Key Constraint

Francis, Kelsey *Martin, Christopher*

Abstract

The Key Constraint project aims to analyze a piece of recorded music to determine its dominant key and utilize this information to produce a simple electronic instrument to accompany the recording, playing only notes from same key of the original song.

Progress Thus Far

The code for this project is hosted on GitHub:

https://github.com/kelseyfrancis/keyconstraint

Analysis

Related Work

Many techniques for identifying the tonality of recorded music have been previously investigated. Krumhansl and Kessler introduced *pitch class profiling* [1], a longstanding approach that has been built upon by many, including Temperly [2]. Rizo et al. proposed a *tree model* approach [3] that is limited to Western music, i.e., 12 semitones per octave. E. Chew presented a *spiral array model* that can also detect changes in key [4], also restricted to music that exhibits tonic triad dominance as is typical in music in the Western tradition. Sheh and Ellis employ EM-Trained Hidden Markov Models [5]. Many other techniques have been proposed beyond those discussed here.

Gomez and Herrera presented a technique [6] similar to the one we plan to take in this project: extract features from the audio in a form similar to the pitch class profile, train a learner with those features based on some model, and use the trained learner to classify features extracted from the key-unknown audio signal.

I/O and Sample Representation

Our plan to identify key hinges on first extracting musically meaningful features from the audio signal in question through spectrum analysis. The first step to analyzing an audio signal is, of course, to read an audio file into memory. We did not use a library for this task, because we wanted to learn the WAVE format. We read in a WAVE files into an array of double-precision floating point samples. We then downmix from stereo to mono, if we detect that the file is indeed multi-channel. Again, we wrote the downmixing code ourselves to learn how it works.

Spectrum Analysis

Once the audio is in memory and downmixed, we perform a spectrum analysis on the audio by running an FFT, using the JTransforms library [7], on overlapping frames of audio to which a window function has been applied. We again implemented the window functions we are experimenting with in order to learn about them in more detail; we have so far considered Hamming and the function used by Ogg Vorbis.

We ensure that, given the sample rate as read in the WAVE header, the number of FFT buckets will provide a resolution of 1 Hz. We will likely refine the number of buckets if we determine we do not need so high a resolution, but using 1 Hz makes certain that each pitch class (discussed below) covers at least one bin, even in the very low octaves, and minimizes overlap of pitch classes between bins.

Pitch Class Profile

Once we have these FFT results, we construct from it a distribution of pitches within the signal. This pitch class profile is comprised of how often each of the the 12 equal-temperament semitone pitches typical of Western music occur within the the signal. Such a profile is made that considers each pitch detected in any octave, e.g., how often the note A occurs in any octave, and another profile considers each octave separately, e.g., A4 vs. A5, etc.

We have a working implementation up through pitch class profile extraction. We have created a simple example program and a 440 Hz sine wave file that demonstrates extracting the pitch class profile and outputs the single-octave profile in human-readable form:

identifykey/pcp.sh

Notice in the output that the profile is expectedly entirely dominated by the pitch class A, the musical note that corresponds to the frequency 440 Hz. Please note that this program was compiled and run with Java 1.6 and built with Maven 3.0.2.

Machine Learning

The next key identification step is to extract the pitch class profile (and any other feature we discover might be valuable) of many audio files of which we know the musical key. We will create a file (likely in ARFF format) of these features and label each song with its key. We will label at least one song in each of the 24 typical western keys (12 notes, each in major and minor modes).

With these labeled features, we will train a learner, using the Weka library [8], which we will use to classify features extracted from a song not previously labeled. We will characterize how well this works with various numbers of songs labeled in each key and using various classification algorithms.

Synthesis

Web Audio

The preliminary idea was to package the product as a web application to which a user could upload an audio file and then play along in a web browser after the server analyzed it. The new Web Audio API implemented by Chromium seemed suitable, but some experimentation demonstrated that it is not quite up to the task.

The "webaudio" directory in the source repository contains a preliminary attempt at realtime synthesis with Web Audio. It simply plays a tone while a key is pressed. This is almost a promising result, but the latency between keypress and audio events is unacceptable for an instrument. A demonstration of this experiment can be viewed at:

http://chris-martin.github.com/webaudio-sandbox/

Python

The Python implementation has been more successful. It is not yet an instrument, but is functioning as a programmatic synthesizer with an understanding of basic music theory.

This section of the project is not utilizing any audio libraries, primarily just because we wanted the experience of developing our own synthesis software from the ground up. It supports oscillators in sine, triangle, and square waveforms and modulation of amplitude and frequency.

We are currently able to play scales in arbitrary keys. This functionality is demonstrated by "play/play.py" in the source repository. This script accepts one parameter: the name of key. Capital/lower case letters denote major/minor keys. "#" and "b" denote sharps and flats. For example, to play a scale in C major followed by another in G-sharp minor:

```
python play.py 'C' && python play.py 'g#'
```

Please note that the audio playback requires that you have "aplay" installed, and that this has been tested only with Python 2.7.2.

The next step is to play notes based on live user input. The plan is to read input from a MIDI keyboard. The "pygame" Python library is well-equipped to assist with this task. It will then require some glue to allow the synthesizer to invoke the detection algorithm. Once that functionality is in place, ideally we should have some time remaining to tweak the synthesis configuration to improve the sound quality.

Upcoming Milestones

July 6

Have the synthesizer controlled via MIDI keyboard.

Extract features for and label at least one song in each of the 12 major keys. Train a learner with this data and do an initial investigation into its effectiveness.

July 13

Connect the analysis and synthesis portions to build a command-line application that accepts the path of an audio file as an argument, performs key analysis, and plays the audio and synthesizer simultaneously.

Extract features for several songs in each of the 12 major and minor keys. Compare effectiveness with less labeled data.

July 20

Add filtering capability to synthesizer. Work on improving overall quality of synthesized sound.

Tweak the classification algorithm by running experiments with several to see which is best with this type of data.

July 25

Final deliverables due.

References

[1] C. L. Krumhansl. *Cognitive Foundations of Musical Pitch*. Oxford University Press, 1990.

[2] D. Temperley. Improving the krumhansl-schmuckler key-finding algorithm. In *22nd Annual Meeting of the Society for Music Theory*. Atlanta. 1999.

[3] D. Rizo, J.M. Inesta, and P.J. Ponce de Leon. Tree model of symbolic music for tonality guessing. In *Proceedings of the Int. Conf. on Artificial Intelligence and Applications, AIA*. Innsbruck, Austria, 2006.

[4] E. Chew. The spiral array: an algorithm for determining key boundaries. In *Proceedings of the Second International Conference on Music and Artificial Intelligence*, ICMAI. 2002.

[5] A. Sheh and D. P. Ellis. Chord segmentation and recognition using EM-trained hidden Markov models. In *Proceedings of the International Symposium on Music Information Retrieval*. Baltimore. 2003.

[6] Gomez, E., and Herrera, P. Estimating the tonality of polyphonic audio files: cognitive versus machine learning modelling strategies. In *Proceedings of the 5th International Conference on Music Information Retrieval (ISMIR04)*. Barcelona. 2004.

[7] JTransforms. http://sites.google.com/site/piotrwendykier/software/jtransforms.

[8] Weka. http://www.cs.waikato.ac.nz/ml/weka/.