UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

SISTEMA PARA CLASSIFICAÇÃO DE OBJETOS SONOROS

ANDRÉ FELIPE DE ALMEIDA

BLUMENAU
2013

2013/1-07

ANDRÉ FELIPE DE ALMEIDA

SISTEMA PARA CLASSIFICAÇÃO DE OBJETOS SONOROS

Trabalho de Conclusão de Curso submetido à Universidade Regional de Blumenau para a obtenção dos créditos na disciplina Trabalho de Conclusão de Curso II do curso de Ciência da Computação — Bacharelado.

Prof. Aurélio Faustino Hoppe, Mestre – Orientador

BLUMENAU
2013

2013/1-07

SISTEMA PARA CLASSIFICAÇÃO DE OBJETOS SONOROS

Por

ANDRÉ FELIPE DE ALMEIDA

Trabalho aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II, pela banca examinadora formada por:

	_____
Presidente:	Prof. Aurélio Faustino Hoppe, Mestre – Orientador, FURB
	_____
Membro:	Prof. Dalton Solano dos Reis, Mestre – FURB
	_____
Membro:	Prof. Rion Brattig Correia, Mestre – FURB

Blumenau, 08 de Julho de 2013

Dedico este trabalho a todos aqueles que me ajudaram diretamente na realização deste.

AGRADECIMENTOS

A minha mãe pelo incentivo.

Aos meus irmãos, pelo exemplo.

Sábio é quem não se aflige com o que lhe falta
e se alegra com o que possui.

Demócrito

RESUMO

Para a identificação de objetos pelo som são necessárias algumas etapas de processamento computacional, sendo: aquisição do áudio a ser gravado, extração das características, registro das informações coletadas e identificação dos dados informados. Este trabalho apresenta um software para efetuar a identificação de objetos utilizando os sons por eles produzidos. A obtenção do áudio é feita através do uso de microfone ou de arquivos de áudio. A extração das características do som é feita utilizando transformada de Fourier. Para a identificação dos objetos são utilizados algoritmos de similaridades. Os resultados demonstram que a abordagem escolhida não trouxe bons resultados. Apesar dos algoritmos de similaridades terem apresentado bons resultados no reconhecimento de músicas, a comparação de características apresenta baixa precisão, considerando a quantidade de características extraídas.

Palavras-chave: Identificação de objetos. WAV. *Audio fingerprint*. Transformada de Fourier.

ABSTRACT

To identify objects by sound requires some computational processing steps, as follows: acquisition of audio to be recorded, feature extraction, registration and identificate information collected from the data reported. This paper presents a software to perform the identification of objects using the sounds they produce. Obtaining audio is done through the use of microphone or audio files. The sound feature extraction is performed using Fourier transform. For identificating objects similarities algorithms are used. The results demonstrate that the approach chosen have not brought good results. Despite the similarities algorithms have shown good results in music recognition, comparison of features has low accuracy, considering the amount of extracted features.

Key-words: Object identification. WAV. Audio fingerprint. Fourier transform.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE SIGLAS

API – *Application Programming Interface*

DFT – *Discrete Fourier Transform*

DTFT – *Discrete Time Fourier Transform*

EA - *Enterprise Architect*

FT – *Fourier Transform*

FFT – *Fast Fourier Transform*

RF – Requisito Funcional

RNF – Requisito Não Funcional

UML – *Unified Modeling Language*

SUMÁRIO

1 INTRODUÇÃO

Atualmente, a segurança pública nacional encontra-se em situação crítica, principalmente pelo elevado índice da criminalidade em todo o país. Estudo realizado em 2010 pelo Ministério da Saúde confirma esta situação, isso porque "a taxa de morte por homicídios no Brasil aumentou aproximadamente 32% em 15 anos" (VEJA, 2010).

Este é um dos principais motivos que obriga a sociedade a buscar formas de proteção, seja em sua casa, trabalho ou até mesmo em recintos comerciais, os quais têm a maior necessidade em atrelar segurança/proteção aos seus clientes.

Com base nisso, grandes avanços vem sendo alcançados na tecnologia voltada para a área de segurança, da qual se destaca os Circuitos Fechados de Televisão (CFTV), que têm por objetivo monitorar os ambientes e registrar toda movimentação dos locais de interesse (CARLASSARA, 2011, p. 14).

Apesar dos registros gerados pelo CFTV serem de grande importância para a área de segurança, possui alguns malefícios em relação ao armazenamento de dados, pois permanecem em funcionamento durante muito tempo e registrando todas as atividades cotidianas, as quais por muitas vezes são irrelevantes. Se isso já não bastasse, torna-se prudente o uso de equipamento de custo elevado, considerando que a imagem gerada pelo sistema deverá ser de alta qualidade. Como alternativa para essa situação, pode ser feito o registro do vídeo em qualidade baixa e com uma taxa de frames configurada para o mínimo necessário para reconhecimento facial, contudo, prejudicando a almejada proteção.

Os sensores de movimentos e os detectores de imagem são exemplos de tecnologias importantes para a melhoria de todo o sistema, pois apresentam bons resultados, seja por ser capaz de detectar a presença em condições diversas de luz no primeiro caso e a identificação de pessoas no segundo, assim, ambos descartam registros que não trarão informações relevantes.

Por esses motivos, se torna importante o estudo de novas formas de filtrar a informação captada mantendo a quantidade de informação gravada por qualidade da mídia da informação.

Portanto, como alternativa aos métodos já adotados, este trabalho propõe a identificação por objetos sonoros, pois na maioria das vezes, o(s) objeto(s) de interesse estão associados ao som ou ruído ocorridos no ambiente.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é desenvolver um sistema que classifique os objetos sonoros através do áudio produzido pelo seu impacto no ambiente.

Os objetivos específicos do trabalho são:

a) capturar e gravar o som ocorrido no ambiente para posterior classificação;
b) gerar um modelo de representação baseado nas características estruturais do áudio (áudio *fingerprints*);
c) implementar técnicas de similaridade para identificar objetos sonoros.

1.2 ESTRUTURA DO TRABALHO

Este trabalho encontra-se dividido em 4 capítulos. O segundo capítulo apresenta a fundamentação teórica que dá embasamento ao desenvolvimento. Aborda técnicas de extração de caracteristicas de áudio conhecidas como *fingerprints* e algoritmos de comparação. Após isso apresenta as informações relevantes a respeito dos trabalhos correlatos.

O terceiro capítulo apresenta os mecanismos utilizados para a implementação do sistema. São descritos os casos de uso implementados, os diagramas de classe e diagramas de sequência. Após a apresentação destas informações, são discutidos os resultados obtidos.

O quarto capítulo apresenta as conclusões obtidas durante o desenvolvimento do trabalho e as sugestões de trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Nas seções seguintes são apresentados os conceitos referentes a classificação e extração de dados provenientes de sons capturados. A seção 2.1 fundamenta o que são fingerprints de áudio. A descrição sobre Transformada de Fourier é apresentada na seção 2.2. Na seção 2.3 são apresentados os trabalhos correlatos.

2.1 ÁUDIO FINGERPRINTING

Um *fingerprint* de áudio é uma assinatura compacta baseada em conteúdo que resume uma gravação de áudio (CANO et al., 2005, p. 23). Uma f*ingerprint* de áudio é um resumo digital, deterministicamente gerado com um sinal de áudio, que pode ser utilizado para identificar uma amostra ou localizar rapidamente itens semelhantes em uma base de dados de áudio.

Tecnologias de *fingerprint* de áudio extraem as características acústicas mais relevantes de um pedaço de áudio e armazenam estas informações em uma base de dados. Quando um trecho de áudio desconhecido é apresentado, estas características são extraídas e são comparadas com as informações já persistidas em base.

A técnica de *fingeprint* trabalha diferente da técnica de marca d'água embutida no áudio, onde uma marca d'água é embutida no arquivo sem alterar a qualidade do áudio. Enquanto que a marca d'água precisa ser inserida manualmente para um motivo específico, as características de *fingerprint* podem ser extraídas de qualquer áudio, sem prévias modificações.

Segundo Larsen (2006, p. 12), os seguintes aspectos são desejados e alguns destes fortemente interconectados para *fingerprints* são:

a) compacto: *fingerprints* devem ser compactos para serem facilmente armazenados e pesquisáveis;
b) posicionamento espacial: a distância Euclidiana define o quão similar são os áudios;
c) robusto: ruídos de intensidade fraca a média não devem afetar o resultado obtido da *fingerprint* significativamente. Se a *fingerprint* não é robusta, na comparação este resultará em um falso negativo;
d) granularidade: o quanto de áudio é necessário para construir uma *fingerprint*. As características devem ser obtidas de um som completo ou de apenas parte dele;
e) destrutividade: uma *fingerprint* não deve ser passível de ser reconstruída para o seu áudio original.

2.2 TRANSFORMADA DE FOURIER

O áudio como conhecemos, em seu formato bruto é conhecido como domínio de tempo. Para se extrair características deste precisa-se ter o áudio no domínio de frequência. O termo “domínio da frequência” é utilizado para descrever as amplitudes das ondas de seno e cosseno (Bunnel, 1996a).

A transfomada de Fourier tem como objetivo tornar um sinal com representação no domínio tempo e representá-lo no domínio frequência. Isto é feito separando todas as frequências que aparecem no sinal do domínio tempo (Mello, 2012).

Existem quatro tipos distintos de transformada de Fourier, resultando em quatro tipos básicos de sinais que podem ser encontrados. Esses tipos Segundo Smith (1997), podem ser contínuos ou discretos e periódicos ou aperiódicos.

a) aperiódico-contínuo: inclui exponenciais decadentes e curva de Gauss. Estes sinais se extendem para infinito positivo e negativo sem repetir em um padrão periódico. A transformação para este tipo de sinal é chamada de Transformação de Fourier (FT);
b) periódico-contínuo: incluem ondas senoidais, ondas quadradas, e qualquer forma de onda que se repete em um padrão regular de negativo ao infinito positivo. Esta versão da transformada de Fourier é chamado de Série de Fourier;
c) aperiódico-discreto: são definidos apenas em pontos discretos entre o infinito positivo e negativo, e não se repetem de forma periódica. Este tipo de transformada de Fourier é chamado de Transformação Discreta de Tempo de Fourier (DTFT);
d) periódico-discreto: são sinais discretos que se repetem de forma periódica de negativo ao infinito positivo. Esta classe de transformada de Fourier é às vezes chamado a série de Fourier discreta, mas é mais freqüentemente chamada de transformada discreta de Fourier (DFT).

Estes quatro tipos de sinais se estendem ao infinito positivo e negativo. Ondas de seno e cosseno são definidas como uma extensão do infinito negativo ao infinito positivo. Não é possível utilizar um grupo de sinais infinitamente longos para sintetizar algo finito de comprimento. A solução para contornar este é fazer os dados finitos se parecerem com uma onda de sinal infinito. Isto é feito imaginando que o sinal tem um número infinito de amostras, à esquerda e à direita dos pontos reais. Se todas estas amostras imaginárias têm um valor zero, o sinal parece discreto e aperiódico aplicando-se assim a DTFT. Como alternativa,

as amostras imaginárias podem ser uma duplicação dos pontos reais. Neste caso, o sinal parece discreto e contínuo, exigindo a utilização da DFT.

Um número infinito de sinusóides são necessários para sintetizar um sinal aperiódico. Isso torna impossível calcular o DTFT em um algoritmo de computador. Por eliminação, a única transformada de Fourier restante, que pode ser utilizada é a DFT. Em outras palavras, os computadores digitais só podem funcionar com a informação que é discreta e de comprimento finito.

Cada uma das quatro transformadas de Fourier pode ser subdividida em versões reais e complexas. A versão real é a mais simples, utilizando-se números ordinários e álgebra para a sintese e decomposição. As versões mais complexas das quatro transformadas de Fourier são mais complicadas, exigindo a utilização de números complexos. Estes números são, tais como: `1 + 2i`, onde i é igual a √−1.

O termo matemático transformação, é amplamente utilizado em processamento digital de sinais. Uma transformada é uma função. Uma função é um algoritmo ou procedimento que altera um valor para outro valor. Por exemplo, `y = x + 2` é uma função. Você escolhe algum valor para `x`, liga-o na equação, e aparece um valor para `y`. As funções também podem alterar vários valores em um único valor, tais como: `y = 2a + 3b + 4c`, onde `a`, `b` e `c` são alteradas em `y`.

2.3 TRABALHOS CORRELATOS

Esta seção apresenta trabalhos relacionados ao reconhecimento e identificação de objetos sonoros. Os trabalhos relacionados, em sua grande maioria, apresentam como característica mais importante o fato de serem utilizados para o reconhecimento de músicas. Dentre os trabalhos encontrados pode-se citar: “Mecanismo de Busca Semântica de Áudio” (DOROW, 2011), “Similaridade de Músicas Baseado no Conteúdo” (WIGNALL, 2003) e “Pesquisa Baseada em Áudio: Do *Fingerprinting* à Recuperação Semântica do Áudio” (CANO, 2006).

O trabalho desenvolvido por Dorow (2011) trata do reconhecimento de músicas, utilizando a técnica de áudio *fingerprint* e similaridade. O primeiro passo trata da obtenção do áudio, oriundo da gravação de um microfone. Após receber o áudio bruto, utiliza a transformada de Fourier para extração de características do áudio, gera os *fingerprints* e submete para a comparação com outros áudios previamente cadastrados em uma base de dados. Os resultados indicam que as técnicas de similaridade adotadas por Dorow são suficientes para obter resultados positivos. Dorow (2011) conclui que o tamanho do trecho de

áudio influência na eficiência do algoritmo. Ou seja, quanto maior for o trecho de áudio informado como entrada para o mecanismo de busca, mais características serão extraídas para fins de comparação.

Wignall (2003) baseou-se na técnica de identificação de semelhanças entre audios para criar um sistema de classificação de músicas por gênero. Com este propósito, ele desenvolveu os seguintes passos: o primeiro responsável por extrair *fingerprints* de trechos musicais e agrupá-los em um conjunto de características espectrais; o segundo utiliza as *fingerprints* encontradas para gerar um banco de dados permitindo que trechos musicais possam ser comparados para realizar a classificação criando assinaturas para as mesmas utilizando técnica de similaridades. De acordo com as informações coletadas, o sistema é capaz de criar listas de música agrupadas po gênero, utilizando qualquer música como ponto inicial.

Cano (2006) apresenta dois principais objetivos: estudar a relevância de técnicas como *fingerprint* e propor um novo método para suprir as limitações semânticas dos identificadores atuais. Para resolução de seu primeiro problema Cano estudou técnicas de baixo nível para descrição de áudio. Ele sugere que ao estender a busca de conteúdo nos áudios pode classificar o conteúdo por similaridades. Na segunda parte ele se refere a falta de inteligência nos sistemas de classificação de áudio. Para tal ele propõe um classificador de som geral capaz de gerar descrições detalhadas em uma representação que computadores e usuários possam entender.

O Quadro 1 mostra de forma resumida as principais características destes sistemas, tendo como base critérios extraídos a partir dos conceitos descritos no decorrer desta seção.

Quadro 1 – Características dos trabalhos relacionados

características / trabalhos	Dorow (2011)	Wignall (2003)	Cano (2006)
Objetivos	Identificação de músicas	Classificação de músicas por gênero	Exploração de limitações semânticas
pré-processamento	Transformada de Fourier	Transformada de Fourier	Transformada de Fourier
extração de características	*Fingerprint*	*Fingerprint*	*Fingerprint*

A partir do quadro 1 é possível observar que as técnicas empregadas para identificação de sons são semelhantes. Com o propósito inicial de extração de características do áudio, os três autores citados seguiram o mesmo caminho. Apesar de possuírem objetivos finais distintos todos efetuaram os mesmos passos para recolhimento dos dados iniciais.

Após analisar trabalhos distintos, fica perceptível que não há estudos de detecção de sons específicos para identificação de objetos. Os trabalhos encontrados tendem a seguir o caminho de classificação/identificação de músicas.

Nos trabalhos estudados, é necessário converter o áudio do domínio de tempo para o domínio de frequência, utilizando a transformada de Fourier. Em seguida, são extraídas as características, conhecidas como *fingerprints*. Após a extração destes dados, estes são comparados com a base já existente utilizando técnicas de similaridade. Ao encontrar características semelhantes, identifica-se o áudio. Em caso de não existir outro áudio com semelhança satisfatório, classifica-o como novo som, para futuras comparações.

Levando em consideração a baixa aplicação dos trabalhos já existentes para classificação de sons não verbais, este trabalho tem por objetivo abranger esta área. A utilização de algoritmos para a classificação dos sons e identificação o que é relevante tende a resultar em menor quantidade de informação irrelevante persistida, ocasionando assim um melhor aproveitamento dos espaços em disco e facilitando quando uma verificação manual se faz necessária.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas para o desenvolvimento do sistema proposto. Na seção 3.1 é apresentado o levantamento de requisitos. Na seção 3.2 é apresentada a especificação do trabalho. Na seção 3.3 é apresentado a implementação do sistema de identificação. Por fim, na seção 3.4 são discutidos os resultados encontrados através dos experimentos realizados.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O sistema de classificação de objetos sonoros deverá:

a) ler um arquivo sonoro no formato *Windows Audio-Visual* (WAV) (Requisito Funcional – RF);
b) estabelecer relações espaciais (*fingerprints*), identificando elementos de interesse a partir da estrutura do áudio (RF);
c) utilizar métodos estatísticos para estabelecer o grau de similaridade entre as características espaciais encontradas com informações previamente cadastradas para identificar o objeto sonoro (RF);
d) ser implementado utilizando a linguagem Java (Requisito Não Funcional – RNF);
e) ser implementado utilizando ambiente de desenvolvimento Eclipse (RNF);
f) ser multiplataforma (RNF).

3.2 ESPECIFICAÇÃO

A seguir é apresentada a especificação do protótipo de identificação de objetos, que foi modelado na ferramente *Enterprise Archtiect* (EA). O sistema foi desenvolvido seguindo a análise orientada a objetos, utilizando a notação *Unified Modeling Language* (UML) para a criação dos diagramas de casos de uso, classe e de sequência. A Figura 1 mostra o processo da solução proposta.

Figura 1 - Processo geral da solução

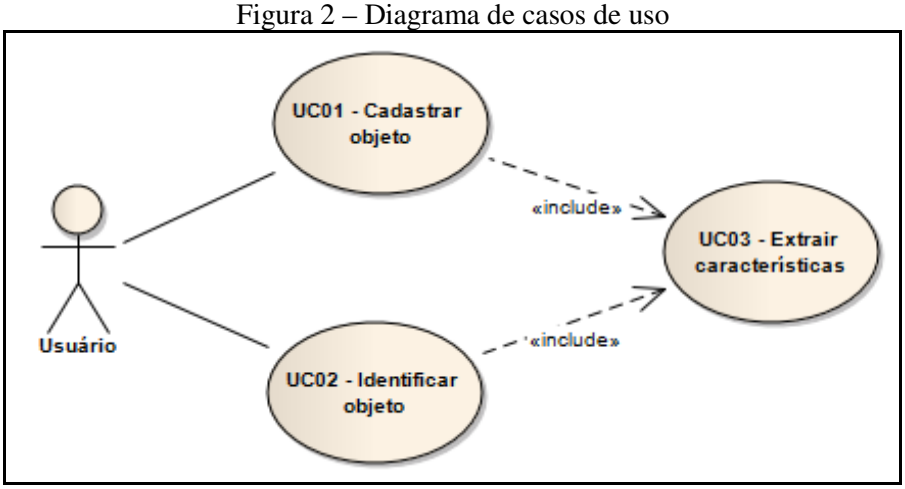
Conforme pode ser visto na Figura 1, como ponto de partida tem-se a captura do áudio do objeto, oriunda de arquivo ou microfone. É aplicada a transformada de fourier no áudio obtido para obter suas frequências. Após obter as frequências são extraídos os *fingerprints*. Ao possuir as *fingerprints* é possível realizar a persistência dos dados já obtidos ou efetuar a identificação. Na identificação os *fingerprints* obtidos são relacionados com os já existentes na base aplicando técnicas de similaridade para identificar o objeto mais similar.

As seções seguintes trazem informações referentes a especificação do sistema. São apresentados os casos de uso na seção 3.2.1, em seguida a seção 3.2.2 descreve os diagramas de classes. A seção 3.2.3 apresenta o diagrama de sequência e por fim, a seção 3.2.4 exibe a modelagem do banco de dados.

3.2.1 Diagrama de casos de uso

O sistema possui três (3) casos de uso que são apresentados na Figura 2.

Figura 2 – Diagrama de casos de uso



Na Figura 2 é possível observar os dois (2) principais casos de uso, o UC01 de cadastro de objetos e o UC02 de identificação de objetos. Para auxiliar nesses dois cenários é utilizado o terceiro caso de uso, UC03, responsável por extrair as características do áudio.

Abaixo são apresentados os quadros que descrevem os casos de uso citados. O Quadro 2 apresenta o UC01, o UC02 é apresentado no Quadro 3 apresenta e o Quadro 4 apresenta o UC03.

Quadro 2 – Caso de uso 1

UC01 – Cadastrar objeto	
Pré-condições	Não há pré-condições.
Cenário principal	01) O usuário inicia o aplicativo. 02) O usuário seleciona a aba de gravação. 03) O usuário informa o caminho do arquivo de áudio a ser gravado. 04) O sistema realiza a conversão do arquivo para o formato utilizado pelo sistema. 05) O sistema divide o arquivo em partes, gerando fingerprints para cada trecho de áudio (através do UC03). 06) O sistema salva as informações geradas para o áudio. 07) O sistema exibe mensagem indicando que o áudio foi registrado.
Fluxo alternativo 01	No passo 03: 03.1) O usuário solicita a gravação do áudio do microfone para o registro. 03.2) O sistema grava o som do microfone até o usuário pressionar a tecla parar.
Exceção 01	No passo 04, o arquivo pode ter um formato não reconhecido pelo sistema: 04.1) O sistema exibe mensagem indicando que um erro ocorreu.
Pós-condição	O áudio e os fingerprints gerados devem estar salvos.

O Quadro 2 informa o procedimento utilizado para gravação e geração dos *fingerprints*. O usuário tem a opção de efetuar a gravação utilizando um arquivo de áudio ou gravar a partir do microfone.

Quadro 3 – Caso de uso 2

UC02 – Identificar objeto	
Pré-condições	Existir objetos pré-cadastrados na base.
Cenário principal	01) O usuário inicia o aplicativo. 02) O usuário seleciona a aba de identificação. 03) O usuário efetua a gravação durante o tempo desejado. 04) O sistema realiza a conversão do arquivo para o formato utilizado pelo sistema. 05) O sistema divide o arquivo em partes, gerando fingerprints para cada trecho de áudio (através do UC03). 06) O sistema executa comparação do áudio com o conteúdo cadastrado na base. 07) O sistema exibe mensagem indicando as similaridades encontradas.
Pós-condição	Objetos similares identificados, de acordo com a técnica utilizada.

Conforme descrito no Quadro 3, o caso de uso 2 descreve o processo de identificação dos objetos. A identificação utiliza o áudio registrado pelo microfone no procedimento.

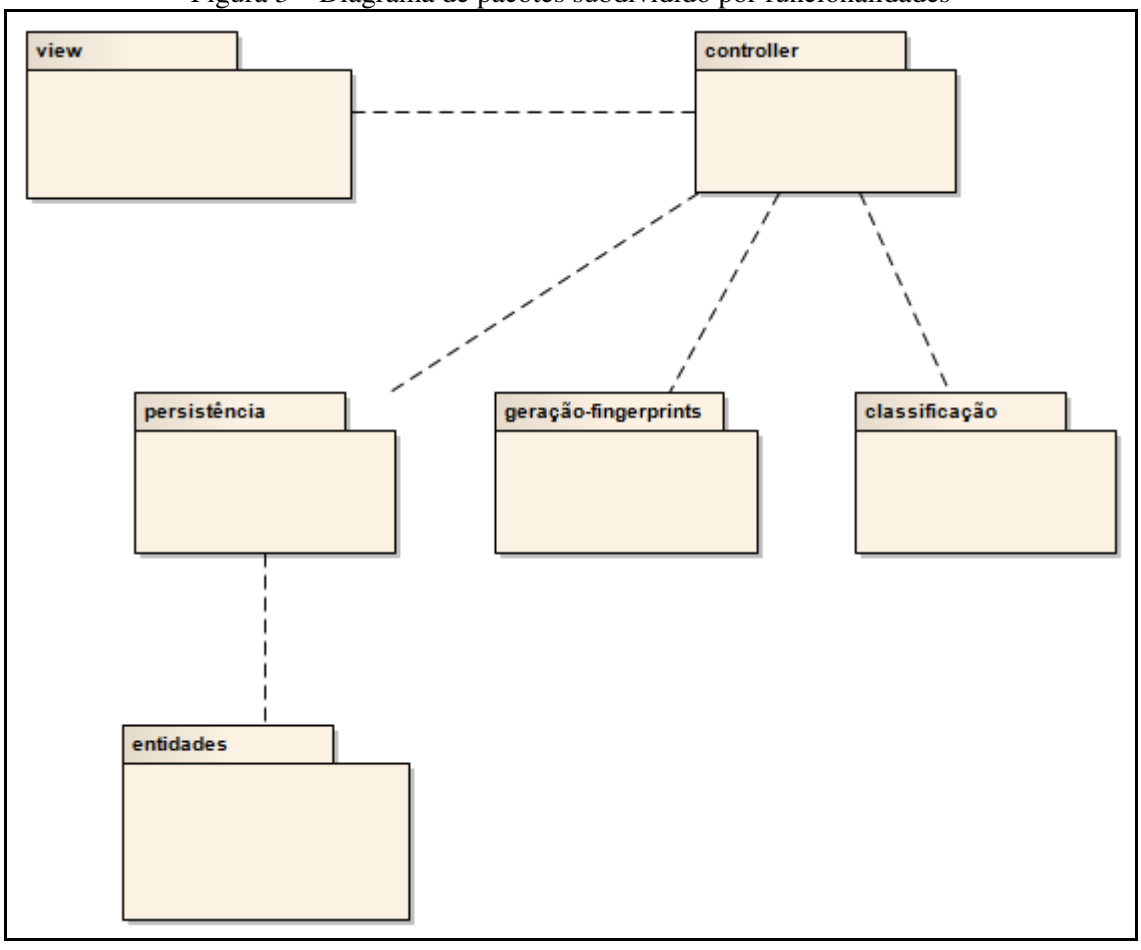
Quadro 4 – Caso de uso 3

UC03 – Geração de *fingerprints*	
Pré-condições	Frequências obtidas de um áudio devem ser informadas.
Cenário principal	01) O sistema identifica os pontos de frequência mais importantes dentre as recebidas. 02) O sistema gera um *hash* baseado nos pontos identificados.
Pós-condição	*Hash* do *fingerprint* gerado.

3.2.2 Diagrama de Classes

O diagrama de classes trás uma visão geral do relacionamento entre as classes do sistema. Inicialmente é apresentado na Figura 3 o diagrama de classes simplificado de todo o sistema. O diagrama está separado por pacotes lógicos para melhor entendimento. Em seguida são apresentados diagramas separados por funcionalidades facilitando a leitura.

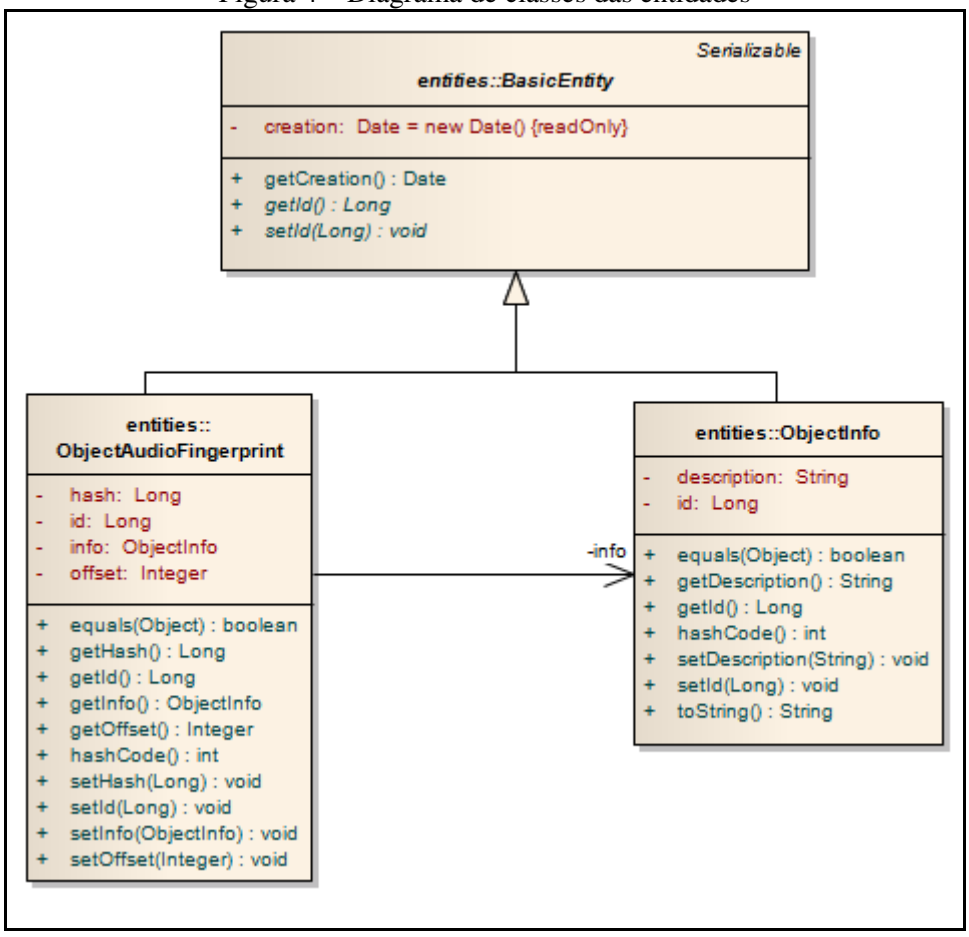
Figura 3 – Diagrama de pacotes subdividido por funcionalidades



A Figura 3 exibe a separação lógica de pacotes dentro do sistema. O pacote `view` é responsável pela apresentação visual e respostas ao usuário. O pacote `controller` é encarregado de fazer a ligação entre as funcionalidades e a `view`. A persistência trata dos acessos de gravação e leitura ao banco de dados, através do uso das entidades em seu respectivo pacote. O `controller` utiliza o pacote de geração de `fingerprints` para efetuar a classificação e a gravação de objetos. A classificação é efetuada utilizando o pacote de `classificação`.

Na Figura 4 são apresentadas as classes referentes as entidades utilizadas para persistência.

Figura 4 – Diagrama de classes das entidades



As classes apresentadas são as seguintes:

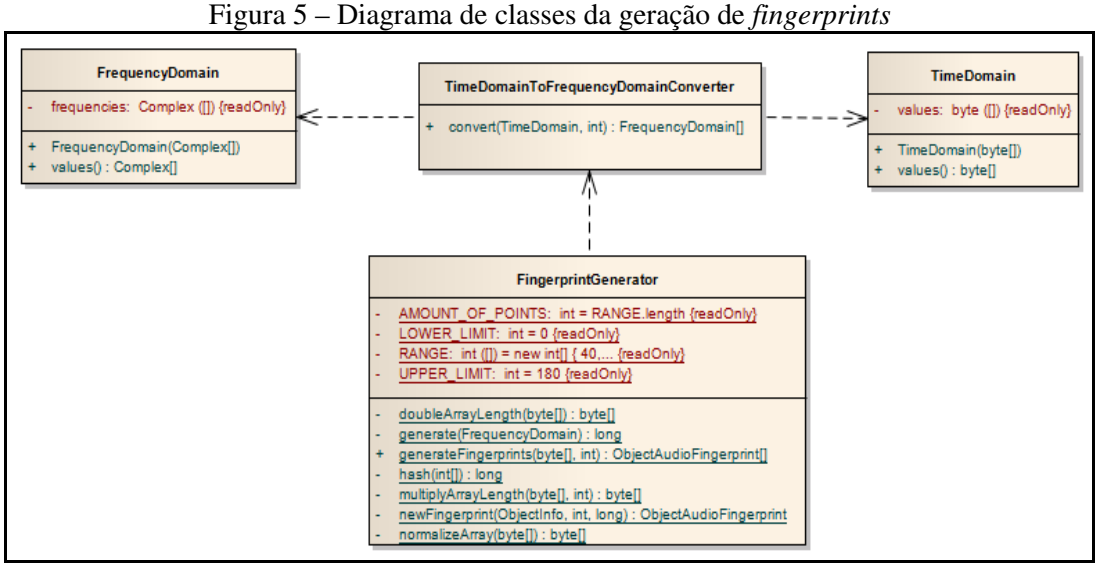
a) `BasicEntity`: é a classe base para as entidades a serem persistidas, contendo informações relevantes para a base de dados;

b) `ObjectInfo`: representação das informações do áudio. Guarda um identificador e o nome descritivo do áudio;

c) `ObjectAudioFingerprint`: representação das características extraídas de um áudio. Registra para qual áudio (`ObjectInfo`) foi gerado, a identificação espacial dentro do áudio de origem e o hash gerado a partir das características.

A funcionalidade a seguir apresenta a geração de *fingerprints* necessário nas etapas de registro e identificação dos objetos. A funcionalidade é apresentada na Figura 5.

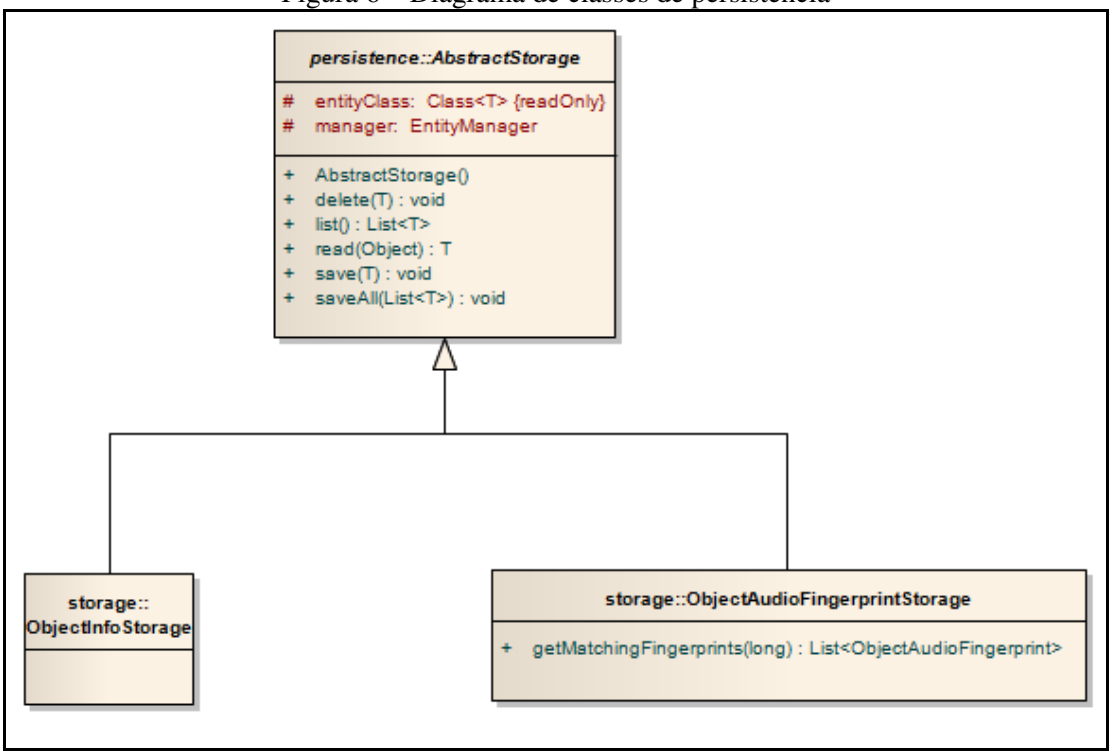
Figura 5 – Diagrama de classes da geração de *fingerprints*



Para poder representar o domínio do tempo e o domínio da frequência, foram especificadas duas classes: `TimeDomain` e `FrequencyDomain`, que podem ser vistas na Figura 5. Para efetuar a conversão da informação no domínio do tempo para o domínio da frequência, foi definida a classe `TimeDomainToFrequencyDomainConverter`, que por sua vez utiliza a transformada de Fourier para realizar a conversão. A classe `FingerprintGenerator` foi definida para realizar a geração de hashes de fingerprints a partir das informações do domínio da frequência. Para extrair a criação das fingerprints, e tornar esta lógica reutilizável, foi definida a classe `FingerprintGenerator`.

As informações extraídas são armazenadas com utilização das classes de entidades citadas. Estas são persistidas utilizando a biblioteca *Hibernate* para facilitar a implementação. A Figura 6 apresenta o as classes responsáveis pela persitência.

Figura 6 – Diagrama de classes de persistência



Na Figura 6 podemos elencar as seguintes classes:

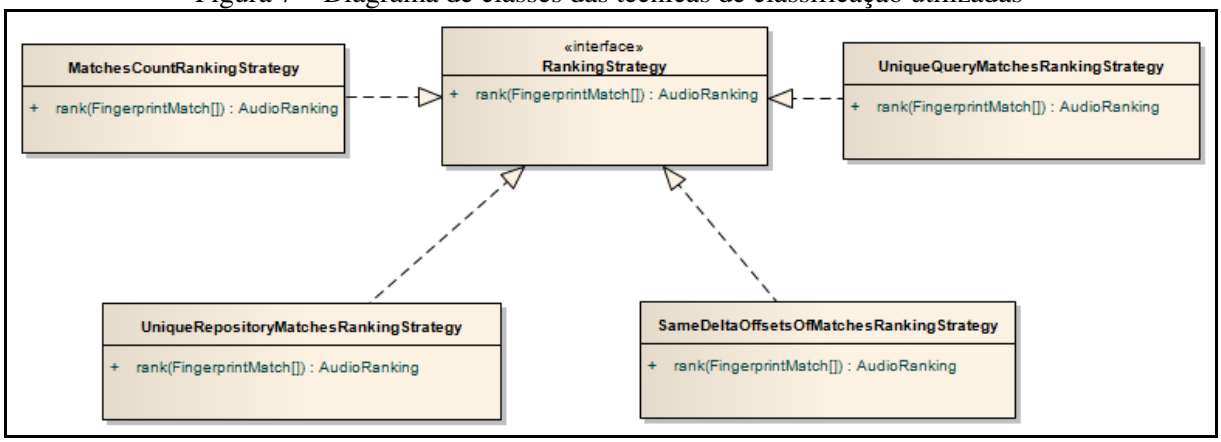
a) `AbstractStorage`: é a classe base para as classes de persistência. Utiliza a API fornecida pelo *Hibernate* para persistir as informações;

b) `ObjectInfoStorage`: classe de persistência das informações do áudio. Responsável por persistir as informações específicas de cada áudio coletado;

c) `ObjectAudioFingerprintStorage`: classe responsável pela persistência das características extraídas do áudio. Ainda responsável pelas consultas específicas utilizadas pelo sistema de busca e identificação.

As classes utilizadas pelo algoritmo de identificação estão destacadas na Figura 7.

Figura 7 – Diagrama de classes das técnicas de classificação utilizadas

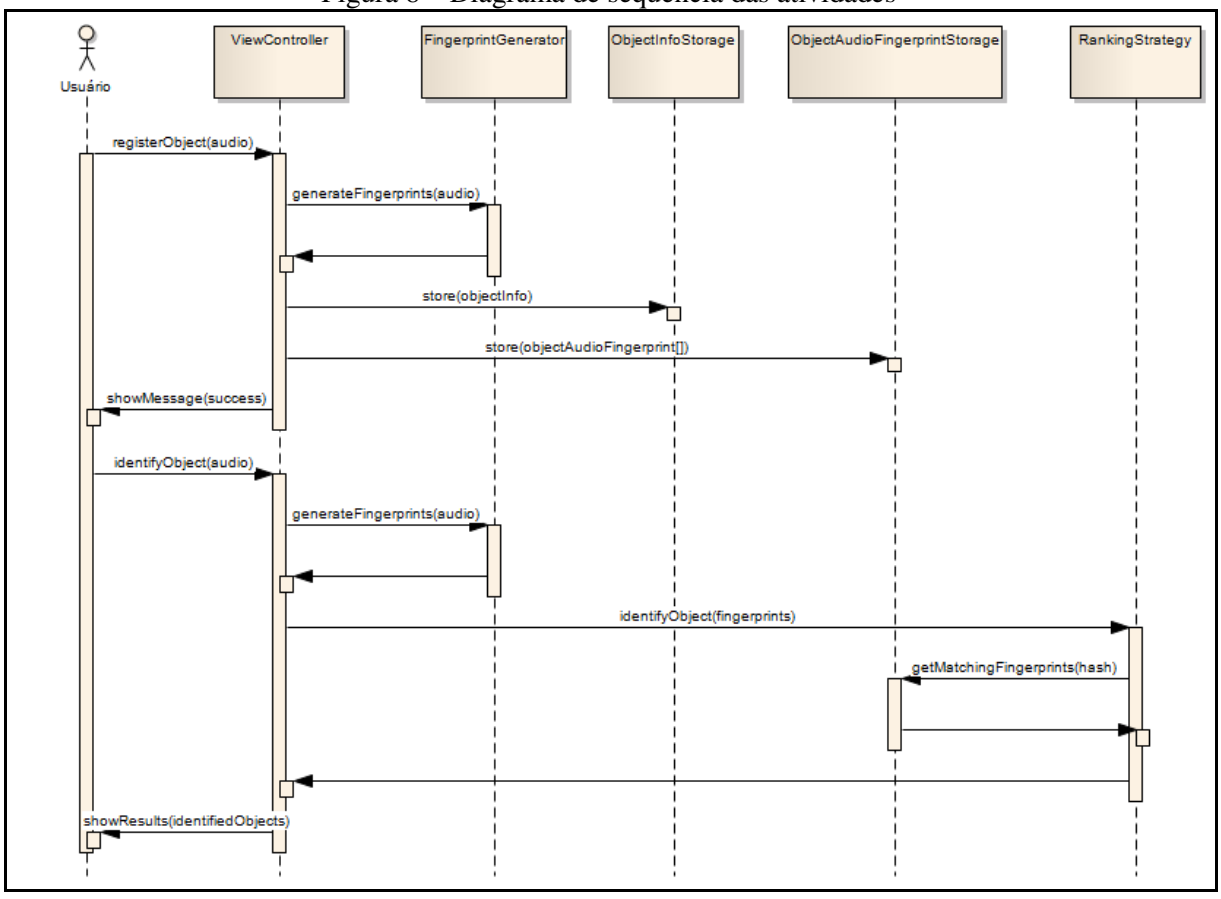


Conforme descrito por Dorow (2011, p 33), cada uma das realizações da interface `RankingStrategy` exibidas utiliza um algoritmo diferente para identificar os áudios mais relevantes. Para tal são utilizados pares de *fingerprints* de mesmo *hash*, buscados da base de dados.

3.2.3 Diagrama de sequência

Na Figura 8 é apresentado o diagrama de sequência das atividades do sistema.

Figura 8 – Diagrama de sequência das atividades



O diagrama da Figura 8 apresenta de forma simplificada os fluxos padrões permitidos pelo sistema. A figura exibe um fluxo de registro de objetos e outro para a identificação, que serão detalhados a seguir.

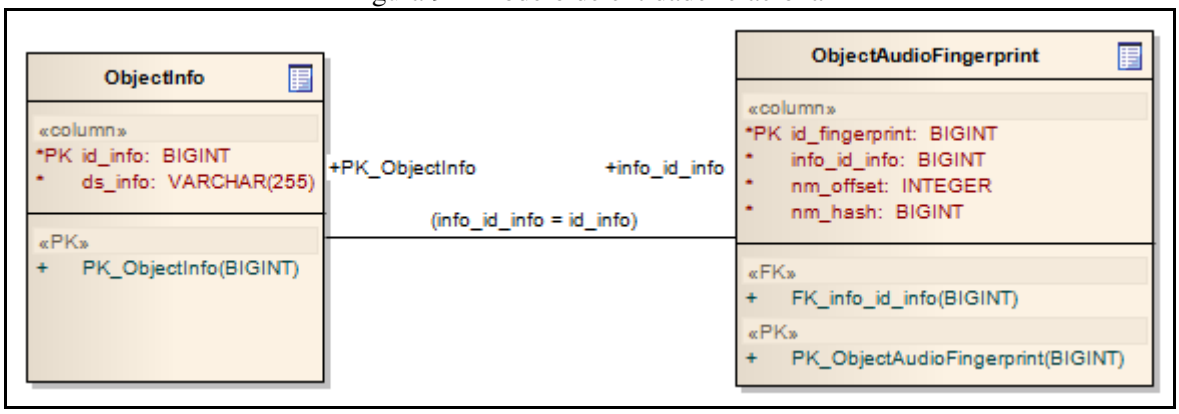
No fluxo de cadastro de objetos, o cliente solicita ao controlador efetue o registro de um objeto na base, informando o áudio obtido ou de microfone ou de arquivo. Neste momento é solicitado ao gerador de *fingerprints* que extraia as características. Após isso o controlador solicita ao *Storage* para armazenar em base as *fingerprints* extraídas. A *view* do usuário exibe mensagem de sucesso.

O fluxo de identificação se assemelha muito ao de gravação. O cliente solicita ao gerador de *fingerprints* para extraír as características. As informações aqui extraídas são enviadas para os algoritmos de comparação, que devolveram o resultado. A *view* exibe os resultados obtidos ao usuário.

3.2.4 Modelagem do banco de dados

Para o sistema de identificação foram utilizadas apenas duas tabelas para registro das informações. As tabelas `ObjectInfo` e `ObjectAudioFingerprint` são detalhadas na Figura 9.

Figura 9 – Modelo de entidade relacional



As tabelas exibidas foram utilizadas para armazenamento das informações dos objetos cadastrados. A tabela `ObjectInfo` guarda o nome e um identificador único, enquanto a tabela `ObjectAudioFingerprint` guarda as fingerprints geradas, o índice dessa característica e um identificador único.

3.3 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas e a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

O sistema de classificação foi desenvolvido utilizando a linguagem Java. Foi utilizado o ambiente Eclipse na versão 3.7 para o desenvolvimento dos fontes. Tanto o Java quanto o eclipse são livres para serem usados para o desenvolvimento de aplicações. Para o armazenamento das informações registradas foi utilizado o banco de dados embarcado *H2 Database Engine*, por não se fazer necessário um banco com maior robustez.

3.3.2 Implementação do sistema

A seguir serão apresentados fragmentos de código descrevendo as etapas do desenvolvimento do sistema.

3.3.2.1 Obtenção do áudio para gravação

Para a identificação de objetos é necessário o cadastro prévio de objetos conhecidos na base. Para tal existem duas possíveis formas de adquirir o áudio. Para a primeira forma se faz necessário possuir um microfone conectado ao computador. Esta forma utiliza como entrada o áudio capturado do microfone. Na segunda forma, é necessário informar um arquivo de áudio contendo os sons do objeto a ser cadastrado.

3.3.2.1.1 Obtenção do áudio utilizando microfone

O processo de gravação de áudio inicia com a requisição ao controlador da `view` para começar a gravação. A gravação do áudio para registro de objetos é feito de forma assíncrona, necessitando o usuário informar o início e fim da gravação. O Quadro 5 demonstra o código utilizado para iniciar a gravação.

Quadro 5 – Requisição do início da gravação

```
1       @Override
2       protected Runnable getRunnableAction(ActionEvent actionEvent) {
3              return new Runnable() {
4                     @Override
5                     public void run() {
6       registerPanel.getRecorder().startRecord(AudioUtils.getRecordingFormat());
7                     }
8              };
9       }
```

Para a obtenção do áudio foi utilizada a Java Sound API (Oracle, 2013). A gravação do áudio utiliza uma `Thread` nova para sua execução, que é criada e retornada pela chamada do método `getRunnableAction` como pode ser visto na linha 2.

O objeto retornado pela chamada a `registerPanel.getRecorder` na linha 6, é responsável pela gravação do áudio. Após a chamada para `startRecord` na mesma linha, é

iniciada a captura do áudio do microfone até que a gravação seja interrompida, conforme o Quadro 6.

Quadro 6 – Obtenção do áudio através do microfone

```
1      public void startRecord(AudioFormat format) throws AudioRecordingException {
2             TargetDataLine line = getMicrophoneTargetDataLine(format);
3             try {
4                    line.open(AudioUtils.asJavaSoundAudioFormat(format));
5             } catch (LineUnavailableException e) {
6                    throw new
AudioRecordingException(AppMessageBundle.get("error.record.lineunavailable"), e);
7             }
8
9             line.start();
10            isRecording = true;
11            byte[] buffer = new byte[line.getBufferSize()];
12
13            ByteArrayOutputStream out = new ByteArrayOutputStream(4096);
14            try {
15                   while (shouldKeepRecording()) {
16                          int count = line.read(buffer, 0, buffer.length);
17                          if (count > 0) {
18                                 out.write(buffer, 0, count);
19                          }
20                   }
21                   out.close();
22                   line.stop();
23                   line.close();
24            } catch (IOException ioe) {
25                   throw new
AudioRecordingException(AppMessageBundle.get("error.record.linereader"), ioe);
26            }
27            recorded = out.toByteArray();
28     }
29
30     public void stopRecord() {
31            isRecording = false;
32     }
```

O Quadro 6 demonstra o código que efetua a obtenção do áudio do microfone. Ao iniciar a execução, o estado é alterado para gravando conforme linha 2. Para indicar que o recurso do microfone deve ser reservado para a aplicação é efetuada a chamada ao método `open()` na linha 5 para o microfone, que é representado pela variável `line`. Em seguida, na linha 10, a chamada ao método `start()` inicia o recebimento dos pacotes de áudio oriundos do microfone. Os dados vindos do microfone chegam em pacotes de 4096 bytes, conforme informado no formato de áudio especificado.

Para armazenamento temporário dos dados vindos do microfone é utilizado um *buffer* intermediário, que capta o áudio e armazena na variável `out`, que é um buffer que cresce conforme a necessidade. Esse processo continua até que seja feita uma chamada para `stopRecording()`na linha 30, que altera o estado para não gravando. Ao parar a rotina de gravação, o buffer do áudio gravado fica pronto para ser recuperado e passsar pelo respectivo tratamento.

3.3.2.1.2 Obtenção do áudio utilizando arquivo

Para a leitura de áudio a partir de arquivo, é utilizado o método exibido no Quadro 7, que descreve o método `readFile(file, targetFormat)`.

Quadro 7 – Método de leitura de arquivo

```
1      public static byte[] readFile(File audioFile, AudioFormat targetFormat)
throws AudioConversionException {
2             AudioInputStream ais = getAudioInputStream(audioFile, targetFormat);
3
4             byte[] data = new byte[8192];
5             int bytesRead;
6
7             final ByteArrayOutputStream out = new ByteArrayOutputStream(4096);
8
9             try {
10                   while ((bytesRead = ais.read(data, 0, data.length)) > -1) {
11                          out.write(data, 0, bytesRead);
12                   }
13            } catch (Exception e) {
14                   throw new AudioConversionException(e);
15            }
16
17            return out.toByteArray();
18     }
```

A rotina é responsável por ler o conteúdo do arquivo e devolver um *buffer* contendo os dados. O arquivo é lido de acordo com o tipo de dados informado no parâmetro `targetFormat`. Na linha 2 é possível ver que é aberto um *stream* para a leitura dos dados do arquivo informado.

3.3.2.2 Algoritmo de interpolação

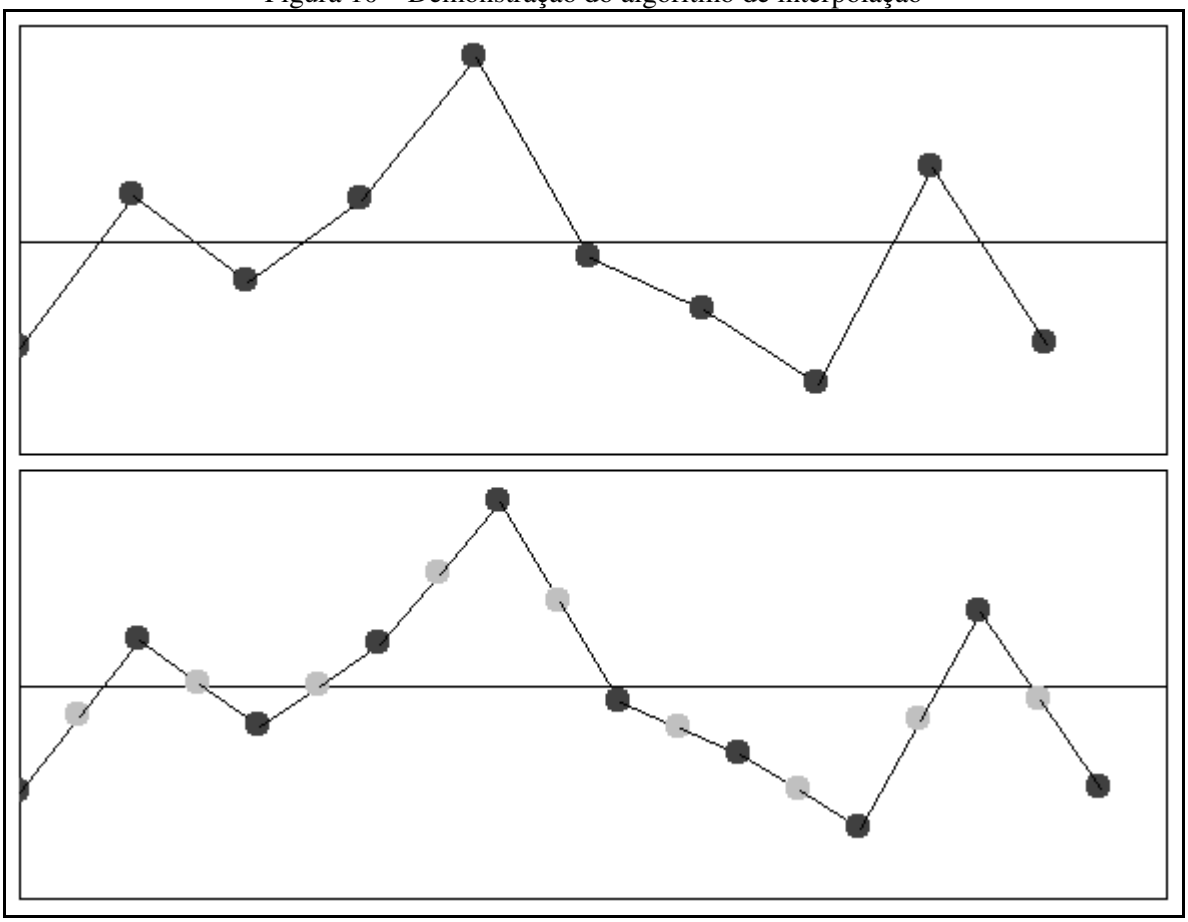
Os algoritmos de reconhecimento de objetos baseiam-se nas características extraídas dos objetos coletados. Levando-se em consideração que sons de objetos são curtos, fez –se necessário a utilização de um algoritmo de interpolação do áudio coletado do microfone ou de arquivo para a extração de um número maior de características.

Durante o desenvolvimento deste trabalho não foi encontrado nenhum algoritmo para tal finalidade, fazendo-se necessário o desenvolvimento de um. O algoritmo descrito tem por objetivo aumentar a quantidade de valores obtidos da entrada.

Para cada par de valores da entrada, é retirado sua média e adicionada ao *buffer* de valores encontrados. A representação gráfica do algoritmo pode ser observada na Figura 10.

A figura demonstra de maneira ampliada um exemplo de onda de áudio relacionando seus pontos em um cartesiano sendo x o tempo e y o valor em si. A parte superior da figura demonstra o áudio antes da execução do algoritmo, enquanto a imagem inferior mostra o valor resultante.

Figura 10 – Demonstração do algoritmo de interpolação



O algoritmo é aplicado antes de iniciar o processo de extração de *fingerprint*, tanto para a gravação do áudio na base para posterior reconhecimento, quanto no áudio utilizado para a identificação. Os pontos em cinza claro exibidos na parte inferior da figura demonstram os pontos inseridos pelo algoritmo. O Quadro 8 exibe o código fonte do algoritmo.

Quadro 8 – Algoritmo de interpolação

```
1       private static byte[] doubleArrayLength(byte[] timeDomainData) {
2               int size = timeDomainData.length;
3               byte[] resized = new byte[size * 2 - 1];
4               int currentIndex = 0;
5               for (int i = 0; i < size - 1; i++) {
6                       resized[currentIndex++] = timeDomainData[i];
7       resized[currentIndex++]=(byte)((timeDomainData[i]+timeDomainData[i+1])/ 2);
8               }
9               resized[currentIndex] = timeDomainData[size - 1];
10              return resized;
11      }
```

Conforme é possível observar no quadro, para o *array* de *bytes* na entrada informado é criado um novo *array* com tamanho igual a duas vezes o tamanho da entrada menos um. O array criado na linha 3 será utilizado para guardar os valores gerados e os informados.

Cada valor do *array* e somado ao próximo valor e dividio por dois, para obter assim o ponto médio entre eles. Estes valores são armazenados no *array* instanciado. O resultado da execução pode ser visto na Figura 10.

3.3.2.3 Transformação de domínio do áudio

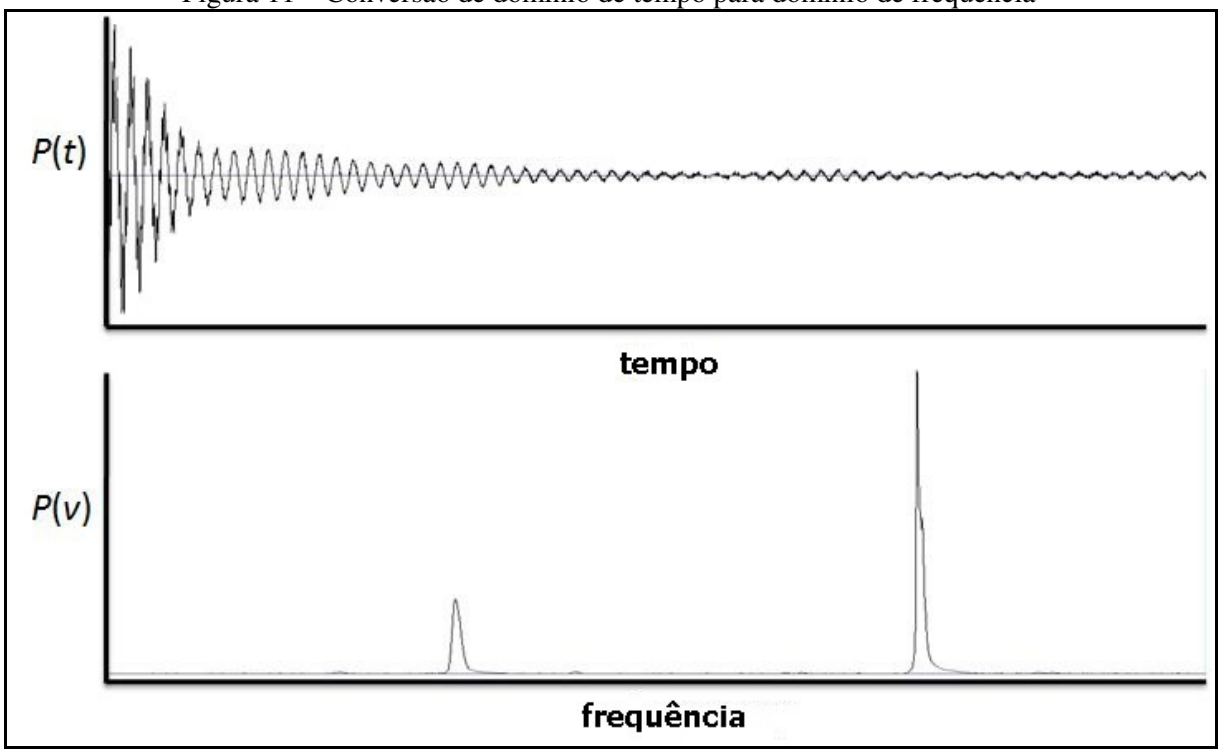
O áudio obtido através de gravação ou arquivo, é apresentado sob o domínio de tempo. Para a extração de características deste, se faz necessáro a conversão para o domínio de frequência. Para efetuar essa conversão, é utilizada a transformada discreta de Fourier. O algoritmo utilizado foi extraído da biblioteca Apache Commons Math (APACHE SOFTWARE FOUNDATION, 2013). Esta biblioteca utiliza o algoritmo chamado FFT. A única restrição deste algoritmo é que o número de valores da entrada seja uma potência de 2. A rotina de transformação do áudio é explicada no Quadro 9.

Quadro 9 – Transformação de domínio de tempo para domínio de frequência

```
1      public FrequencyDomain[] convert(TimeDomain timeDomain, int chunkSize) {
2             byte[] tdValues = timeDomain.values();
3             int tdSize = tdValues.length;
4
5             chunkSize = tdSize < chunkSize ? tdSize : chunkSize;
6             final int fdSize = tdSize / chunkSize;
7
8             FrequencyDomain[] fds = new FrequencyDomain[fdSize];
9             FastFourierTransformer fft = new FastFourierTransformer();
10
11            for (int fdIndex = 0; fdIndex < fdSize; fdIndex++) {
12                   Complex[] complex = new Complex[chunkSize];
13                   int offset = (fdIndex * chunkSize);
14                   for (int i = 0; i < chunkSize; i++) {
15                          complex[i] = new Complex(tdValues[offset + i], 0);
16                   }
17                   Complex[] transformed = fft.transform(complex);
18                   fds[fdIndex] = new FrequencyDomain(transformed);
19            }
20            return fds;
21     }
```

Para utilização da transformada de Fourier é necessário utilizar números complexos. Para a execução do algoritmo foi utilizado como número complexo o valor obtido como parte real do número e zero para a parte imaginária, conforme mostra a linha 15 que instância os números complexos. Após execução da transformada executada na linha 17 através da chamada para `fft.transform(complex)`, obtem-se o áudio representado pelo domínio de frequência onde não há informações referentes a tempo. Por esse motivo o áudio é dividido em partes menores de tamanho definido por `chunkSize`. A Figura 11 demonstra o funcionamento da função.

Figura 11 – Conversão de domínio de tempo para domínio de frequência



3.3.2.4 Geração de *fingerprints*

Após a transformação do domínio do áudio, tem-se ele em forma de frequências, de onde as características serão extraídas. O quadro 10 demonstra o algoritmo utilizado.

Quadro 10 – Geração de fingerprints

```
1      private static final int LOWER_LIMIT = 0;
2      private static final int UPPER_LIMIT = 180;
3      private static final int[] RANGE = new int[] { 40, 80, 120, UPPER_LIMIT };
4      private static final int AMOUNT_OF_POINTS = RANGE.length;
5
6      static long generate(FrequencyDomain fd) {
7             Complex[] frequencies = fd.values();
8             double[] highscores = new double[AMOUNT_OF_POINTS];
9             int[] recordPoints = new int[AMOUNT_OF_POINTS];
10            int freqRange = 0;
11            for (int freq = LOWER_LIMIT; freq <= UPPER_LIMIT; freq++) {
12                   if (RANGE[freqRange] < freq) {
13                          freqRange++;
14                   }
15                   double magnitude = frequencies[freq].abs();
16                   if (magnitude > highscores[freqRange]) {
17                          highscores[freqRange] = magnitude;
18                          recordPoints[freqRange] = freq;
19                   }
20            }
21            return hash(recordPoints);
22     }
23     private static long hash(int[] p) {
24            int p0 = (RANGE[0] - p[0]) & 0xFE;
25            int p1 = (RANGE[1] - p[1]) & 0xFE;
26            int p2 = (RANGE[2] - p[2]) & 0xFE;
27            int p3 = (RANGE[3] - p[3]) & 0xFE;
28            return (p0) | ((p1) << 8) | ((p2) << 16) | ((p3) << 24);
29     }
```

Para geração das *fingerprints* são utilizadas as frequências de 0 a 40, 41 a 80, 81 a 120 e 121 a 180 conforme a constante da linha 3 demonstra. Para cada frequência resultante é extraída sua magnitude como mostra a linha 15. As frequências de maior magnitude em cada frequência são selecionadas, resultando em 4 números. Os valores obtidos são unidos em um único valor utilizando operações de *shift*. As operações podem ser vistas no método `hash(int[] p)` que é descrito na linha 24 em diante. O número resultante, juntamente com o *offset* dele dentro do áudio resultarão no audio *fingerprint*.

3.3.2.5 Obtenção do áudio para a identificação

A obtenção de áudio para identificação é feita através da captura ininterrupta de áudio do microfone. Ao iniciar o processo de identificação é criada uma nova *Thread* de execução que é responsável por registrar todo áudio captado pelo microfone, conforme código do Quadro 11. Ao detectar que algo foi recebido, esse trecho é enviado para quem está escutando aos eventos de áudio capturado.

Quadro 11 – Gravação do áudio para a identificação

```
1      public void record()throws AudioRecordingException,LineUnavailableException{
2
. . .
18
19           ByteArrayOutputStream out = new ByteArrayOutputStream(4096);
20           try {
21                 int count = 0;
22                 shouldKeepRecording = true;
23
24                 do {
25                       count = lineReadBuffer(line, buffer);
26                       if (count > 0) {
27
28                             if (isValidBuffer(buffer)) {
29                                   if (!isRecording) {
30                                         out = new
ByteArrayOutputStream(4096);
31                                         isRecording = true;
32                                   }
33                                   out.write(buffer, 0, count);
34                             } else {
35                                   if (isRecording) {
36                                         if (++errorBuffers < errorMargin) {
37                                               out.write(buffer, 0, count);
38                                         } else {
39                                               out.close();
40                                   adviceAudioRecorded(out.toByteArray());
41                                               errorBuffers = 0;
42                                               isRecording = false;
43                                         }
44                                   }
45                             }
46
47                       }
48                 } while (shouldKeepRecording());
```

A rotina de gravação do áudio para reconhecimento é descrita no Quadro 11. Após iniciado o processo de identificação, a *Thread* inicia a gravação contínua. Ao identificar que um áudio foi recebido, é iniciado o registro do áudio coletado para um *buffer* temporário, que é instânciado na linha 30.

Toda entrada considerada válida é adicionada ao buffer de gravação. A chamada na linha 28 à `isValidBuffer()` é quem verifica a validade do buffer. Uma entrada é considerada válida quando seus valores se distanciam do valor zero. A validação de uma entrada pode ser vista no Quadro 12.

Quadro 12 – Validação dos valores de entrada do microfone

```
1      private boolean isValidBuffer(byte[] buffer) {
2             for (byte b : buffer) {
3                    if (!isInInvalidRange(b)) {
4                           return true;
5                    }
6             }
7             return false;
8      }
9
10     private boolean isInInvalidRange(byte b) {
11            return -2 <= b && b <= 2;
12     }
```

Todos os valores do *buffer* informado são verificados. Se existe ao menos um valor válido no *buffer* conforme mostra a linha 3, este é dado como válido. Caso não encontre nenhum é retornado falso. A tolerância para os valores válidos pode ser vista na linha 11.

Cada entrada coletada do microfone passa pela rotina de validação. Para todas as leituras existe uma tolerância de *buffers* inválidos para o caso de haver pausas no som produzido pelo objeto.

3.3.2.6 Reconhecimento dos objetos

Para o reconhecimento dos objetos foram utilizados os algoritmos extraídos do trabalho correlato de Dorow (2011). Foram utilizados 4 algoritmos diferentes de reconhecimento. O Quadro 13 enumera os algoritmos utilizados.

Quadro 13 – Algoritmos de reconhecimento de objetos

```
MatchesCountRankingStrategy
UniqueRepositoryMatchesRankingStrategy
UniqueQueryMatchesRankingStrategy
SameDeltaOffsetsOfMatchesRankingStrategy
```

A implementação de classificação por total de resultados, `MatchesCountRankingStrategy` é a técnica na qual apenas é contabilizado a quantidade de elementos da lista de similaridades que correspondem a um determinado áudio da base. O algoritmo de `UniqueRepositoryMatchesRankingStrategy` conta para o resultado final a

quantidade de *fingerprints* da base sem aceitar repetições. Na técnica utilizada no algoritmo `UniqueQueryMatchesRankingStrategy`, cada *fingerprint* gerado e que possui relação com o áudio base será contado apenas uma vez. A última técnica utilizada, `SameDeltaOffsetsOfMatchesRankingStrategy`, verifica as variações de tempo entre os *fingerprints* gerados e das similaridades encontradas na base.

Na técnica de classificação por total de resultados são contados a quantidade de elementos da lista de similaridades que correspondem a um determinado áudio da base. A aplicação desta técnica leva em consideração a ocorrência/relacionamento entre os hashes, desconsiderando sua distribuição em relação ao tempo.O algoritmo inicialmente cria uma lista para manter a quantidade de resultados para cada áudio do repositório. Realiza-se uma iteração sobre os elementos da lista de similaridades, onde o áudio de origem de cada fingerprint da base é inserido na lista. Uma instância de `AudioRankingBuilder` é criada para gerar a classificação final dos áudios. A partir disso, é contraído cada áudio contido na lista. Estas informações são inseridas no `AudioRankingBuilder`. Após realizar este procedimento em todos os áudios é retornada a classificação gerada pelo `AudioRankingBuilder`.

Na técnica de classificação por acertos únicos de *fingerprints* similares na base são contados para o resultado final, apenas a quantidade de fingerprints do repositório contidos na lista de similaridades de um áudio, sem aceitar repetições. Para o funcionamento do algoritmo primeiro é criado um mapeamento para referenciar um áudio com um conjunto de fingerprints. A lista com as similaridades é iterada, o áudio da base é obtido, e buscado no mapa por este item. Caso ainda não houver um conjunto relacionado a este áudio, um novo conjunto é criado, e associado a ele. Se já houver um conjunto associado, este conjunto é obtido. Tendo este conjunto, o *fingerprint* da base contido no par de *fingerprints* similares é adicionado ao conjunto, sendo que, se este *fingerprint* já foi adicionado previamente, não irá gerar nenhuma alteração no conjunto, pois o conjunto não adiciona itens duplicados. Após executar esta ação para todos os itens da lista de similaridades, é criada uma instância de `AudioRankingBuilder`, nela é adicionada cada áudio contido no mapa, junto da quantidade de itens no conjunto associado a ele. Então é retornada a classificação gerada pelo `AudioRankingBuilder`.

Na técnica de classificação por acertos únicos de fingerprints da *query* , cada fingerprint gerado a partir da query e, que possui alguma relação com o áudio base, será contado apenas uma vez na definição do grau de similaridade. Na execução do algoritmo, primeiro é criado um mapeamento para referenciar um áudio com um conjunto de

fingerprints. A lista com as similaridades é iterada, o áudio da base é obtido e buscado no mapa. Se ainda não houver um conjunto relacionado a este áudio, um novo conjunto é criado e associado a ele, se já houver um conjunto associado, este conjunto é obtido. Tendo este conjunto, o *fingerprint* da query contido no par de *fingerprints* similares é adicionado ao conjunto, sendo que, se este *fingerprint* já foi adicionado previamente, não gerará nenhuma alteração no conjunto, pois o conjunto não adiciona itens duplicados. Após executar esta ação para todos os itens da lista de similaridades, é criada uma instância de `AudioRankingBuilder`, nela é adicionada cada áudio contido no mapa, junto da quantidade de itens no conjunto associado a ele. Então é retornada a classificação gerada pelo `AudioRankingBuilder`.

Na técnica de classificação por variação de tempo entre *fingerprint* da base e fingerprint da *query* são verificadas as variações de tempo entre os fingerprints da query e das suas similaridades encontradas no repositório. Para cada variação de tempo que se repete, é somado um na pontuação da classificação para o áudio correspondente. O primeiro passo da execução do algoritmo é criar um mapa para armazenar a lista de similaridades de cada áudio. Para alimentar este mapa, é iterado pelo vetor fingerprints correlatas, identificado o áudio original do fingerprint da base, e então o par de fingerprints similares é adicionado na lista daquele áudio. Uma instância de `AudioRankingBuilder` é criada para armazenar o grau de similaridade de cada áudio, e em seguida, para cada áudio do mapa criado é obtida a lista de similaridades gerada para ele. Após isto, é iterado pela lista de similaridades deste áudio, onde a diferença entre a posição do *fingerprint* da base e a posição do fingerprint da *query* é calculada, a diferença de posições que mais ocorrer tem sua quantidade utilizada como grau de similaridade do áudio. O áudio e este grau de similaridade são adicionados ao `AudioRankingBuilder`. Depois de repetir tal procedimento em todos os áudios, é retornada a classificação gerada pelo `AudioRankingBuilder`.

A descrição completa dos algoritmos listados pode ser encontrada no trabalho redigido por Dorow (2011), nas páginas 46 à 59.

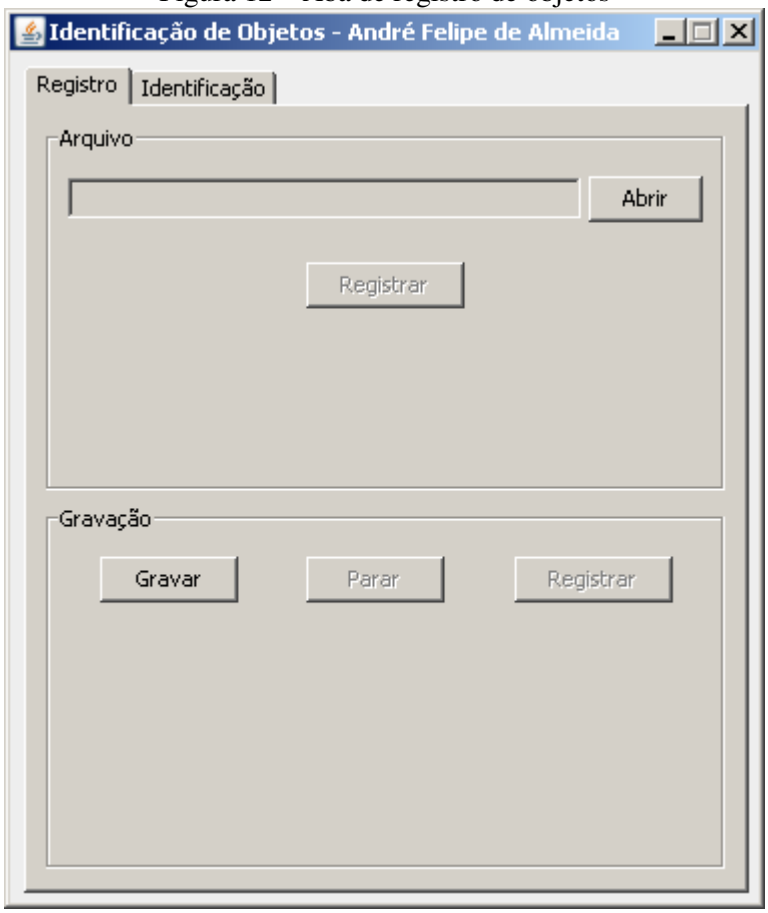
3.3.3 Operacionalidade da implementação

Esta seção apresenta a operacionalidade da implementação do ponto de vista do usuário. Esta seção é dividida em duas partes: gravação de áudio e reconhecimento.

3.3.3.1 Gravação de áudio

A primeira fase da execução do sistema consiste em efetuar a gravação dos objetos a serem posteriormente identificados. O usuário pode escolher entre efetuar a gravação dos áudios por arquivo ou por gravação do microfone. A tela de registros é exibida na Figura 12.

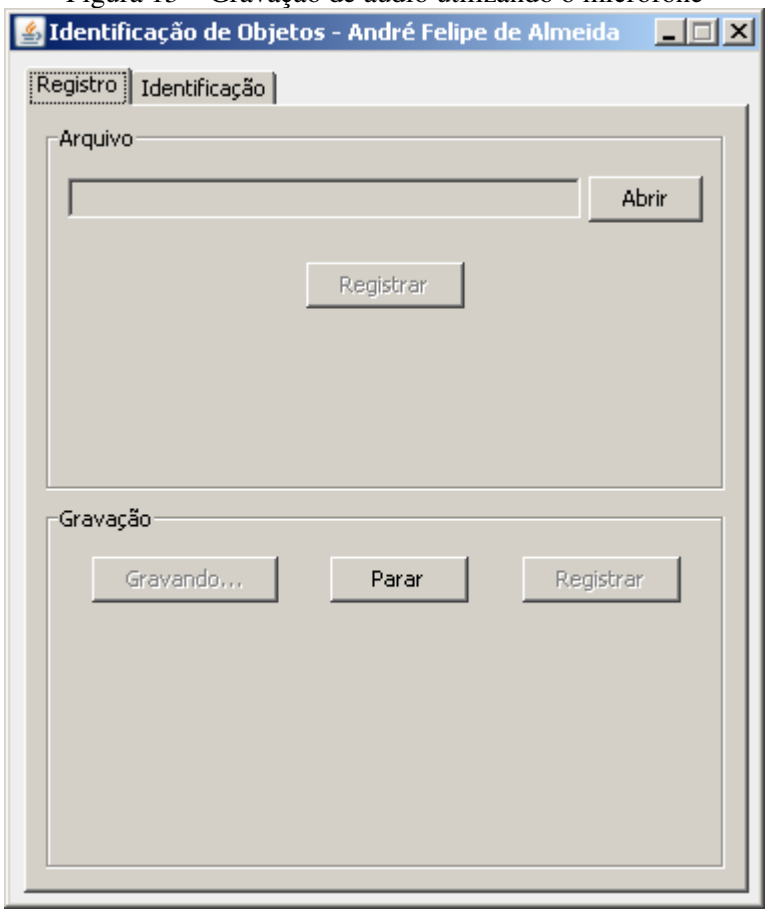
Figura 12 – Aba de registro de objetos



Para escolher a gravação por arquivo, o usuário deve clicar no botão `Abrir`, apresentado na Figura 12, na parte superior da janela. Após o clique é exibida a janela de seleção de arquivo. O usuário deve selecionar o arquivo de áudio a ser registrado e pressionar `OK`.

Para efetuar a gravação de objetos utilizando o microfone, o usuário deve clicar no botão `Gravar`, situado na parte inferior da janela, exibida na Figura 13.

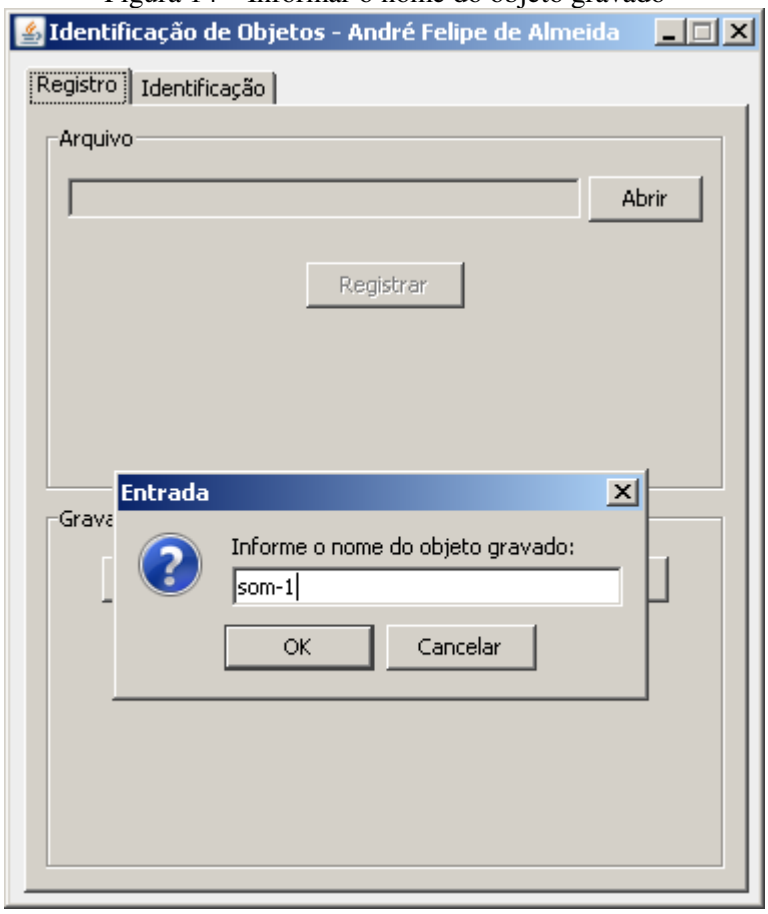
Figura 13 – Gravação de áudio utilizando o microfone



Ao pressionar o botão `Gravar` uma `Thread` de execução é iniciada que captura todo áudio registrado pelo microfone. Após o áudio registrado o usuário deve pressionar o botão `Parar`. Após ter o áudio gravado, o botão `Registrar` deve ser pressionado.

Ao pressionar o botão de registro será solicitado ao usuário o nome do objeto gravado, conforme mostra a Figura 14. O passo seguinte enviará essas informações ao controlador da `view` que iniciará o processo de extração das características para o registro.

Figura 14 – Informar o nome do objeto gravado

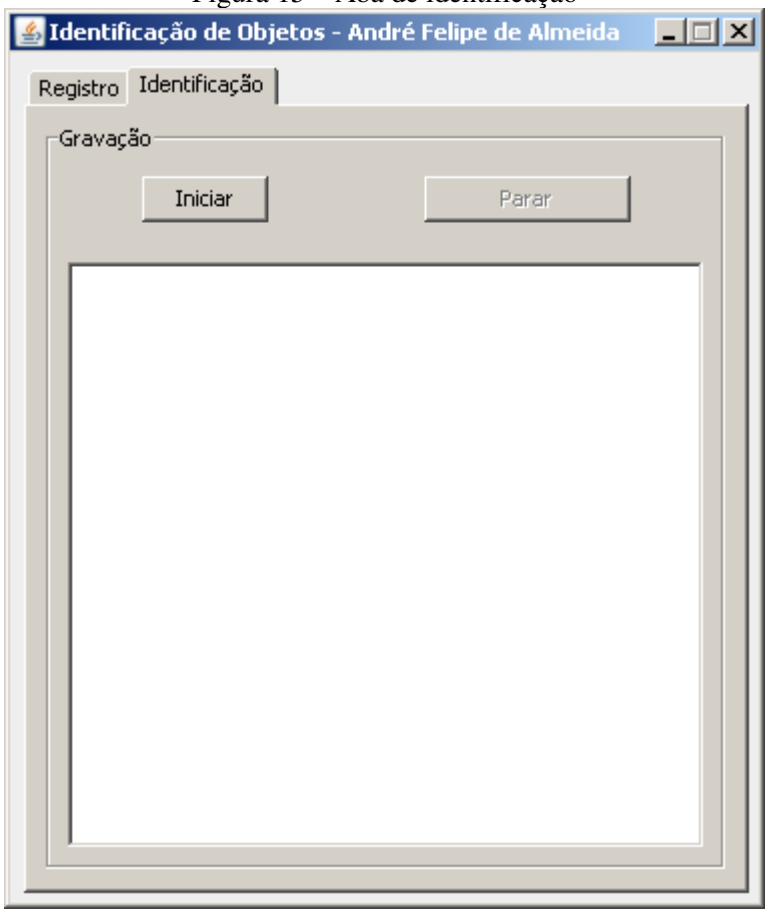


3.3.3.2 Reconhecimento de objetos

Com os objetos cadastrados na base, o usuário pode iniciar a etapa de identificação. Para iniciar o processo o usuário deve selecionar a aba `Identificação`.

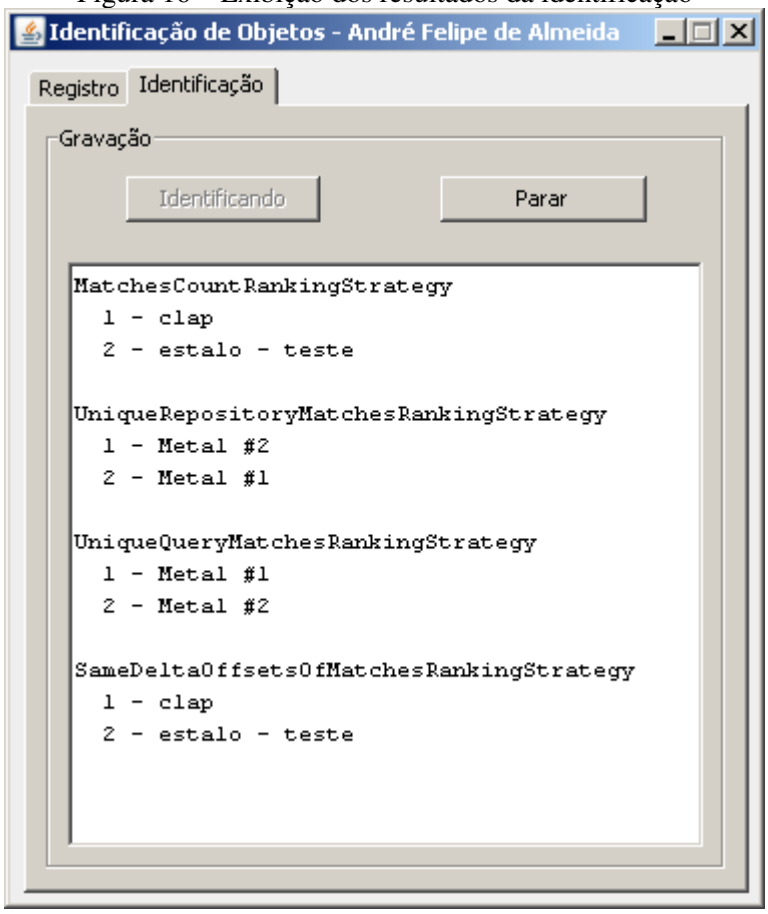
Ao pressionar o botão “Iniciar” na aba de identificação é iniciada um *Thread* de execução responsável por receber todo o áudio captura do microfone de forma contínua. Ao detectar que houve algum som na entrada, o controlador inicia o processo de identificação de objeto, enquando a *Thread* continua escutando o microfone para identificar novas entradas. A Figura 15 exibe a aba de identificação do sistema.

Figura 15 – Aba de identificação



Após o controlador ser informado de que existe som na entrada e o algoritmo de reconhecimento ter sido executado, é exibido para o usuário o resultado das comparações. O resultado é exbido no campo de texto logo abaixo dos botões de maneira descritiva. A exibição das respostas pode ser vista na Figura 16.

Figura 16 – Exibição dos resultados da identificação



O resultado obtido informa os dois principais resultados da consulta. Ao receber uma nova entrada no microfone, o processo de identificação é executado novamente e os resultados anteriores são sobreescritos.

3.4 RESULTADOS E DISCUSSÃO

Para a validação do sistema desenvolvido foi necessário realizar um experimento para validar a capacidade em identificar os objetos de acordo com sons gravados pelo próprio sistema. O teste utiliza apenas as ferramentas disponibilizadas pelo próprio sistema.

O teste propoposto foi dividido em duas etapas. A primeira de controle, sem a utilização do algoritmo de interpolação desenvolvido, gerando menos caracteristicas. A segunda etapa utiliza o algoritmo para aumentar a quantidade de *fingerprints* geradas tentando assim, aumentar a precisão.

Para execução dos testes foram utilizados dois grupos de sons. Para o grupo mais controlado foram utilizadas duas amostras das notas das cordas de um violão comum. Como segundo grupo, para verificação da efetividade foram utilizados sons não controlados como palmas e estalos.

3.4.1 Testes sem a utilização do algoritmo de interpolação

A primeira etapa dos testes foi realizada utilizando cadastro de características e comparação, sem a utilização do algoritmo de interpolação. A quantidade de características extraídas de cada som é reduzida, conforme demonstram os testes.

3.4.1.1 Teste em ambiente controlado

Para os testes controlados foram gravadas dois registros de áudio para cada corda do violão. O resultado das gravações é exibido na Tabela 1, relacionando as notas com a quantidade de características extraídas.

Tabela 1 – Base de amostras controladas sem interpolação

AMOSTRAS CONTROLADAS SEM INTERPOLACÃO	
Amostra	Quantidade de características
Nota mi (aguda) #1	128
Nota mi (aguda) #2	32
Nota si #1	64
Nota si #2	64
Nota sol #1	64
Nota sol #2	64
Nota ré #1	256
Nota ré #2	128
Nota lá #1	128
Nota lá #2	128
Nota mi (grave) #1	128
Nota mi (grave) #2	128

As amostras colhidas apresentam números próximos de quantidade de características extraídas. O número de *fingerprints* geradas resultar em potências de dois se dá pelo fato da baixa quantidade de informação de áudio capturada.

Após recolhidas estas amostras, foram efetuados cinco testes de identificação para cada nota. Os resultados são exibidos na Tabela 2, relacionando os acertos por nota para cada uma das duas coletas de cada nota. Os testes apresentam a soma de resultados onde a nota foi identificada como sendo a primeira nota mais provável e a segunda nota mais provável. São somadas ainda sem distinguir o algoritmo utilizado que a reconheceu.

Tabela 2 – Acertos por nota sem utilizar interpolação

ACERTOS POR NOTA SEM INTERPOLAÇÃO		
Nota	Amostra #1	Amostra #2
Mi (agudo)	4	4
Si	10	10
Sol	2	1
Ré	19	4
Lá	1	11
Mi (grave)	2	1

Os resultados obtidos podem variar de zero, nenhum reconhecimento, até vinte, que significa ser reconhecido por todos os algoritmos em todos os testes. Os resultados obtidos para os sons controlados sem utilização de interpolação variou entre 5% e 95% de acerto, sendo que a maioria não chegou 25%. O método utilizado se mostrou ineficiente para o reconhecimento efetivo de objetos.

3.4.1.2 Teste com sons não controlados

Para sons não controlados foi escolhido um grupo menor de amostras que é apresentado na Tabela 3.

Tabela 3 – Base de amostras não controladas sem interpolação

AMOSTRAS NÃO CONTROLADAS SEM INTERPOLACÃO	
Amostra	Quantidade de características
Palma #1	4
Palma #2	2
Estalo #1	1
Estalo #2	1
Plástico #1	4
Plástico #2	4
Metal #1	16
Metal #2	32

Novamente é possível verificar que as amostras apresentam números semelhantes por serem extraídas de áudios muito curtos. A quantidade de fingerprints geradas é ainda menor pois o áudio também é mais curto.

As amostras recolhidas foram submetidas aos mesmos testes dos sons controlados. Os resultados são demonstrados na Tabela 4.

Tabela 4 – Acertos por som não controlado sem utilizar interpolação

ACERTOS POR SOM NÃO CONTROLADO SEM INTERPOLAÇÃO		
Nota	Amostra #1	Amostra #2
Palma	0	0
Estalo	0	0
Plástico	0	0
Metal	16	17

Relacionando o número de características extraídas com os acertos obtidos, é possível ver que o sistema não foi capaz de identificar nenhum dos sons cadastrados. O objeto de metal possui um alto número de acertos por se tratar do objeto com maior número de características extraídas e gravadas na base, para utilização na comparação.

3.4.2 Teste utilizando algoritmo de interpolação

Para a segunda etapa do teste as amostras foram coletadas utilizando o algoritmo de interpolação, aumentando assim a quantidade de características geradas. O algoritmo de interpolação é executado na entrada de áudio oriunda da gravação do microfone ou da leitura de arquivo e provê um aumento na quantidade de características a serem extraídas sem perder a curva original de áudio obtida.

3.4.2.1 Teste em ambiente controlado

Os testes controlados de interpolação utilizam novamente os sons das notas de um violão. As características extraídas são submetidas ao algoritmo de interpolação proposto. A Tabela 5 apresenta as amostras coletadas e o número de características extraídas de cada uma.

Tabela 5 – Bases de amostras coletadas com interpolação

AMOSTRAS CONTROLADAS COM INTERPOLAÇÃO	
Amostra	Quantidade de características
Nota mi (aguda) #1	379
Nota mi (aguda) #2	268
Nota si #1	462
Nota si #2	482
Nota sol #1	916
Nota sol #2	894
Nota ré #1	830
Nota ré #2	1613
Nota lá #1	1481
Nota lá #2	1706
Nota mi (grave) #1	1896
Nota mi (grave) #2	1764

Ao analisar a Tabela 5 podemos notar que ao registrar as ocorrências de som, após a interpolação quanto mais grave maior a quantidade de *fingerprints* geradas. Com base nos dados obtidos é possível ver que a quantidade de características extraídas subiu conforme o esperado.

Após obtenção dos dados o mesmo teste aplicado anteriormente foi efetuado. Os resultados para os testes utilizando o algoritmo de interpolação em sons controlados é exibido na Tabela 6.

Tabela 6 – Acertos por nota utilizando interpolação

ACERTOS POR NOTA COM INTERPOLAÇÃO		
Nota	Amostra #1	Amostra #2
Mi (agudo)	0	0
Si	1	0
Sol	1	0
Ré	5	5
Lá	12	5
Mi (grave)	12	4

Com base nos dados obtidos é possível ver que a quantidade de acertos não aumentou conforme o esperado, ocorrendo ainda uma queda no número de acertos.

3.4.2.2 Teste com sons não controlados

As amostras de sons não controlados extraídas executando o algoritmo de interpolação são descritos na Tabela 7.

Tabela 7 – Base de amostras não controladas sem interpolação

AMOSTRAS NÃO CONTROLADAS COM INTERPOLACÃO	
Amostra	Quantidade de características
Palma #1	62
Palma #2	78
Estalo #1	80
Estalo #2	7
Plástico #1	64
Plástico #2	103
Metal #1	420
Metal #2	372

Novamente a quatidade de características extraídas subiu como o esperado, por se tratar de um som interpolado.

Os sons interpolados foram submetidos aos testes de identificação e o resultado é apresentado na Tabela 8.

Tabela 8 – Acertos por som não controlado utilizando interpolação

ACERTOS POR SOM NÃO CONTROLADO COM INTERPOLAÇÃO		
Nota	Amostra #1	Amostra #2
Palma	1	1
Estalo	12	0
Plástico	0	1
Metal	19	17

Conforme demonstrado na Tabela 8, apesar da execução do algoritmo de interpolação os resultados para identificação de sons não controlados continuou se demonstrando insatisfatório. Os sons de metal se destacam por serem reconhecidos por possuírem a maior quantidade de *fingerprints* extraídas.

3.4.3 Seção de discussões

Diante dos resultados obtidos, pode-se notar que o objetivo do trabalho que tratava de identificar os objetos não foi alcançado. O principal fator para a determinação da falha no atingimento dos resultados se dá pelo fato do número baixo de características geradas pelos áudios utilizados tanto na gravação quanto no processo de reconhecimento.

Apesar da falta de precisão, destaca-se o fato de que sons controlados sem a utilização de interpolação tiveram o maior índice de acerto entre os testes efetuados. Ao utilizar o algoritmo de interpolação para aumentar o número de características extraídas essa precisão foi perdida. O algoritmo de interpolação se mostrou eficáz na sua tarefa, porém afetou negativamente os resultados obtidos.

Pode-se notar também que os resultados obtidos tendem a identificar como som atual aquele que possui mais características extraídas e salvas na base. Esse resultado foi obtido por se tratarem de poucas características, tendendo a reconhecer o que possui mais informações para serem comparadas.

4 CONCLUSÕES

Este trabalho se propôs a implementar um sistema de identificação de objetos que extraísse caracteristicas através do som produzido e as utilizasse para identificá-los após o registro de informações de objetos na base.

A partir dos resultados alcançados, é possível ver que o objetivo principal não foi alcançado. Ao utilizar amostras de áudio de objetos, cujas durações são demasiadamente curtas, foi observado que não é possível extrair características suficientes para apresentar precisão na identificação.

Os testes demonstraram que a utilização de algoritmos de identificação aplicados a poucas características geram um resultado insatisfatório. O ambiente que possui melhores resultados é o de sons controlados sem utilização de algoritmos de interpolação. Ainda assim a precisão foi baixa.

Como limitações, dada a abordagem do sistema, somente é possível efetuar o reconhecimento de sons oriundos do microfone, efetuando o reconhecimento em tempo real. É possível destacar ainda que o cadastro de objetos e suas características deve ser feito manualmente um a um, o que dificultou a execução de testes. Ao utilizar o algoritmo de interpolação para aumentar a quantidade de características extraídas, a precisão caiu.

No decorrer do desenvolvimento deste trabalho se mostrou difícil encontrar trabalhos correlatos com o mesmo propósito deste. Para contornar tal situação buscou-se trabalhos que se referiam a identificação de músicas. Essa abordagem não se mostrou eficiente para o propósito de idenficação de sons de menor duração nos testes executado. Outra dificuldade foi observada pelo fato de não utilizar uma massa de dados para teste. Com testes manuais, se tornou difícil validar a precisão do algoritmo, resultando assim em um baixo número de testes.

De maneira geral, este trabalho poderá ser usado para consultas referentes aos resultados obtidos em se aplicar técnicas de extração de características e de reconhecimento para sons de curta duração.

4.1 EXTENSÕES

Como sugestões para trabalhos futuros ao sistema de identificação de objetos propõe-se:

a) aprimoramento das técnicas de similaridade;
b) utilização de técnicas de inteligência artificial;

c) extensão da função de reconhecimento para permitir reconhecer objetos informados em arquivos de áudio, utilizando uma massa de arquivos com pequenas variações para testar a eficiência dos algoritmos de similaridades em ambiente controlado e quantitativo.

REFERÊNCIAS BIBLIOGRÁFICAS

APACHE SOFTWARE FOUNDATION. **Math – commons math:** the apache commons mathematics library. [S.l.], 2013. Disponível em: <http://commons.apache.org/proper/commons-math/>. Acesso em: 25 mai. 2013.

BUNNELL, Timothy. **Sound in the frequency domain**. [S.l.], fev. 1996a. Disponível em: <http://www.asel.udel.edu/speech/tutorials/acoustics/freq_domain.html>. Acesso em: 11 jun. 2013.

CANO, Pedro et al. **A review of audio fingerprinting**. Journal of VLSI Signal Processing, 2005. Disponível em: < http://www.music.mcgill.ca/~ich/classes/mumt611_08/fingerprinting/cano05review.pdf>. Acesso em 29 maio 2013.

CANO, Pedro. **Content-based audio search:** from fingerprinting to semantic audio retrieval. 2006. 214 f. Tese (Doutorado em ciências da computação e comunicação digital) – Technology Department, Pompeu Fabra University, Spain.

CARLASSARA, Diogo. **Visualização de imagens capturadas em um circuito fechado de televisão (cftv) no iphone**. 2009. 68 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Santa Catarina.

DOROW, Anderson. **Mecanismo de busca semântica de áudio**. 2011. 70 f. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Santa Catarina.

LARSEN, Vengard A. **Combining Audio Fingerprints**. 2006. 137 f. Dissertação (Mestrado em ciências da computação) - Department of Computer and Information Science, Norwegian University of Science and Technology, Noruega.

MELLO, Jorge W. K. **Arquitetura de hardware para transformada rápida de Fourier aplicada ao tratamento de sinais do sistema nervoso**. 2012. 113 f. Trabalho de Conclusão de Curso (Engenharia de Computação) –Instituto de Informática, Universidade Federal do Rio Grande do Sul, Rio Grande do Sul.

ORACLE. **Java sound API**. [S.l.], [2013]. Disponível em: <http://docs.oracle.com/javase/tutorial/sound/>. Acesso em: 23 maio 2013.

SMITH, Steven W. **The scientist and engineer's guide to digital signal processing**. [S.l.]: California Technical Publishing, 1997. Disponível em: <http://www.dspguide.com/pdfbook.htm>. Acesso em: 29 maio 2013.

VEJA. **Mortes por homicídio aumentaram 32 % no país em 15 anos.** [2010]. Disponível em: <http://veja.abril.com.br/noticia/brasil/mortes-por-homicidio-aumentaram-32-no-pais-em-15-anos>. Acesso em: 09 ago. 2013.

VIEIRA, José et al. **Localização de fontes sonoras com tolerância a ambientes reverberantes**. 2003. 132 f. Dissertação (Mestrado em ciências da fala e audição) – Escola Superior de Saúde, Universidade de Aveiro, Portugal.

WIGNALL, Michael D. **Content-based music similarity**. 2003. 175 f. Trabalho de Conclusão de Curso (Bacharelado em Engenharia de Software) - School of Information Technology and Electrical Engineering , The University of Queensland, Austrália.