *bMelShift* = 0, *bRatioShift* = 1, *pertAmp* = all 0.3's and *pertPhi* = all 0's, then the perturbation will be a uniform 30% increase in F1 of the vowel.

The mappings from F2 to A and φ are specified in the form of look-up tables by the three parameters *pertf2*, *pertAmp* and *pertPhi*, which are all 1×257 vectors. The hard-coded number 257 may look a little peculiar and arbitrary. It is selected to enable efficient binary search for mapping unperturbed F2 values to perturbation vectors. Linear interpolation is used to calculate the magnitude and angle of the perturbation vectors. See the demo script "audapterDemo_triphthong.m" for an example of how to use the perturbation field (see also Sect. 2.1.4)

The design of the perturbation is general enough to allow flexible F2-dependent perturbations. However, your project may concern with only fixed perturbation to a steady-state vowel, and hence not require this flexible setup. If that's the case, you can simply set both *pertAmp* and *pertPhi* as constant. For example, if you want to introduce a 300-mel downward shift to the F1 of a steady-state vowel (e.g., /ɛ/), you can simply set *bMelShift* = 1, *bRatioShift* = 0, and let *pertAmp* be a 1×257 vector of all 300's and let *pertPhi* be a 1×257 vector of all π's. Here, *pertF2* should be a 1×257 linear ramp from *f2Min* to *f2Max*.

You should also keep in mind that the parameters *f1Min*, *f1Max*, *f2Min*, *f2Max*, *LBk*, *LBb*, *pertF2*, and *pertAmp* all have units that are dependent on *bMelShift*, despite the fact that the formant frequency outputs in the data structure (Sect. 6) and other parameters of Audapter (e.g., *srate*, *fn1*, *fn2*, *wgFreq*, see Table 1) always have the unit of Hz.

6. Data Structure of the .mat Files

The runExperiment script generates a .mat file for each trial. Each of those .mat files contains a variable by the name of "data". This variable is obtained through the MATLAB script *AudapterIO*:

```
data = AudapterIO('getData');
```

It is a structure containing a number of fields. The following is a description of the meaning of those fields.

Table 3. Data fields and their meanings in the output of AudapterIO('getData') and the saved data files

Field name	Meaning

data.signalIn	Input signal after downsampling. This can either be from the audio interface for an online trial, or from the "runFrame" option for an offline trial.
data.signalOut	Output signal, before upsampling, possibly perturbed, depending on the perturbation configuration. If an online trial is involved, this is the signal delivered to the output channel of the audio interface, i.e., to the headphone
data.intervals	The onset sample numbers (integers) for the data frames.
data.rms	Column 1: Short-time root-mean-square (RMS) intensity values, smoothed, denoted **rms_s**. The amount of smoothing is determined by the value of the parameter "rmsFF" (RMS forgetting factor). Column 2: Short-time RMS intensity, pre-emphasized (i.e., high-pass filtered) and smoothed, denoted **rms_p**. The amount of pre-emphasis (i.e., high-pass filtering) is determined by parameter "preemp", whose value is 0.98 by default. The amount of smoothing is the same as in column 1. Column 3: Non-smoothed, non-pre-emphasized short-time RMS intensity, denoted **rms_o.**
data.fmts	Formant frequencies. The values are non-zero only for the moments in which the smoothed short-time RMS intensity rms_s and the ratio between rms_s and rms_p (i.e., rms_ratio) satisfy the following condition: (rms_s > 2 * rmsThr) && (rms_ratio > rmsRatioThresh / 1.3) || (2 * rmsThr > rms_s > rmsThr) && (rms_ratio > rmsRatioThresh) The use of rmsRatioThresh in this condition is for excluding the unvoiced segments from the formant tracking. The value of rms_ratio should be greater during vowels and other voiced sounds than during unvoiced sounds such as fricatives. If you do not wish to include this ratio in the determination of formant-tracking intervals, you can set rmsRatioThresh to 0.
data.rads	The phase angles of the poles in the Z-plane in the LP results.
data.dfmts	Rate of change (velocity) of the formants.
data.sfmts	Formant frequencies in the output signal (i.e., auditory feedback). The values are non-zero during and only during the time intervals involving non-zero formant shifts.
data.rms_slope	Short-time slope of rms_s (the smoothed short-time RMS intensity values). This slope has a unit of s^{-1}, and is obtained through Pearson linear regression of the rms_s values against the time. The size of the regression window is adjustable in the first part of an OST file (see Sect. 7). The reason why this parameter is configured in OST files is because rms_slope is used in certain heuristic rules in OST.
data.ost_stat	OST status number, determined by using the configurations in the OST file loaded into Audapter. If no OST file has been loaded, the values in ost_stat will be all zero.
data.pitchShiftRatio	The ratio of pitch shift as a function of time. 1.0 - no shift; >1.0 – up-shift; <1.0 down-shift. Note: Audapter currently doesn't support the extraction of pitch shift values in the multi-voice feedback mode.

	Under that mode, only the pitch shifting ratio of the first feedback voice is recorded.
data.params	Parameter settings during the trial. See Table 1 for a full description of the parameters.
data.vowelLevel (may or may not exist depending on the version of runExperiment.m)	Mean intensity of the vowel (in dB SPL A).
data.uiConfig (may or may not exist depending on the version of runExperiment.m)	GUI configuration during this trial
data.timeStamp	A time stamp created shortly after the end of the trial

7. Online Status Tracking (OST)

For certain psychophysical AFP applications, you may wish to use a multisyllabic speech utterance and impose the perturbation during specific sounds or syllables of the utterance. Online status tracking (OST) is a functionality of Audapter that serves this purpose. You can design a set of heuristic rules based on signal properties such as intensity to detect the onset and offset of various sounds in the utterance. With OST, Audapter assigns a non-negative integer status number to each input frame in real time. In post-processing, these state numbers are stored in data.ost_stat (see Sect. 6). You can map these state numbers to various types of perturbations in by using perturbation configuration (PCF) files, a topic covered in Sect. 8. Therefore OST and PCF work together to enable the online automatic triggering of perturbation events.

An OST file is an ASCII text file that configures the set of heuristic rules for tracking the progress of a speech utterance. It can be loaded into Audapter with the 'ost' option:

```
Audapter('ost', ost_fn, 0);
```

The second input argument is the name of the OST configuration file. The third argument is a Boolean flag for verbose mode.

Code Sample 1 below is an example OST file. You should follow this formant when creating your own OST files. This file consists of three parts. Part 1 is a single line that begins with “rmsSlopeWin =”. This sets the window size (in seconds) for computing the slopes of short-time RMS intensity. Part 2 begins with a line such as “n = 3”. This compulsory line specifies the number of OST rules in the OST file. The number of following lines in this part must match the value of n. Each of the following lines specifies a tracking rule. These rules are engaged sequentially during an online trial.

Code Sample 1. An example online status tracking (OST) configuration file.

```
# Online status tracking (OST) configuration file
rmsSlopeWin = 0.030000

# Main section: heuristic rules for tracking
n = 3
0 INTENSITY_RISE_HOLD 0.02 0.0200 {} # Detect the onset of the first word
2 INTENSITY_FALL 0.01 0.0100 {} # Detect the end of the first word
4 OST_END NaN NaN {}

# maxIOICfg
n = 1
2 0.2 4
```

Each line of OST rule consists of five fields that are words or numbers, separated by single spaces. The first field is the starting state value. The second field selects the mode of tracking. It can be either a number from the first column of Table 2 or an all-upper-case string from the second column of the same table. Table 2 lists the currently supported modes of tracking. They are based mostly on short-time intensity, its rate of change (slope), and the ratio of spectral intensity in high- and low-frequency bands (e.g., mode numbers 30 and 31). If you wish to include new and/or more sophisticated modes of tracking, changes to the C++ source code of Audapter will have to be made. Specifically, the OST functions are package in header and source files ost.h and ost.cpp. The third and fourth fields of the line are the two mode-specific parameters that can be configured by the user. For example, in the tracking INTENSITY_RISE_HOLD, the user needs to set the intensity threshold and the hold duration in the third and fourth fields, respectively. Table 2 contains descriptions of these parameters. Note that some tracking modes are associated with two parameters, while others are associated with one or none. In the cases wherein fewer than two parameters are required, use the first several ones of the third and fourth fields, and leave the rest at NaN or arbitrary values. The fifth field of the line is a pair of curly brackets. This compulsory field serves no purpose in the current version of Audapter and is reserved for potential future uses.

Each tracking mode is associated with a fixed increment in status number at the end of the mode. For example, the mode INTENSITY_RISE_HOLD involves an increment of 2 from the beginning to the end of the tracking. The last column of Table 2 lists these increment amounts. In Part 2 of the OST file, the first fields of the consecutive lines must match these increment values, otherwise unexpected and unpredictable tracking errors may occur. In other words, if the onset ost_stat value of an INTENSITY_RISE_HOLD rules is 0, for example, then the beginning ost_value of the next rule, specified in the following line, must be 2. It should also be noted that each set of OST rules must end with a rule of the OST_END tracking mode (e.g., see the code sample above).

Part 3 of the OST file is for the maximum-inter-onset-interval (maxIOI) mode of tracking. The maxIOI mode of tracking is a quite *ad hoc* way of dealing with possible tracking failure. It is essentially a way of telling the OST module of Audapter that "you should proceed to a different state forcefully, regardless of the tracking rule, if a certain amount of time has elapsed from the onset of a given state". As you probably have come to realize, this is not an elegant way of approaching the tracking problem and should be used only as a last resort.

This part begins with a line which specifies the number of maxIOI rules. The number of the trailing lines in this section must match the value of n in this first line. In each of the trailing lines, there are three numbers. The first number is the onset ost_stat number. The second one is the maximum wait time, in seconds. The third onset is the value of ost_stat that Audapter will automatically jump to when this wait period has elapsed.

Consecutive parts in the OST file are separated by blank lines. You can insert comment lines in OST file. These comment lines should begin with the hash (#) character. You can also add comments to the end of uncommented lines (as in certain programming languages such as Python or MATLAB).

Table 2. A list of supported heuristic modes for online status tracking (OST)

Mode Number	**Mode Name**	**Description and Example Usage**	**Parameters**	**Increment in OST state number**
0	OST_END	Serves as an ending rule. Once a rule of this mode is reached, OST halts and the status number freezes at the current value until the end of the trial. Each OST file **must** end with this rule.	(None)	0
1	ELAPSED_TIME	Elapsed time from previous state Example: Wait for a fixed amount of time (e.g., 100 ms) after voicing onset	prm1: duration	+1
5	INTENSITY_RISE_HOLD	Crossing an intensity (RMS) threshold from below and hold Example: Detect the onset of a vowel or a voiced consonant-vowel cluster	prm1: rmsThresh prm2: minDur (s)	+2
6	INTENSITY_RISE_HOLD_POS_SLOPE	Crossing an intensity threshold from below and hold, during positive RMS slope Example: Detect the onset of a vowel, with added security	prm1: rmsThresh prm2: minDur (s)	+2
10	POS_INTENSITY_SLOPE_STRETCH	Stretch of positive intensity slope, with only a stretch count threshold Example: This is still another way to detect the onset of a vowel or voiced consonant	prm1: stretchCntThresh	+2

11	NEG_INTENSITY_SLOPE_STRETCH_SPAN	Stretch of negative intensity slope, with a stretch count threshold and a stretch span threshold Example: Detect the end of a vowel or the silent interval between a vowel and a stop consonant	prm1: stretchCntThresh prm2: stretchSpanThresh	+2
20	INTENSITY_FALL	Fall from a certain intensity threshold Example: Detect the end of a vowel	prm1: rmsThresh prm2: minDur (s)	+1
30	INTENSITY_RATIO_RISE	Intensity ratio cross from below and hold. See rows “data.rms” and “data.fmts” in Table 3 for further details. Example: Detect the onset of a sibilant (e.g., [s])	prm1: rmsRatioThresh prm2: minDur (s)	+2
31	INTENSITY_RATIO_FALL_HOLD	Intensity ratio: fall from a threshold and hold Example: Detect the end of a sibilant (e.g., [s])	prm1: rmsRatioThresh prm2: minDur (s)	+2

8. Perturbation Configuration (PCF)

Once you have the OST heuristics configured, the next step in enabling focal perturbation of AF is supplying Audapter with a perturbation configuration (PCF) file. Similar to OST files, you can input a PCF into Audapter by using the following syntax:

```
Audapter('pcf', pcf_fn, 0);
```

The perturbation settings in a PCF file are divided into two parts:

1) Time warping settings
2) Settings for pitch, intensity and formant perturbations to be delivered at each specific state number

This two-part organization is reflected in the structure of the PCF files. See the following code example:

Code Sample 2. An example perturbation configuration (PCF) file.

```
# Part 1 (Time warping): (state number), tBegin, rate1, dur1, durHold, rate2
2
0.94, 0.1, 0.1, 0.1, 1.5 # Time warping 1
1.502, 0.1, 0.1, 0.1, 1.5 # Time warping 2


# Part 2: stat pitchShift(st) gainShift(dB) fmtPertAmp fmtPertPhi(rad)
6
0, 0.0, 0, 0, 0
1, 0.0, 0, 0, 0
2, 0.0, 0, 0, 0
3, 0.0, 0, 0, 0
4, 0.0, 0, 0, 0
5, 2.0, 0, 0, 0 # Two-semitone upward pitch perturbation during the last word
```

This example PCF file defines two types of perturbations during a single utterance: two temporally non-overlapping time warps in the first section and a two-semitone pitch shift in the second section.

The syntax of Part 1 (time warping) is as follows. You begin by including a line consisting of a single positive integer, specifying the number of time warping events in the utterance. Following this line, a correct number of trailing lines needs to be entered, defining details of each time-warping event. There are two possible formatting for each line. In the first format, five numbers are included in the line. These five numbers provide Audapter with the following pieces of information, respectively,

1) *tBegin*: The onset time of the warp event (relative to utterance onset)
2) *rate1*: The rate of initial time warping, with <1 being time dilation. In fact, this number has to be ≤ 1.0 in order for the system to be causal.
3) *dur1*: Duration of the initial warping at *rate1*.
4) *durHold*: Duration of the hold (i.e., no-warping) period following *dur1*
5) *rate2:* rate of the time warping in the catch-up (or recovery) period. This number has to be ≥ 1.0.

In total, a time warping event configured in this format lasts for a total duration of

$$tBegin + dur1 + durHold + dur2$$

wherein

$$dur2 = (1 - rate1) / (rate2 - 1) \times dur1$$

When more than one warping events are specified in this format, Audapter will check to make sure that there are no temporal overlap between them. It will report an error if an overlap exists.

In format 2, six, instead of five, numbers are included in each line. The first number should be an integer and it specifies the OST status number the time-warping event resides in. In this format, the onset timing of the warping event is relative to the onset time of the specified status number, not the onset of the utterance. The following five numbers have the same meaning as the numbers in line format 1. As in format 1, Audapter will look for temporal overlaps between time-warping events of the same OST status number and report an error if it finds any. However, because the onset timing of different OST status numbers cannot be predicted beforehand, Audapter will not attempt to check overlaps between time-warping events between different OST numbers or between time-warping events specified with different formats.

Note that the sample PCF file above includes only format 1.

In Part 2 of the PCF file, you define the amount of the following three types of non-time-warping perturbations at each OST status number. This section needs to start with a line consisting of a single integer that specifies the total number of different OST status numbers. Since OST status numbers always begin at 0, this integer should be one plus the maximum OST status number in the OST file you are using.

The following lines have a fixed format, namely five numbers separated by commas and/or spaces. These five numbers, in order, define the following perturbation settings:

1) 1st number: The OST status number. Note that this has to be sequential. You cannot skip status numbers or include status numbers that are outside the possible range.
2) 2nd number: The amount of pitch shifting, in semitones. Positive values correspond to upward pitch shifts, while negative ones correspond to downward shifts.
3) 3rd number: The amount of intensity perturbation, in dB.
4) 4th number: The magnitude of joint F1-F2 perturbation vector, in the formant plane spanned by F1 and F2. The unit of this depends on the *bRatioShift* parameter set in Audapter (see Section 4). If it is set to 0 (false), the unit will be Hz. Otherwise this number is dimensionless and specifies the ratio (fraction) of formant shifts.
5) 5th number: The angle of the formant perturbation vector, in radians. For example, an angle of 0 leads to a pure upward F1 perturbation. An angle of $-\pi/2$ leads to a pure downward F2 perturbation. An angle of $\pi/4$ is used to specify equal amounts of perturbation to F1 and F2.

As in an OST file, you can add comments to the PCF file with the "#" character (see Code Sample 6).

9. Hardware Wiring and Configuration (Non-MRI Experiments)

The following contains a photo and descriptions of the hardware wiring and configurations for a typical psychophysical (non-MRI) AFP experiment based on Audapter.

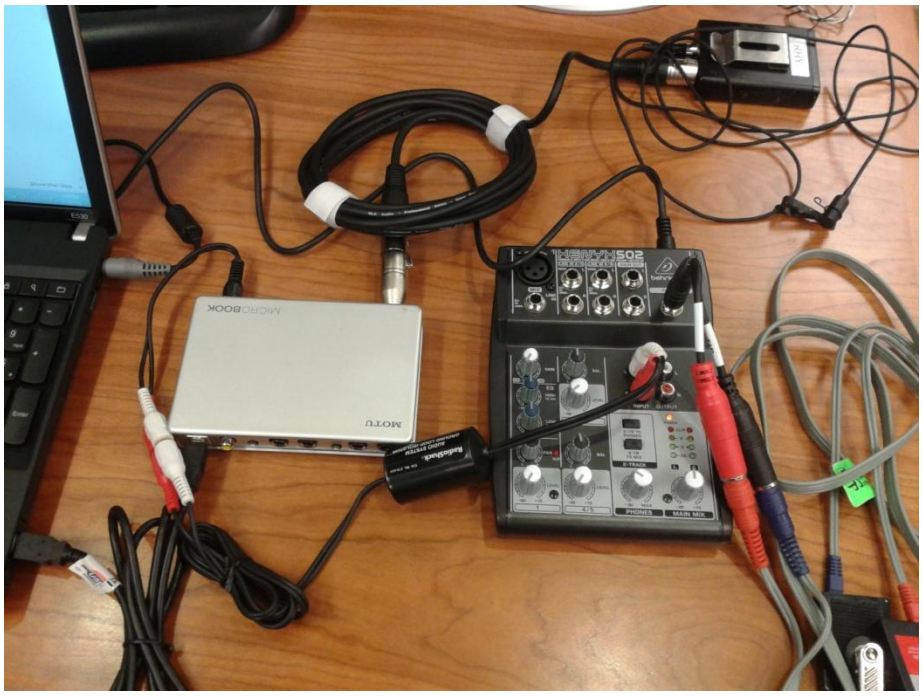

Figure 5. A snapshot of the basic hardware connections for a psychophysical AFP experiment.

- MOTU MicroBook USB <--> ThinkPad USB on the right-hand side
- Xenyx502 AC Power In <--> Power outlet
- MicroBook headphones (front) --> 1/8 inch/RCA adapter --> Ground loop isolator --> Xenyx502 2-TRACK Input L/R
- Xenyx502 Phones --> headphones (whatever model you use)
- Microphone (whatever model you use) --> (Mic XLR extension cable --> XLR/quarter inch adapter -->) MOTU Mic (front)
- Make sure that Xenyx502 the power indicator light is one
- Xenyx502 "2-TR to phones" button: down
- Xenyx502 "2-TR to mix" button: up
- Xenyx502 main mix: @ 12 o'clock
- Xenyx502 PHONES knob: determined by headphones calibration

10. Running an Auditory-Feedback Perturbation Experiment

1. Instructions for obtaining the Audapter software package and building the core C++ MEX program can be found in Appendix 1 and Appendix 2, respectively.
2. Open MATLAB
   Make sure that Internet is available if a non-standalone MATLAB license is used.
3. Type in command:
   ```
   addpath c:/speechres/commonmcode
   cds('audapter_matlab');
   ```
4. (Optional) Verify that the path to Audapter is found by typing:
   ```
   which Audapter
   ```
   and see that the return value is not empty. It should be something like:
   ```
   C:/speechres/audapter/Audapter-2.1/BIN/Release
   ```
5. (Optional) Verify that Audapter recognizes the sound card:
   ```
   Audapter info
   ```

If the sound card (e.g., MicroBook) is connected properly, you should see the name of the device in the printed message. The "match" / "non-match" info can be ignored for now.

6. Edit expt_config.txt, for example:
   ```
   edit expt_config_fmt.txt
   ```
   Edit the subject info, device name (MicroBook or UltraLite), data directory, experiment design, single-trial perturbation design, etc.
   Note that the above file name "expt_config_fmt.txt" does not exist initially. You can create it by copying one of the example configuration files that comes with the package (see below).
7. To start a behavioral (non-MRI) experiment, type in:
   ```
   runExperiment expt_config_fmt.example.txt
   ```
   or
   ```
   runExperiment expt_config_pitch.example.txt
   ```
   or
   ```
   runExperiment expt_config_fmt.example.txt twoScreens
   ```
   or
   ```
   runExperiment expt_config_pitch.example.txt twoScreens
   ```

   To start a sparse-sampling fMRI experiment, do:
   ```
   runExperiment expt_config_fmt_fmri.example.txt twoScreens
   ```

   The second input argument is the name of the experiment configuration file to be used. It determines the type of the experiment you will run. See Section 11 "Experiment Configuration File" for details.
   The input argument "twoScreens" can be used when a second monitor is connected. This argument will make the program display word stimuli and the visual feedback to the subject on the second monitor. If so, make sure that the second screen is to the right of the main screen and has a resolution of 1024x768.
8. The command "runExperiment" opens up three windows:
   a) Data monitor, which displays the input and output waveforms, spectrograms and associated data after each trial
   b) Participant window: this is the window that displays stimuli and certain visual feedback to the subject
   c) Control window, in which the experimenter can see and adjust certain settings of experiment workflow

   The subject is supposed to see only the participant window, but not the data monitor and control window.
9. If you entered 0 into the PRE_REPS, PRACT1_REPS and PRACT2_REPS in the expt_config.txt file, the practice phases will be skipped. Hit Enter until you reach the rand (i.e., main) phase.
10. Hit Enter again upon entering the rand phase. Click the play button in the control window and the first trial will start.
11. To quit the experiment before the completion of the experiment, press Ctrl+C in the main MATLAB window and then enter command:
    ```
    Audapter stop;
    ```

11. Experiment configuration (expt_config) file

This section is a description of some of the most important fields in the experiment configuration files (e.g., expt_config_fmt.example.txt);

- DATA_DIR is the base directory in which the experiment data will be saved. For example, if the SUBJECT_ID is "TS_01" and DATA_DIR is "C:/DATA/APE", the data from the subject's experiment will be saved at C:/DATA/APE/TS_01. Note that if a subject undergoes

multiple experiments, each experiment needs to have a unique SUBJECT_ID in order to prevent the overwriting of previous data.

- To switch between a formant perturbation experiment and a pitch experiment one, modify the fields PITCH_SHIFT_CENT, F1_SHIFTS_RATIO and F2_SHIFTS_RATIO. If the values are zero in PITCH_SHIFT_CENT and non-zero in either F1_SHIFTS_RATIO or F2_SHIFTS_RATIO, the experiment will be a pitch perturbation experiment. If the converse is the case, the experiment will be a formant perturbation one.
  The above-mentioned three fields are for specifying the randomized-perturbation part of the experiment. For specifying the sustained-perturbation part[4], modify fields: SUST_PITCH_SHIFTS_CENT, SUST_F1_SHIFTS_RATIO and SUST_F2_SHIFTS_RATIO. They work the same way as the three previously mentioned fields.
  Note that in principle, you could do an experiment with pitch perturbation in its randomized part and formant perturbation in its sustained part (or vice versa), but such a design probably would not makes sense in most situations.
  Also, you could mix pitch and formant perturbation by having non-zero values in both the pitch and formant (F1/F2) fields.
- To specify whether an experiment will contain randomized perturbation, sustained perturbation, or both, you can modify these fields: N_RAND_RUNS and {SUST_START_REPS, SUST_RAMP_REPS, SUST_STAY_REPS, SUST_END_REPS}. If N_RAND_RUNS is non-zero, the experiment will contain a randomized part (after the pre, pract1 and pract2 phases). If the other four fields are non-zero, the experiment will contain a sustained part. Note that the sustained part always goes after the randomized part. So if you want to do an experiment with only a sustained part, you should set N_RAND_RUNS to zero. If N_RAND_RUNS and the other four fields are all non-zero, the experiment will consist of both a randomized part and a sustained one, in that particular order.
- To specify whether the experiment is an behavioral or fMRI session, set the field TRIGGER_BY_MRI_SCANNER to 0 or 1, respectively. Other relevant fields are MRI_TRIGGER_KEY and FMRI_TA. The program will wait for the trigger key issued by the MRI session at the onset of every scan. The key is specified in MRI_TRIGGER_KEY. After the program receives the key, it will wait for a period specified in FMRI_TA before presenting the word stimulus. This ensures the sparse-sampling paradigm in the fMRI session. Note (important) for the program to receive the trigger key properly, you need to always focus on the "Control window".

Acknowledgements

Below is a potentially incomplete list of grants that have supported involved personnel during the development of Audapter:

- N.I.H. R01-DC001925 (P.I.: Joseph Perkell)
- N.I.H. R01-DC007683 (P.I.: Frank Guenther)
- N.I.H. R56- DC0010849 (P.I.: Joseph Perkell)
- N.S.F. Doctoral Dissertation Improvement Grant (DDIG) 1056511 (P.I.: Joseph Perkell and Shanqing Cai)

[4] AFP experiments can be divided into two categories based on design. The first category is what we call the randomized design. In this design, the non-perturbed and perturbed trials are mixed together and randomized in order. This design is suitable for studying the online AF-based control of speech movements. Examples of this design can be found in Cai et al. (2011; 2012). The second category is called sustained design. A sustained perturbation experiment is organized as a number of phases, typically called "start", "ramp", "pert" and "end". The perturbation trials are congregated into the ramp and pert phases, while the other phases contain non-perturbed trials. This design is used to study long-term modification of the speech commands under consistent feedback errors, sometimes referred to as auditory-motor learning. Examples of this design can be found in Cai et al. (2010).

- M.I.T. Edward Austin Endowed Fellowship (Recipient: Shanqing Cai)
- M.I.T. Chyn Duog Shiah Memorial Fellowship (Recipient: Shanqing Cai)
- Keith North Memorial Fund of the Research Laboratory of Electronics, M.I.T.

Appendix 1. Instructions for obtaining and setting up the Audapter package

1. Download and install Git Bash (http://git-scm.com/downloads) or other preferred Git programs
2. Open Git Bash
3. Create directory: `mkdir /c/speechres`
4. `mkdir /c/speechres/audapter`
5. `cd /c/speechres`
6. `git clone https://github.com/shanqing-cai/commonmcode.git`
7. `git clone https://github.com/shanqing-cai/audapter_matlab.git`
8. There are two alternative approaches for obtaining the core C++ MEX program of Audpater.
   a) *Download the source code and build it on your own*. See Appendix 2 for details.
   b) *Download pre-built binary MEX files*. Download the compiled core MEX program of Audapter from the author's website (http://people.bu.edu/scai/). Create directory such as "C:\speechres\audapter\Audapter-2.1\BIN\Release" and put the MEX file (Audapte.mexw32 or Audapter.mexw64) in that directory.

In addition, make sure that at least one ASIO sound card driver is installed on your computer, as Audapter MEX program requires such a driver to run. This is necessary even if you plan to use Audapter only in the offline mode. Installing the ASIO driver does not require you to actually have the sound card hardware. You can download MOTU's universal audio installer for free at: http://www.motu.com/download. Alternatively, you can try ASIO4ALL: http://www.asio4all.com/.

To update the code through Git in the future, do in Git Bash:

```
cd /c/speechres/audapter_matlab

git pull origin master
```

Note that Git pulling can execute successfully only if you have not changed any of the tracked files in the Git repository or if you have made some changes and committed them. If you are new to Git and want to learn about it, there are several good sources online, such as http://git-scm.com/docs/gittutorial.

Appendix 2. Instructions for building the core MEX program of Audapter in Microsoft Visual C++

1. Request the C++ source code from Shanqing Cai (scai@bu.edu), the author of this manual and the current maintainer of Audapter.
   (Courtesy notice: Please check with the author before sharing the files or links with other labs or research groups.)
2. Extract the Audapter-2.1 directory in the zip archive to C:/speechres/audapter/
3. Open the solution in Visual C++ 2010 or later

4. In Visual C++, select the correct architecture (win32 or x64). You may need to manually set the linker output format to either mexw32 or mexw64, depending on your architecture.
5. There are a number of configurations in the solution, such as "Release", "Release_USyd" and so on. The main difference between these configurations are the include and library paths. You can use an existing configuration and make necessary modifications to it. You can modify the configuration by right-clicking "Audapter" in the Solution Explorer and select "properties".
   Below are the most important settings for ensuring successful debugging and compiling:
   a. General / Configuration type = Dynamic Library (.dll)
   b. C/C++ / Additional include directories should contain $(SolutionDir)\audioIO, $(SolutionDir)\SibShift and C:\Program Files\MATLAB\R2011a\extern\include. The last directory may vary depending on your MATLAB installation path.
   c. Linker / General / Additional library dependencies should include C:\Program Files\MATLAB\R2011a\extern\lib\win32\microsoft. This directory may vary depending on your MATLAB installation path and your CPU architecture.
   d. Linker / Input / Additional dependencies should include libmx.lib, libmex.lib and libmax.lib
   e. Other settings that are described in this helpful webpage for guiding beginners through MEX building in VC++: http://coachk.cs.ucf.edu/GPGPU/Compiling_a_MEX_file_with_Visual_Studio2.htm
6. Use menu option: "Build → Rebuild" to rebuild both the audioIO and Audapter projects (in that order) in the solution. A number of warning messages are expected. Most of them should be harmless and can be ignored.
7. If your build result is in a directory other than BIN/Release, e.g., in BIN/Release/USyd, move the mexw32 (or mexw64) file into BIN/Release, so that the cds script can find the Audapter MEX file.

References

Boucek M. (2007). The nature of planned acoustic trajectories. *Unpublished M.S. thesis.* Universität Karlsruhe.

Cai S, Boucek M, Ghosh SS, Guenther FH, Perkell JS. (2008). A system for online dynamic perturbation of formant frequencies and results from perturbation of the Mandarin triphthong /iau/. In *Proceedings of the 8th Intl. Seminar on Speech Production*, *Strasbourg, France*, Dec. 8 - 12, 2008. pp. 65-68.

Cai S, Beal DS, Ghosh SS, Guenther FH, Perkell JS. (In press). Impaired timing adjustments in response to time-varying auditory perturbation during connected speech production in persons who stutter. *Brain Lang.*

Cai S, Beal DS, Ghosh SS, Tiede MK, Guenther FH, Perkell JS. (2012). Weak responses to auditory feedback perturbation during articulation in persons who stutter: Evidence for abnormal auditory-motor transformation. *PLoS ONE*. 7(7):e41830.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2010). Adaptive auditory feedback control of the production of the formant trajectories in the Mandarin triphthong /iau/ and its patterns of generalization. *J. Acoust. Soc. Am.* 128(4):2033-2048.

Cai S, Ghosh SS, Guenther FH, Perkell JS. (2011). Focal manipulations of formant trajectories reveal a role of auditory feedback in the online control of both within-syllable and between-syllable speech timing. *J. Neurosci.* 31(45):16483-16490.

Kalinowski J, Armson J, Stuart A, Gracco VL. (1993). Effects of Alterations in Auditory Feedback and Speech Rate on Stuttering Frequency. *Lang. Speech.* 36(1):1-16.

Larson CR, Altman KW, Liu H, Hain TC. (2008). Interactions between auditory and somatosensory feedback for voice F0 control. *Exp. Brain Res.* 187:613-621.

Tourville JA, Cai S, Guenther FH (2013) Exploring auditory-motor interactions in normal and disordered speech. *Proceedings of Meeting on Acoustics*. 9:060180. Presented at the *165th Meeting of the Acoustical Society of America*, Montreal, Quebec, Canada, June 2 – June 7, 2013.

Xia K, Espy-Wilson C. (2000). A new strategy of formant tracking based on dynamic programming. In *ICSLP2000*, Beijing, China, October 2000.