FINAL YEAR PROJECT

Using a PDA as an Audio Capture Device for Music Genre Classification

James Steward

Year 2004/2005

Supervisor: Dr Dan Smith
Abstract

Acknowledgements
I would like to thank my supervisor, Dr Dan Smith for his support and guidance throughout this project.

Nick Ryan – Initial Recorder Code
Ling Ma – Context Application
Table of Contents

1 INTRODUCTION 5

1.1 AIMS AND OBJECTIVES 5
1.2 OUTLINE 6

2 RELATED WORK 7

2.1 CONTEXT AND CONTEXT-AWARE COMPUTING 7
2.2 SOUND RECOGNITION 9
2.3 MUSIC RECOGNITION AND CLASSIFICATION 9

3 TECHNOLOGIES 11

3.1 MOBILE COMPUTING 11
  3.1.1 PERSONAL DIGITAL ASSISTANTS 11
  3.1.2 WINDOWS CE 11
  3.1.3 JAVA MICRO EDITION 11
  3.1.4 PERSONAL JAVA 13
3.2 THE JAVA NATIVE INTERFACE 13
3.3 HIDDEN MARKOV MODELS 15
3.4 HIDDEN MARKOV MODEL TOOLKIT 15
3.5 THE WAVE FILE FORMAT 16

4 SYSTEM DESIGN 17

4.1 PROBLEM THE PROJECT IS ADDRESSING 17
4.2 SOLUTION 17
  4.2.1 PDA APPLICATION 18
  4.2.2 SERVER APPLICATION 25

5 IMPLEMENTATION AND TESTING 30

5.1 CLIENT IMPLEMENTATION 30
  5.1.1 AUDIO RECORDER 30
  5.1.2 PDA JAVA APPLICATION 30
5.2 SERVER IMPLEMENTATION 32
  5.2.1 NOISE DATABASE 32
  5.2.2 NOISE CLASSIFICATION AND RECOGNITION 32
5.3 TESTING 36

6 EXPERIMENTAL WORK 37

6.1 DATA COLLECTION 37
6.2 EXPERIMENTATION AND RESULTS 37
  6.2.1 MUSIC GENRE CLASSIFICATION 37
  6.2.2 PREDOMINANT INSTRUMENT TYPE CLASSIFICATION 39
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>**7 **EVALUATION AND CONCLUSIONS</td>
<td>41</td>
</tr>
<tr>
<td>7.1 EVALUATION</td>
<td>41</td>
</tr>
<tr>
<td>7.2 CONCLUSIONS</td>
<td>41</td>
</tr>
<tr>
<td>7.3 FURTHER WORK</td>
<td>41</td>
</tr>
<tr>
<td>**8 **REFERENCES</td>
<td>42</td>
</tr>
<tr>
<td>**9 **APPENDIX I: HMM PROTOTYPE DEFINITION</td>
<td>45</td>
</tr>
<tr>
<td>**10 **APPENDIX II: CLASS DIAGRAMS</td>
<td>46</td>
</tr>
<tr>
<td>10.1 PDA APPLICATION CLASS DIAGRAM</td>
<td>46</td>
</tr>
<tr>
<td>10.2 SERVER APPLICATION CLASS DIAGRAM</td>
<td>47</td>
</tr>
<tr>
<td>**11 **APPENDIX III: AUDIO SAMPLE SPECTROGRAMS</td>
<td>48</td>
</tr>
<tr>
<td>11.1 CHAMBER</td>
<td>48</td>
</tr>
<tr>
<td>11.2 ORCHESTRA</td>
<td>48</td>
</tr>
<tr>
<td>11.3 ROCK</td>
<td>48</td>
</tr>
<tr>
<td>11.4 SPEECH</td>
<td>48</td>
</tr>
<tr>
<td>11.5 JAZZ</td>
<td>48</td>
</tr>
<tr>
<td>**12 **APPENDIX IV: TRAINING AND TEST DATA SOURCES</td>
<td>49</td>
</tr>
<tr>
<td>12.1 TRAINING DATA</td>
<td>49</td>
</tr>
<tr>
<td>12.1.1 CHAMBER</td>
<td>49</td>
</tr>
<tr>
<td>12.1.2 ORCHESTRA</td>
<td>51</td>
</tr>
<tr>
<td>12.1.3 JAZZ</td>
<td>53</td>
</tr>
<tr>
<td>12.1.4 ROCK</td>
<td>55</td>
</tr>
<tr>
<td>12.1.5 SPEECH</td>
<td>58</td>
</tr>
<tr>
<td>12.2 TESTING DATA</td>
<td>59</td>
</tr>
<tr>
<td>12.2.1 CHAMBER</td>
<td>59</td>
</tr>
<tr>
<td>12.2.2 ORCHESTRA</td>
<td>61</td>
</tr>
<tr>
<td>12.2.3 JAZZ</td>
<td>63</td>
</tr>
<tr>
<td>12.2.4 ROCK</td>
<td>65</td>
</tr>
<tr>
<td>12.2.5 SPEECH</td>
<td>67</td>
</tr>
</tbody>
</table>
1 INTRODUCTION

Mobile devices are an area of technology which is making huge progress in terms of capabilities and size. Together with the advances in wireless technology, allowing some devices to almost be constantly in communication with other devices or the internet has lead to the expansion of mobile computing in general. However, these devices still have some severe hardware limitations; user interaction is restricted by these limitations as well as their activity.

It is no surprise that mobile devices are designed easily portable; suggesting the ability for use in almost any location the user is situated. With this in mind the situation or location a device is in can change many times over a period of time. It is this that makes contextual information important to mobile computing. Yet many of the applications that are used on these devices are still developed as if they were to be used on traditional desktop computers [2] i.e. they are not aware of the context in which the applications are being used. Context is information that can be used to characterise the environment in which the user is running the application [3] and encompasses a number of things including lighting, noise, network connectivity, communication costs, and communication bandwidth amongst others. Context-aware applications are able to adapt its behaviour according to different contextual cues.

Acoustic noise is a form of contextual information which can be captured easily and can provide valuable information about the current situation. Previous work has been conducted using environmental noise to determine a user’s location situation (e.g. on a bus, in a car, etc.) with promising results [Ling paper]. This project proposes to determine whether classification of styles of music and speech in proximity to a device can be conducted accurately.

Musical genres are categorical descriptions used to describe music. Genre classification can be an important aspect of music information retrieval. It is used by retailers, librarians, musicologists and listeners in general. The latter was particularly highlighted by the work of Adrian North [paper] which shows that the liking of listeners is influenced more by the style a particular piece is played in rather then the piece itself.

1.1 Aims and Objectives

Initial aims of the project – weather forecasting!

Changes of focus…

Over the course of the project the aims have changed significantly. Originally the project was entitled “Delivering Weather Forecast Information on PDAs”. A system was to be developed to provide a weather forecast to the user in a form appropriate to the current situation the user is in. This involved implementing a version of the system described in [Ling paper – acoustic environ as an…]

The purpose of this project is to determine whether accurate classification of styles of music and speech can be made using a Personal Digital Assistant (PDA).

- Recorder for PDA
  - Capture samples of defined length (seconds) and frequency automatically until disabled.
  - Transmit the recorder sample to a ‘remote server’ for classification.
• Develop a set of HMMs for use in classification experiments
• Capture date for the experiments
• Conduct experiments using genre of the music to classify samples

• Breakdown of the project in logical aims and objectives
• Understanding of their importance and relation

1.2 Outline
This section will explain the contents and the structure of the remainder of this report.

Section 2 discusses existing work relating to the subject area covered by this project. It is split into three parts. The first is a general overview of the ideas of what context is and the forms of information to which it is related. Various works utilising the notions of context and gather contextual information are described. The two subsequent parts discuss works completed regarding sound and music recognition respectively.

In section 3 the technologies available for used in this project will be discussed in four separate parts. The first part will introduce the PDA as a mobile device and the platforms by which they are operated, followed by an introduction into application development using Java, specifically J2ME (Java 2 Micro Edition). The second part will discuss the technology of the Java Native Interface (JNI). Part 3 will focus on the use of Hidden Markov Models as a method of sound recognition while part 4, to conclude the section, will introduce the Hidden Markov Model Toolkit (HTK) as a tool for creating and manipulating HMMs.

The topic for discussion in section 4 will be the design of a system, utilising the technologies discussed in section 3, as a solution to the aims and objectives discussed in section 1.1. This is followed by the implementation and testing of the designed system in section 5.

Section 6 details the experimental work carried out using the implemented system. This includes descriptions of the methods used, analysis of the samples captured and the results and conclusion obtained.

An overall evaluation of the project as a whole is detailed in section 7, with conclusion drawn and a suggestion of further work.
### 2 RELATED WORK

Section introduction...

#### 2.1 Context and Context-Aware Computing

When humans interact with other people and the surrounding environment we make use of implicit situational information. We are able to intuitively deduce and interpret the context of the current situation and react appropriately. An example of this is when someone in a discussion with another person automatically observes the gestures and voice tone of the other party and reacts in an appropriate manner.

Computers are not as good as humans in deducing situational information from their environment and in using it in interactions. They cannot easily take advantage of such information in a transparent way, and if they can they usually require that it be explicitly provided.

Context is basically situational information. A more formal and well-regarded definition is stated in [14]:

> “Context is any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.”

Almost any information available at the time of an interaction can be seen as contextual information. Some examples are:

- Identity
- Spatial information
- Temporal information
- Environmental information
- Social situation
- Resources that are nearby
- Availability of resources
- Physiological measurements
- Activity
- Schedules and agendas

Schmidt gives a definition of ‘situation’ and ‘context’ in [8]; using these, he defines how a situation belongs to a context:

> “Definition: Situation

A situation is the state of the real world at a certain moment or during an interval in time at a certain location.”

> “Definition: Context

A context is identified by a name and includes a description of a type of situation by its characteristic features.”
“Definition: Situation S belongs to a Context C

A situation S belongs to a context C where all conditions in the description of C evaluate to true in a given situation S.”

Using these definitions Schmidt regards context as a pattern, which can be used to match situations of the same type.


Expand of Schilit work a bit more here… just a sentence or two…

Dey and Abowd [13] describes them as restricting the definition from applications that are simple informed about context to applications that adapt themselves to context. Dey gives a general definition of context-aware computing in [14]:

“A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s tasks.”

A system that can recognise the identity of the user can be dramatically personalised. Different user interface models can be used allowing information to be displayed in an appropriate language for example. Generally, the identity can be confirmed using a variety of sensors including biometric sensors such as retinal scanners, speech recognition, and identity cards. For mobile or wearable computing, methods such as pin codes may be more appropriate.

Location-awareness is an area very popular in context-aware computing research and is one of the most important forms of context in pervasive computing. When the location of a system is known, systems like ‘where am I?’ applications are available. Location can be determined via a number of means. These include navigational hardware and compasses, GPS, and indoor triangulation techniques using RF and sound. Active Map [16] [17] [18] uses location information. The room number is collected directly from the PARC Tab system because the PARC Tab network uses a separate wireless base station for each room. The individual faces are shown at the locations of people on the map, which is updated every few seconds, allowing people to be located quickly, or to notice a meeting one might want to attend.

Time is not a difficult context type to obtain and is available from built-in clocks of computers. Many applications correlate location or schedule information with timestamp, such as Active Badge [20], Cyberguide [21] [22] and Office Assistant [15]. Other forms of time context include day of the week, day of the month, month of the year, season of the year, and time zone.

Physical contexts such as temperature, light or sound levels can be very useful sources of information. Researchers of the TEA project built a multi-sensor prototype to sense more contexts [23], including a photodiode to detect light level, two accelerometers to provide tilt and vibration measurements, a passive IR sensor to detect the proximity of humans, a omni directional microphone to detect sound, and other built-in sensors for temperature, pressure, and CO gas.

An extended version of the DUMMBO (Dynamic Ubiquitous Mobile Meeting Board) project [9] is an example making use of social types of context. DUMMBO is an instrumented digitising whiteboard that aids the capture and access of informal and spontaneous meetings. It captures the ink added to and erased from the board and records the audio discussions. The context-aware element was introduced when the process of the data capture was triggered when a group of two or more people gathered around the whiteboard. As well as the social situation context, the context used in this project included identities of the participants, the time of the meetings and when participants joined and left the meetings and the location of the meetings.
Network connectivity and bandwidth can be an important computational context. There is no easy way, however, for applications to adapt to the bandwidth changes without underlying system support. The Odyssey system [10] provides API calls by which applications can be notified when the network bandwidth changes. More recently, work has been carried out on the Congestion Manager [11] that measures the bandwidth and notifies applications. Examples of other forms or resources that could potentially be used as contextual information are battery life and display availability/resolution.

The Adaptive GSM phone and PDA [24] use the user’s activity as a source of contextual information, as well as light and pressure levels and the proximity to other people. In the PDA scenario a notepad application is changed to adapt to the font size to the activity of the user (a large font when the user is walking, small font when stationary) as well as to environmental conditions. In the phone scenario, the profiles of the mobile phone are selected automatically based on the recognized context. The phone rings, vibrates, adjusts the ring volume, or keeps silent, depending on whether the phone is in hand, on a table, in a suitcase, or outside.

The context-aware office assistant [15] uses the office owner’s current activity and schedule as the source of context. The assistant is an agent that interacts with visitors at the office door and manages the office owner’s schedule. It is activated when a visitor approaches, detected by two pressure sensitive mats placed on both side of the office door, and it will adapt its behaviour to such contextual information as the identity of the visitor, the office owner’s schedule status and busy status, and the owner’s willingness to see the current visitor. It is intrusive, however, to recognise visitor’s id by a name asking process. A system that could be used as an alternative approach is the Smart Floor [19].

### 2.2 Sound Recognition

- Introduce Various Methods

Environmental noise recognition – Ling!

### 2.3 Music Recognition and Classification

Experimentation of classification of melodies of 4 categories of Musical Instrument Digital Interface (MIDI) files using discriminant feature extraction and fusion [Ming Li, Ronan Sleep] yielded some interesting results. This used a combination of melody features, the bi-gram pitch features and knowledge-based pitch features derived from a selection of top-level properties. Fusing the discriminant features of different sets yielded a 5% higher classification accuracy than the single un-combined feature sets.

Tin Lay Nwe and Ye Wang [paper] propose a method for automatically detecting the vocal segments of songs. The technique they propose employs the use of HMMs to separate the samples into vocal and non-vocal classifications. The conventional method of training is to create one model for each class; Nwe utilises multi-model HMMs for improved accuracy. They specify a number of specialised HMM models for each vocal and non-vocal class dividing the class, for example, chorus and verse. The technique used for feature extraction is based on sub-band processing using Log Frequency Power Coefficients (LFPC) to determine the energy distribution across sub-bands. Experimentation was conducted to compare the performance of the LFPC features and the MFCC and the results indicated the higher capability of LFPC features for extracting spectral characteristics of audio signals.
The classification of MIDI recordings by genre is presented by McKay and Fujinaga using 109 different musical features [paper]. The classifications are based on the high level “root genres” such as Jazz and Classical with specialisation of these into subcategories or “leaf genres” forming a tree hierarchy of classifications. The leaf genres are the various forms or styles within the root genre; Bebop for example is a leaf genre to Jazz. A total of 950 MIDI recordings were using for the training and testing of 3 root genres and 9 subcategories which produced a 98% accuracy for the root genre classification and a 90% accuracy for the subcategory classification.

West and Cox [paper]

Rock, Classical, Heavy Metal, Drum and Bass, Reggae and Jungle Music.

Vincent and Rodet [paper] conducted an investigation into the identification of solo and ensemble music using non-linear Independent Subspace Analysis (ISA).
3 TECHNOLOGIES

Introduce Section...

3.1 Mobile Computing

3.1.1 Personal Digital Assistants

Personal Digital Assistants or PDAs encompass a number of devices that are essentially handheld computers. There are two major groups of these devices, those with small keyboards and those without, relying instead on a special pen and some type of handwriting recognition.

PDA's are commonly used to keep track of appointments, addresses and to-do lists, but many have the capability to do much more.

Currently there are three major PDA operating systems in the market:

- Palm OS; created by Palm Computing, now separated into PalmSource and palmOne [1], is the market leader in PDA sales and the originator of the Palm Pilot, the first popular handheld device.
- Microsoft's Windows CE PocketPC OS [2] is a relatively new and fast growing operating system that is gaining large shares of the market, particularly in the United States.
- Symbian OS [3], formally known as EPOC, was developed by the Symbian consortium comprising of Ericsson, Motorola, Nokia and Psion, for usage in a variety of mobile devices including handheld PCs.

For this project Windows CE Pocket PC 2002 will be used on a HP iPAQ, model: h5450.

3.1.2 Windows CE

The most commonly used version of Windows CE for handheld computers is the PocketPC version, which is tailored specifically for this type of device. This is the version that will be used on the iPAQ for this project. The PocketPC package comes complete with mini versions of Microsoft Word and Excel and a Personal Information Management (PIM) Pocket Outlook. Pocket Internet Explorer is available and can be utilised for both online and offline browsing as well as supporting ActiveX and JavaScript. The synchronisation software for Windows CE is ActiveSync. This is used to keep mobile data consistent and up-to-date with data on the users corresponding PC. Communication with ActiveSync is supported via USB, infrared, the serial port as well as Ethernet LAN [4].

3.1.3 Java Micro Edition

J2ME stands for Java 2 Micro Edition. Whereas J2EE (Java 2 Enterprise Edition) refers to the specifications that govern the use of java in distributed server side environments, and while J2SE (Java 2 Standard Edition) addresses the needs of the desktop, J2ME was developed to target the extremely diverse range of consumer and handheld devices and cope with the hardware limitations of these devices [5].

The way J2ME is implemented there is no single specification. It is, instead, a family of related specifications that define what Java technology looks like on resource-constrained devices giving it a very modular form. It utilises profiles targeted at specific devices, but with portability maintained across configurations.
3.1.3.1 Configurations
Configurations are targeted towards groups of devices that have similar memory constraints, interface requirements, network capabilities, etc. They are the minimum virtual machine and core java classes that will support a relatively broad range of similar devices.

Currently there are two configurations:

- Connected Device Configuration (CDC)
- Connected Limited Device Configuration (CLDC)

The intention of the CDC was to bring the Java platform to network-connected consumer and embedded devices like Symbian EPOC Communicators and high end PDAs. These typically have 32-bit microprocessors/controllers and have at least 2 MB of RAM and 2.5 MB of ROM available to the Java application environment [6].

The CLDC support mobile devices which are significantly more hardware and performance limited than the ones that the CDC supports. These can include low end PDAs, mobile phones and pagers. The CLDC specification identifies devices in this category as having the following characteristics [7]:

- 160 to 512 KB total memory available for the Java platform
- 16-bit or 32-bit processor
- Low power consumption, often battery powered
- Intermittent network connectivity (often wireless) with potentially limited bandwidth

3.1.3.2 Profiles
Figure 1 shows the relationships between the different J2ME configurations and profiles.

Profiles sit above a particular configuration, but in order to function they require the availability of the underlying configuration. They contain the java classes that focus on specific implementations of a particular class of mobile device.

![Figure 1: Relationships between J2ME configurations and profiles](https://www.blueboard.com/j2me)

Foundation Profile is a set of Java APIs that support resource-constrained devices without a standards-based GUI system. Combined with the Connected Device Configuration (CDC), Foundation Profile provides a complete J2ME application environment for consumer products and embedded devices.

The Personal Profile provides an environment with full AWT support. The intention of its creators is to provide a platform suitable for Web applets. Personal Profile version 1.0 requires an implementation of the Foundation Profile version 1.0. Personal Profile is a subset...
of the J2SE version 1.3.1 platform allowing the upward compatible of Personal Profile applications with J2SE version 1.3.1.

The MIDP targets mobile information devices, such as mobile phones, two-way pagers, etc. The PDA Profile will produce optional packages for features that are commonly found on PDAs and other mobile. The final release for this profile only just became available in June 2004.

3.1.4 Personal Java

- Pre-J2ME
- Insignia Jeode Runtime...

3.2 The Java Native Interface

The Java Native Interface (JNI) is an interface part of the JDK (Java Development Kit) that allows a Java application running within a Java Virtual Machine to operate with applications and libraries written in other languages. These can include C, C++ as well as assembly. Additionally, the Invocation API allows the embedding of the Java Virtual Machine into native applications.

The JNI is used to write native methods to handle those situations when an application cannot be written entirely in the Java programming language. Liang [book] suggests four examples when the JNI would be useful in this way:

- The Java API might not support certain host-dependent features needed by an application. An application may want to perform, for example, special file operations that are not supported by the Java API, yet it is both cumbersome and inefficient to manipulate files through another process.

- You may want to access an existing native library and are not willing to pay for the overhead of copying and transmitting data across different processes. Loading the native library in the same process is much more efficient.

- Having an application span multiple processes could result in unacceptable memory footprint. This is typically true if these processes need to reside on the same client machine. Loading a native library into the existing process hosting the application requires less system resources than starting a new process and loading the library into that process.

- You may want to implement a small portion of time-critical code in a lower-level language, such as assembly. If a 3D-intensive application spends most of its time in graphics rendering, you may find it necessary to write the core portion of a graphics library in assembly code to achieve maximum performance.

The benefits gained in functionality by using the JNI do have consequences. When using the JNI there are risks that two benefits of the Java platform can be lost. JNI use compromises the applications ability to run on multiple host environments; the parts of the application written in the native language will require recompiling in order to operate. The Java programming language is type-safe and secure. Other languages such as C and C++ are not. Consequently, Java applications using the JNI require extra care as native methods can corrupt the entire application. Liang gives a general rule for JNI implementation saying
native methods should be “defined in as few classes as possible. This entails a cleaner isolation between native code and the rest of the application”.

There are six steps for writing native methods for Java applications; this example is using C as the native programming language [Liang] [JNI Tutorial]:

1. Write the Java program. Create a Java class that declares the native method; this class contains the declaration or signature for the native method. It also includes a main method which calls the native method.

2. Compile the Java class that declares the native method and the main method.

3. Generate a header file for the native method using javah with the native interface flag -jni.

4. Write the implementation of the native method in the programming language of your choice, such as C or C++.

5. Compile the header and implementation files into a shared library file.

6. Run the Java program.

The Java program has two requirements; declaration of the native method(s), and loading the shared library. The declaration uses the keyword ‘native’, indicating to the Java compiler that it is a native method. It provides no implementation for the method, this in the native language source file. An example of a native declaration is:

```java
public native void myNativeMethod();
```

The native language code is compiled onto a shared library. This is later loaded into the Java class that required it at runtime. The loading of the library maps the implementation of the native method to its declaration. This is done by using the `System.loadLibrary` method with its argument being the name of the library as a String, placed within a static initialiser.

An example of a static initialiser is:

```java
static
{
    System.loadLibrary("library");
}
```

The Java classes declaring native methods are compiled as normal using the Java compiler. A C header file is created using the tool javah on the class containing the native methods. The header contains the function prototype for the native method. What is important here is the prototype for the native methods declared in the java program. The name of the native language function that implements the native method consists of the prefix `Java_`, the package name (if applicable), the class name, and the name of the native method. Between each name component is an underscore "_." separator [Liang]. For example:

```c
JNIEXPORT void JNICALL Java_MyJavaClass_MyNativeMethod(JNIEnv *, jobject);
```

The JNIEXPORT and JNICALL macros ensure that this function is exported from the native library and C compilers generate code with the correct calling convention for this function.

The implementation functions always take two arguments. The first is a JNIEnv interface pointer; this pointer is a handle to the current thread in the Java virtual machine, and contains mapping and other housekeeping information. The second argument is a reference to the method that called the native code. If the calling method is static, this parameter would be type jclass instead of jobject.
The basic process of writing the implementation of the native method is fairly straightforward. The functions that are written must follow the prototypes specified in the generated header file. When using C as the native language, it should include at least two header files for the purposes of the JNI, jni.h and the any header files generated in the previous step, plus any other header files required by the C implementation.

The native implementation code is then compiled into a shared library. This is the library that is loaded by the Java application calling the native methods. On this Windows environment the library is in the form of a dynamic linked library (DLL).

Given that the previous steps have been completed, the Java application would now be ready to run, calling the native method from main when required.

The process here outlines only the very basics of a native method implementation. Aspects such as native method return types and conversions between Java, JNI and native object and primitive types have not been discussed. For more information on these technical aspects see [Liang], [Stearns (Online Tutorial)], [Gordon, R. (1998)] and [Austin, C. and Pawlan, M. (2000)]

3.3 Hidden Markov Models
- Introduce HMMs

3.4 Hidden Markov Model Toolkit

HTK script files are used when tools very long list of files. This is the case for both feature extraction and HMM training. Two script files will be required. For feature extraction the script file (code_mfcc.scp) will contain a list of all the paths of the files samples in the noise database and destination paths for the extracted MFCC files. For training using the HCompV tool the script file will be required to simply contain a list of all the MFCC files to be used for training.

A list of the models to be trained is required by the tool HERest. This is simply a text file containing a single model name on each line.

Grammar
- gram
- snet (created by parser)

• Dictionary
  - dict

• Labels
  - labels.mlf
3.5 The Wave File Format

The audio file will be written as a Pulse Code Modulation (PCM) Waveform Audio Format (WAVE) sound file. This format is a subset of Microsoft’s Resource Interchange File Format (RIFF) specification for the storage of multimedia files. A RIFF file starts out with a file header followed by a sequence of data chunks. A WAVE file is often just a RIFF file with a single “WAVE” chunk which consists of two sub-chunks, a “fmt” chunk (where the fourth character is a ‘space’) specifying the data format and a “data” chunk containing the actual sample data. The specification of the basic PCM WAVE file to be used is detailed in Table 1.

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ChunkID</td>
<td>4</td>
<td>Contains the letters “RIFF” in ASCII form</td>
</tr>
<tr>
<td>ChunkSize</td>
<td>4</td>
<td>(= 4 + (8 + \text{SubChunk1Size}) + (8 + \text{SubChunk2Size})) This is the size of the entire file in bytes minus 8 bytes for the two fields not included in this count: ChunkID and ChunkSize.</td>
</tr>
<tr>
<td>Format</td>
<td>4</td>
<td>Contains the letters “WAVE”</td>
</tr>
<tr>
<td>Subchunk1ID</td>
<td>4</td>
<td>Contains the letters “fmt”</td>
</tr>
<tr>
<td>Subchunk1Size</td>
<td>4</td>
<td>16 for PCM. This is the size of the rest of the Subchunk which follows this number.</td>
</tr>
<tr>
<td>AudioFormat</td>
<td>2</td>
<td>PCM = 1 (i.e. Linear quantization). Values other than 1 indicate some form of compression.</td>
</tr>
<tr>
<td>NumChannels</td>
<td>2</td>
<td>Mono = 1, Stereo = 2, etc.</td>
</tr>
<tr>
<td>SampleRate</td>
<td>4</td>
<td>8000 = 8 kHz, 44100 = 44.1 kHz, etc.</td>
</tr>
<tr>
<td>ByteRate</td>
<td>4</td>
<td>(= \text{SampleRate} \times \text{NumChannels} \times \text{BitsPerSample} / 8)</td>
</tr>
<tr>
<td>BlockAlign</td>
<td>2</td>
<td>(= \text{NumChannels} \times \text{BitsPerSample} / 8). The number of bytes for one sample including all channels.</td>
</tr>
<tr>
<td>BitsPerSample</td>
<td>2</td>
<td>8 bits = 8, 16 bits = 16, etc</td>
</tr>
<tr>
<td>Subchunk2ID</td>
<td>4</td>
<td>Contains the letters “data”</td>
</tr>
<tr>
<td>Subchunk2Size</td>
<td>4</td>
<td>(= \text{NumSamples} \times \text{NumChannels} \times \text{BitsPerSample} / 8) This is the number of bytes in the data.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Field Size (bytes)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>Subchunk2Size</td>
<td>The actual sound data.</td>
</tr>
</tbody>
</table>

Table 1: Basic PCM WAVE File Format Specification
4 SYSTEM DESIGN

Introduction to section

4.1 Problem the Project is Addressing

- Clear understanding of the issues which the project is trying to solve
- Breakdown of the problem into logical components
  - Audio Recorder
  - PDA Java UI
  - HTK HMMs
  - Server application and network transmissions

4.2 Solution

The approach to the solution will employ a client-server framework based on the system created by Ling Ma [ling paper]. The client, being the PDA device, will consist of an audio recorder and a controlling application with the functionality of transmitting the file over a (wireless) network. The 'server' will provide the noise models, a database of recorded samples, and a noise classification and recognition application. Figure 1 is an overview if the system structure to be developed showing the general components of the client and the server sides of the system.

On the server side the noise classification system will retrieve the data from the database and create HMMs. When sample recognitions occur, the sample will be processed and compared against the HMMs created.

The client application will initialise the recorder and capture the data for one of two purposes; recognition of a sample whose category is unknown to the system, or addition of a sample to

Figure 1: System Structure Overview
the noise database to be used as training data. When the latter is the case, the category of the model the sample is to be added will be required.

This will generate a text file containing the classification of the sample which will be read and passed back to the PDA. When a training sample is recorded, the sample and category (specified by the user) will be transmitted to the server from the PDA application and the sample will be added to the appropriate section of the noise database.

### 4.2.1 PDA Application

On Pocket PC 2002, the platform in which the PDA operates, there is no support for the Java Sound API on Pocket PC 2002. Therefore, a recorder is to be developed using the C++ programming language which will interface with the controlling Java application via the Java Native Interface (JNI). When the application is started, the shared library of the recorder will be loaded to enable the Java access to the native methods. The recorder itself will be initialised were all the require parameters (e.g. the storage directory, audio sample quality, etc) are set-up ready for sample recording to occur. On exiting of the application the shared library will be unloaded to free the resource. This is shown in Figure 1.

![Figure 1: Initialisation and finalisation requirements](image)

From the point of view of the Java PDA application, the recorder is simply a method that is to be called to generate one audio sample. Therefore the use of a Java Thread will be employed to call the ‘recorder method’ multiple times with a sleep function serving as the duration in between sample captures. Therefore the user will operate the recorder with two simple buttons, one to start and the other to stop the thread as shown in Figure 1.

As would be expected, the recorder will produce an output file. The format to be used will be the PCM WAVE format; see section X for more details. The file will be written into the root directory of the PDA application designated “path here”. The sample name will be “sample.wav”. When multiple files are recorder the new data will overwrite the any existing file, therefore not allowing a build-up of samples as storage capacity is limited on the PDA.
The recorder will have two variables set by the user, the duration of each sample to be recorded and the time delay between recordings. The delay between recordings will be handled on the Java side via a Sleep function in the Java thread. However the duration of each sample will need to be handled on the C++ side. Therefore a native method will be required. Figure 1 is a diagram of the processes required.

When a sample is recorded, transmission of the sample to the server will be required. This will be for one of two possible cases:

- New training data addition
- Sample recognition

Discussed earlier was fact that new samples recorded would overwrite the wave file of a previous recording to prevent issues with the (limited) storage capacity being exceeded. As this is the case, the transmission of the file to the server must occur immediately after the new sample has been written.

Data training will require the user to select which model the data is to be added. This is to be transmitted to the server so the file can be added to the required directory in the noise database. Samples to be transmitted for recognition will receive two items of information from the server, the model classification of the sample sent (e.g. rock) and the confidence level (discussed later).

The application is to employ a very basic user interface, consisting of a single ‘screen’. The following information is required to be displayed to the user:

- The current setting of:
  - the sample recording length
  - the time delay between recordings
• the model category to which data is to be added (training only)

- The recorder status (enabled/disabled)
- The success/failure of the capture of the last audio sample
- A count of the number of files recorded in current session (both successful and unsuccessful.
- Current operational status taking the form of one of the following:
  - Recording. The recorder is in the process of capturing new data and writing a new file
  - Sleeping. The recorder is waiting for the time delay between recordings to expire
  - Sending. The transmission of the audio sample to the server and either the context information or the selected training category is in progress
  - Stopping. The recorder had been disabled but is waiting for the current recording process to complete
  - Stopped. The recorder is inactive
- The model classification and confidence level of the last recorded sample (sample recognition only)

4.2.1.1 The Native Recorder Physical Design

Source code was made available for use in this project. This was written in C++ by Nick Ryan of the Computing Laboratory, University of Kent and was intended to be used as a starting point for the application. This code made use of the VoiceRecorderControl API for its implantation. Unfortunately this API relies on the user input with a Graphical User Interface (GUI) to start and stop the recorder for each individual sample. As a consequence this approach was abandoned and a new approach using the Windows Waveform Audio Functions [MSDN internet link] was initiated.

The recorder has the following requirements:

- To periodically record multiple samples without user input once enabled.
- To allow the user to specify the length of the samples to be recorded.
- To output single file in the WAVE format, overwriting any existing file of the same name and path.

The approach to the design of the recorder was to design it in such a way that the recording of a single sample would require only one method call to the recorder, allowing the method call to be implemented into a Java thread calling the recorder as required. As this is the case, the duration between recordings will be handled on the Java side with the timing of the thread calling the recorder.

The required processed shown in Figure 1, Figure 1 and Figure 1 indicate the requirement of 3 native method, one to initialise the recorder, one to set the duration of the sample recordings and a third to record the sample itself.

initialiseRecorder is used to setup the parameters by which the device will record the samples using the WAVEFORMATEX object and is to be called when the shared library is loaded into the Java application. These details are to be ‘hard coded’ as it is expected that the sample rates, number of channels, etc, are to remain constant for testing purposes, therefore dynamic
setting of the sample parameters is not required. Given the nature of the PDA device and its limited resources, it was decided that the parameters of the sample recordings should reflect this; therefore the following specification was chosen for use:

- 8-bit quantisation
- 22050 samples per second
- Single channel (mono)

`setRecDuration` is a simple method to set the length of which the samples are to be recorded. This relates to item 4 in ...

`recordAudioSample` is where the actual recording takes place. A single call of this method will record a single audio sample; therefore for multiple sample recordings, multiple calls of this method will be required. This is handled in the Java application and is discussed in section 4.2.1.2.

As mentioned earlier the Windows Waveform Audio Functions are to be used for the recorder. To use these, a buffer will be allocated to hold enough data for the required length of file required at the time. When recording, the buffer will be allowed to completely fill. When this occurs the API generates a message signalling the recording has stopped. In order to capture this message a background thread will be created to monitor for this. When the signal is received the thread will write the wave file.

**What is the best way to present the processes shown below??**

**recordAudioSample()**

- Create Thread `waveInProc`
- `waveInOpen`
- allocate memory buffer
- `waveInPrepareHeader`
- `waveInAddBuffer`
- `waveInStart`
- Wait for buffer to fill and thread to set completion flags (timeout after approx 5 seconds)
- `waveInClose`
- Free allocated memory
- Return success or failure string to Java

**waveInProc()**

- Start while(true)
- Get message
- If message is waveform-audio input message MM_WIM_DATA
  - Prepare variables for write to output file
  - Open output file
- Write WAVE Format data
- Set completion or failure flag
- If message is waveform-audio input message MM_WIM_OPEN
  - Do nothing…
- If message is waveform-audio input message MM_WIM_CLOSE
  - Break from while loop so thread returns (and therefore exits)
- End while loop
- Return
The requirements of the application are as follows:

- Access the native recorder to capture audio samples periodically for either:
  - Noise recognition, or
  - Noise model training
- Transmit the sample to the server
- Receive the sample classification and confidence level and display to the user.
- Display the sample classification and the confidence level.

The following four classes will be required for the Java application, also illustrated in the class diagram in Appendix II: Class Diagrams:

- **App** – The main class and entry point into the application
- **Recorder** – Java Thread that calls the native recorder methods
- **RecorderControl** – Control class for the Recorder class
- **Client** – Class to send the captured audio sample to the ‘server’ application and receive the context classification information

App extends Frame and consisting of a single panel containing the entire user interface of the application. On the top will be a simple MenuBar containing a single Menu and MenuItem used to exit the application. The ‘App Panel’ will contain a single Panel that extends from the class RecorderControl.

RecorderControl is a panel to provide the UI by which the user can control the recorder. It will be formatted with a GridBagLayout Manager containing 4 separate panels:

- A switch panel providing two buttons for enabling and disabling the recorder thread. These will call `start()` and `stop()` respectively in the recorder class.
- A settings panel providing dropdown boxes (Java Choices) to select the length of the samples to be recorded and the delay duration in between recordings. Choices require a set of pre-determined selections. Available selections for the delay time will be multiples of 10 seconds between 10 and 60 seconds. On selection of a delay time the event generated will call `setDelay()` of the Recorder class. The sample length choice will be all integer values between 1 and 5 seconds (inclusive), defaulting to 3 seconds. Selection of the sample length will call `setDuration()` in class Recorder.
- An Information panel providing a Label notifying the status of the recorder (whether it is disabled or enabled).
- The Recorder Panel (class Recorder).

The Recorder class provides the native methods to run the recorder. It will also be implemented as a panel, allowing the status of the recorder to be presented to the user. As Recorder will extend Panel, in order to create and run a Thread the class will have to implement Runnable. As well as the initialising the UI components, the constructor will call the native method `initialiseRecorder`, therefore setting up the recorder parameters on start-up. The client class is also constructed here. The class will contain five methods:

- `setDelay` Sets the day time between recordings to the value in the methods argument.
setDuration  Calls the native method setRecDuration to set the recording length to the value in the methods argument.

Start  If a recorder thread is not already running, a new thread will be created and started.

Stop  If a thread recorder is running the method will it will disable the recorder.

Run  The run method of the Thread. The method is in a while loop while recording is enabled. It calls the native method recordAudioSample to record a single at a time. After which it calls the various methods in the Client class to transmit the recorded file after each recording has been made. After completion (or failure) of transmission of the sample to the server the method ‘sleeps’ for the number of seconds set by the delay variable. As the method progresses it sets the operational statuses discussed earlier in the statusLabel as appropriate.

The Client class provides the role of transmitting the recorded audio sample from the PDA to the server and retrieving the string values of the recognised HMM classification and confidence level. The class used here is a slightly modified version of that written by Ling Ma for the environmental noise classification system. The class employs 8 methods, 3 for sending the audio file, a further 3 for retrieving the strings and two ‘getter’ methods for the sample classification and confidence values, these are:

- audioConnect and stringConnect
- audioTransfer and stringTransfer
- audioClose and StringClose
- getScene and getConfidence

The ‘connect’ and ‘transfer’ methods return boolean values indicating success or failure to enable error checking. The ‘close’ methods are void and therefore return no value. The two connect methods initialise the connection between the PDA and the server; the connections are initiated on the PDA side so the server is just a ‘listener’. This enables the server to not require ‘prior knowledge’ of the IP address to transmit the strings to. The audio sample is transferred via the audioTransfer method, this used a BufferedReader to read the source file to be transmitted, then ‘written’ to a BufferedReader obtained from the communication Socket. The String transfers will utilise a BufferedReader to read the values from the InputStream and setting appropriate local class variables, scene and confidence. The ‘close’ methods will close the communication sockets and input or output streams are required. Finally, the two getter methods are to enable the Recorder class to obtain the scene and confidence variables for displaying to the user.

The matching methods on the server involving this interaction between the two applications are discussed in section 4.2.2.

**User Interface Structure**

As Jeode Runtime is being used there is no support for Java Swing, therefore the GUI is to be implemented using the Java Abstract Window Toolkit (AWT) class libraries. Figure 1 shows the complete GUI containment hierarchy for the application.
4.2.2 Server Application

As discussed earlier the server side of the system features a noise database, noise classification and recognition. The context classification will be conducted using the Hidden Markov Model Toolkit (HTK), a toolkit developed at the Speech, Vision and Robotics Group of the Cambridge University Engineering Department as a toolkit for building and manipulating Hidden Markov Models (HMMs).

4.2.2.1 Noise Database

The noise database will not be an actual ‘database’ implemented using a Database Management System (DBMS); it will be a basic directory structure storing the recorded wave samples in a subdirectory was a particular model category. The structure is shown in Figure 1. The “data” directory is the root of the structure. Category 1, 2 and 3, etc will be named (in all lowercase) with the category for which the samples contained are related, for example “rock”. The naming of the audio samples contained in each category directory will have two conditions; a prefix of the category it belongs and must end with “.wav”. There are no other conditions for naming except the standard Windows constrains. An example of a valid name is “rock_rasmus_23.wav”. The structure will allow for any number of both category directories and any number of samples with each directory with the only restriction being the physical storage capacity.
4.2.2.2 Noise Classification and recognition

The core of the HMM classification and recognition system is that developed by Ling Ma for classifying environmental noise [ling paper]. The system uses a left-to-right HMM topology consisting of 11 states and is implemented using HTK.

HMM Creation

The creation of the HMMs will not be implemented as part of the ‘server’ application, instead relying on a user to manually start the creation process. The HMMs to be created will consist of 11 states utilising a left-to-right topology, selected as a result of experimental results in [ling paper]. HMM creation involves two main processes, feature extraction and the model training, shown in Figure 1.

For the creation of HMMs using HTK the following file types are required:

- Configuration
- Dictionary
- Labels
- Model List
- Prototype HMM definition
- Script
Task Grammar

Configuration files are used for customising the HTK working environment. They consist of a list of parameter-values pairs along with an optional prefix which limits the scope of the parameter to a specific module or tool. For this system three configuration files will be required, for feature extraction, HMM initialisation and HMM training.

![Image of HMM Diagram]

**Figure 1: Some HMM Diagram**

The configuration and HMM prototype definition files, after initial creation, will not require modification for the life of the system; however the remaining types will require customisation dependent on the number of HMMs to be created and their names, as well as the quantity of samples in the noise database and their names. Therefore these files are to be dynamically created using a Java script.

The Java class Script is to read the contents of the ‘data’ directory of the noise database and extract the directory names contained within, giving a) the number of models to be created, and b) the names of the models. From this information the dictionary, label, grammar and model list can be written in the appropriate formats. The script then finds the names of the files within each of the model directories to write the file paths into two HTK script files, one containing the source and destination file paths for feature extraction, the other containing the extracted feature paths for HMM creation and training.

The sample sampled data must be parameterised into sequences of feature vectors. Here, Mel-Frequency Cepstral Coefficients (MFCCs) will be used. This coding is to be conducted using the HTK tool, HCopy outputting the parameterised files into a ‘mfcc’ directory structure similar to that of the noise database. HCopy will require a configuration file determining the characteristics of the wave source file and the target of extracting the MFCC features.

The method of HMM creation to be used will be the ‘flat start’ approach available with the HCompV HTK tool. This approach initially makes the models for each scene equal, and then embedded training will update the models using all the training data available. HCompV only creates a new version of the proto definition. Two files are required to be created manually, a new model file containing a copy of the new prototype definition, and a global options macro file. These are to be completed using a Java seeding application written by Ling Ma for her Noise classification system. This is implemented in three classes, SeedProto, Macros and Models. Macros reads the new prototype definition and the file vFloors created by HCompV to write the macro file, while Models reads only the proto definition to create the file models containing a HMM definition for each HMM to be created. Currently this class relies on the models to be created to be explicitly defined in the code, this will be modified to read the list of models in the file created by the Script discussed earlier. SeedProto is a simple class
calling the Models and Macros. Other than the change already mentioned no modifications to these classes will be made. The model and macro files manually created are required by the HTK tool HERest. This is the method of conducting the embedded training of the HMMs. HERest is to be executed at least three times, changing the input and output directories to create the final set of initialised HMMs.

**Recognition**

The recognition system will consist of two parts; the Java application and the HTK processing.

For HTK to match a sample the features must first be extracted using the tool HCopy, as they were for the training data. The features extracted are then matched against the created HMMs to produce a text file containing the first two likelihood matches as shown in Figure 1. The first two likely matches are retrieved to enable the calculation of the confidence level (discussed later).

![HTK Recognition Diagram](image1)

*Figure 1: HTK Recognition Diagram*

To match the sample against the set of HMMs the tool HVite will be used.

![Application Recognition Diagram](image2)

*Figure 1: Application Recognition*
**Confidence level**

The confidence level is a scoring mechanism initially developed as part of an environmental noise classifier system [Acoustic environment as an indicator of social and physical context]. It works using a simple method of N-best likelihoods of the HMMs. It is computed using the following formula where \( L_1 \) is the log likelihood of the best matching model and \( L_2 \) is the second best matching model.

\[
\text{confidence} = \frac{L_1 - L_2}{L_1}
\]

**New Training Data**

The addition of the new data to the HMMs is completed by the same process of initial HMM creation, just including the new data in the database.

When a new samples are to be added to the database a string of which model it is to be added to is also retrieved from the client. This allows the server to format the file as required (adding the model name prefix and the correct file extension) before adding it to the correct directory in the database, shown in Figure 1.

*Figure 1: Application Training*
5 IMPLEMENTATION AND TESTING

The following section details the implementation of the designed system and the testing of the implemented components. Include code listings in Appendix???

- Good implementation of designs
- Logical process of implementation
- Intelligent choice of platform for implementation

5.1 Client Implementation

5.1.1 Audio Recorder

As the recorder was to be written in a different language (C++) to the controlling UI (Java), the decision was made to firstly develop the recorder as a standalone Pocket PC executable application. This was possible as the aim was to be able to run the recorder as if it was a single method, with multiple recordings simple requiring multiple calls. This gave the ability to test the code without the need to develop the UI first.

The methods initialiseRecorder and recordAudioSample (see section X) were written, and calls were made from the WinMain method (the entry point into the application). The setRecDuration method was omitted here and the sample length was temporarily ‘hard coded’, the reason being that it will be a variable set by the Java UI, which is yet to be developed and was therefore not implemented until the UI was developed. On successful implementation running the application produced a single WAVE format file containing a 3 second audio recording. Full testing later.

Once the functionality of the audio recorder code had been verified in testing, attention was turned to implementing it as a shared library for use via the JNI. The two methods implemented in the standalone application were extracted and modified to include the required JNI headers. For example Figure 1 shows the method initialiseRecorder written for use via the JNI.

```java
JNINAME void JNICALL Java_Recorder_initialiseRecorder(JNIEnv *env, jobject obj, jstring path)
{
    filePath = (char *)env->GetStringChars(path, NULL);
    // Set up the format for the WAVE audio recording
    // 22050, mono, 8 bit, PCM WAVE
    waveFormat.wFormatTag = WAVE_FORMAT_PCM;
    waveFormat.nChannels = 1;
    waveFormat.nSamplesPerSec = 22050;
    waveFormat.wBitsPerSample = 8;
    waveFormat.nBlockAlign = waveFormat.nChannels * (waveFormat.wBitsPerSample / 8);
    waveFormat.nAvgBytesPerSec = waveFormat.nSamplesPerSec * waveFormat.nBlockAlign;
    waveFormat.cbSize = 0;
}
```

Figure 1: Code Listing of JNI method to initialise the audio recorder

5.1.2 PDA Java Application

The four classes detailed in the design were implemented in the following order:

1. App
2. Recorder
3. RecorderControl
4. Client

App, the entry point to the application was chosen for implementation first. Although it could not been completed without the implementation of the other classes first, therefore not making it necessarily the logical choice for first implementation, the process of creating a main application Frame and MenuBar is a relatively simple one. The intention was to use this to test the correct functionality of the Jeode Runtime Environment on the PDA, allowing any initial issues to be resolved before the more complex parts of the application were developed.

Figure 1 shows the implemented interface running on the PDA.
5.2 Server Implementation

This section describes the physically implemented server side of the system. It should be noted that all directory paths were implemented relatively so to avoid file path related issues if changing development systems during the implementation process.

5.2.1 Noise Database

The noise database is a simple directory structure containing the audio sample to be used to create the HMMs. The structure was created as illustrated in Figure 1 as required for the experimentations completed detailed in section 6.

5.2.2 Noise Classification and recognition

5.2.2.1 HMM Creation

HTK uses some standard options in the tool commands. The following are used throughout the project:

- `-C config` Declares the configuration file “config”
- `-S script` Declared the script file “script”
- `-T x` Tool tracing. The value ‘x’ is replaced with an octal value indicating the level of tracing to be used
- `-D` Display the configuration settings

The features are extracted from samples using the HTK tool HCopy. This is invoked using a batch file with the following command line:

```
HCopy -T 1 -C config/config_mfcc -S scripts/code_mfcc.scp
```

Figure 1 is the configuration file used “config_mfcc” and Figure 1 shows a small extract of the script file “code_mfcc.scp” created by the Java class `Script`.

```
# coding audio sample set
TARGETKIND = MFCC_0_E
SOURCEKIND = WAVEFORM
SOURCEFORMAT = WAV
HNET:TRACE = 1
TARGETRATE = 100000.0
SAVECOMPRESSED = F
SAVETIMEWITHCRC = F
WINDOWSIZE = 250000.0
USEHAMMING = T
ENORMALISE = F
ZMEANSOURCE = F
FREQUENCYCOEF = 0.97
NUMCHANS = 23
CEPLIFTER = 22
NUMCEPS = 12
LOPASS = 64
```

Figure 1: Configuration file config_mfcc
Initial creation of the HMMs is conducted using the tool HCompV. This is initiated by the command line:

The HERest command uses the configuration file config_train, given in Figure 1. The options –I, –t, –H and –M were used when using HERest in addition to the standard options discussed earlier:

HCompV -T 2 -D -C config/config_initiate -o hmmdef -f 0.01 -m -S scripts/train_all.scp -M hmm0 proto/proto.11

The call to HCompV uses the configuration file config_in addition to the standard options, when using HCompV the options –o, –f, –m and –M were used:

–o sets the name of the output HMM. In this case it is called “hmmdef”
–f creates variance floor macros in a file called vFloors with values equals to 0.01 times the global variance
–m indicates HCompV to update all the HMM component means with the sample mean computed from the training files
–M sets the output HMM macro and model files in directory “hmm0”

The prototype definition from which the HMMs are to be created is “proto.11”.

The implementation of this tool revealed one a small issue. The initial creation of the HMM model was to utilise all the samples contained in the noise database, listed in the script file train_all.scp. However, when the number of samples reached 500 (ish) the HMM definition file hmmdef started to be generated incorrectly. At this time it is still unknown why this issue occurred, leading to the requirement of a workaround. A third script file, train_initiate.scp, containing a small sample of source files was created. This contains the same number of samples for each HMM classification with a maximum of 500 samples in total. The creation
of this file was added to the java Script class for automatic generation. The number of samples listed for each classification is based on the following formula:

\[
\text{No of Samples} = \frac{500}{\text{No of Models}}
\]

The command to create the initial HMMs was then modified to:

```
HCompV -T 2 -D -C config/config_initiate -o hmmdef -f 0.01 -m -S scripts/train_initiate.scp -M hmm0 proto/proto.11
```

**Seeding**

Here the classes written by Ling Ma are used, including Seed, Models and Macros. Seed calls the other two classes to generate the full untrained HMMs for each classification. To generate the models for the specific models of the system, the script was ‘hard coded’ with the model classification names. This was altered to read the list of model names in the file modellist, allowing any changes in the model to be reflected here automatically. Other than this small modification the classes are unchanged.

**Re-estimation**

First three iterations:

```
HERest -D -C config/config_train -I labels/labels.mlf -t 250.0 150.0 1000.0 -S scripts/train_all.scp -H hmm0/macros -H hmm0/models -M hmm1 modellist/modellist
```

```
HERest -D -C config/config_train -I labels/labels.mlf -t 250.0 150.0 1000.0 -S scripts/train_all.scp -H hmm1/macros -H hmm1/models -M hmm2 modellist/modellist
```

```
HERest -D -C config/config_train -I labels/labels.mlf -t 250.0 150.0 1000.0 -S scripts/train_all.scp -H hmm2/macros -H hmm2/models -M hmm3 modellist/modellist
```

```
TARGETKIND = MFCC_E_D_A
DELTAWINDOW = 3
ACCWINDOW = 2
HNET:TRACE = 2
```

*Figure 1: Configuration file config_train*

The HERest command uses the configuration file config_train, given in Figure 1. The options –I, –t, –H and –M were used when using HERest in addition to the standard options discussed earlier:

- **–I** is used to load the master label file “labels.mlf”
- **–t** sets the pruning threshold to 250.0. In the event of a pruning error, the threshold is increased by 150.0. In the event of continued errors the threshold is increased until the limit of 1000.0 is reached.
- **–H** is used to load the HMM macro and model files
- **–M** sets the output HMM macro and model files to the directory specified.

This tool requires the list of the model classifications contained in the file “modellist”.

The HERest tool is called a further 6 times to complete the training with the final trained models being written into a directory hmm9.
At this point the HMMs can be created. Full testing is detailed in section XXX.

5.2.2.2 Recognition
The recognition process has 4 parts.

1. Retrieve the audio file from the PDA
2. Match the sample against a set of created HMMs (producing a text file containing the matched model)
3. Read the text file to obtain the matched HMM and the confidence level
4. Transmit the context information back to the PDA

The recognition process to create the text file containing the context information requires to HTK command calls. The first, using HCopy is for the feature extraction as discussed in section 5.2.2.1, the second is the comparison against the created HMMs using the tool HVite. The commands are as follows:

HCopy -T 1 -C hmmdef/config/config_mfcc -S scripts/codesample.scp

HVite -D -H hmmdef/macros -H hmmdef/models -S scripts/sample.scp -C hmmdef/config/config_tr -w hmmdef/grammar/snet -i test/rep.txt -p -20.0 -s 0.0 -n 2 20 hmmdef/dictionary/dict hmmdef/modellist/modellist

5.2.2.3 New Training Data
- Server
  - Retrieve Recognise/Train flag added to server and client
  - Retrieve File for training
5.3 Testing

- Careful design of testing procedures
- Comprehensive testing of project
- Successful testing of the project
6 EXPERIMENTAL WORK

This section details the experimental work carried using the application discussed in the previous sections. The experiments are to determine if various types of music can be classified using a HMM based recognition system. Two experiments were carried out using different methods of categorising the sound samples; by music genre and by predominant instrument.

6.1 Data Collection

The experiments use single channel (mono) audio recordings sampled at 22.050 kHz using 8-bit quantisation. The recordings were collected by placing the PDA in front of a desktop PC connected to an external stereo hi-fi amplifier and loudspeaker. The sound source was played from the PC with care to ensure that the position of the PDA and the source volume remained constant for all audio samples. Any inconsistency in the volume of the sound source would be due to the variances of the recording level of the source. The PDA recorder application was run capturing three second sound samples with a delay of 5 seconds between samples. The three second sample length was chosen as a likely duration to be used as a practical application.

6.2 Experimentation and Results

6.2.1 Music Genre Classification

Five different HMMs were created for this experiment, 4 for music environments and a single model for a speech environment. Descriptions of these are in Table 2.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chamber</td>
<td>Classical music of the chamber style</td>
</tr>
<tr>
<td>Orchestra</td>
<td>Classical music consisting of symphony orchestras</td>
</tr>
<tr>
<td>Rock</td>
<td>Music of the rock genre containing predominately guitars, drums and vocals</td>
</tr>
<tr>
<td>Speech</td>
<td>Human speech</td>
</tr>
<tr>
<td>Jazz</td>
<td>Modern style jazz music</td>
</tr>
</tbody>
</table>

Table 2: Genre classification descriptions

Given the nature of music in that, to a person listening to it, certain sections of a particular song or track can sound distinctly different to other sections; to avoid confusion due to these differences a large quantity of data is required for training. For each category 2000 audio samples were collected to be used as the training data, with a further 1000 to be used as testing data. Therefore, for the five categories, a total of 15,000 audio samples were recorded for the experiment (with 10,000 examples for training and 5000 for testing). A complete listing of the sources of the training and testing data can be found in Appendix IV: Training and Test Data Sources.

The spectrograms in Figure 1 show very the samples of each category to have quite distinctive characteristics. The spectrograms show time along the abscissa with time along the ordinate. The darker regions show more energy at the particular moment in time and frequency.

The chamber sample shows horizontal lines, again towards the lower end of the frequency spectrum. This is consistent with the instruments being played at constant notes for periods of time. The orchestra sample shows similar horizontal as seen in the chamber sample, however there are more of them and cover a wider frequency range. The lines themselves would be for the same reason, and the fact that there are more of them could be explained by the larger number of instruments that make up the orchestra, each producing sounds as different
frequencies. In the speech sample the majority of the energy is focused around the lower frequencies of the spectrum with a number of spikes or pulses of energy. These spikes are due to the individual words of the spoken sentence; between each sentence there is a slight pause. There is a small pause in the sentence in this sample showed by the lack of these characteristic spikes between approximately 1.5 seconds and 2.3 seconds in the sample. The Rock sample shows completely different characteristics. This sample shows distinct vertical structures caused by the regular drum beat of the style. A similar pattern is seen in the Jazz sample; in this particular one the beats appear to be of a faster tempo and covering a wider frequency range.

It should be noted that the samples featured in Figure 1 are only examples of the differences between the categories and are not entirely representative of all the samples for each category type. Further examples of sample spectrograms for each model can be found in Appendix X. Based on this information gathered, an estimation of possible confusions between categories can be made:

- Rock and Jazz due to the regular beats of the styles.
- Orchestra and Chamber in ‘calmer’ sections of orchestral music.

The trained HMMs were tested using 1000 samples of each category. The testing provided a 61.1% success rate in correctly classifying the samples. The accuracy of each category is shown in Table 3 and the success rate ranged from 41.7% to 93.0%.

<table>
<thead>
<tr>
<th></th>
<th>Chamber</th>
<th>Orchestra</th>
<th>Rock</th>
<th>Speech</th>
<th>Jazz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chamber</td>
<td>64.5</td>
<td>19.4</td>
<td>6.9</td>
<td>1.9</td>
<td>7.3</td>
</tr>
<tr>
<td>Orchestra</td>
<td>45.5</td>
<td>41.7</td>
<td>5.3</td>
<td>3.5</td>
<td>4.0</td>
</tr>
<tr>
<td>Rock</td>
<td>4.9</td>
<td>4.4</td>
<td>52.6</td>
<td>2.2</td>
<td>35.9</td>
</tr>
<tr>
<td>Speech</td>
<td>0.0</td>
<td>0.1</td>
<td>2.3</td>
<td>93.0</td>
<td>4.6</td>
</tr>
<tr>
<td>Jazz</td>
<td>12.3</td>
<td>5.2</td>
<td>28.0</td>
<td>0.5</td>
<td>53.7</td>
</tr>
</tbody>
</table>

Overall Accuracy: 61.1%

Table 3: HMM genre classification results and confusion matrix
As expected, there is a lot of confusion between the chamber and orchestral styles of the classical genre, with the orchestral samples being classified as orchestral and chamber almost equally. Approximately half of the rock and jazz samples were classified correctly. These two classifications also showed some confusion with just over 1/3 of the rock samples recognised as jazz and just over _ of jazz classified as rock. The best classification performance was the speech samples giving an accuracy of 93%. This was expected to be the most accurate due to the spectrograms in Figure 1 showing the speech samples to have the most unique characteristics.

For a base of comparison, a human listening test was conducted using a small sample of audio recording from the same dataset. 15 subjects were employed for the test, comprising of 9 males and 6 females between the ages of 20 and 24 years, each with normal hearing and no previous experience of this kind of test. The subjects were required to listen to 25 randomly ordered audio samples (5 from each classification), which were selected from the test set at random. The confusion matrix in Table 4 shows the results of the listening tests which gave an overall accuracy of 82.4%.

The human listening test provided an overall accuracy greater than the HMM-based system by 21.3%, a substantial improvement. A number of subjects reported relative ease in classifying all categories with the exception of chamber and orchestra with many unsure as to the real difference between the two categories. This relates to the subjects lack of experience with the genre as a whole. This perceived confusion between the chamber and orchestral categories is shown in the results with the chamber samples being classified as chamber and orchestra almost equally.

<table>
<thead>
<tr>
<th></th>
<th>Chamber</th>
<th>Orchestra</th>
<th>Rock</th>
<th>Speech</th>
<th>Jazz</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chamber</td>
<td>46.7</td>
<td>53.3</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Orchestra</td>
<td>17.3</td>
<td>82.7</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Rock</td>
<td>6.7</td>
<td>1.3</td>
<td>86.7</td>
<td>2.7</td>
<td>2.7</td>
</tr>
<tr>
<td>Speech</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Jazz</td>
<td>1.3</td>
<td>2.7</td>
<td>0.0</td>
<td>0.0</td>
<td>96.0</td>
</tr>
</tbody>
</table>

*Table 4: Human listening genre classification results and confusion matrix* 

The 61.1% average accuracy of the HMM-based system indicates that the classification of music by genre using only 3 second samples has promise, although for any practical application this level of accuracy is far too low. A possible case for the misclassifications is the variable nature of music. As mentioned earlier the characteristics seen in samples from the song or track vary a lot over the course of the entire song; as a consequence the characteristics may be too varied for the HMM to model when training. This is true not only for individual songs, but also when comparing different songs and tracks from within the same genre. An example of this is Beethoven’s piano concerto and Beethoven’s string quartets, both within the chamber group, but the samples recorded look very different on the spectrogram. Based on this thought, it was decided to repeat the tests grouping the samples by the predominant instrument type, details of which are discussed in the next section.

### 6.2.2 Predominant Instrument Type Classification

Many of the individual samples recorded for the classification by music genre feature the same instruments in the foreground, for example, the predominant instrument used in some samples of Jarrett’s Trios (categorised as jazz) and in Beethoven’s piano trio’s (categorised as chamber) are both the piano. This experiment involves the re-organisation of these samples
into categories by the most predominant instrument or instrument type with the aim of improving the overall classification accuracy that the 61.1% achieved using the classification by genre.

5 HMMs were created for testing the classification by instrument type, described in Table 5. For each model a total of 180 sample were collected, 150 to be used for training the HMMs and a further 30 for testing, (giving a total of 900 samples in all). The smaller amounts for this experiment are due to the lack of available time to re-organise the samples.

<table>
<thead>
<tr>
<th>Category</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DGV</td>
<td>Drums, Guitars and Vocals</td>
</tr>
<tr>
<td>Piano</td>
<td>Music containing predominantly the piano</td>
</tr>
<tr>
<td>Saxophone</td>
<td>Music containing the saxophone in the foreground</td>
</tr>
<tr>
<td>Speech</td>
<td>Standard human speech</td>
</tr>
<tr>
<td>String</td>
<td>Music containing only string instruments</td>
</tr>
</tbody>
</table>

Table 5: Instrument classification descriptions

The testing of the trained models resulted in the confusion matrix in Table 6. The accuracy for each category varied from 70% (DGV and Piano) and 100% (Speech) giving an overall accuracy of 84%, a 22.9% improvement over the original genre classification of 61.1%.

<table>
<thead>
<tr>
<th>Category</th>
<th>DGV</th>
<th>Piano</th>
<th>Saxophone</th>
<th>Speech</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>70.0</td>
<td>0.0</td>
<td>16.7</td>
<td>6.7</td>
<td>6.7</td>
</tr>
<tr>
<td>DGV</td>
<td>0.0</td>
<td>70.0</td>
<td>0.0</td>
<td>0.0</td>
<td>30.0</td>
</tr>
<tr>
<td>Piano</td>
<td>0.0</td>
<td>10.0</td>
<td>90.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Saxophone</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Speech</td>
<td>0.0</td>
<td>3.3</td>
<td>6.7</td>
<td>0.0</td>
<td>90.0</td>
</tr>
</tbody>
</table>

Overall Accuracy: 84.0%

Table 6: HMM instrument classification results and confusion matrix

As with the genre classification, a human listening test was conducted for the classification by instrument, conducted at the same time as the human listening test previously discussed, therefore completed by the same 15 subjects under the same conditions. The test produced an accuracy of 100% by this method of classification, shown in Table 7. This again is an improvement (17.6%) over the human listening test of classification by genre, resulting in an accuracy of 82.4%. Numerous subjects commented this method of classification being much easier to determine with confidence.

<table>
<thead>
<tr>
<th>Category</th>
<th>DGV</th>
<th>Piano</th>
<th>Saxophone</th>
<th>Speech</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>DGV</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Piano</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Saxophone</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Speech</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
<tr>
<td>String</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
<td>100.0</td>
</tr>
</tbody>
</table>

Overall Accuracy: 100.0%

Table 7: Human listening instrument classification results and confusion matrix
7 EVALUATION AND CONCLUSIONS

This section describes a detailed evaluation of the progress and success of the work conducted and draws some conclusions from this. Suggestions of further work to both improve aspects of the project completed and to extend the project further are made here.

7.1 Evaluation

- Evaluation of the success of the project
- Relation to initial aims and objectives

7.2 Conclusions

- Good conclusion of completed project in relation to the wider field
- Explicit mention of key successes in the project

7.3 Further Work

- Logical suggestion for further work

Experimentation yealded results of 61% accuracy which is too low for and practical applications. Work should be done on increasing the accuracy.

- Application Improvements
  - File Training transmission – backup buffer maybe to transfer more than one file in a single connection?
  - Recording one large file and breaking down on server side

- Experimental
  - Experimentation using LFPC features [see nwe paper!]
  - Specialising the model classes. [see nwe paper]
  - Varying amounts of training data for each model.
REFERENCES


[12] Schilit, B and Theimer, M. (1994); Disseminating Active Map Information to Mobile Hosts; IEEE Network; 8(5); p22-32


[14] Dey, A. K. (2001); Understanding and Using Context; in: Personal and Ubiquitous Computing, Volume 5, Number 1; pp4-7


[23] Schmidt, A., et al. (1998); There is More to Context than Location; in: Proceedings of Workshop on Interactive Applications of Mobile Computing (IMC’98); Rostock, Germany; Neuer Hochschulschriftverlag


[26] Li, M. and Sleep, R. (2004); Improving Melody Classification by Discriminant Feature Extraction and Fusion; in: 5th International Conference on Music Information Retrieval (ISMIR)

[27] West, K. and Cox, S. (2004); Features and Classifiers for the Automatic Classification of Musical Audio Signals; ISMIR 2004

[28] McKay, C. and Fujinaga, I. (2004); Automatic Genre Classification Using Large High-Level Musical Feature Sets; ISMIR 2004

[29] Nwe, T. L. and Wang, Y. (2004); Automatic Detection of Vocal Segments in Popular Songs; ISMIR 2004


Further Reading

APPENDIX I: HMM PROTOTYPE DEFINITION

```
<BeginHMM>
<NumStates> 11 <VecSize> 39 <MFCC_E_D_A>

<State> 2
<Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 3
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 4
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 5
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 6
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 7
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 8
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 9
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 10
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

-State> 11
-Variance> 39
<Mean> 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0 0.0
<TransP> 11

<EndHMM>
```
10  APPENDIX II: CLASS DIAGRAMS

10.1 PDA Application Class Diagram

**App**

- panel : Panel
- recorderControl : RecorderControl

+ generateMenu() : MenuBar

**RecorderControl**

- recorder : Recorder
- control : Panel
- settings : Panel
- info : Panel
- enableRecorder : Button
- disableRecorder : Button
- delayChooser : Choice
- durationChooser : Choice
- recorderStatusLabel : Label
- delay : int
- duration : int

+ itemStateChanged(event : ItemEvent) : void

**Client**

- audioSocket : Socket
- stringSocket : Socket
- host : String
- audioPort : int
- stringPort : int
- fr : File
- fis : FileInputStream
- in : BufferedInputStream
- out : BufferedOutputStream
- reader : BufferedReader
- scene : String
- confidence : String

+ audioConnect() : boolean
+ stringConnect() : boolean
+ audioTransfer(filePath : String) : boolean
+ stringTransfer() : boolean
+ audioClose() : void
+ stringClose() : void
+ getScene() : String
+ getConfidence : String

**Recorder**

- client : Client
- result : String
- status : String
- delay : int
- duration : int
- count : int
- resultLabel : Label
- recordingCountLabel : Label
- statusLabel : Label
- sceneLabel : Label
- confidenceLabel : Label

+ run() : void
+ start() : boolean
+ stop() : boolean
+ setDelay(delay : int) : void
+ setDuration(duration : int) : void
10.2 Server Application Class Diagram

![Class Diagram](image)

- **Server**
  - audioServerSocket : ServerSocket
  - stringServerSocket : ServerSocket
  - audioSocket : Socket
  - stringSocket : Socket
  - audioPort : int
  - stringPort : int
  - out : BufferedOutputStream
  - out2 : PrintWriter
  - in : BufferedReader
  - fos : FileOutputStream
  - file : File
  - clientIP : String
  - scene : String
  - confidence : String
  - fmt : DecimalFormat
  - +audioConnect() : void
  - +stringConnect() : void
  - +audioTransfer() : void
  - +stringTransfer() : void
  - +audioClose() : void
  - +stringClose() : void

- **Recogniser**
  - reader : ResultReader
  - scene : String
  - confidence : double
  - +recognise() : void
  - +getScene() : String
  - +getConfidence() : double

- **ResultReader**
  - firstScene : String
  - secondScene : String
  - firstLog : double
  - secondLog : double
  - confidence : double
  - tempScene : String
  - tempLog : double
  - +readOneLine() : void
  - +readResult() : void
  - +getScene() : String
  - +getConfidence() : double

- **HTKParser**
  - rt : Runtime
  - fileName : String
  - +get() : void

- **GeneralFileReader**
  - st : StringTokenizer
  - reader : BufferedReader
  - filename : String
  - +readLine() : void
  - +close() : void
APPENDIX III: AUDIO SAMPLE SPECTROGRAMS

11.1 Chamber

11.2 Orchestra

11.3 Rock

11.4 Speech

11.5 Jazz
12 APPENDIX IV: TRAINING AND TEST DATA SOURCES

[What about the copyright protection for recording the data??]

12.1 Training Data

12.1.1 Chamber

- Beethoven – The Late String Quartets III

  String Quartet No 14 in C sharp minor, Op 131

  1. I Adagio ma non troppo e molto espressivo
  2. II Allegro molto vivace
  3. III Allegro moderato
  4. IV Andante ma non troppo e molto cantabile
  5. V Presto
  6. VI Adagio quasi un poco andante
  7. VII Allegro

  String Quartet No 15 in A minor, Op 132

  8. I Allegro sostenuto – Allegro
  9. II Allegro ma non tanto
  10. III Molto adagio
  11. IV Alla Marcia, assai vivace
  12. V Allegro appassionato

- Bach – Violin Concertos – Concerto For Oboe & Violin

  Violin Concerto in E, BWV 1042

  1. Allegro
  2. Adagio
  3. Allegro assai

  Violin Concerto in A minor, BWV 1041

  4. (Allegro)
  5. Andante
  6. Allegro assai

  Concerto for 2 Violins in D minor, BWV 1043

  7. Vivace
  8. Largo, ma non tanto
  9. Allegro
Concerto for Violin and Oboe in D minor, BWV 1060

10. Allegro
11. Adagio
12. Allegro

• Beethoven – Piano Trios Volume 4

Trio in B Flat Major, Op. 97 ("Archduke Trio")

1. Allegro moderato
2. Scherzo: Allegro
3. Andante cantabile ma però con moto
4. Allegro moderato – Presto

Kakadu Variations in G Major, Op. 121a

5. Adagio assai – Allegretto

6. Allegretto in B Flat Major, WoO 39
12.1.2 Orchestra

- **Haydn – Famous Symphonies, Vol. 1**
  
  Capella Istropolitana, Barry Wordsworth

  *Symphony No. 82 in C Major ‘The Bear’*
  
  1. Vivace assai
  2. Allegretto
  3. Menuet
  4. Finale: Vivace

  *Symphony No. 96 in D Major ‘The Miracle’*
  
  5. Adagio – Allegro
  6. Andante
  7. Menuetto. Allegretto
  8. Finale: Vivace assai

  *Symphony No. 100 in G Major ‘Military’*
  
  9. Adagio – Allegro
  10. Allegretto
  11. Menuet: Moderato
  12. Finale: Presto

- **BBC Legends – Menuhin Rostropovich**

  *Johannes Brahms*

  *Concerto for Violin, Violonchello and Orchestra in A minor, Op. 102*
  
  1. I Allegro
  2. II Andante
  3. III Vivace non troppo

  *Felix Mendelssohn*

  *Concerto for Violin and Orchestra in E minor, Op. 64*
  
  1. I Allegro molto appassionato
  2. II Andante
  3. III Allegro non troppo – Allegro molto vivace

  *Johann Sebastian Bach*

  *Concerto for Violin, Strings and Continuo in E major, BWV1042*
  
  1. I Allegro
  2. II Adagio
  3. III Allegro assai
• Mozart – Divertimenti K. 131 & K. 287

*Divertimento in B Flat Major, K. 287*

1. Allegro
2. Andante grazioso con variazioni
3. Menuetto
4. Adagio
5. Menuetto
6. Andante – Allegro molto

*Divertimento in D Major, K. 131*

7. Allegro
8. Adagio
9. Minuetto
10. Allegretto
11. Menuetto
12. Adagio – Allegro molto – Allegro assai

• The World of the Symphony – Felix Mendelssohn

*Symphony No. 4 in A major, Op. 90 “Italian”*

1. Allegro vivace
2. Andante con moto
3. Con moto moderato
4. Saltarello presto

*Symphony No. 3 in A minor, Op. 56 “Scottish”*

5. Andante con moto
6. Allegro un poco agitato – Andante come prima
7. Vivace non troppo
8. Adagio
9. Allegro vivacissimo
10. Allegro maestoso assai
12.1.3 Jazz

- Miles Davis – Kind of Blue
  
  Columbia/Legacy – CK 64935

  Sony Jazz
  
  1. So What
  2. Freddie Freeloader
  3. Blue In Green
  4. All Blues
  5. Flamenco Sketches
  6. Flamenco Sketches (Alternate Take)

- Thelonious Monk – The Paris Concert
  
  1. Lulu’s Back In Town
  2. Just A Gigolo
  3. I Am Getting Sentimental Over You
  4. Sweet And Lovely
  5. Off Minor
  6. Crepuscule With Nellie
  7. Epistrophy

- Michel Petrucciani – Both Worlds
  
  Dreyfus
  
  1. 35 Seconds Of Music And More
  2. Brazilian Like
  3. Training
  4. Colors
  5. Petite Louise
  6. Chloé Meets Gershwin
  7. Chimes
  8. Guadeloupe
  9. On Top Of The Roof

- Charlie Parker – Bird Lives!

  Charlie Budget
  
  1. Move
  2. Ornithology
  3. Out Of Nowhere
  4. Hot House
  5. How High The Moon
6. Bebop  
7. Scrapple From The Apple  
8. Street Beat  
9. ‘Round Midnight  
10. Koko  
11. Groovin’ High

**Art Blakey And The Jazz Messengers – Moanin’**

1. Moanin’  
2. Soulful Mr. Timmons  
3. My Deal  
4. Free For All  
5. Angel Eyes  
6. Bitter Dose  
7. Jodi  
8. Wheel Within A Wheel  
9. Gypsy

**Count Basie – Vol. 1**

1. Clap Hands, Here Comes Charlie  
2. Tickel Toe  
3. Broadway  
4. Jump The Blues Away  
5. Harvard Blues  
6. It’s Sand, Man!  
7. Ain’t It The Truth?  
8. For The Good Of Your Country  
9. High Tide  
10. Queer Street  
11. Rambo  
12. Stay Cool  
13. Stay On It  
14. Golden Bullet  
15. Song Of The Islands  
16. One O’clock Jump  
17. Tootsie
12.1.4  Rock

• The Killers – Hot Fuss

Lizard King – LIZARD011

1. Jenny Was a Friend of Mine
2. Mr. Brightside
3. Smell Like You Mean It
4. Somebody Told Me
5. All These Things That I’ve Done
6. Andy, You’re a Star
7. On Top
8. Glamorous Indie Rock & Roll
9. Believe Me Natalie
10. Midnight Show
11. Everything Will Be Alright

• Stereophonics – Proformance & Cocktails

V2 – VVR1004492

1. Roll Up and Shine
2. The Bartender and the Thief
3. Hurry Up and Wait
4. Pick A Part That’s New
5. Just Lookin
6. Half the Lies You Tell Ain’t True
7. I Would Believe Your Radio
8. T-Shirt Sun Tan
9. Is Yesterday, Tomorrow, Today?
10. A Minute Longer
11. She Takes Her Clothes Off
12. Plastic California
13. I Stopped To Fill My Car Up

• The Rasmus – Dead Letters

Playground Music - 9818057

1. In The Shadows
2. Guilty
3. First Day of My Life
4. Standing Still
5. In My Life
6. Time To Burn
7. Not Like the Other Girls
8. The One I Love
9. Back in the Picture
10. Funeral Song
11. F-F-F-Falling

• **Razorlight – Up All Night**

**Mercury Records Limited - 8966804**

1. Leave Me Alone
2. Rock N Roll Lies
3. Vice
4. Up All Night
5. Which Way Is Out
6. Rip It Up
7. Don’t Go Back to Dalston
8. Golden Touch
9. Stumble and Fall
10. In the City
11. To the Sea
12. Fall, Fall, Fall
13. If You Ever
14. What Ever

• **Queen – Greatest Hits**

**EMI – 0777 7 89504 2 4**

1. Bohemian Rhapsody
2. Another One Bites The Dust
3. Killer Queen
4. Fat Bottomed Girls
5. Bicycle Race
6. You’re My Best Friend
7. Don’t Stop Me Now
8. Save Me
9. Crazy Little Thing Called Love
10. Somebody To Love
11. Now I’m Here
12. Good Old-Fashioned Lover Boy
13. Play The Game
14. Flash
15. Seven Seas of Rhye
16. We Will Rock You
17. We Are The Champions
• Queen – Greatest Hits II

EMI – 0777 7 97971 2 7

1. A Kind of Magic
2. Under Pressure
3. Radio Ga Ga
4. I Want It All
5. I Want To Break Free
6. Innuendo
7. It’s A Hard Life
8. Breakthru
9. Who Wants To Live Forever
10. Headlong
11. The Miracle
12. I’m Going Slightly Mad
13. The Invisible Man
14. Hammer To The Fall
15. Friends Will Be Friends
16. The Show Must Go On
17. One Vision

• Eve6 – Horrorscope

RCA

1. Rescue
2. Promise
3. On the Roof Again
4. Sunset Strip Bitch
5. Here’s to the Night
6. Amphetamines
7. Enemy
8. Nocturnal
9. Jet Pack
10. Nightmare
11. Bang
12. Girl Eyes
12.1.5 Speech

• BBC Radio 4
  1. Broadcasts between XXX and XXX
  2. Broadcasts between XXX and XXX

• BBC Radio 5 Live
  1. Broadcasts between XXX and XXX
  2. Broadcasts between XXX and XXX
12.2 Testing Data

12.2.1 Chamber

- Beethoven – Chamber Music for Horns, Wind and Strings
  Naxos – 8.553090
  *Septet in E Flat Major for Violin, Viola, Chello, Double Bass, Clarinet, Horn and Bassoon, Op. 20*
  1. Adagio – Allegro con brio
  2. Adagio cantabile
  3. Tempo di Menuetto
  4. Tama con veriazioni: Andante
  5. Scherzo: Allegro molto e vivace
  6. Andante con moto alla Marcia – Presto

Quintet in E Flat Major for three Horns, Oboe and Bassoon, H. 19
  7. Allegro
  8. Adagio maestoso
  9. Minuetto

Sextet in E Flat Major for two Horns and String Quartet, Op. 81b
  10. Allegro con brio
  11. Adagio
  12. Rondo

- Beethoven – The Late String Quartets II

HMV Classics – HMV 572839 2
  *String Quartet No 13 in B flat, Op 130*
  1. I Adagio ma non troppo – Allegro
  2. II Presto
  3. III Andante con moto, mar non troppo
  4. IV Alla danza tedesca (Allegro assai)
  5. V Cavatina (Adagio molto espressivo)

Grosse Fuge in B flat, Op 133
  6. VI Finale (Allegro)

- Beethoven – Sonatas for Chello and Piano

Naxos – 8.550478
  *Sonata in C Major, Op. 102, No. 1*
1. Andante –
2. Allegro vivace
3. Adagio – Tempo d’Andante –
4. Allegro vivace

**Sonata in D Major, Op. 102, No 2**
5. Allegro con brio
6. Adagio con molto sentimento d’affetto
7. Allegro fugato

**Sonata in A Major, Op. 69**
8. Allegro, ma non tanto
9. Scherzo: Allegro molto
10. Adagio cantabile
11. Allegro vivace
12.2.2  Orchestra

•  
Beethoven – Violin Concerto-Romances Nos. 1 & 2

Privilege – 427 197-2

Concerto for Violin and Orchestra in D major, Op. 61

1. Allegro ma non troppo
2. Larghetto
3. Rondo. Allegro
4. Romance for Violin and Orchestra no. 1 in G major, Op. 40
5. Romance for Violin and Orchestra no. 2 in F major, Op. 50

•  
Beethoven – Symphonies Nos. 1 & 4

Deutsch Grammophon/Galleria – 419048-2

Symphony no. 1 in C major, Op. 60

1. Adagio molto – Allegro con brio
2. Andante cantabile con moto
3. Menuetto. Allegro molto e vivace
4. Adagio – Allegro molto e vivace

Symphony no. 4 in B flat, Op. 60

5. Adagio – Allegro vivace
6. Adagio
7. Allegro vivace
8. Allegro ma non troppo

Overture

9. Egmond Op. 84 – Sostenuto, ma non troppo - Allegro

•  
Beethoven – Symphonies Nos. 7 & 8

EMI Classics – 7243 5 57570 2 9

Symphony no. 7 in A major, Op. 92

1. Poco sostenuto – Vivace
2. Allegretto
3. Presto – Assai meno presto
4. Allegro con brio

Symphony no. 8 in F major, Op. 93

5. Allegro vivace e con brio
6. Allegretto scherzando
7. Tempo di menuetto
8. Allegro vivace
12.2.3  Jazz

- Jan Garbarek – Madar
  ECM – ECM 1515 519075-2
  1. Sull lill
  2. Madar
  3. Sebika
  4. Bahia
  5. Ramy
  6. Jaw
  7. Jaron
  8. Qaws
  9. Epilogue

- John Coltrane – Giant Steps
  Atlantic Masters – 8127 3610-2

  Original Album
  1. Giant Steps
  2. Cousin Mary
  3. Countdown
  4. Spiral
  5. Syeeda’s Song Flute
  6. Naima
  7. Mr P.C.

  Bonus Tracks
  8. Giant Steps (Alternate Version 1)
  9. Naima (Alternate Version 1)
  10. Cousin Mary (Alternate Take)
  11. Countdown (Alternate Take)
  12. Syeeda’s Song Flute (Alternate Take)
  13. Giant Steps (Alternate Version 2)
  14. Naima (Alternate Version 2)
  15. Giant Steps (Alternate Take)

- Dave O’Higgins – All Good Things
  EFZ – EFZ1002
  1. Buzz
  2. Waltz For Anna Christina
  3. No Prizes For Guessing
  4. Incidentally
5. Lost In The Post
6. Ev’ry Time We Say Goodbye
7. Dear Lord
8. Quids In
9. All Good Things Come To An End

• Keith Jarrett Trio – Standards in Norway

ECM Recordings

*(Keith Jarrett, Gary Peacock, Jack DeJohnette)*

1. All Of You
2. Little Girl Blue
3. Jist In Time
4. Old Folks
5. Love Is A Many-Splendored Thing
6. Dedicated To You
7. I Hear A Rhapsody
8. How About You?

• Keith Jarrett – Staircase

ECM Recordings

1. Staircase (Part 1)
2. Staircase (Part 2)
3. Staircase (Part 3)
4. Hourglass (Part 1)
5. Hourglass (Part 2)
6. Hourglass (Part 3)
7. Sundial (Part 1)
8. Sundial (Part 2)
9. Sundial (Part 3)
10. Sand (Part 1)
11. Sand (Part 2)
12. Sand (Part 3)
12.2.4 Rock

- Third Eye Blind - Third Eye Blind

Elektra – 7559-62012-2

1. Losing A Whole Year
2. Narcolepsy
3. Semi-Charmed Life
4. Jumper
5. Graduate
6. How’s It Gonna Be
7. Thanks A Lot
8. Burning Man
9. Good For You
10. London
11. I Want You
12. The Background
13. Motorcycle Drive By
14. God Of Wine

- The Darkness – Permission To Land

Atlantic – 5050466-7452-2-4

1. Black Stuff
2. Get Your Hands Off My Woman
3. Growing On Me
4. I Believe In A Thing Called Love
5. Love Is Only A Feeling
6. Givin’ Up
7. Stuck In A Rut
8. Friday Night
9. Love On The Rocks With No Ice
10. Holding My Own

- Sum 41 – All Killer No Filler

1. Introduction To Destruction
2. Nothing My Back
3. Never Wake Up
4. Fat Lip
5. Rythmus
6. Motivation
7. In Too Deep
8. Summer
9. Handle This
10. Crazy Amanda Bunkface
11. All She’s Got
12. Heart Attack
13. Pain For Pleasure

• Snow Patrol – Final Straw

Polydor - 9866089

1. How To Be Dead
2. Wow
3. Gleaming Auction
4. Whatever’s Left
5. Spitting Games
6. Chocolate
7. Run
8. Grazed Knees
9. Ways & Means
10. Tiny Little Fractures
11. Somewhere A Clock Is Ticking
12. Same
13. We Can Run Away Now They’re All Dead and Gone
14. Half the Fun

• Savage Garden – Affirmation

Columbia – 494935 2

1. Affirmation
2. Hold Me
3. I Knew I Loved You
4. The Best Thing
5. Crash and Burn
6. Chained To You
7. The Animal Song
8. The Lover After Me
9. Two Beds and A Coffee Machine
10. You Can Still Be Free
11. Gunning Down Romance
12. I Don’t Know You Anymore
12.2.5 Speech

- BBC Radio 4
  1. Something here…
  2.