**A Manual of Audapter**

Shanqing Cai

[scai@bu.edu](mailto:scai@bu.edu), [shanqing.cai@gmail.com](mailto:shanqing.cai@gmail.com)

Speech Laboratory,

Department of Speech, Language and Hearing Sciences,

Sargent College of Health and Rehabilitation Sciences,

Boston University

January 2014

1. Overview: What is Audapter?[[1]](#footnote-2)

Audapter 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

Perturbation types a – d 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. X.X). Certain combinations of perturbation types (e.g., a and c, b and e) can be delivered simultaneously.

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 includes graphical user interfaces for stimulus presentation, experimental workflow control, data preprocessing, as well as basic analysis of ths data. Although Audapter has only be thoroughly used and tested on Windows PCs (both 32- and 64-bit), it should be portable to other platforms, including Mac.

Audapter was 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 (Ref) 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 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 will serve as a general guide to the usage of Audapter.

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, these references can be also be cited.

2. Getting Started - Running 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, compile and set up Audapter can be found in Section X.X. To set up the environment properly, you need to add path to Shanqing Cai's MATLAB toolkit, by entering in MATLAB a command such as the following:

addpath [e:/speechres/commonmcode](file:///e:\speechres\commonmcode);

Then, use the cds script in the toolkit to set up the paths and environment automatically:

cds('ape');

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 perturbation 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');

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”.

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 lengths 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, more than one time-warping events can be included in the same utterance.

X. 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 an integer status number to each input frame in real time. In post-processing, these state numbers are stored in data.ost\_stat (see Sect. X.X). You can map these state numbers to various types of perturbations in by using perturbation configuration (PCF) files, a topic covered in Sect. X.X. 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 Audpater 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 X 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 configures 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 (ost\_stat) value. The second field selects the mode of tracking. It can be either a number from the first column of Table 1 or an all-upper-case string from the second column of the same table. Table 1 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-freuency 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. The fourth column of Table 1 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 field serves no purpose in the current version of Audapter, but are 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 1 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 when necessary.

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 1.** 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 (TODO: Explain) cross from below and hold  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 |

X. Perturbation configuration files (PCFs)

Once you have the OST heuristics configured, the next step in enabling focal perturbation of AF is supplying Audapter with a perturbation configuration file (PCF). 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 is 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 X2. An example online status tracking (OST) configuration file.

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

|  |
| --- |
| # Section 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  # Section 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 Section 1 and a two-semitone pitch shift in Section 2.

The syntax of Section 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, the correct number of lines need 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) × 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 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 formant 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 Section 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 corresponds 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 XX and Table XX). 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 -π/2 leads to a pure downward F2 perturbation. An angle of π/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 X2).

References

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

Universität Karlsruhe.

Xia K, Espy-Wilson C. (2000). A new strategy of formant tracking based on dynamic

programming. In ICSLP2000, Beijing, China, October 2000.

1. List of abbreviations: F0 – fundamental frequency; F1- 1st formant frequency; F2 – 2nd formant frequency; OST – Online status tracking; PCF – Perturbation configuration; RMS – Root mean square. [↑](#footnote-ref-2)