Audio Processing In Game Of Thrones' Mywatchbegins.com: A Technical Walkthrough

by Alessandro Saccoia / Dinahmoe

Introduction

This walkthrough shows how the server side audio processing used for mywatchbegins.com has been created by using several open source libraries and programs. The orchestration of this free software has allowed us to achieve the goal of the project in a very short time, without ever reinventing the wheel but just through focusing on the more creative aspects.
Besides compiler and build system toolchains, the free software packages used in the project are:

- CMU sphinx 3
- glpk
- freeverb
- ffmpeg

The project

The goal of mywatchbegins.com is to allow users to record their voice while reciting the oath of the TV series Game Of Thrones. After the oath and recording have ended, the user can hear his voice mixed either with the actor's voice, or along with the voices of other users who have created a recording. Mixing one's recording with those of other users creates a choir-like effect.

The initial thoughts about the system have showed two basic needs:
- the implementation of a basic speech recognition system, in order to discard unintelligible recordings and recordings where the users have deliberately used different words.
- the need to time stretch the files in order to realign them to the reference oath. A simple test with the TIMIT speech database (http://www.ldc.upenn.edu/Catalog/CatalogEntry.jsp?catalogId=LDC93S1) proved that as the number of recordings used in a mix grows, so does the need for an almost perfect timing in order to preserve the intelligibility of the final choir.

These needs had to be fulfilled while taking other constraints into account:
- the system had to be as fast as possible: the TV series ranks among the most viewed on the planet and it's very difficult to anticipate the number of users that could be using the system concurrently.
- Time-to-market: the solution had to be ready in a very limited timeframe

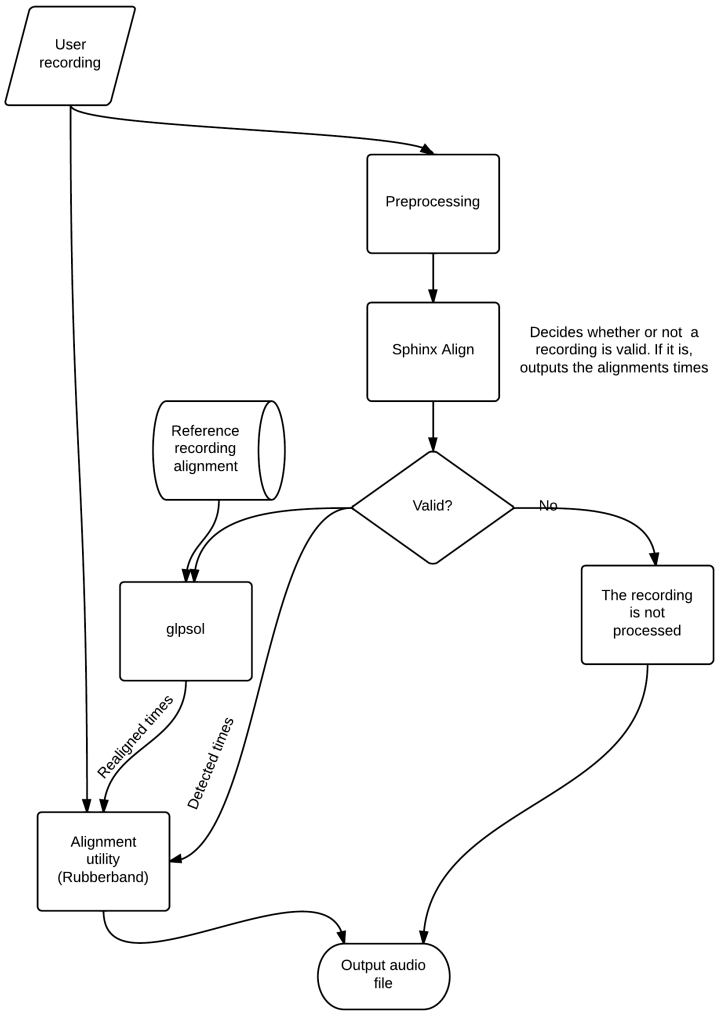

Figure 1: overview of the system

The last constraint has excluded the possibility of the creation of a custom speech recognition system, and has moved the attention towards the use of the CMU Sphinx recognition system version 3, written in the C language. C/C++ has been chosen for the whole system for its performance, and for its ease of deployment on the target Linux servers. The overview of the system can be seen in Figure 1.

Listening tests have been repeatedly performed during the implementation phase, and other needs emerged. In next sections we will explain exactly the reasons behind these needs and what approaches have been used to address them.

Speech validation with CMU Sphinx 3

Sphinx V3 is a library for speech recognition based on Hidden Markov Models developed at the Carnegie Mellon University.

The library comes with a series of executables that, used together, address most of the needs of the researchers in speech recognition. One peculiar utility, called sphinx_align, is interesting because it analyzes an audio file and compares it to a transcript of the sentence that is supposed to be pronounced. Sphinx_align then outputs the timings relative to the beginning of the audio file for each phoneme or word inside the sentence.

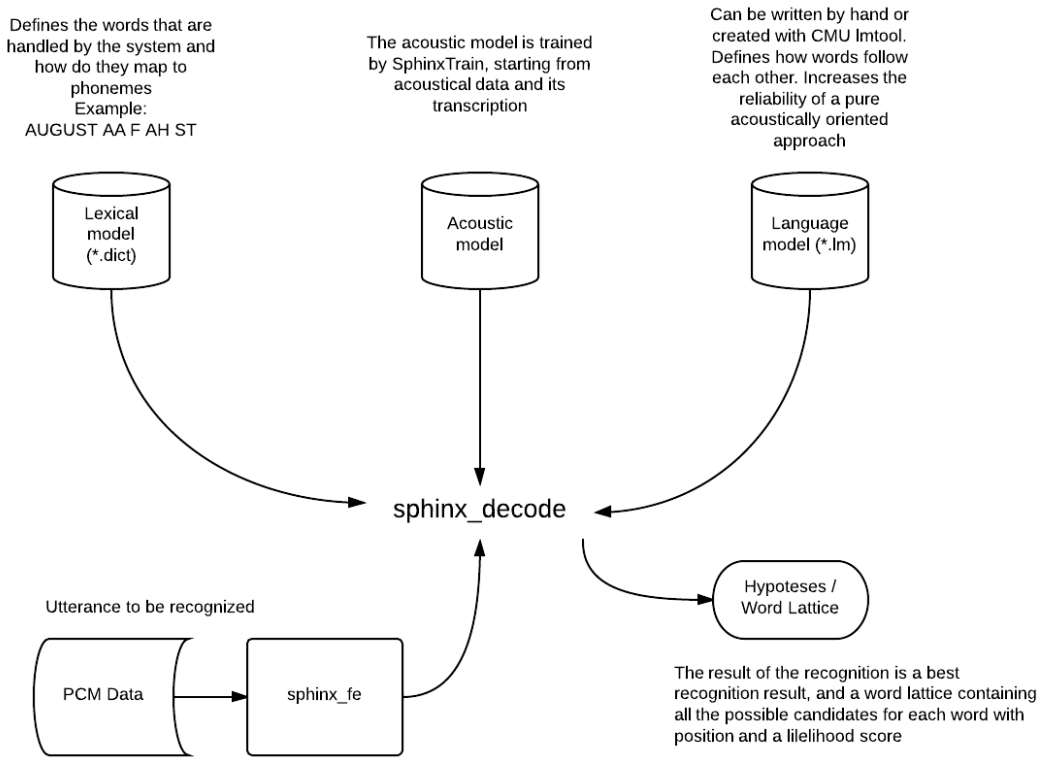

Figure 2: data flow in Sphinx

Using sphinx_align doesn't completely solve the speech recognition problem, as the tool wasn't built to be used for sentence verification. The tool expects to always find the sentence whose transcript was given. The alignment procedure is similar to the basic recognition task as pictured in figure 2, with the difference that the engine will try to "force" the detected acoustical data to fit the language model, that in this case could be reduced to contain only the sentence that has to be validated. As suggested on the Voxforge forums (http://www.voxforge.org/home/forums/message-boards/speech-recognition-engines/sentence-validation-with-sphinx-3), the verification task involves computing the probability of two different hypothesis, not just one as sphinx_align does, and it's itself a well known problem (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.93.6890).

The first experiments with the TIMIT database have highlighted a high number of false positives in the validation. As the number of false positives decreases with the length of

the sentence to be validated, it has become clear that the length of the sentence of the oath would have counter-balanced the fact that the alignment engine was being too optimistic in the recognition phase, given the small language model. Indeed, using long sentences, the probability that the engine will force a positive interpretation creating a false positive decreases, and the results become satisfactory.

The result given by sphinx_align used for validation instead of just for the alignment is very accurate, but the alignment does suffer from sudden background noises, slight mispronunciations, and long silences. In order to improve the reliability of the detected alignment times, a noise gate with a high threshold (-20dB) has been used on the input files (normalized to 0 dB) in order to remove as much noise as possible. Trimming the noise gate parameters has been crucial for the results of the alignment, because the engine seems very sensitive to the start of the words, that could be cut with a more aggressive noise reduction, generating a false negative. Noise gating has been nonetheless very important to improve the accuracy of the alignment, even though it hasn't made it completely reliable. For this reason, the glpk library has been used, as it will be explained in the next section.

An implementation note: even though Sphinx delivered as two separate libraries, all the core functionality is bloated in huge amounts of file in-out handling code, which makes it basically impossible to access the speech recognition routines of the library with PCM data from a custom program. As a result, sphinx_align has been modified to write the results of the performed analysis to disk.

Alignment using Rubberband

The Rubberband time-stretching and pitch shifting library (http://rubberbandaudio.com) provided invaluable help in minimizing the time-to-market of the product.
This software library written in C++ implements all the best features that allow one to have artifact-free time stretching based on the phase-vocoder approach.

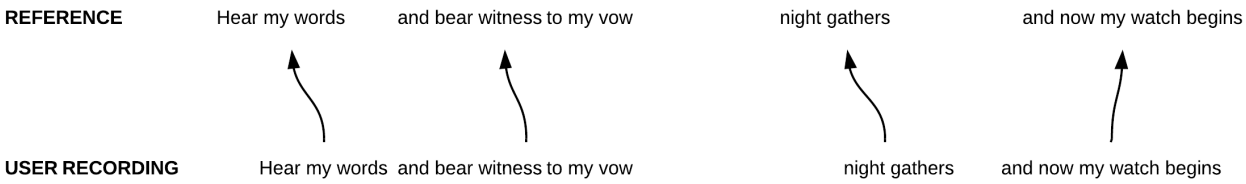

Figure 3: alignment with Rubberband

The audio alignment tool transforms the input PCM data according to the timings stored in two parallel arrays (figure 3). The time stretching ratio for each "pivot" point is computed depending on the ratio between the old and the new durations of a segment. The granularity of the pivot points has been left configurable, in order to do several tests. In the production code, eventually, the alignment points have been set to be at the beginning and at the end of each set of consecutive words.

One thing worth pointing out when using Rubberband is that for sudden changes in the time stretching ratio it's better to have a short window, at the expense of a result that is a bit more "phasey". This is achieved using the *OptionWindowShort* on the initialization of the library.

Realignment using glpsol

As seen in the section about Sphinx, it has proven to be impossible to achieve a 100% confidence in the alignment. The misalignment shows itself in the output utterance with parts that are noticeably too stretched or compressed. It's an unwanted behavior that even if it only happened seldom, would affect the quality of the delivered system in a major way.

A way to bound the time-stretching ratio for each of the segments in which the oath has been divided is needed. Imagining that we want to align the sentence "Shall I compare thee to a summer's day" on the words Shall, thee and summer. The reference oath has timings r1, r2, r3. The analyzed oath (Q) has timings q1,q2,q3. The solution to the problem began with the following question:

I am trying to align an ordered set of n real, strictly positive numbers

$$Q = q_1, q_2, \ldots, q_n$$

to a reference set of the same size and with the same properties

$$R = r_1, r_2, \ldots, r_n$$

I am looking for an analytical solution to find the resulting set

$$S = s_1, s_2, \ldots, s_n$$

that minimizes the differences between S and R

$$F(S) = \sum_{1}^{n} |r_i - s_i|$$

$$argmin_S F(S)$$

but preventing the length of each "segment" $s_n s_{n+1}$ from stretching too much from the original length $q_n q_{n+1}$, keeping the ratio between two numbers α and β.

$$\alpha \leq \frac{s_{n+1} - s_n}{q_{n+1} - q_n} \leq \beta$$

with $0 < \alpha \leq 1$ and $\beta \geq 1$ that are input data of the problem.

Figure 4: initial formulation of the realignment problem

The problem can be solved using Linear Programming (http://en.wikipedia.org/wiki/Linear_programming), and the open source library that implements the simplex algorithm is called glpk (GNU Linear Programming Kit).
Note that the problem contains an absolute value, and the mathematical formulation is not straightforward (http://lpsolve.sourceforge.net/5.1/absolute.htm). The system of equations needed to align the two sets of points varies with the length of the two sets, and it has to be created at runtime using the functions `glp_set_xxx`. This requires a complete understanding of the underlying mathematical method.

Fortunately, the glpsol program included in the source code distribution of glpk can solve problems that use the MathProg language syntax, a human readable language to specify linear programming problems. Even though this isn't the best solution performance-wise, this has allowed to solve the problem in a very short time, by having the modified sphinx_align writing the data file, and parsing the results file of glpsol. The code was given by Jeffrey Kantor on the glpk-help mailing list. Note: the model can be separated from the data, and that's exactly what has been done in mywatchbegins.

```
set N;
param q{N};
param r{N};

param a;
param b;

var z{N} >= 0;
var s{N};

s.t. c1 {n in N}: z[n] >= r[n] - s[n];
s.t. c2 {n in N}: z[n] >= s[n] - r[n];
s.t. c3 {n in 1..(card(N)-1)}: s[n] - s[n-1] >= a*(q[n]-q[n-1]);
s.t. c4 {n in 1..(card(N)-1)}: s[n] - s[n-1] <= b*(q[n]-q[n-1]);

minimize obj: sum{n in N} z[n];
solve;

data;

param a:= 0.95;
param b:= 1.05;

param : N : q :=
   0    3
   1    5
   2    8
   3   12 ;

param r :=
   0    2
   1    5
   2    7
   3   11 ;

end;
```

The introduction of bounding the time-stretching had an excellent result on the quality of the alignment, making it much more robust to the problems in the speech recognition.

Conclusions

The final performance of the delivered system suffered from the fact that most of the communication between the various components had to be carried on by writing and parsing files with a custom formatting. Even the excellent ffmpeg, that has be used extensively to resample and change the format of the audio files, doesn't have a proper API and can be used just through its command-line front-end.
One note of merit goes to the partitioned convolution algorithm implemented in freeverb, which is really fast.
Free software has made this a fun project, letting us focus on the actual problem-solving, rather than just rewriting algorithms. Credit goes to those who helped in directing my efforts on the glpk-help and voxforge mailing lists. Thank you.
In case the reader is interested in getting some source code, I will be glad to provide it. Compiling sphinx is not trouble-free and I have written some patch files and a batch script to automate the process on OS X and Linux systems. Releasing the entire source code would be useless as the API of the server side code is peculiar to the needs of this project and depends strictly on the web front end.

Alessandro Saccoia (Berlin)
http://www.keplero.com
alex@keplero.com

Dinahmoe (Stockholm)
http://www.dinahmoe.com

Berlin,
Feb 21, 2013