Java Visual Speech Components for Rapid Application Development of GUI based Speech Processing Applications

Stefan Steidl[1,2], Korbinian Riedhammer[2], Tobias Bocklet[2], Florian Hönig[2], and Elmar Nöth[2]

[1]International Computer Science Institute (ICSI) at Berkeley, CA, U. S. A.
[2]Department of Computer Science, University of Erlangen-Nuremberg, Germany

`stefan.steidl@informatik.uni-erlangen.de`

Abstract

In this paper, we describe a new Java framework for an easy and efficient way of developing new GUI based speech processing applications. Standard components are provided to display the speech signal, the power plot, and the spectrogram. Furthermore, a component to create a new transcription and to display and manipulate an existing transcription is provided, as well as a component to display and manually correct external pitch values. These Swing components can be easily embedded into own Java programs. They can be synchronized to display the same region of the speech file. The object-oriented design provides base classes for rapid development of own components.

Index Terms: Speech Processing Applications, Java, Graphical User Interfaces (GUI), Rapid Application Development (RAD)

1. Introduction

Speech processing is a very large area of research and the number of tools that are used in this community is tremendously high. Tools are needed to play back audio signals, to visualize them, for spectral analysis, for transliteration and various annotations, for acoustic feature calculation, and many more purposes. Some tools are needed for manual user interaction, some for batch processing in order to apply the same processing steps to a large number of files. There are some standard tools for certain applications, e. g., the cross-platform tools *Wavesurfer*[1] for audio playback and analysis and *Praat*[2], which is widely used for calculation of acoustic features. Both have graphical user interfaces (GUI). A batch processing tool for acoustic features extraction is *openSMILE*[3] [1].

Often, standard tools are used if no specialized tools are available. For transliteration, e. g., in principle a simple text editor and an audio playback tool are sufficient. Certainly, this is not an efficient way of processing large amounts of (small) audio files since each file has to be loaded separately. It would be much more efficient to load a list of files and to be able to jump from one file to the next one having the application take care of loading and saving the required files. In general, these tools provide a graphical user interface (GUI) that can be controlled by the mouse. Mouse control is certainly a very intuitive way of controlling an application, but it causes wrist strain much faster than keyboard control, which in most cases is even faster than mouse control.

[1]`http://sourceforge.net/projects/wavesurfer/`
[2]`http://www.fon.hum.uva.nl/praat/`
[3]`http://opensmile.sourceforge.net/`

Annotation of naturally occurring emotions for vocal emotion recognition is one example where a lot of specialized GUI based applications are needed. There are many ways how emotions can be labeled. Emotions can be labeled with categorical labels. A large list of possible categories exist. Which categories should be used depends on the emotional states that actually appear in a given scenario. Emotions can be labeled in a dimensional space, too. Different numbers and types of dimensions have been proposed. Labels can be assigned to the whole segment assuming that the emotion is constant within this segment. Segments can be turns in a human-human or human-machine dialog, utterances, chunks, or words (s. [2]). Segments can be stored in separate audio files or an audio file can contain more than one segment. If the emotional state changes within one segment, these changes have to be tracked. It is obvious that there is no tool that is suited for all of these annotation possibilities. In most scenarios, specialized tools have to be designed. However, developing own software requires programming skills and – even more important – a considerable amount of time.

In this paper, we present a framework for rapid development of own GUI based speech processing applications: the Java Visual Speech Components. For this purpose, the proposed framework provides Java classes for various tasks, e. g. loading and displaying the speech signal, showing a spectrogram, etc. Furthermore, its object-oriented design and the existence of appropriate base classes allow for a rapid and easy development of own visual components.

2. *Java Speech Toolkit*

The Java Visual Speech Components are part of the Java Visual Speech Toolkit (JSTK) and use, for example, the JSTK classes to read an audio file. The JSTK is a Java framework for automatic speech recognition. It provides classes for feature extraction as well as a speech recognition system with classes for training a speech recognizer. Currently, the development of the ASR decoder is in progress. The Java Speech Toolkit including the Java Visual Speech Components and the JSTK Transcriber application described in this paper is available under the GNU General Public License version 3. The project website is `http://code.google.com/p/jstk/`.

3. *Java Visual Speech Components*

3.1. Object-oriented design

The design of the available Java Visual Speech Components follows an object-oriented approach, where the base classes `VisualComponent` and `FileVisualizer` contain all the necessary attributes and methods that are in general needed by

all visual components. `VisualComponent` is derived directly from `javax.swing.JComponent` and has the following features:

- It has the ability to display a coordinate system. The range of values on the x and y axis can be customized as well as the size of the borders and various color settings.

- The class offers double buffering to avoid flickering when displaying information. Another buffer is used to draw a horizontal and/or a vertical line at the position of the mouse cursor. The size of the buffered images are adjusted after resize events.

- Objects implementing the `MouseMotionVisualizationListener` interface can be registered, which are informed about movements of the mouse.

- Methods are provided to convert x and y coordinates into pixel coordinates and vice versa.

Components that are derived from `VisualComponent` directly – for examples components to show the spectrum or the autocorrelation function – display information of one frame (or a small number of frames) independently of other components.

The class `FileVisualizer` is derived from `VisualComponent` and is intended to display more information than just one frame, for example whole audio files. The following additional features are offered:

- The component supports the use of a `JScrollBar` in order to control the visible region of the audio file.

- It allows to zoom into and out of the signal.

- Components derived from `FileVisualizer` can communicate with each other in order to synchronize the visualization.

- Two ways of highlighting regions of the signal are provided: (1) The user can select a region by dragging the mouse. (2) A region can be highlighted by calling the appropriate method with a given start and end sample. The latter way can be used, for example, to walk through a transcription and to highlight the current word. Both features can be used independently of each other and color settings can be specified separately.

- Objects implementing the `SignalSectionSelectedListener` interface can be registered and are then informed if regions are selected by mouse dragging.

- Listeners can be registered to be able to react to the selection of single sample values (`SampleSelectedListener` interface).

- The component can react differently to mouse actions. Mouse clicks and mouse dragging can be used to zoom into and out of the signal in the `ZOOM_MODE`, they can be used to select regions in the `SELECTION_MODE`, or they can be ignored in the `NORMAL_MODE`.

Figure 1 shows the class hierarchy of these two base classes as well as the Java Visual Speech Components that are described in the following section.

3.2. Available components

At the moment, the following Java Visual Speech Components are available:

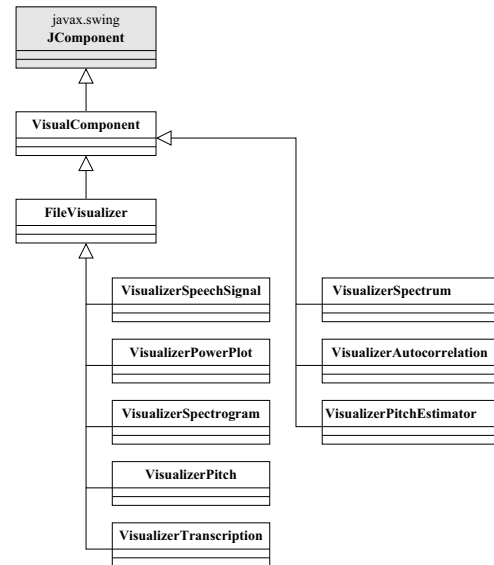**VisualizerSpeechSignal** displays the waveform of the speech signal.

Figure 1: Class hierarchy of the available Java Visual Speech Components

VisualizerPowerPlot displays the short-term root mean square energy (RMS) in decibel.

VisualizerSpectrogram shows the spectrogram of the signal on a gray scale or in pseudo-colors. Methods are provided to set the window function and the length of the window as well as to adjust the parameters for brightness and contrast.

VisualizerPitch is able to display external pitch values. Most often, errors of pitch extraction algorithms are either voiced/unvoiced errors or octave errors. The user can select pitch values using the mouse, which then can be set to zero, to the double or to the half of their value. Values between selected pitch values can also be interpolated using a spline interpolation. New values, which can be estimated with another component, can be assigned to single pitch values, too. The component displays both the original and the manually corrected pitch values.

VisualizerTransliteration is a component to create, to visualize, and to manually change the transcription of the speech signal as well as the segmentation, i.e. the start and the end times of the words. The user can jump from one word to the next or previous one and the current word is highlighted in all synchronized components.

VisualizerSpectrum shows the spectrum for a single frame. Again, the window function and the window length can be controlled using method calls.

VisualizerAutocorrelation displays the autocorrelation function for a given frame.

VisualizerPitchEstimator shows the magnified speech signal of the selected frame and, depending on the zoom factor and the width of the component, its adjacent neighboring frames. The user can define pitch periods using the mouse. The length of the periods is converted into their frequency values, which are then averaged over all periods. This average value can be assigned to the selected pitch value of the `VisualizerPitch` component.

3.3. Using available components

Listing 1 demonstrates how easily the existing Java Visual Speech Components can be used. First, an instance of class `AudioFileReader` is created in order to read the audio file. The first argument of the constructor is the file name, the second one is a boolean controlling whether a `BufferedInputStream` is used for reading the file or not. The `AudioFileReader` implements the `AudioSource` interface allowing to access the audio samples in sequential order. However, the visual components require to access the audio file in general more than once and in random order. Hence, the audio data is buffered using an instance of class `BufferedAudioSource`. Then, instances of the two classes `VisualizerSpeechSignal` and `VisualizerSpectrogram` are created. The buffered audio source is passed to both constructors as well as a String name for identification of the component, mainly used in the `toString` method of both classes. The dimensions of the visual components are set using the `setPreferredSize` method.

As in most cases only a part of the speech signal will be displayed, a JScrollBar is used to control the visible region. The scrollbar is attached to only one of the two Java Visual Speech Components. In order to synchronize the visualization of both components, each component is registered as `VisualizationListener` of the other one. Finally, all components are added to a `Box` which then can be added to a `JFrame`, for example.

Listing 1: Instantiation of two Java Visual Speech Components

```
1  Box box = Box.createVerticalBox();
2
3  AudioFileReader reader =
4      new AudioFileReader("file.wav", true);
5  BufferedAudioSource audiosource =
6      new BufferedAudioSource(reader);
7  VisualizerSpeechSignal signal =
8      new VisualizerSpeechSignal("signal",
9          audiosource);
10 VisualizerSpectrogram spectrogram =
11     new VisualizerSpectrogram("spectrogram",
12         audiosource);
13
14 signal.setPreferredSize(
15     new Dimension(400,100));
16 signal.showCursorY = false;
17 signal.switchMode(
18     FileVisualizer.SELECTION_MODE);
19 spectrogram.setPreferredSize(
20     new Dimension(400,158));
21
22 JScrollBar scrollbar = new JScrollBar(
23     JScrollBar.HORIZONTAL);
24 signal.setScrollbar(scrollbar);
25
26 signal.addVisualizationListener(spectrogram);
27 spectrogram.addVisualizationListener(signal);
28
29 box.add(signal);
30 box.add(Box.createVerticalStrut(10));
31 box.add(spectrogram);
32 box.add(scrollbar);
```

In order to synchronize n Java Visual Speech Components, $n(n-1)$ connections have to be established. To reduce this amount of connections, a `Visualiza-tionInformer` object can be used. Calling the `set-VisualizationInformer` method establishes a bidirectional connection between the visual component and the `VisualizationInformer` object, which forwards all visualization update requests that it receives from one component to all other registered components. Listing 2 shows how lines 26 and 27 in Listing 1 can be replaced.

Listing 2: Synchronizing Java Visual Speech Components

```
1  VisualizationInformer informer =
2      new VisualizationInformer();
3  signal.setVisualizationInformer(informer);
4  spectrogram.setVisualizationInformer(informer);
```

3.4. Creating custom components

The presented framework should not only encourage programmers to write their own programs based on the existing Java Visual Speech Components, but also to create their own visual components. Listing 3 demonstrates how easy new components can be created by deriving them from the existing super classes `VisualComponent` and `FileVisualizer`. The class `MyVisualizer` in the given example inherits the ability to display a coordinate system. As customized in the constructor, values in the range from 0 to 100 are displayed on the y axis. The component also inherits the ability to communicate with other existing Java Visual Speech Components. It is a fully functional Java Visual Speech Component, only the ability to display the intrinsic information is still missing. For this purpose, the components inherits double buffering and methods to convert samples and y values into pixel coordinates and vice versa. In order to display information, the method `drawSignal` has to be overridden. This method is always called if a repaint event is triggered. If time consuming computations are required to display the information, these computations can be done in the method `recalculate`, which is only called if resize event occurs.

Listing 3: Creating your own Java Visual Speech Components

```
1  import java.awt.Graphics;
2  import de.fau.cs.jstk.io.BufferedAudioSource;
3  import de.fau.cs.jstk.vc.FileVisualizer;
4
5  public class MyVisualizer
6      extends FileVisualizer {
7
8      public MyVisualizer(String name,
9              BufferedAudioSource audiosource) {
10         super(name, audiosource);
11         yMin = 0;
12         yMax = 100;
13         ytics = 25;
14     }
15
16     @Override
17     protected void recalculate() {
18     }
19
20     @Override
21     protected void drawSignal(Graphics g) {
22     }
23
24     @Override
25     public String toString() {
26         return "MyVisualizer '" + name + "'";
27     }
28 }
```
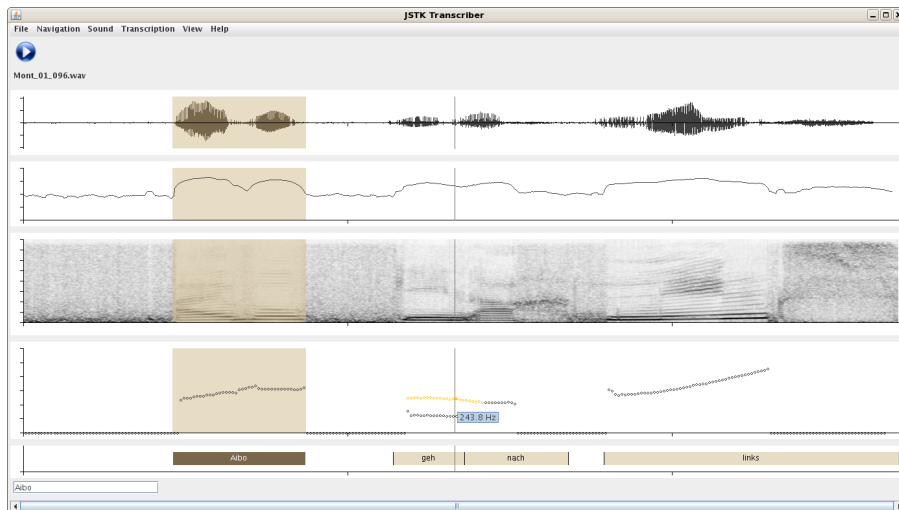
(a) Main application window
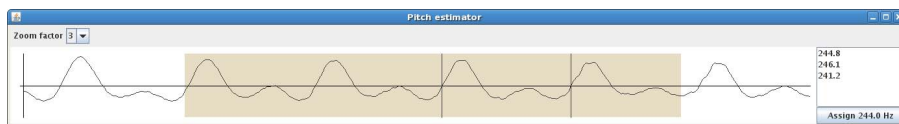
(b) Pitch estimation window

Figure 2: JSTK Transcriber is a fully functional tool for transliteration and manual pitch correction demonstrating the functionality of the available Java Visual Speech Components

4. Application *JSTK Transcriber*

The main purpose of this application is not to propose a new transcription tool, but to demonstrate the usability of the Java Visual Speech Components. However, the application is fully functional and well-suited for manual processing of large amounts of audio files. The user opens a transcription file, a text file that contains one line for each audio file. Each line contains the name of the audio file, followed by the transcription, which is a sequence of triples containing the word, the number of the first sample, and the number of the last sample of the word. As the transcription can be empty, the user can start with a simple list of audio file names.

Figure 2a shows the main window, which contains the components `VisualizerSpeechSignal`, `VisualizerPowerPlot`, `VisualizerSpectrogram`, `VisualizerPitch`, and `VisualizerTranscription`. The tool is suited to create own transcriptions with a word segmentation, or to display and manually correct a given transliteration obtained, e. g. by forced-alignment of the sequence of uttered words. Furthermore, the tool allows to display and manually correct external fundamental frequency values, which are read (automatically) from separate files. Both the original and manually corrected pitch values are shown. Thus, the tool can be used to study errors of pitch extraction algorithms, laryngealizations, regions where the fundamental frequency differs from the perceived pitch, etc. The tool can also be used for comparing two different pitch extraction algorithms. The spectrum and the autocorrelation function can be shown in separate windows (not shown in Figure 2). The tool can be used in teaching to study the effect of window type and window length on the spectrum and the autocorrelation function.

Figure 2b shows how the user can manually define pitch periods by mouse dragging. The new pitch value estimated over all manually defined pitch periods can be assigned to the current pitch value. An older version of this tool has been used for annotation of the FAU Aibo Emotion Corpus [3]. The word segmentation and the pitch values have been manually corrected [3, 4].

5. Conclusions

We proposed a new Java framework for rapid development of GUI based speech processing applications. The set of available visual components can be easily extended by writing own components that are derived from the given super classes and that inherit all the necessary functionality to communicate with already existing components. The application *JSTK Transcriber* is a fully functional tool for transliteration and manual pitch correction, which proves the functionality of the Java Visual Speech Components.

6. References

[1] Eyben, F. and Wöllmer, M. and Schuller, B., "openSMILE – The Munich Versatile and Fast Open-Source Audio Feature Extractor", Proc. ACM Multimedia 2010, pp. 1459-1462, 2010.

[2] Batliner, A. and Seppi, D. and Steidl, S. and Schuller, B., Segmenting into adequate units for automatic recognition of emotion-related episodes: a speech-based approach, Advances in Human-Computer Interaction, 2010

[3] Steidl, S., "Automatic Classification of Emotion-Related User States in Spontaneous Children's Speech", Logos Verlag, Berlin, 2009.

[4] Steidl, S. and Batliner, A. and Nöth, E. and Hornegger, J., Quantification of Segmentation and F0 Errors and Their Effect on Emotion Recognition, Proc. TSD 2008, pp. 525-534, 2008.