**Georgia Institute of Technology**

**2014 Spring MUSI 8903 Audio Software Engineering**

**Homework #5**

**Name:** Chih-Wei Wu  **Student ID:** 902931924

**Name:** Xinquan Zhou  **Student ID:** 902998187

**1. Decide on the class structure [15]**

**Compare for final class structure with possible alternative and describe why you have settled for this design with respect to (i) ease of use, (ii) readability, (iii) maintainability, (iv) expandibility (more elaborate is not necessarily better). Take into account the following requirements:**

1. **you will have both time domain and frequency domain features (see below)**
2. **the feature extractor itself takes a block of audio data (or the magnitude spectrum) and computes the one feature value for this block**
3. **the feature extractor works with a fixed block size set during the initialization**
4. **the feature extractor should allow to extract any feature individually, any subset of the features, or all features at once**

Please refer to “FeatureExtract”, “FeatureCalculate”.

The class hierarchical structure:

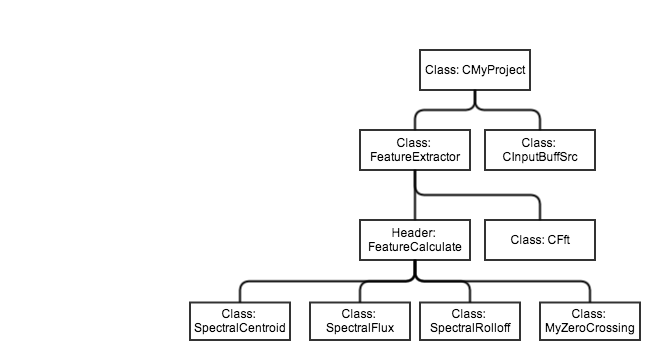


Fig 1. Class structure

In terms of ease of use and readability, this structure allows user to interact only with FeatureExtractor class to get both time and frequency domain features, and it is less confusing without knowing the details of each feature class. In terms of maintainability and expandability, whenever the developer implements more features, they only need to include these classes under FeatureCalculate header (which is included in FeatureExtractor class). The need to tweak FeatureExtractor in order to allow more features is the drawback of this structure. However, a class method to query features under FeatureCalculate header might be a way to solve this problem…

**2. Declare your feature extractor interface and implement your class hierarchy [30]**

**Spectral Centroid**

**Spectral Flux**

**Spectral Rolloff**

**Zero Crossing Rate**

Please refer to FeatureCalculate.cpp for the implementation of each feature class. A better way to do this might be creating base classes for frequency-domain features and time-domain features. But since we only have four features, we decided to keep things simple and straightforward.

**3. Implement tests for your feature extractor [20]**

**One for each feature plus one zero input test for all features.**

Please refer to Test\_MyProject.cpp.

Besides four zero input tests, there are four more tests:

Note that in these tests, when the FeatureExtractor->setTest is set to true, the input buffers will be treated as FFT buffers directly; when it is set to false, the input buffers will be treated as time-domain signals.

1. Spectral Flux: pure Tone test: input one single sine wave. Check if the output is equal or close to zero.
2. Spectral Centroid: Pure tone test: input one single sine wave. Check if the output is at the corresponding frequency bin.
3. Spectral Rolloff: Increasing signal test: generate a gradual increasing signal (x = y), use matlab to calculate the rolloff point at kappa = 0.85 (default value). Check and see if the results are equal.
4. Zero Crossing: One alternative value test: create a signal that only contains +1 and -1. Check and see if the zero crossing rate is equal to 1/(num of FFT).

**4. Modify CMyProject   [30] Add your feature extrator(s) as a member ot MyProject and call the from CMyProject's process function. Use the InputBufferSrc to ensure a fixed block size for the feature extraction. Integrate and understand the FFT class. Store the resulting feature matrix and add two functions: getSizeOfResult and getResult to return the final feature matrix to the command line.**

Please refer to CMyProject.cpp.

**5. write the feature matrix in a text output file [5]**

Please refer to CMyProjectMain.cpp. Also, please run the matlab program main.m. The program will read in the txt file generated from cpp implementation and compare the results.