6th Congress of Alps-Adria Acoustics Assosiation
16.-17. October 2014
Graz, Austria

Austrian Acoustics Association
Österreichische Gesellschaft für Akustik

Alps Adria Acoustics Association

ISBN: 978-3-200-03921-6

CHRONOS-TRAINER: A RAPID PROTOTYPING FRAMEWORK FOR REALTIME AUDIO PATTERN RECOGNITION

Bernhard Rettenbacher, Moritz Fišer

JOANNEUM RESEARCH Forschungsgesellschaft mbH (bernhard.rettenbacher@joanneum.at)

Abstract: *We present the Chronos-Trainer Framework for developing audio pattern recognition systems with a focus on both offline prototyping and realtime execution. The framework covers feature extraction, labelling, classifier training, evaluation and execution. It is implemented as a hybrid Simulink - MATLAB - C++ - Java - Framework which enables seamless transformation of algorithms from MATLAB- and Simulink-prototypes to highly optimized binary code that can be integrated into online (realtime) and offline applications.*

Keywords: audio feature extraction, pattern recognition, annotation, realtime, MPEG-7, MATLAB, Simulink

1. INTRODUCTION

For rapid prototyping of audio pattern recognition algorithms MATLAB provides a convenient and efficient environment. Many signal processing and pattern recognition toolboxes are available and the rich scripting and visualisation environment makes it easy to interact with the analysis data.

In many signal processing and pattern recognition applications multiple algorithms have to be chained together. This creates a demand for frameworks that provide the "glue code" enabling data flow between the algorithms. In Speech Processing and Music Information Retrieval communities frameworks have been developed either directly in MATLAB or in other programming languages, mainly C++. MATLAB toolboxes are generally easy to use and to modify while C++ frameworks promise a more computationally efficient execution.

A popular MATLAB toolbox from the Music Information Retrieval (MIR) community is MIRtoolbox[1], which provides a wide range of feature extraction algorithms and includes additional toolboxes: the AuditoryToolbox[2] provides auditory features, NetlabToolbox[3] provides classifiers and SOMToolbox[4] is used for Self-Organizing-Maps. Major disadvantages of these MATLAB toolboxes are poor processing speed and difficulties when integrating toolbox functionality into a standalone application. A common solution is to reimplement the algorithm prototypes in a more efficient programming language like C++. This task often takes a significant amount of time and requires verfication of the ported algorithms to prove the equality of implementations.

For further development and especially for evaluation tasks it is advantageous to reintegrate these ported algorithms into MATLAB. Therefore additional effort has to be taken to create interfaces using MATLAB APIs or by file exchange. The practical need for this kind of integration can be seen when looking at actual Frameworks from the Music Information Retrieval and Speech Processing communities. Many of theses frameworks provide interfaces to scripting languages or other interactive environments.

MARSYAS[5] is implemented in C++ and uses the MATLAB Engine to make calls to the MATLAB environment. It can be also integrated with MAX/MSP[1] as a so called "external" or by using one of the SWIG[2] bindings for more than 20 programming languages. For data exchange with the WEKA[6] data mining software it supports the ARFF textual file format. openSMILE[7], a C++ framework focusing on feature extraction also supports the ARFF file format and in addition the file formats for the Hidden Markov Toolkit (HTK)[8], a speech recognition toolkit and LIBSVM[9], a Support Vector Machine library. Essentia[10], also written in C++ provides a Python binding and can be wrapped in a SonicVisualiser[11] VAMP Pugin for visualisation.

The availability and performance of easy-to-use native C++ frameworks highly support the reuse of existing algorithms

[1] http://cycling74.com
[2] http://www.swig.org

but that they appear to limit the motivation for development of new feature algorithms. This originates in the fact that people working on audio pattern recognition problems often are signal processing experts but have limited experience in advanced programming techniques like object oriented programming. To overcome this constraints we present a code-generation approach to feature development which is part of our internal framework called Chronos Trainer.

Chronos trainer is a hybrid Simulink - MATLAB - C++ - Java - Framework that almost eliminates the need for porting algorithms to a different programming language. Algorithms may be prototyped in MATLAB or Simulink and transformed automatically into highly optimized native binary code. The framework tightly integrates with an MPEG-7 Database for storing semantic annotations and a pattern recognition toolbox that enables classifier training within MATLAB and classifier execution within the binary code. To integrate with other frameworks, a wide range of file and API interfaces is provided.

2. OVERVIEW

Chronos-Trainer is a software framework supporting the development of audio pattern recognition systems. It covers all phases of development including audio data management, annotation, feature extraction, classifier training, evaluation and execution. The framework consists of

- a MATLAB Toolbox for interactively accessing audio data, annotating audio segments, extracting features, and training classifiers
- an MPEG-7 Database with MATLAB and Java APIs to manage audio file metadata and temporal semantic annotations
- a Simulink Blockset containing blocks for feature algorithms, classifier execution, signal managment, file- and network-IO
- a Simulink Code Generation Target for generating shared libraries

Chronos-Trainer uses additional tools and Toolboxes to complete its functionality. For classifier training and execution the perClass[3] library is used. This library consists of a MATLAB Toolbox for classifier training and a C-API for directly executing trained classifiers. For temporal annotation interfaces to WaveSurfer[12], SonicVisualiser[11] and Audacity[4] are available.

3. AUDIO DATA MANAGEMENT

In Chronos-Trainer, all audio data is managed by an MPEG-7 Database. Every audio file is accompanied with an MPEG-7 XML document that contains at least an ID and an URI to the audio file. The ID is used for referencing the audio file and its associated metadata throughout the framework. All operations on the MPEG-7 document as well as retrieving audio data can be made using an object oriented MATLAB interface. The interface also +provides methods for calling anotation tools directly. The interface transparently manages annotation file format conversion for the external tools so the user never needs to perform any manual export/import operations. When editing with the external tool is finished a database update with the changed annotations is triggered.

4. Annotation

We decided on MPEG-7 as the central metadata storage format because it combines technical metadata with document level descriptive metadata and content based temporal descriptions needed for audio annotation. In [13] we investigated whether MPEG-7 is usable for audio pattern recognition tasks in general with a focus on technical metadata like recording conditions and signal error description.

MPEG-7 is well suited for temporal descriptions. An audio file can be described as a sequence of *AudioSegment*s grouped in *TemporalDecomposition*s. Each AudioSegment may contain structured descriptions using free text, keywords, structured annotation or semantic descriptors. We decided to use the more complex semantic descriptors to formalize segment annotation and make the annotation accessible to semantic tools like semantic reasoners.

*AudioSegment*s are identified by a segment id. They contain a sample-based time code for starting time and duration. The segments may not have any temporal overlap but may be distributed over different *TemporalDecomposition*s. This structure maps to the segment layout of multitrack audio editors like Audacity and speech transcription tools like Wavesurfer.

To make the semantic description more human readable, easier to write and to make use of audio annotation tools from the Speech Processing domain we created a domain specific language named “AnnotationLine” that directly maps to MPEG-7 semantic descriptors. The language provides a syntax for key- value pairs representing semantic entities described by a label, its semantic type and a list of properties.

[3] http://perclass.com
[4] http://audacity.sourceforge.net

Here is an example of a MPEG-7 semantic description inside an *AudioSegment* which describes “a large black dog barking agressively”:

```
<Semantic>
  <SemanticBase xsi:type="ObjectType">
    <Label>
      <Name>dog</Name>
    </Label>
    <Property>
      <Name>large</Name>
    </Property>
    <Property>
      <Name>black</Name>
    </Property>
  </SemanticBase>
  <SemanticBase xsi:type="EventType">
    <Label>
      <Name>bark</Name>
    </Label>
    <Property>
      <Name>agressive</Name>
    </Property>
  </SemanticBase>
</Semantic>
```

The equivalent *AnnotationLine* would look like that:

```
obj = dog[large, black]; ev = bark[agressive]
```

5. FEATURE EXTRACTION

For audio feature extraction the *FeatureProcessor* class transparently manages feature processing and storage. Feature implementations reside in a *ProcessingStage*. The features have to be implemented in a Simulink model which can be loaded into the *ProcessingStage*. Simulink provides many potential feature algorithms as part of the Signal Processing and DSP System Toolboxes. Chronos-Trainer also provides a toolbox containing popular features like MFCC, Zero Crossing Rate, Spectral Flux, etc. With Simulink Coder[5], the Simulink model can be transformed to C Code and subsequently to an executable. We created a custom Simulink Coder Target to enable the creation of executable libraries for all operating systems supported by MATLAB. Other implementations or even external processors and frameworks are supported by inheritance.

A *ProcessingStage* contains a signal processing stub with one input port and one output port. This signal processing stub can compute one or more features. A Simulink implementation of MFCCs, Zero Crossing Rate and Spectral Centroid is show in Figure 1. The model has a special interface block called *ProcessingStageInterface* managing additional parameters needed for using the model within a *ProcessingStage*.

For processing features this stub is automatically connected with a file source and a file sink. Depending on the number of input channels the model is also reconfigured. This process is hidden from the user and only takes place at processing

[5]http://www.mathworks.com/products/simulink-coder/

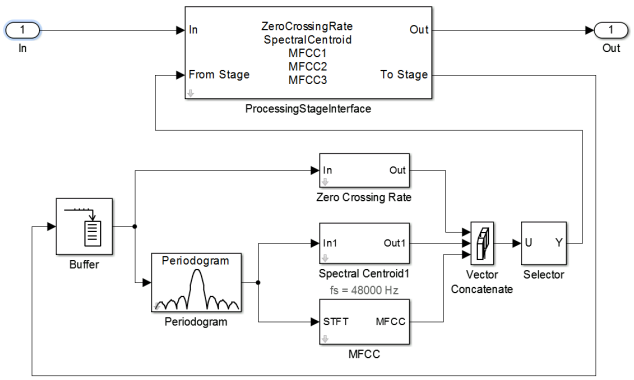

Figure 1: Simulink model for a *ProcessingStage* calculating MFCCs, Zero Crossing Rate and Spectral Centroid

time. The file sink writes feature files which are stored in a *feature repository* maintained by the *FeatureProcessor*.

Feature processing is triggered either explicit by calling the *process* method or implicit by calling a *read* method. Features are processed lazily which means that features are processed only when a feature file does not exist in the repository or the feature algorithms have changed. Algorithmic changes can be detected by the Simulink model versioning mechanism. The model version is written into the feature file and used for change detection. This mechanism ensures that the feature files are always consistent with the algorithms and only minimum computing power (and execution time) is needed.

*ProcessingStage*s can also be stacked to form a processing pipeline. When the features of a stacked stage have to be processed the *FeatureProcessor* manages automatic processing for all parent *ProcessingStage*s. Attached stages inherit feature name prefixes from their predecessors and can compute new features either by *extension* or by *combination*.

Extension means that every new feature is calulated for each input feature. Figure 2 shows an example for using *extension*. When appended to the example stage from Figure 1 which calculates the features named “ZeroCrossingRate”, “SpectralCentroid”, “MFCC1”, “MFCC2” and “MFCC3” the calculated features in the second stage are named “Mean_ZeroCrossingRate”, “Std_ZeroCrossingRate”, “Mean_SpectralCentroid”, “Std_SpectralCentroid” and so on.

Combination uses all or a selection of input features to calculate a new feature. The second mode becomes handy when a classifier is used as a feature. A classifier needs a feature vector as an input and the output is e.g. one feature representing a class index.

For online processing the Simulink model either has to be embedded into a wrapper model containing the necessary IO (e.g. sound card input or network input) or Simulink Coder has to be used to generate C++ Code or an executable library

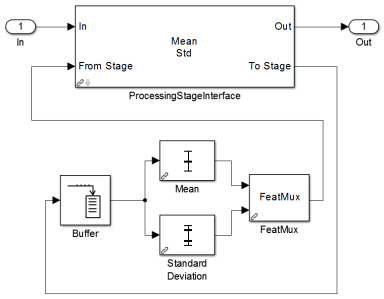

Figure 2: Simulink model for a *ProcessingStage* calculating Mean and Standard Deviation for each input feature (*extension*)

(e.g. a DLL). The *ProcessingStageInterface* block contains C++ interface blocks that are added to the generated code (using the Target Language Compiler). Through the API exposed by those interface blocks, a C++ application can exchange data with the model.

For offline processing features are always stored in feature files. Also when stacking *ProcessingStage*s, processed features from one stage are stored in a file and the next stage reads this file to process its features. This approach allows access to intermediate results, allows sharing features between child stages and avoids unnecessary computation.

The features are stored using the HDF5[6] hierarchical data format. This file format has several advantages over proprietary binary or text based formats. An API for several programming languages including C/C++, Java, Fortran, MATLAB, Python and R is freely available. HDF5 is hierarchically organized in groups and can contain multidimensional datasets of arbitrary size. The data is stored binary providing different datatypes. Datasets and groups can be annotated with attributes. We store feature data structured by feature and audio channel. In addition to the features we store a timestamp dataset and creation information like sample rate, frame size, model version etc.

6. CLASSIFIER TRAINING

To train and evaluate classifiers, feature datasets of labelled data are needed. To obtain labels directly from the semantic annotations we created a rule-based mechanism. These rules are defined using the XPath query language. For example, if we want to train a dog barking detector a query for the barking label could look like this:

```
SemanticBase[@xsi:type="EventType"]/Label/Name
  ="bark"
```

[6]http://www.hdfgroup.org/HDF5/

To simplify queries a few XPath variables exist for common query terms. E.g. the variable *$ev* is a shortcut for *SemanticBase[@xsi:type="EventType"]/Label/Name*. This simplifies the above query to:

```
$ev="bark"
```

This labelling mechanism is integrated in the *FeatureReader* class which can read feature datasets supplemented by associated metadata like semantic annotations, frame count or timestamps from the MPEG-7 Database.

For classifier training we mainly use perClass Toolbox. This MATLAB toolbox provides a wide range of classifiers and supports the complete classifier design lifecycle including training, operating point optimization and classifier execution. Trained classifiers can be exported and embedded in C/C++ code. The library is focused on practical application of pattern recognition tools which makes it well suited for real-world applications.

7. APPLICATIONS

The Chronos-Trainer framework follows a generic approach to audio pattern recognition. It has been used mainly for the detection of environmental sounds and machine sounds but also for speech/music detection and vibration analysis. We use Chronos-Trainer for classifier development of AKUT[14], an acoustic monitoring system for road tunnels which started as a research project more than ten years ago and now will be enrolled in multiple tunnels in Austria over the next years.

In the future we are planning to integrate other frameworks. A first step in this direction has been made by creating an "external" *ProcessingStage* which does not do processing by itself but is able to read from processed ARFF-Files residing in the feature repository. This makes it possible to easily integrate Frameworks like openSMILE or MARSYAS.

8. CONCLUSION

Chronos-Trainer has been developed over the last eight years with a main focus on applied research and industrial applications. The main issue that had to be solved was the fact that most customers require practically working solutions. It is not sufficient to proof that a classifier can reach a certain accuracy, false alarm rate etc. on a given dataset. Moreover complete classification systems able to work online in real-world conditions have to be developed. For achieving this goal an iterative approach is necessary that touches all development stages from annotation to online execution. It is necessary that the framework focuses on minimizes the required effort to go through this cyclic workflow.

Compared to all frameworks that were referenced in this article, Chronos-Trainer delivers a more complete frame-

work because it addresses the whole classifier development workflow. Especially the semantic annotation and rule-based label-mapping approach exceeds the scope of other frameworks. We believe that with this approach it is easier to use the same tools within different domains. Furthermore it avoids the need for a compromise when having to decide between a computationally efficient framework and an experimental prototyping framework by using a code generation approach.

For Chronos-Trainer we also tried to make it easy to include other feature frameworks. Chronos-Trainer provides workflow management and a single interface to the external framework by inheriting a *ProcessingStage*. External annotation tools and external pattern recognition tools like WEKA can be easily integrated by either file exchange or API calls.

Development on Chronos-Trainer is still ongoing and its functionality grows with every project where it is applied. In the future we want to put more effort in interfacing with other frameworks and we are looking for ways to share parts of the framework with the scientific community.

REFERENCES

[1] O. Lartillot and P. Toiviainen, "A Matlab toolbox for musical feature extraction from audio," in *Proc. of the 10th Int. Conference on Digital Audio Effects (DAFx-07)*, 2007, pp. 1–8.

[2] M. Slaney, "Auditory Toolbox," Interval Research Corporation, Tech. Rep., 1998.

[3] I. Nabney, *NETLAB: algorithms for pattern recognition*. Springer, 2002.

[4] J. Vesanto and J. Himberg, "Self-Organizing Map in Matlab: the SOM Toolbox," in *Proc. of the Matlab DSP Conference*, 1999.

[5] G. Tzanetakis and P. Cook, "MARSYAS: a framework for audio analysis," *Organised Sound*, vol. 4, no. 3, pp. 169–175, Dec. 2000.

[6] I. H. Witten and E. Frank, *Data Mining: Practical machine learning tools and techniques*, 2005.

[7] F. Eyben, "openSMILE - The Munich Versatile and Fast Open-Source Audio Feature Extractor Categories and Subject Descriptors," in *Proc. of the international conference on Multimedia*, 2010, pp. 1459–1462.

[8] S. Young, G. Evermann, M. Gales, and T. Hain, *The HTK book (for HTK version 3.4)*. Cambridge University Engineering Department, 2006, no. July 2000.

[9] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, pp. 27:1—-27:27, 2011.

[10] D. Bogdanov, N. Wack, E. Gómez, and S. Gulati, "Essentia: An Audio Analysis Library for Music Information Retrieval." in *Proc. of ISMIR*, 2013, pp. 2–7.

[11] C. Cannam, C. Landone, and M. Sandler, "Sonic Visualiser : An Open Source Application for Viewing , Analysing , and Annotating Music Audio Files," in *Proc. of the international conference on Multimedia*, 2010, pp. 1467—-1468.

[12] K. r. Sjölander and J. Beskow, "Wavesurferan open source speech tool." in *Proc. of INTERSPEECH*, 2000. [Online]. Available: http://www.speech.kth.se/wavesurfer

[13] B. Rettenbacher, W. Bailer, and P. Schallauer, "Einsatz von MPEG-7 fur die Entwicklung von akustischen Klassifikationssystemen," in *Forschritte der Akustik - DAGA 2006*, 2006, pp. 2–3.

[14] F. Graf, G. Rattei, and G. Ruhdorfer, "Giving tunnels ears - installation of the first acoustic monitoring system for road tunnels worldwide," in *Internationaler Tunnelkongress, Hamburg*, 2011.