Audiroty morphing using minimum STRAIGHT

Hideki Kawahara

29th April 2005

Auditory Media Laboratory
Department of Design Information Sciences
Faculty of Systems Engineering, Wakayama University
930 Sakaedani, Wakayama, Wakayama 640-8510, Japan

Auditory morphing using *STRAIGHT*[1] provides close to natural reproduction of morphed speech sounds when two natural speech examples are given[2, 3]. This text introduces a new set of procedures for making morphing easy for everyone. The procedures are based on minimum STRAIGHT that is the STRAIGHT suite with newly defined APIs.

1 How to install morphing

Please set up the minimum STRAIGHT as the basis. Please read Required environment section of the document for details. The latest minimum STRAIGHTis linked here. It is a compressed archive. A link to the compressed archive of the procedures for morphing is also given. Please read the HTML document of the morphing codes (written in Matlab) for details.

2 Terminology

This section discusses basic concepts for morphing. It can be too detailed, because sometimes it is not separable concepts from their implementation.

2.1 Morphing object: mObject

Morphing is a procedure to generate sounds in between two given sound examples. A set of parametric representations of sounds and reference information to define correspondence between two examples have to be given for the morphing procedure to be functional. Let's call an entity that consists of these information and representations a morphing object (M-object in short).

Ingredients of a morphing object M-object is implemented as a structured variable in Matlab. M-object has the following types of information. They are implemented as fields in a structured variable.

- **Speech waveform**
- **Sampling frequency**
- **STRAIGHT spectrogram**
- **Fundamental frequency**
- **Aperiodicity indices**

- **Frame update interval**
- **Anchoring point**

The other side information such as date of creation, analysis parameters and location of the source material, are also stored in its fields.

2.2 Adding information to a morphing object

Information is added to a morphing object when some action is applied to it. For example, reading a sound from a file, performing STRAIGHT analysis and marking anchoring points are such action.

2.3 Synthesizing speech from a morphing object

A function to synthesize sounds from morphing objects is implemented, because morphing results are also represented as morphing objects.

2.4 Morphing morphing object

Morphing procedure is implemented to generate a morphing object from two morphing objects. This unified implementation makes it possible to apply morphing recursively.

3 Step by step introduction to auditory morphing using STRAIGHT

The following sections introduces auditory morphing procedure in a step-by-step manner using real examples.

3.1 Reading speech from a file

Firs of all, it is necessary to create a morphing object. Then, updating appropriate fields with the speech signal and the fundamental frequency of the read file. In a usual morphing case, it is generally necessary to read two speech samples. However, it is also possible to morph without the target example. The following explanations are based on normal morphing using two sound examples. Let's use a Japanese utterance 'HAI' ('YES' in English). The word is spoken in two different emotional expressions; neutral and angry. They are stored as `tmneu.wav` and `tmang.wav` respectively.

```
neutralHai=createMobject;
angryHai=createMobject;
[x,fs]=wavread('tmneu.wav');
neutralHai=updateFieldOfMobject(neutralHai,'waveform',x);
[x,fs]=wavread('tmang.wav');
angryHai=updateFieldOfMobject(angryHai,'waveform',x);
```

3.2 Speech parameter extraction using STRAIGHT

Speech parameters extracted using STRAIGHT are registered as fields of a structured variable. The usual way to register information to a field is to use `updateFieldOfMobject` function. It also works with STRAIGHT parameters and is a flexible general purpose function. It needs three successive calls to this function. A short cut function is prepared to combine these three steps into a single function `executeSTRAIGHTanalysisM`. This function performs a STRAIGHT analysis using

its default values. (It is possible to use other optional parameters. Please refer to the minimum STRAIGHT document.)

```
neutralHai = executeSTRAIGHTanalysisM(neutralHai);
angryHai = executeSTRAIGHTanalysisM(angryHai);
```

The next step needs somewhat lengthy manual inspection. It is advised to save resulted anchoring points information to a file with STRAIGHT analysis parameters as a mat file storing the structured variable holding all those information. One problem is an excessive size of STRAIGHT parameters. (This actually is a waste of file space. A more memory efficient data format will be introduced later. Please be patient for the time being.)

```
save neutralHai neutralHai
save angryHai angryHai
```

The current fields in a morphing object is listed below. It is designed to have room for extension.

```
neutralHai =

                          date: '27-Feb-2005 01:47:04'
                           pwd: '/Users/kawahara/matlab/newMorph'
                      waveform: [11368x1 double]
              samplingFrequency: 44100
                            F0: [1x258 double]
                   spectrogram: [1025x258 double]
              aperiodicityIndex: [1025x258 double]
            frameUpdateInterval: 1
             anchorTimeLocation: []
          maximumFrequencyPoints: 9
                 anchorFrequency: []
          F0extractionConditions: [1x1 struct]
    SpectrumExtractionConditions: [1x1 struct]
```

3.2.1 STRAIGHT parameter extraction with manual tuning

Sometimes extracted STRAIGHT parameters are not suitable for high-quality morphing due to errors in parameter estimation especially in F0 extraction. If it is the case, it is necessary to manually tune analysis parameters or to use more robust extraction algorithm to estimate relevant parameters instead of using a one shot procedure `executeSTRAIGHTanalysisM`.

3.3 Attributes necessary for morphing

It is necessary to define correspondence between two examples for the morphing to take place. There are M(M-1)/2 possible combinations when there are M examples to be morphed. This number can be prohibiting when those correspondences are to be defined manually and the number of examples are many. (Automatic alignment and matching is one alternative. It will be discussed elsewhere.) In this document, anchoring points are defined as defined solely from each example itself. It also will be defined as a bundle of features to provide sufficient information to define specific correspondence easily.

3.3.1 Where to set anchoring points

Anchoring points should be representative points to describe speech. It is reasonable to use segmental structure which is an essential aspect of speech sounds. Speech sounds have segmental

structure at several different levels; sentence, phrase, clause, word, syllable and phoneme and so on. It is convenient to use label information which is usually available in speech databases. These boundaries of linguistic element are the first candidate of anchoring points, because the primary role of anchoring point is temporal alignment of two example speech.

However, these linguistic elements are not detailed enough for making morphing possible. Spectral structure changes within a linguistic segment. Most commonly found anchoring point is the boundary of spectral transition between consonants and vowels.

Moreover, it can be better to set anchoring point in the mid point of vowels. It may be relevant to set vowel centroid using label information. Or, it may be more relevant to search the most stable point using labeling information to set search range.

3.3.2 Required attributes for anchoring points

The secondary role of anchoring points is frequency alignment. Mismatch in peak locations introduces severe degradation when morphing rate is 0.5 and should be avoided. Mismatch in dip locations may be negligible because of their relatively low power. However it should be noted that zeros around 5kHz may represents individuality, according to some researchers.

It is strongly advised to have phonological knowledge to select proper peaks for anchoring points. The anchoring points are better to be set at formant locations, even if they are not salient enough. On the contrary, they are better not to be set at peaks not corresponding to formant locations. (It is not very likely those non-formant peaks to have counter parts in other example speech.) If labeling information is available, It can be a better alternative to deform phoneme prototype to get frequency anchor points. Anchor point setting not based on solid acoustic-phonetics may be rather harmful than helpful.

Please note that this does not prohibit to set anchoring points by comparing two examples. When it is easier and helpful, please do it. The other purpose of introducing M-object and defining anchor points as an attribute of a M-object is to eliminate error prone process in the previous morphing procedure. (In the old procedure, a user has to keep correspondence information in mind and has to write mapping definition table directly. It was cognitively heavy load.)

3.3.3 Example: anchor selection

This section illustrates how to set anchor points using a real example. A built in Matlab function `ginput` is employed for interactive selection. An object information display function `displayMobject` is designed to assist this interactive anchor point setting. The following command displays a STRAIGHT spectrogram of a M-object. (The speech material is the same one that was used in ICASSP'2003 paper.)

```
displayMobject(angryHai,'spectrogram','angryHai');
```

The next step is to set an appropriate view referring information displayed on the title line of the display. Then, by invoking function `ginput` the initial part of the interactive anchor point selection is initiated.

```
axis([0 300 0 7000])
rawanch  = ginput
```

The information display with cross hair cursor is shown in the following figure. You can set anchor points by clicking mouse button when cross hair cursor is located at each proper point. Please continue this operation while candidate anchor points are left unmarked. This operation is terminated when a key on the keyboard is pressed. This termination operation gives the following output for example.

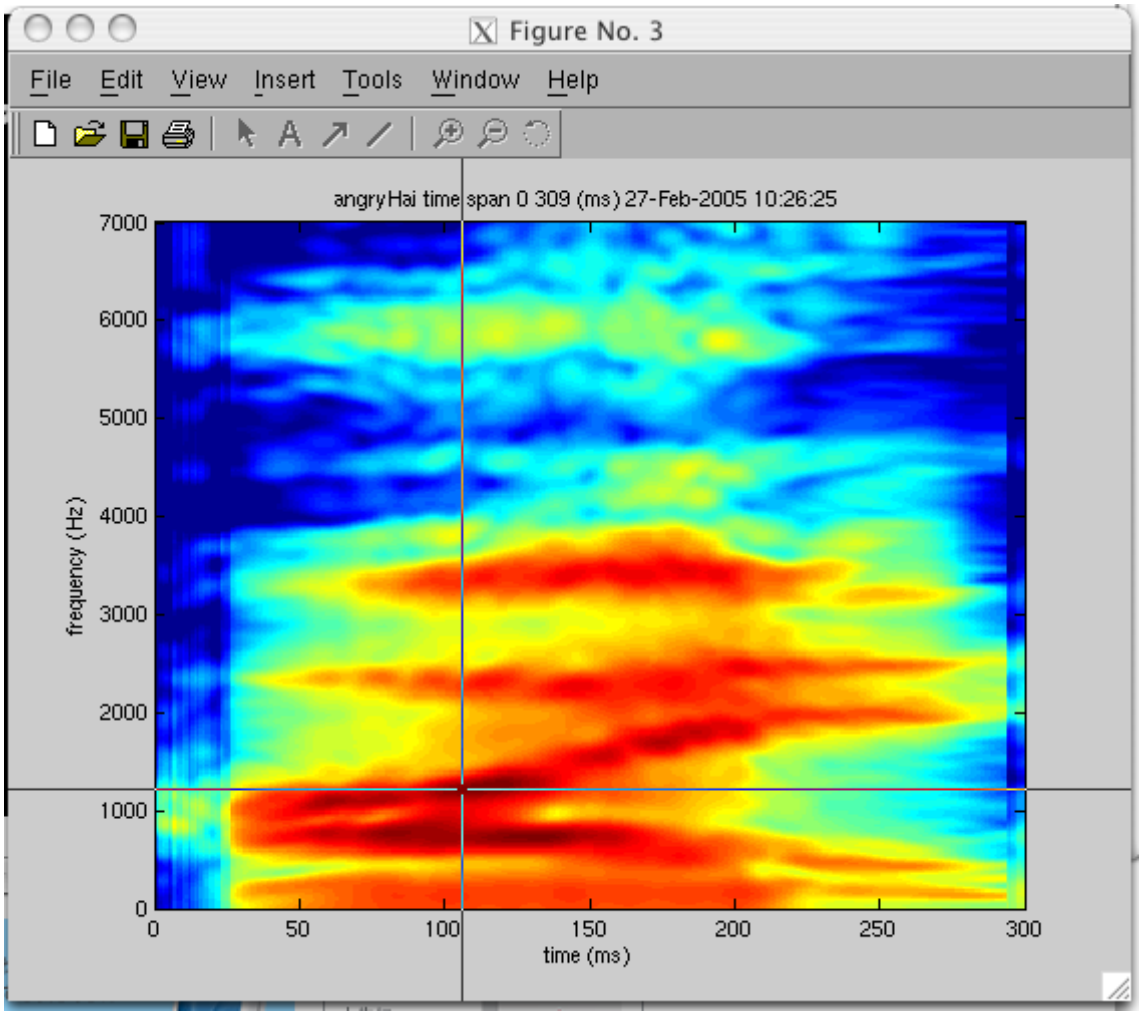

Figure 1: STRAIGHT spectrogram with cross hair cursor.

```
>> rawanch = ginput

rawanch =

   1.0e+03 *

    0.0266   -0.5424
    0.2934   -0.6243
    0.0916    0.7471
    0.0923    1.1360
    0.0916    2.3026
    0.0923    3.2851
    0.0916    5.8845
    0.1621    0.7061
    0.1635    1.5658
    0.1621    2.2617
    0.1628    3.4079
    0.1635    5.8231
    0.1994    0.6038
    0.1987    1.7909
    0.1994    2.4868
    0.1987    3.3874
    0.2008    5.7822
    0.2499    0.4196
    0.2506    1.9751
    0.2492    2.4664
    0.2506    3.2237
```

The left column represents time (in ms) and the right column represents frequency (in Hz). Boundaries of speech segments are marked by clicking outside of frequency range (negative frequency).

Assigning these values to one variable, anchor points are aligned by sorting using time information as the key. The following commands do those steps.

```
[dm1,indsrt] = sort(rawanch(:,1));
rawanch(indsrt,:)

ans =

   1.0e+03 *

    0.0266   -0.5424
    0.0916    2.3026
    0.0916    5.8845
    0.0916    0.7471
    0.0923    1.1360
    0.0923    3.2851
    0.1621    0.7061
    0.1621    2.2617
    0.1628    3.4079
    0.1635    1.5658
    0.1635    5.8231
    0.1987    1.7909
    0.1987    3.3874
```

```
    0.1994    0.6038
    0.1994    2.4868
    0.2008    5.7822
    0.2492    2.4664
    0.2499    0.4196
    0.2506    1.9751
    0.2506    3.2237
    0.2934   -0.6243
```

By inspecting the results, temporal anchor points are found to be set 27ms, 92ms, 162ms, 199ms, 250ms, 293ms. Frequency anchor points have to be assigned to these temporal locations.

As indicated in this example, usually more than two frequency anchor points share the same time location. However, this process is error prone. A post processing function `setAnchorFromRawAnchor` is provided to clean up manually set anchor points to meet this condition. The following commands processes the example given above using this post processing function. The function updates (in this case assigns) anchor point information to a M-object. The results can be checked by typing relevant field name with the M-object name.

```
 >> angryHai = setAnchorFromRawAnchor(angryHai,rawanch)
 >> angryHai.anchorTimeLocation

ans =

   26.6129
   91.8684
  162.7880
  199.4240
  250.0576
  293.4332

 >> angryHai.anchorFrequency

ans =

   1.0e+03 *

   -0.5424         0         0         0         0         0         0         0         0
    0.7471    1.1360    2.3026    3.2851    5.8845         0         0         0         0
    0.7061    1.5658    2.2617    3.4079    5.8231         0         0         0         0
    0.6038    1.7909    2.4868    3.3874    5.7822         0         0         0         0
    0.4196    1.9751    2.4664    3.2237         0         0         0         0         0
   -0.6243         0         0         0         0         0         0         0         0
```

These anchor information are also displayed using the display function for M-object.

```
displayMobject(angryHai,'anchorTimeLocation','angryHai');
axis([0 300 0 7000])
```

The above example is 'anger' speech. Let's perform the similar procedure on 'neutral' speech. It yields the following.

```
>> neutralHai.anchorTimeLocation

ans =
```

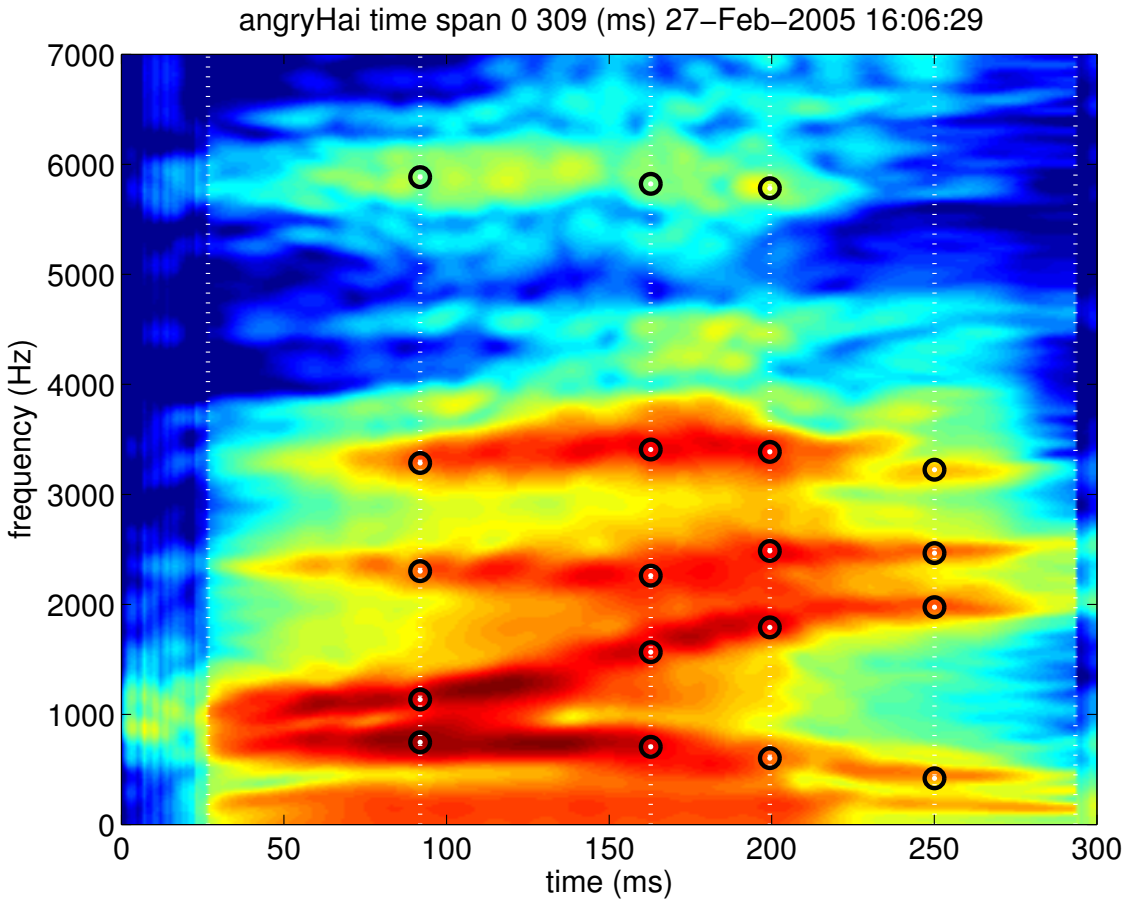

Figure 2: STRAIGHT spectrogram with anchor points. (Anger)

```
   53.2834
   84.8502
  132.6613
  166.8779
  209.2921
  239.9194

>> neutralHai.anchorFrequency

ans =

   1.0e+03 *

   -0.6038         0         0         0         0         0         0         0         0
    0.7471    1.3202    2.4050    3.7558    5.8231         0         0         0         0
    0.5219    1.8728    2.3845    3.8787    5.8231         0         0         0         0
    0.3787    2.1184    3.2032    3.9196    5.7617         0         0         0         0
    0.3173    2.1594    3.2000    3.7968         0         0         0         0         0
   -0.5219         0         0         0         0         0         0         0         0

>> displayMobject(neutralHai,'anchorTimeLocation','neutralHai');
>> axis([0 250 0 7000])
```

3.4 Executing morphing

Morphing is an operation to generate a M-object from given M-object(s) using predefined measure. There are several variations as shown below.

- Using two M-objects: This morphing is similar to the previous morphing.
- Using one M-object and target emotion: This morphing is for real time voice morphing.
- Using three M-objects: This morphing is designed to map a person's emotional expression to.

Other variations of morphing that is other than these also will be discussed in other sections.

3.5 Synthesizing morphed speech and file writing

It is generally useful to provide a function to synthesize speech from a M-object, because morphing result is also a M-object. The synthesis function `executeSTRAIGHTsynthesisM` is invoked using the following commands.

```
syang = executeSTRAIGHTsynthesisM(angryHai);
```

Optional parameters to control synthesis function can be modified assigning proper value to a specific field of a control structure parameter. The structure and fields are described in minimumSTRAIGHT document. One useful parameter is one that control energy normalization. The default is to normalize synthesized sound at -22 dB below clipping level. Disabling this normalization is desirable when morphing emotional speech, where intensity itself has an important role. The following commands disable normalization.

```
prminSyn.levelNormalizationIndicator = 0;
syang = executeSTRAIGHTsynthesisM(angryHai,prminSyn);
```

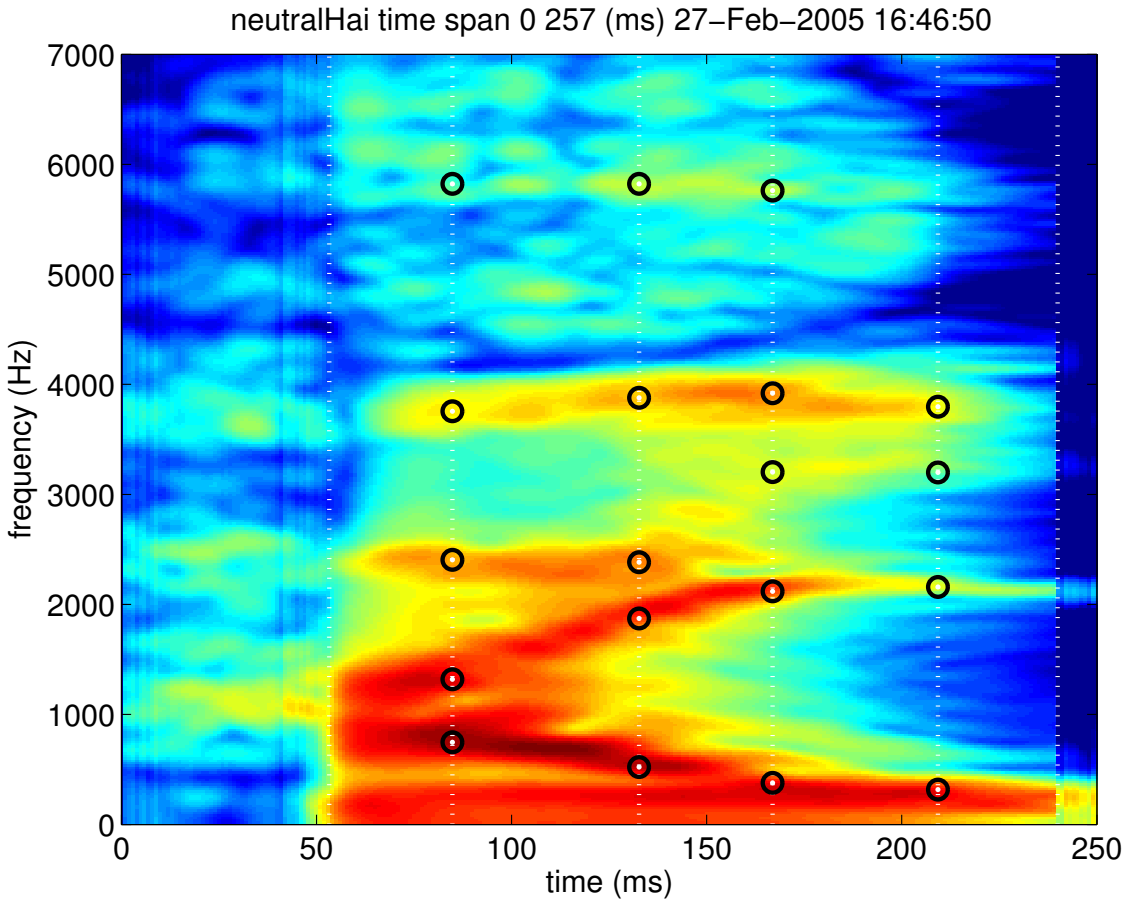

Figure 3: STRAIGHT spectrogram with anchoring points. (neutral)

Please use `wavwrite` (Matlab function) or `aiffwrite` (minimum STRAIGHT) to write synthesized speech into a file. Please keep no clipping to happen.

4 Step by step implementation of morphing function: I

In this section, the morphing procedure introduced in ICASSP'03 is implemented in a step by step manner starting from a simple waveform summation.

4.1 Morphing without side information

It is worthwhile to check that simple waveform superposition does not perform desired morphing function. A superposition function `waveformMorphing` is designed to mix waveform information of two M-objects according to the given amount r. In this case, the waveform of the first M-object is weighted by (1-r) and that of the second M-object is weighted by r and then added together. The following commands does this operation.

```
mixedWave = waveformMorphing(neutralHai,angryHai,0.5);
displayMobject(mixedWave,'waveform','mixedWave');
```

Please note that the output of `waveformMorphing` is also a M-object to maintain compatibility. The morphed signal by this procedure sounds like two persons are speaking at the same time. This is not the desired behavior of auditory morphing.

4.2 Morphing in STRAIGHT parameter domain without side information

The second step is more realistic. Morphing is implemented by interpolation in the STRAIGHT parameter domain. No time alignment is take place in this case. Simple interpolation between corresponding parameters is taken place according to morphing ratio. This morphing is implemented as a function `directSTRAIGHTmorphing`. The first parameter and the second parameter represents M-objects to be morphed. The third parameter represents the morphing rate that is amount of the second component to be mixed (sum of this rate and the amount of the first component is constrained to be one). The last component defines mixing axis. Currently `linear` and `log` are provided.

```
mObject3 = directSTRAIGHTmorphing(neutralHai,angryHai,0.5,'linear');
```

Anomalies develop such as doubled formant trajectories and double plosive explosions, because no time and frequency alignment is introduced. The following figure shows direct interpolation in STRAIGHT spectrum domain. The rate is 0.5. Fundamental frequency is also interpolated on log-frequency axis. Fundamental frequency of voiced counterpart is used as the mixed fundamental frequency when one of example F0s at an instant is classified as unvoiced. The mixed fundamental frequency trajectory is shown below as a solid blue line. Red line and green line represents those of given examples. The resulted STRAIGHT spectrum does not corresponds to any natural sounds. The F0 trajectory is also unnatural. However, the resynthesized speech sound does not sound awfully unnatural. It is an interesting perceptual result. It may indicate that concept of formant as a resonance of a vocal tract does not directly corresponding to perceptual attribute. It can be likely that formant as a representative statistical value of energy aggregation in the frequency domain is more directly related to our perception.

STRAIGHT spectrogram can also be mixed on logarithmic axis. The following command does it.

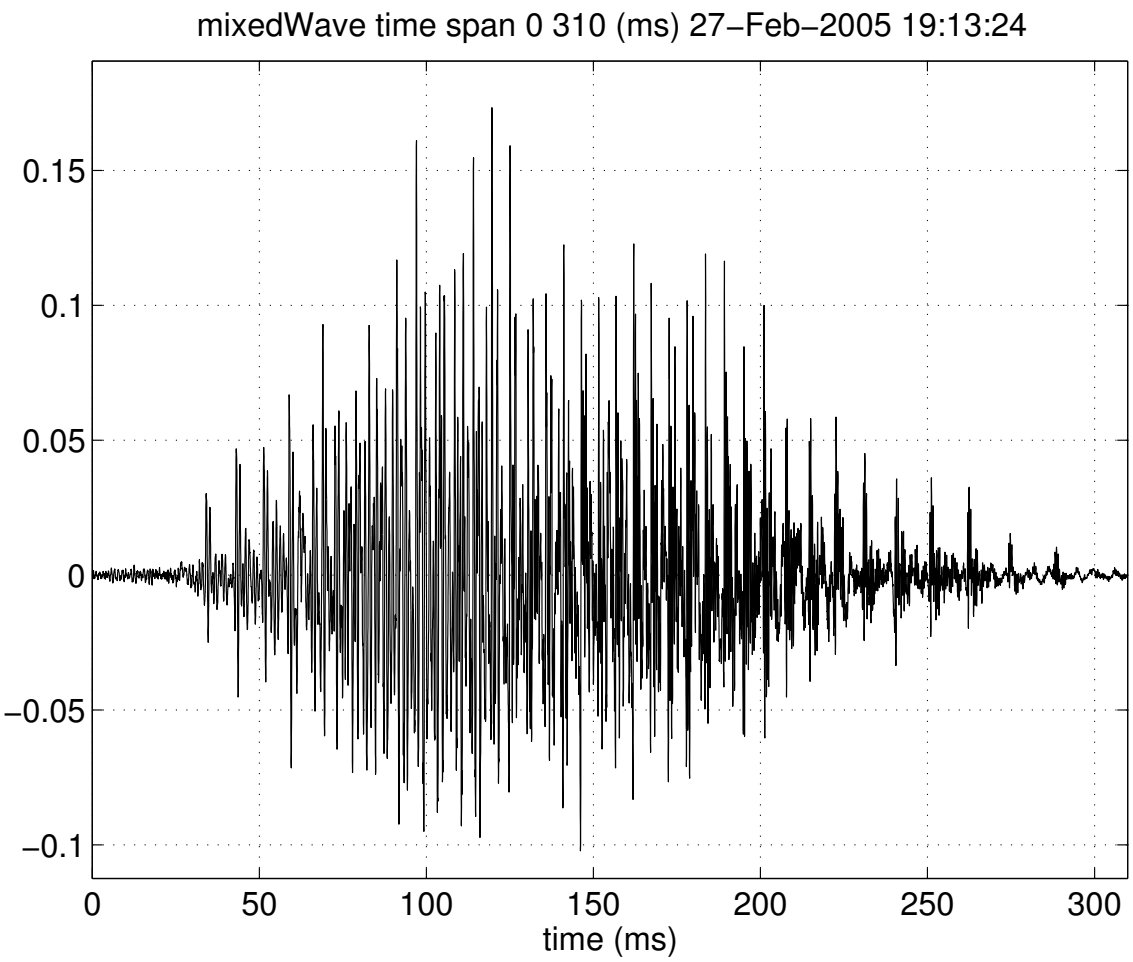

Figure 4: synthesized waveform

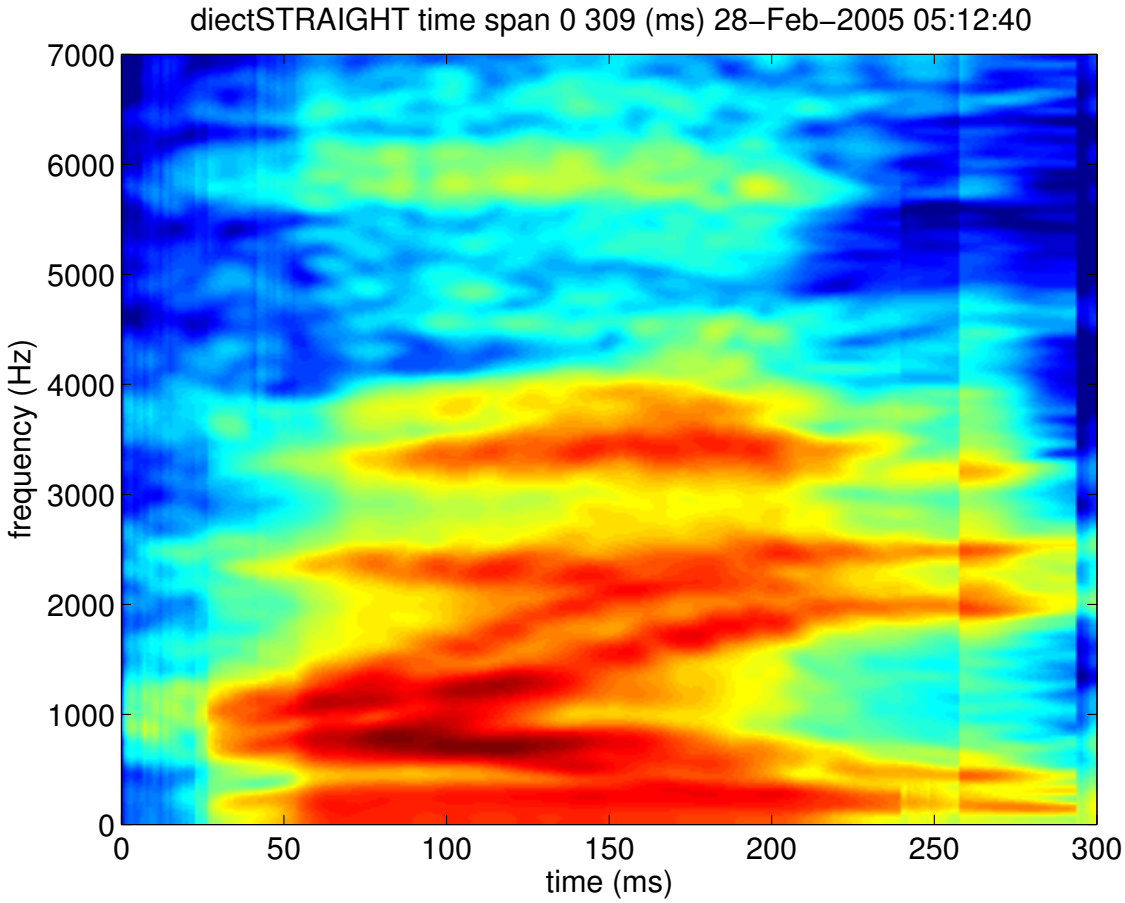

Figure 5: Directly mixed STRAIGHT spectrogram (linear interpolation)

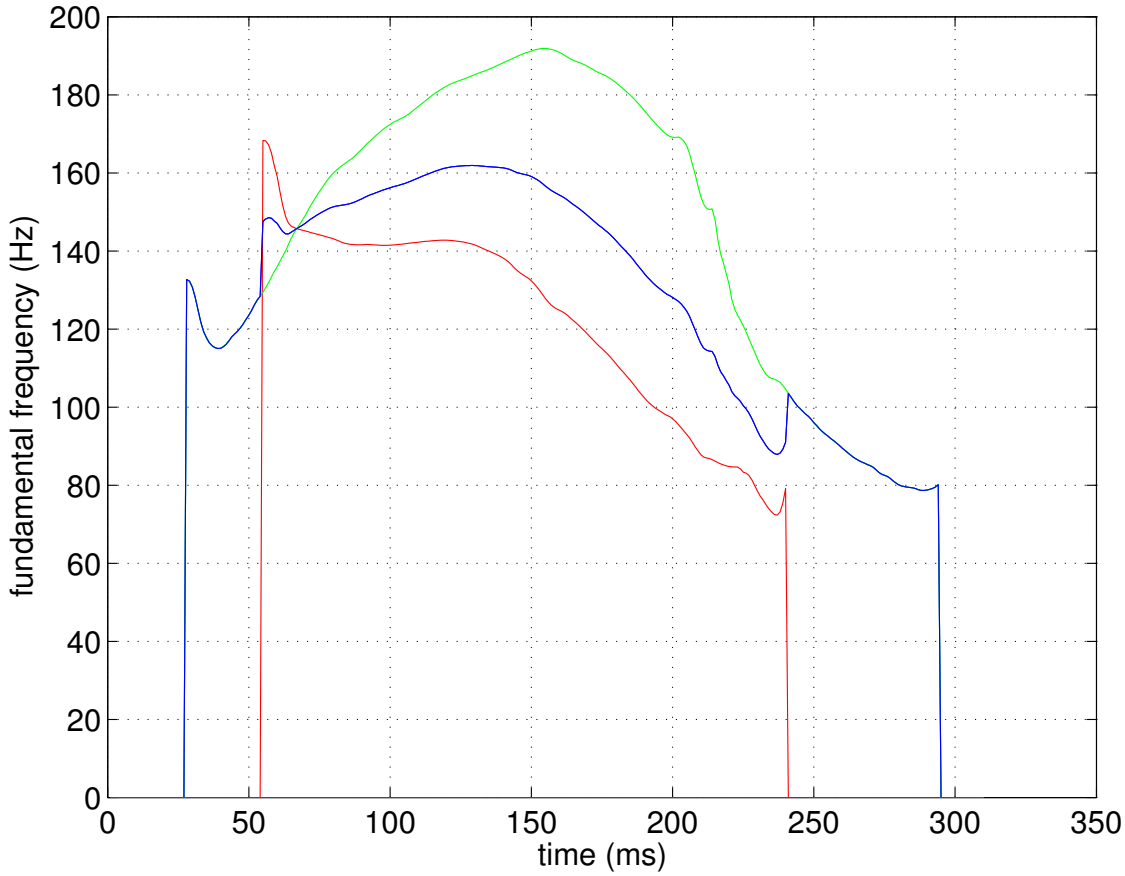

Figure 6: Fundamental frequency trajectories including mixed one on the log-frequency axis

```
>> mObject3 = directSTRAIGHTmorphing(neutralHai,angryHai,0.5,'log');
>> syneu = executeSTRAIGHTsynthesisM(mObject3);
>> wavwrite(syneu/32768,fs,16,'mixSTRAIGHTlog.wav');
```

This mixing is spectral multiplication and shapes of low energy region dominates the shape. The following figure shows the result of logarithmic mixing. Resynthesized speech using this morph-

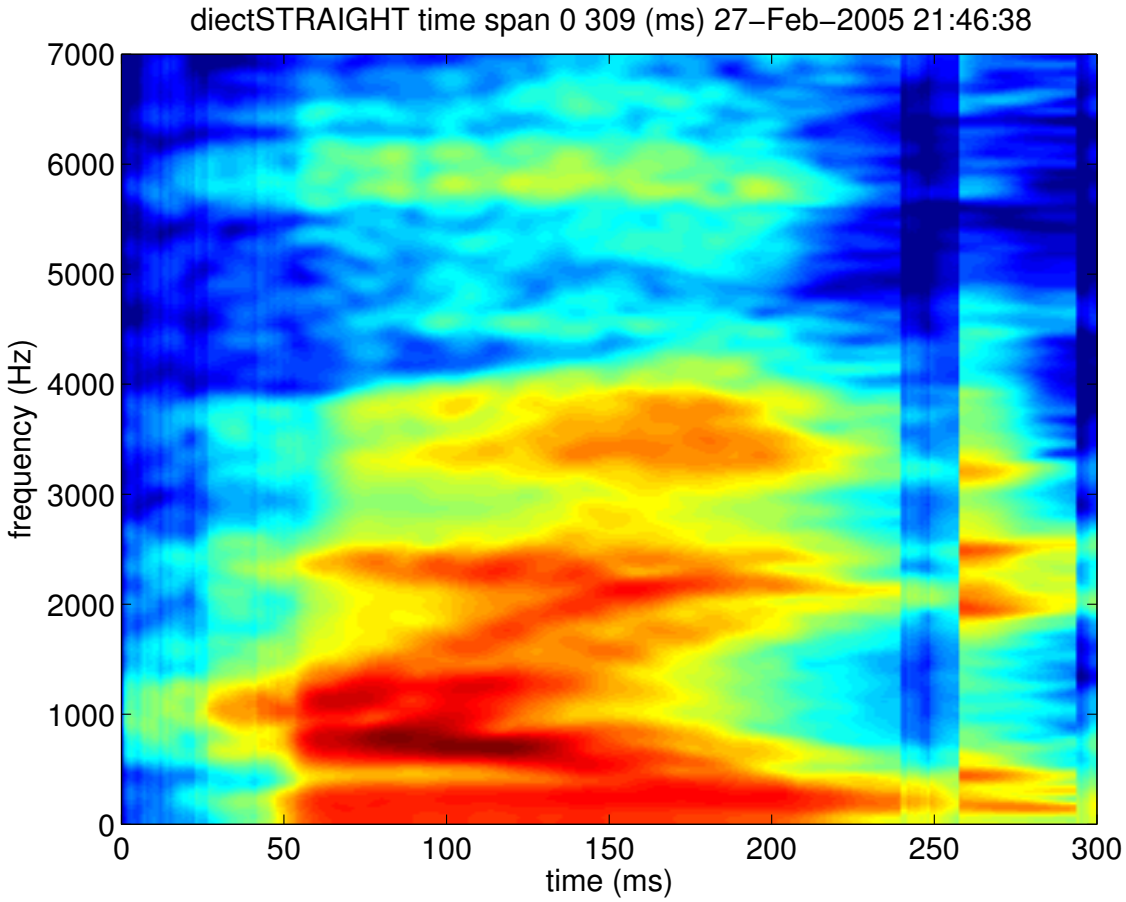

Figure 7: Directly mixed STRAIGHT spectrogram (logarithmic interpolation)

ingsounds to have excessive syllable at the final part.

4.3 Morphing with STRAIGHT parameter and location information

The next extension is time alignment. Frequency axis alignment is not introduced at this stage. It is interesting to observe that synthesized sounds sound reasonably natural even with this incomplete morphing. The morphing function at this stage is implemented as `timeAlignedDirectSTRAIGHTmorphing`. The first parameter and the second parameter represents M-objects to be morphed. The third parameter represents the morphing rate that is amount of the second component to be mixed (sum of this rate and the amount of the first component is constrained to be one). The last component defines mixing axis. Currently `linear` and `log` are provided.

```
mObjectdmy = timeAlignedDirectSTRAIGHTmorphing(neutralHai,angryHai,0.5,'log');
```

The morphed time axis by this procedure is similar to that produced by the final form. A piecewise linear interpolation is used to implement mapping between two M-objects' time axes using location information of anchor points. The following plot shows F0 trajectories for different morphing rates (red line: 0, blue line: 0.5 and green line: 1).

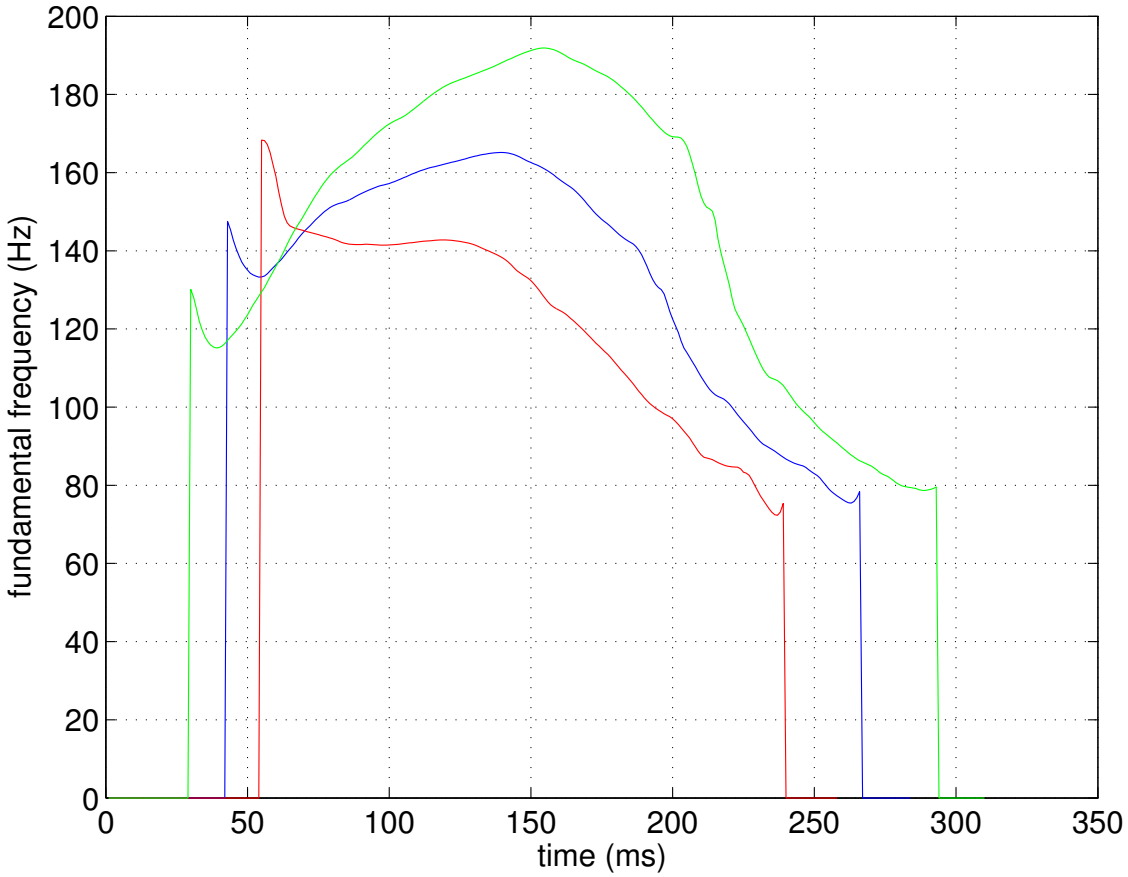

Figure 8: Fundamental frequency trajectories interpolated by piecewise linear functions. Color codes for morphing rates are red: 0, blue:0.5 and green:1.

The following figure shows morphed STRAIGHT spectrum. Formant trajectories look closer to that of natural ones than the previous morphing function without time alignment. Similarly, the M-object resulted by this morphing can be used to generate morphed speech sound. The morphed speech sound by time aligned STRAIGHT spectrogramsounds closer to the natural ones than the previous examples without time alignment.

However, when linear interpolation is used, the morphed STRAIGHT spectrogram still shows doubled formant trajectories. The linearly morphed speechalso sounds more natural than that made without time alignment.

This morphing interpolates anchor points based on their correspondence. This interpolation results in moving of those points. The following commands display relocated anchor points on morphed STRAIGHT spectrogram.

```
displayMobject(mObjectdmy,'anchorTimeLocation','alignedLinSum');
axis([0 283 0 7000])
```

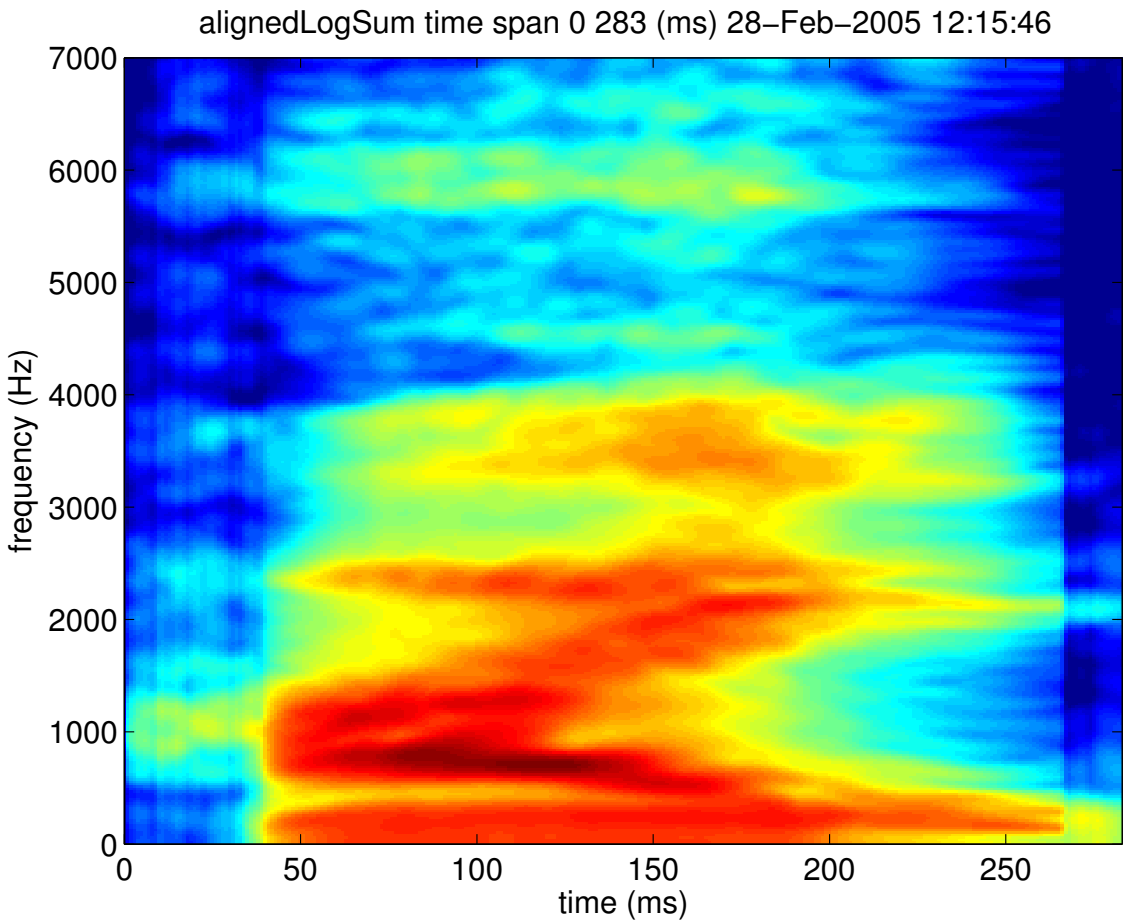

Figure 9: Morphed STRAIGHT spectrum with time alignment only. (logarithmic interpolation)

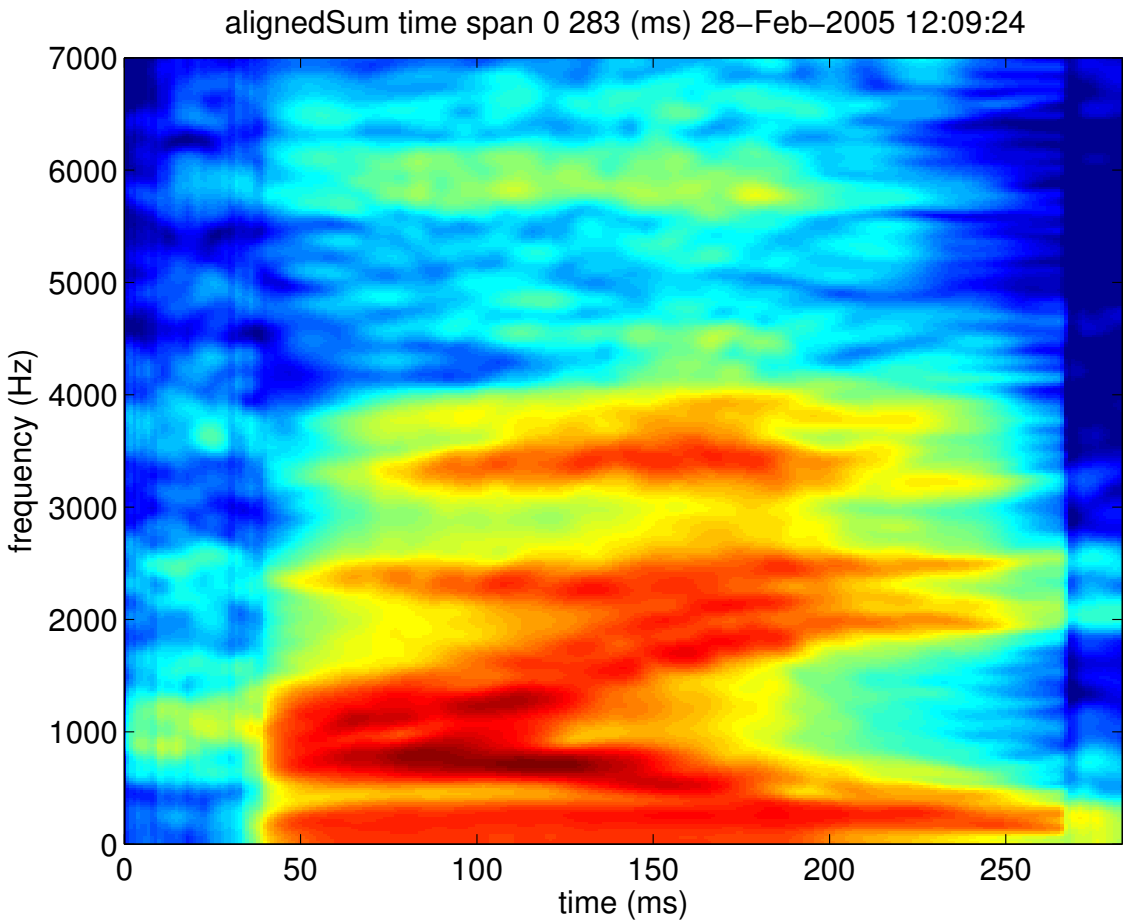

Figure 10: Morphed STRAIGHT spectrum with time alignment only. (linear interpolation)

This interpolation is implemented using linear interpolation. (It may be better to use ERB axis for interpolation when differences between anchor point frequencies are large.) The following figure shows relocated anchor points on morphed STRAIGHT spectrogram. Note that these moved anchor points are not suitably corresponding to characteristic spectral features on the morphed STRAIGHT spectrogram.

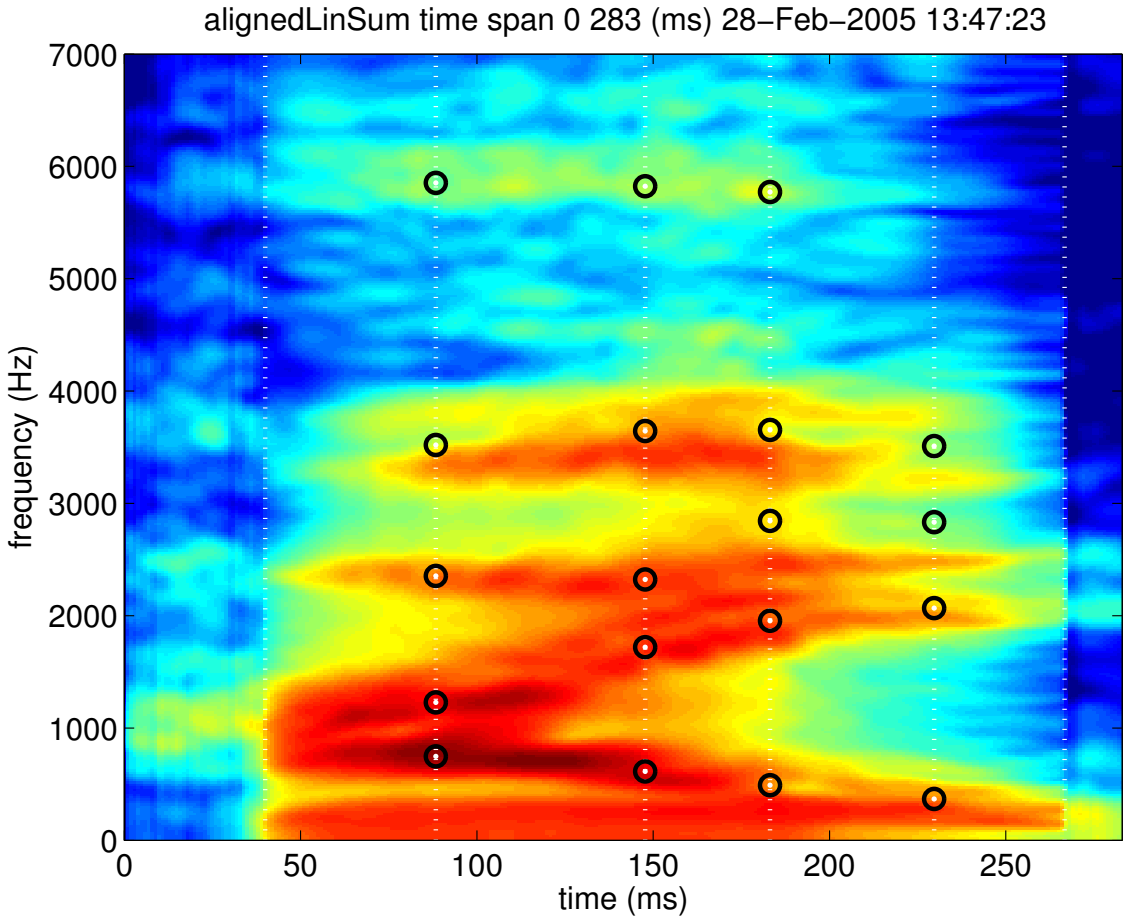

Figure 11: Linearly interpolated anchor points and morphed STRAIGHT spectrogram only with time alignment. (linear level interpolation.)

4.4 Morphing with STRAIGHT parameters and all anchor information

Extending the final example by adding frequency alignment yields the morphing procedure presented at ICASSP2003. One important difference in this implementation is more structured code that is (hopefully) easy to understand and to extend. This function is implemented as `timeFrequencySTRAIGHTmorphing`. The meaning of arguments are the same as the previous section. The following commands perform morphing on logarithmic level morphing with morphing rate 0.5. The commands also display morphed STRAIGHT spectrogram with morphed anchor points and then synthesize the morphed speech sound.

```
mObjectdmy = timeFrequencySTRAIGHTmorphing(neutralHai,angryHai,0.5,'log');
```

```
displayMobject(mObjectdmy,'anchorTimeLocation','Log');
axis([0 283 0 7000])
syneu = executeSTRAIGHTsynthesisM(mObjectdmy);
```

The following figure shows the morphed STRAIGHT spectrogram and anchor points. Note that the interpolated anchor points are suitably corresponding to characteristic points on morphed STRAIGHT spectrogram. These procedures are functionally equivalent to the morphing procedure presented in the Proceedings of ICASSP2003.

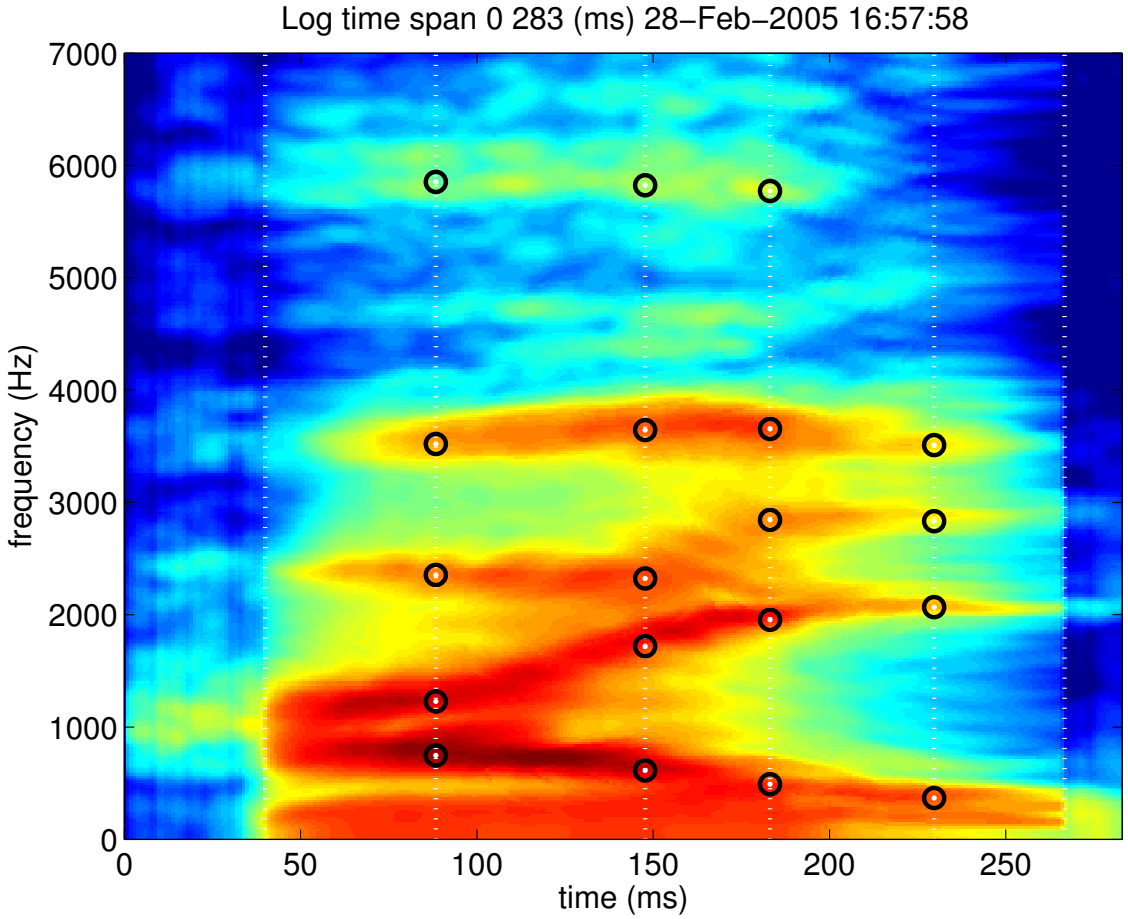

Figure 12: Linearly interpolated anchor points and morphed STRAIGHT spectrogram with time-frequency alignment. (logarithmic level interpolation.)

The following figure shows the morphed STRAIGHT spectrogram with linear level interpolation.

This concludes the step-by-step introduction of the morphing procedure presented in our ICASSP2003 paper. This implementation using M-object as the information representation is designed to introduce flexibility and simplicity to our original complicated procedures. (Actually, it was completely rewrote.)

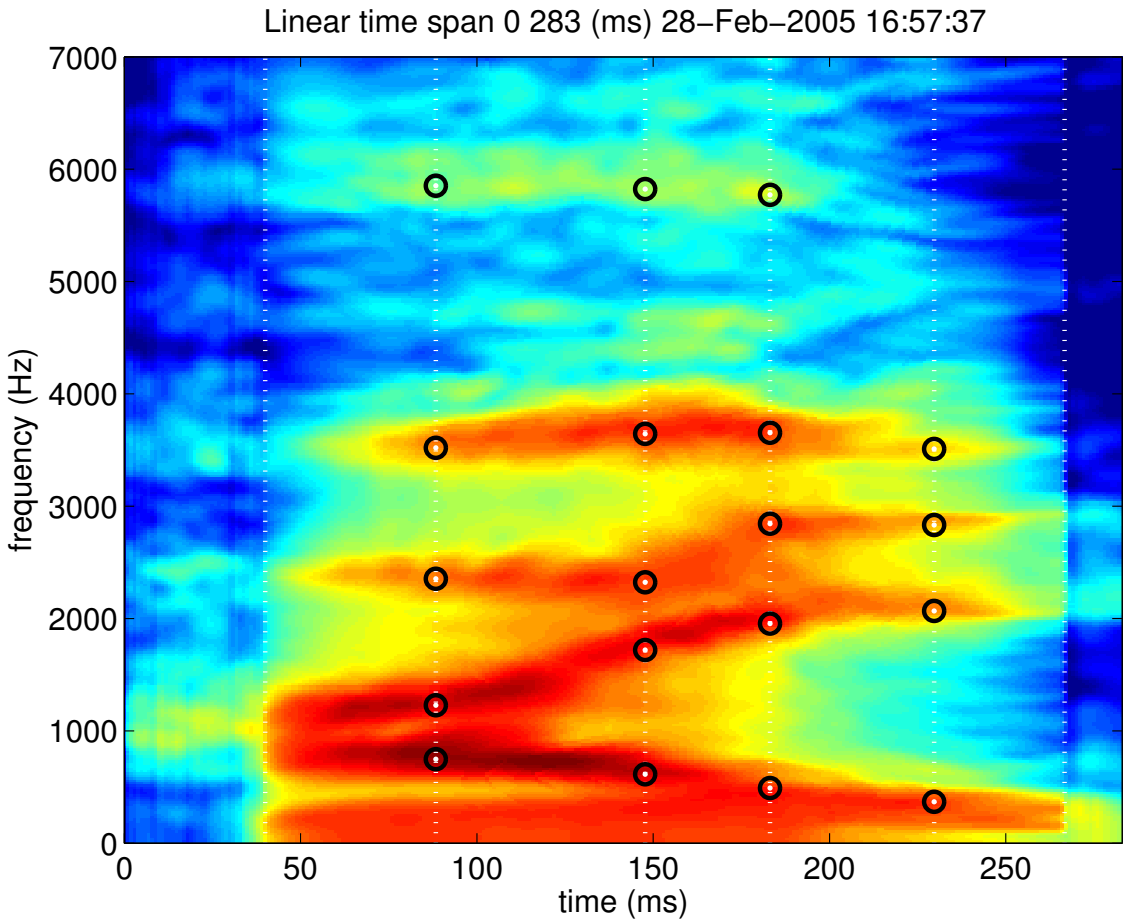

Figure 13: Linearly interpolated anchor points and morphed STRAIGHT spectrogram with time-frequency alignment. (linear level interpolation.)

5 Implementation of morphing without natural target

6 Implementation of mapping other speaker's expression

7 Applications of morphing

References

[1] Hideki Kawahara, Ikuyo Masuda-Katsuse, and Alain de Cheveigné. Restructuring speech representations using a pitch-adaptive time-frequency smoothing and an instantaneous-frequency-based F0 extraction. *Speech Communication*, Vol. 27, No. 3-4, pp. 187–207, 1999.

[2] Hideki Kawahara and Hisami Matsui. Auditory morphing based on an elastic perceptual distance metric in an interference-free time-frequency representation. In *Proc. 2003 IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP 2003)*, Vol. I, pp. 256–259, Hong Kong, 2003.

[3] Hisami Matsui and Hideki Kawahara. Investigation of emotionally morphed speech perception and its structure using a high quality speech manipulation system. In *Eurospeech'03*, pp. 2113–2116, Geneva, 2003.

Contents