IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Luís Miguel Fernandes Valério

novembro | 2015

Escola Superior de Tecnologia e Gestão

Vibroid

RELATÓRIO DE PROJETO

ENGª INFORMÁTICA

Miguel Valério

Novembro | 2015

Politécnico da Guarda
Polytechnic of Guarda

Escola Superior de Tecnologia e Gestão

Ficha de Identificação

Nome: Luís Miguel Fernandes Valério

Nacionalidade: Portuguesa

Telefone: 961445006

Morada: Rua 31 de Janeiro, nº40, 6300-769 Guarda

Correio eletrónico: migu3lval3rio@gmail.com

Curso: Engenharia Informática

Orientador: Professor Doutor Adérito Neto Alcaso & Professor Doutor Paulo Alexandre Vieira

Instituição: Instituto Politécnico da Guarda, Escola Superior de Tecnologia e Gestão

Morada: Av. Dr. Francisco Sá Carneiro 50, 6300-559 Guarda

Telefone/Fax: 271220100 / 271220150

Correio eletrónico: estg-geral@ipg.pt

Agradecimentos

Queria antes de mais, agradecer ao Professor Doutor Adérito Neto Alcaso e ao Professor Doutor Paulo Alexandre Vieira pela disponibilidade e suporte que tiveram no sucesso do desenvolvimento do projeto. Aos dois, um muito obrigado.

Resumo

Este trabalho tem como objetivo o desenvolvimento de uma aplicação para a recolha de dados através dos sensores - acelerómetro e magnético - de um Smartphone Android e também a recolha áudio para análise posterior, com vista a criar-se uma ferramenta auxiliar de diagnóstico ao funcionamento de equipamentos de base eletromecânica.

Índice

Lista de Figuras

Glossário

API	Interface de Programação de Aplicações - É um acrónimo que provém do inglês *Application Programming Interface.* Este interface é constituído por conjuntos de rotinas e padrões de programação que permite a construção de programas.
App	Designação de aplicações informáticas desenvolvidas para plataformas móveis.
AVD	Dispositivo Virtual Android – É um acrónimo que provém do Inglês *Android Virtual Device*. Permite ao utilizador ter uma perspetiva gráfica do dispositivo.
Gradle	É um sistema de automatização de *builds.* Vem por padrão em todos os projetos em Android Studio.
FFT	*Fast Fourier Transform* (Transformada Rápida de Fourier) – ferramenta matemática para análise de dados.
Framework	Conjunto de classes implementadas em linguagem de programação para auxiliar o desenvolvimento de *software*.
HAXM	Intel *Hardware Accelerated Execution Manager* (Intel® HAXM) – É um motor de virtualização que acelera a emulação de uma App.
SENSOR_DELAY_NORMAL	Taxa (padrão) apropriada para as alterações de orientação do ecrã
SENSOR_DELAY_UI	Taxa apropriada para a interface do utilizador
SENSOR_DELAY_GAME	Taxa apropriada para os jogos
SENSOR_DELAY_FASTEST	Obter os dados o mais rápido possível

1- Introdução

Passaram-se alguns anos desde que se definiu o objetivo para que grande parte da população possuísse um computador em sua casa com acesso à internet. Nos dias que correm e com o crescimento exponencial dos Smartphones o objetivo passou a que todos tenham um no bolso…com internet.

Os computadores são um produto da evolução da eletricidade, assim como os motores elétricos. Estes têm uma importância vital em muitas atividades mas estão sujeitos a falhas de funcionamento que, de acordo com as estatísticas, têm como causa os rolamentos. Assim, é de extrema importância a monitorização destes elementos, o que faz sobretudo através da análise das suas vibrações e sensores do tipo acelerómetro. A aquisição de um sistema de análise geral para máquinas não críticas para a indústria, torna-se dispendioso e o retorno financeiro pode ser longo.

O objetivo deste trabalho é o de potenciar um Smartphone através de uma App no sistema operativo Android, para a recolha de dados de máquinas rotativas de baixa tensão, utilizando para isso o sensor já existente em qualquer equipamento moderno – o acelerómetro.

A monitorização periódica, recolha e análise dos dados através destes equipamentos, com baixo custo para a indústria, pode tornar-se assim viável e na nossa opinião muito importante.

Na análise de mercado encontraram-se algumas Apps que mostram as leituras, mas são limitadas, dado que não fazem sequer a recolha e gravação dos dados para ficheiro para futura análise.

Pretende-se com este trabalho que seja a iniciação de um projeto maior, dado que de futuro pretende-se que todo a análise dos dados e conclusões sejam feita no próprio Smartphone ou Tablet.

Este projeto faz parte da unidade curricular de Projeto de licenciatura em Engenharia Informática.

Por parte do aluno, o projeto tem como principais dificuldades de implementação, a inexistência de conhecimento em desenvolvimento para Android, plataformas móveis, sensores, motores elétricos e vibrações. Muito menos ainda na forma como estes vários itens se podem relacionar para desenvolver uma solução com interesse prático e aplicação comercial. Foi possível verificar as consequências que as avarias de máquinas podem causar e a importância do diagnóstico das suas falhas.

Além do mais, para aumentar a fiabilidade do diagnóstico é útil (e possível) usar informação de várias fontes de informação (além da vibração), tornando o sistema ainda mais desafiante de implementar.

2- Estado da Arte

A necessidade de analisar o funcionamento de sistemas eletromecânicos é de uma importância crucial em ambiente industrial. Esta análise pretende determinar se os equipamentos estão nas condições de funcionamento definidas pelas suas características construtivas e se as mantêm ao longo do tempo. Caso assim não seja, será necessário diagnosticar, em tempo oportuno, as novas condições de funcionamento e tomar ações que possam garantir a segurança de máquinas e pessoas.

Existem várias maneiras de proceder à análise do funcionamento deste tipo de equipamentos, sendo a mais comum a análise de vibrações. Tradicionalmente esta análise é feita com base em acelerómetros externos, aplicados nas máquinas, que medem as vibrações nas estruturas. O registo destas vibrações é depois processado em equipamentos dedicados, conhecidos por analisadores de espetros, que determinam as frequências dessas vibrações e, por comparação com resultados já conhecidos, concluir sobre o funcionamento do equipamento

A evolução tecnológica permitiu a integração do acelerómetro (e outros tipos de sensores) nos designados Smartphones e tabletes. Inicialmente estes sensores foram usados no âmbito de jogos, mas tem-se procurado alargar o seu campo de aplicação sendo a área do diagnóstico eletromecânico uma área de interesse.

Figura 1: App de Sensores de Smartphones

Alguns estudos e investigações académicas recentes foram já realizados para analisar a potencialidade desta aplicação, mostrando o interesse em torno desta ideia [1],[2]. Aliás o uso do Smartphone como ferramenta nas mais variadas situações é atualmente objeto de muitas teses de mestrado [3] e até doutoramento.

Em termos de aplicações para Smartphones existem já soluções que permitem visualizar as acelerações temporais aplicadas ao aparelho. Mais recentemente começaram a aparecer algumas soluções que executam também já a análise em frequência dos sinais dos sensores.

Figura 2: App de análise de frequências de vibração para Android

A solução desenvolvida neste projeto vem na sequência destas aplicações e pretende aprofundar o estudo sobre as potencialidades dos Smartphones como ferramenta auxiliar ao diagnóstico de sistemas eletromecânicos.

3- Solução

A solução encontrada para o problema proposto foi a criação de uma App para Smartphones com sistema operativo Android, desenvolvida na plataforma Android Studio.

Esta aplicação pretende ser num futuro próximo, uma ferramenta importante para a monotorização de máquinas, como motores elétricos, sobretudo os de baixa tensão que constituem uma grande parte deste tipo de equipamento.

a. O Android Studio

A escolha pelo Android Studio baseou-se na sua construção direcionada especificamente para o sistema operativo Android.

Além de poder desenvolver Apps para Smartphones e tabletes (que é o principal objetivo do projeto), este sistema possibilita também a construção de Apps para Android tv, Google Glass, Android Auto, entre outros, o que torna a aprendizagem mais aliciante. É um editor de código inteligente capaz de analisar código, *refactoring* e auto completar código avançado.

Em Android Studio, existe um novo conceito de módulos e livraria de módulos, que podem ser executados independentemente. Possui emuladores otimizados pré-configurados (*Android Virtual Device* ou AVD). O AVD Manager, permite selecionar as configurações dos equipamentos mais utilizados, resolução e tamanho de ecrã para um fácil *preview* da App. O Android Studio instala um emulador Intel® x86 Hardware *Accelerated Execution Manager* (HAXM) que permite uma rápida prototipagem da App.

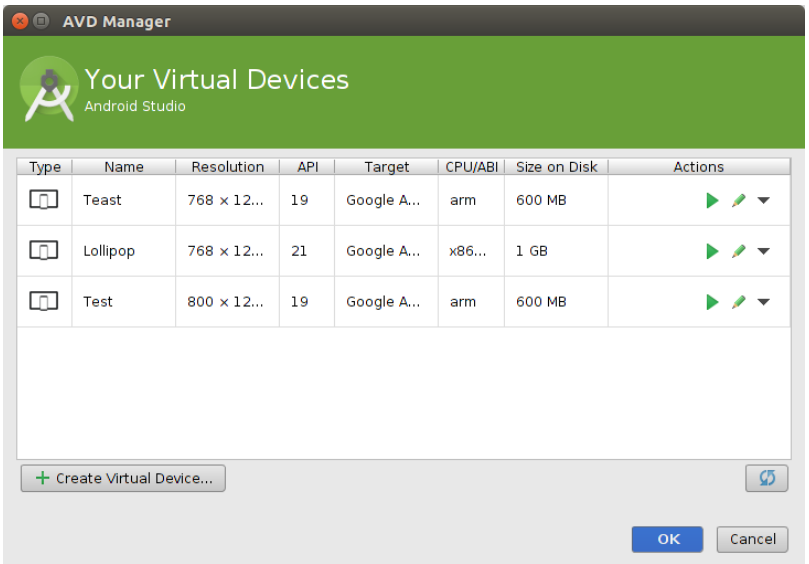


Figura 3: AVD Manager

O *Interface Designer* permite ver todos os componentes relacionados com o nosso projeto.

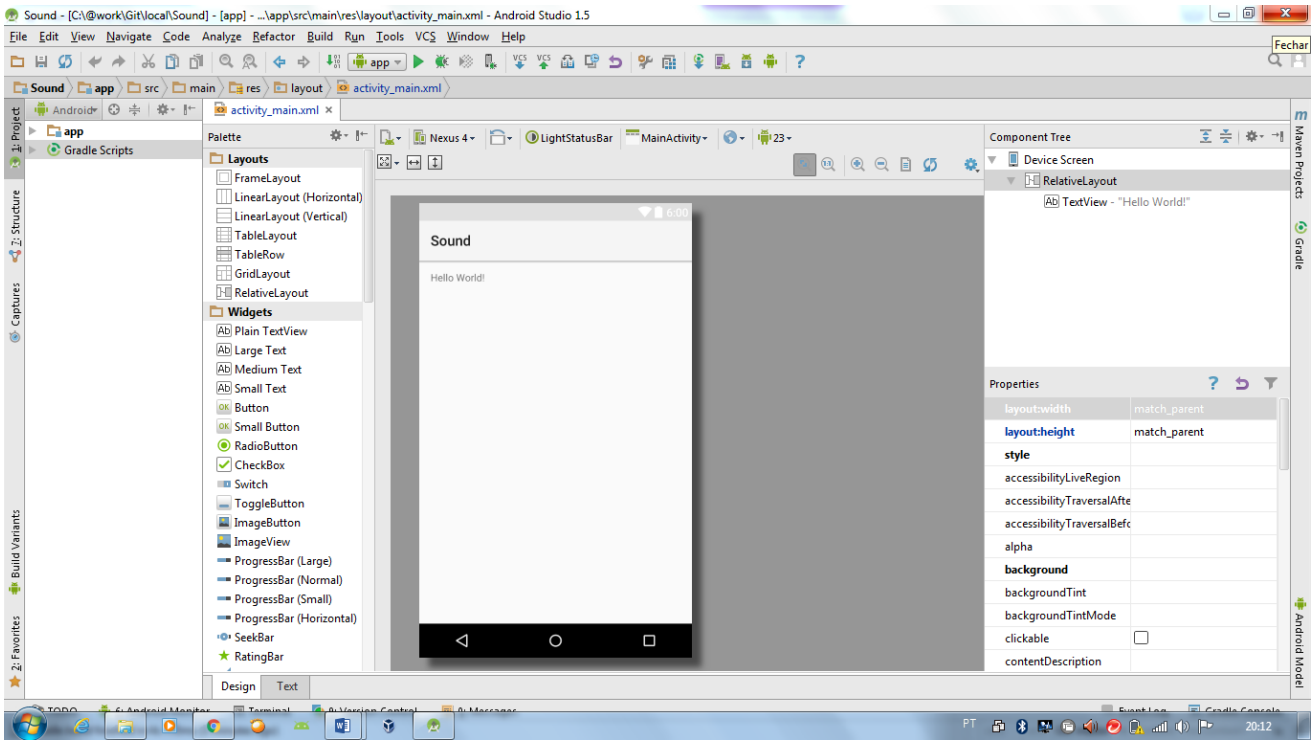


Figura 4: Interface Android Studio

Permite ainda acrescentar, de forma fácil, dependências de .jar criadas por terceiros.

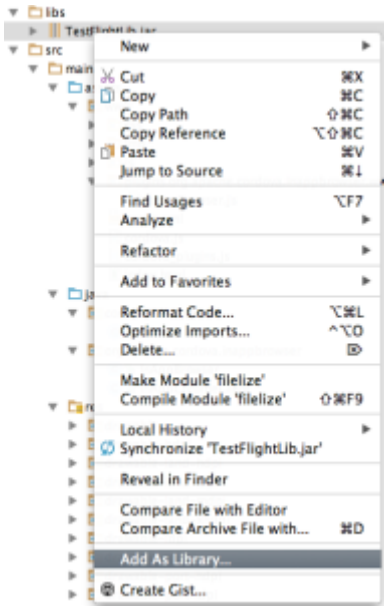


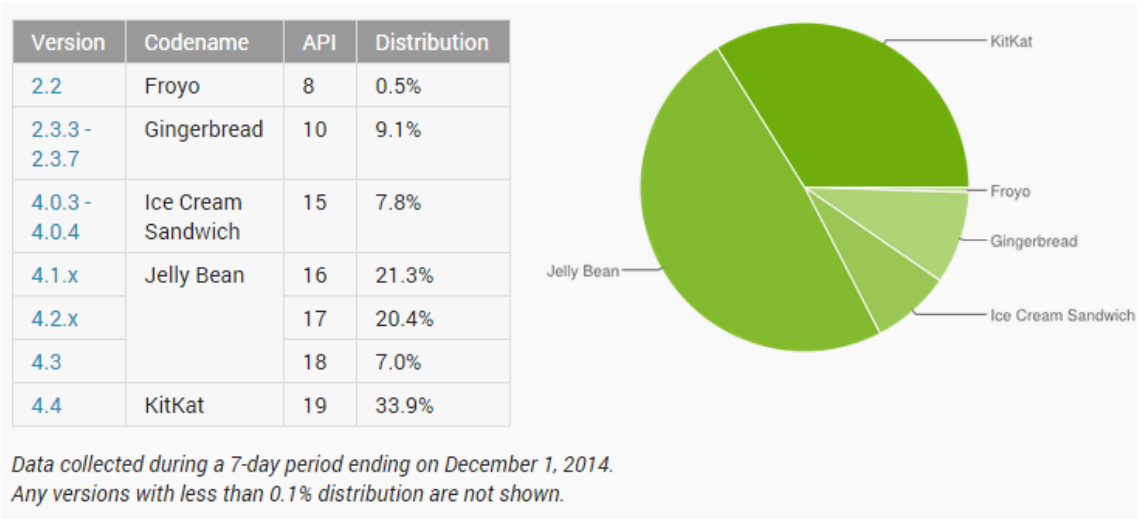
Figura 5: Adicionar Livrarias

Assim, o Android Studio gera automaticamente as entradas no Gradle que necessitamos quando acrescentamos módulos que dependem de outros.

b. Versão de sistema operativo

A escolha da versão do sistema operativo (SO) Android é também importante para o desenvolvimento do nosso projeto, dado que uma escolha num SO recente poderá ter efeitos nefastos para a App a desenvolver (mau funcionamento, crashs de software, bugs, etc.) em equipamentos com SO mais antigos.

Por outro lado a escolha de um SO Android antigo poderá não ter todas as funcionalidades necessárias com o que se pretende implementar. Através da figura 6 podemos concluir que a API 19 é a mais utilizada nos equipamentos Android e vai ser essa a versão que vai ser utilizada no nosso projeto.



Version	Codename	API	Distribution
2.2	Froyo	8	0.5%
2.3.3 - 2.3.7	Gingerbread	10	9.1%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	7.8%
4.1.x	Jelly Bean	16	21.3%
4.2.x		17	20.4%
4.3		18	7.0%
4.4	KitKat	19	33.9%

Data collected during a 7-day period ending on December 1, 2014.
Any versions with less than 0.1% distribution are not shown.

Figura 6: Gráfico de percentagem por distribuição da API

c. Sensores de Movimento

A plataforma Android fornece vários sensores que permitem monitorizar o movimento de um dispositivo. Em termos de *hardware* há dois sensores, o acelerómetro e o giroscópio sensíveis ao movimento direto, podendo o seu controlo ser feito por hardware ou software e originando indiretamente mais três informações, gravidade, aceleração linear e orientação.

Estes sensores aplicados aos equipamentos monitorizam o toque, a rotação e o movimento. Normalmente estes movimentos são uma reflexão dos *inputs* do utilizador, mas podem utilizar-se para refletir o meio ambiente físico, tal como as vibrações de uma máquina industrial.

Todos os sensores de movimento retornam um *array* de valores para cada SensorEvent. No nosso projeto, por exemplo, um evento retorna um float para os três eixos de coordenadas.

Além do acelerómetro usado para medida da vibração, usar-se-á também o de campo magnético de forma a permitir combinar a informação e torná-la mais robusta e objetiva. Assim, os sensores e informação que se vão utilizar neste trabalho são:

Sensor	**Sensor Event Data**	**Descrição**	**Unidade de medida**
Type_Accelerometer	SensorEvent.values[0]	Força da Aceleração no eixo x (inclui gravidade)	m/s^2
	SensorEvent.values[1]	Força da Aceleração no eixo y (inclui gravidade)	m/s^2
	SensorEvent.values[2]	Força da Aceleração no eixo z (inclui gravidade)	m/s^2
Type_Magnetic_Field	SensorEvent.values[0]	Campo magnético no eixo do x	μT
	SensorEvent.values[1]	Campo magnético no eixo do y	μT
	SensorEvent.values[2]	Campo magnético no eixo do z	μT

Por curiosidade e ilustrar as potencialidades de medida e informação que se pode retirar, listam-se os outros sensores existentes na *framework* do Android:

-*AMBIENT_TEMPERATURE* - Sensor de temperatura.

-*GRAVITY* - Sensor de gravidade.

-*GYROSCOPE* - Sensor de Aceleração (não inclui a gravidade).

-*LIGHT* - Sensor de LUZ.

-*LINEAR_ACCELERATION* - Sensor de aceleração linear.

-*ORIENTATION* - Sensor de orientação.

-*PRESSURE* - Sensor de pressão.

-*PROXIMITY* - Sensor de proximidade

-*RELATIVE_HUMIDITY* - Sensor de humidade relativa.

-*ROTATION_VECTOR* - Sensor de rotação.

d. Activity

Uma parte importante do ciclo de vida de uma aplicação em Android é a classe *Activity*. É fundamental a maneira como as *activities* são interligadas e como se iniciam.

Activities são tratadas pelo sistema como uma pilha. Quando uma nova *activity* começa, é colocada no topo da pilha e passa a ser a *activity* em funcionamento. A *activity* anterior passa para baixo da pilha e não volta para primeiro plano enquanto a nova *activity* existir.

Uma *activity* possui quatro estados:

- Se a *activity* está em primeiro plano (no topo da pilha), está ativa e a correr (RUN).
- Se a *activity* está transparente, mas visível, está em pausa (PAUSE). Uma *activity* que está neste estado está completamente "viva", mantém todos os estados e informações. Pode ser, eliminada pelo sistema em caso de situações de falta extrema de bateria.
- Se a *activity* esta completamente sobreposta por outra *activity*, está parada (STOP). Mantém todos os estados e a informação, mas deixa de ser visível para o utilizador e é muitas vezes eliminada quando a memória é necessária noutro local qualquer.
- Se uma *activity* está em PAUSE ou STOPPED, o sistema pode descartar a *activity* da memória ou simplesmente eliminar o processo. Quando é despoletada novamente pelo utilizador, é necessário o reinício completo e restauro do estado anterior.

Vamos mostrar com o diagrama seguinte os importantes estados de uma *activity*.

Os retângulos (cinza) representam os métodos *callback* que podem ser implementados para realizar operações quando a *activity* se move entre os estados.

Os retângulos com os cantos ovais, são os estados principais em que uma *activity* pode estar.

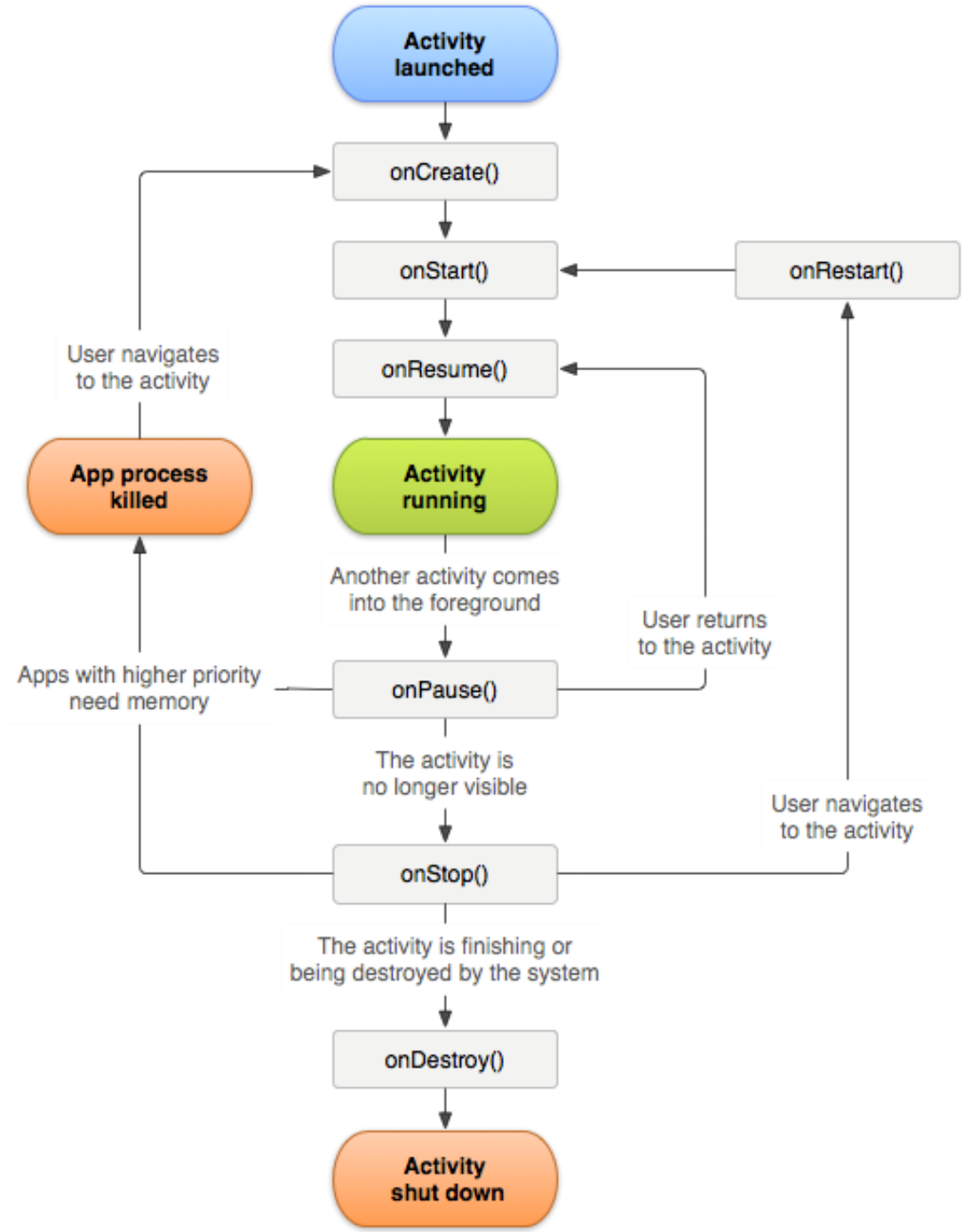


Figura 7: Ciclo de Vida de uma Activity [4]

A evolução de uma *activity* é dividida em ciclos de vida:

- O ciclo de vida acontece entre o onCreate() até o onDestroy(). Todas as atividades "globais" vão ser realizadas no estado onCreate(), e liberta todos os recursos no estado onDestroy(). Por exemplo: quando se tem uma *thread* que corre em *background* para fazer o *download* de dados através da rede, pode ser criada em onCreate() e parar a thread em onDestroy().

- O ciclo de vida visível numa *activity* acontece entre onStart() até a correspondente onStop(). Estes métodos podem ser chamados múltiplas vezes.

- O ciclo de vida em primeiro plano de uma *activity* acontece entre onResume() e onPause(). Nesta fase a *activity* está na frente de todas as outras *activities* e a interagir com os utilizadores. Pode frequentemente ir desde o estado onResume() para o onPause().

Quando se implementam estes métodos, tem que ser sempre chamada a super classe usando o seguinte código:

```
public class Activity extends Vibroid {
    protected void onCreate(Bundle savedInstanceState);

    protected void onStart();

    protected void onRestart();

    protected void onResume();

    protected void onPause();

    protected void onStop();

    protected void onDestroy();
}
```

e. Manifesto – “AndroidManifest.xml”

Todas as aplicações desenvolvidas em Android devem ter um arquivo AndroidManifest.xml, na raiz do diretório. É o arquivo principal do projeto.

Possui todas as configurações necessárias para a execução da aplicação, com o nome do *package*, o nome das classes das activities entre outras configurações.

Exemplo da declaração do *package* principal do projeto:

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"

    package="graph.app.com.vibroid_kappa"

    android:installLocation="preferExternal" >
```

Exemplo da declaração de uma activity:

```
<activity

            android:name=".sA"

            android:label="@string/title_activity_sa"

            android:parentActivityName=".Main" >

            <meta-data

                android:name="android.support.PARENT_ACTIVITY"

                android:value="graph.app.com.vibroid_kappa.Main" />

        </activity>
```

Para usarmos funcionalidades como, escrever no sdcard, gravar áudio, temos que as declarar no AndroidManifest.xml.

Exemplo de declaração para aceder ao SDcard:

```
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

f. Sensor - Acelerômetro

Um sensor de aceleração mede a aceleração aplicada ao aparelho, incluindo a força da gravidade.

De seguida mostra-se em código como obter por defeito uma instância do sensor:

```
private SensorManager mSensorManager;
private Sensor mSensor;
  ...
mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
```

Este sensor determina a aceleração que é aplicada a um equipamento (A_d) medindo as forcas que lhe são aplicadas (F_s), utilizando a seguinte relação:

$$A_d = - \sum F_s / mass$$

A força da gravidade é sempre influenciada de acordo com a seguinte relação:

$$A_d = -g - \sum F / mass$$

Por esta razão quando o equipamento está estático (sem acelaração), o leitor obtém sempre leituras de magnitude g=9.81 m/s^2. Da mesma forma, quando o equipamento está em queda (rápida aceleração) a leitura é de g=0 m/s^2.

No projeto em questão, não é necessário utilizar as leituras com a força da gravidade sendo possível remover esta aceleração da leitura do sensor, aplicando um *high-pass filter* ou isolando o valor, aplicando um *low-pass filter*. Para tal recorre-se ao código seguinte:

```
public void onSensorChanged(SensorEvent event){
  // Neste exemplo, alpha é calculado como t / (t + dT),
  // onde t é o filtro low-pass constante-tempo e
  // dT é a taxa do evento.
  // 0.8 é um valor demonstrativo

  final float alpha = 0.8;

  // Isolar a gravidade com low-pass filter

  gravity[0] = alpha * gravity[0] + (1 - alpha) * event.values[0];
  gravity[1] = alpha * gravity[1] + (1 - alpha) * event.values[1];
```

```
  gravity[2] = alpha * gravity[2] + (1 - alpha) * event.values[2];

  // Remove a força da gravidade com um filtro high-pass
  linear_acceleration[0] = event.values[0] - gravity[0];
  linear_acceleration[1] = event.values[1] - gravity[1];
  linear_acceleration[2] = event.values[2] - gravity[2];
}
```

Acelerómetros usam um sistema de coordenadas tradicional, ou seja, se o equipamento estiver sobre uma mesa e lhe aplicar um deslocamento para a esquerda (ou direita), o valor da aceleração de x vai ser positivo, se o deslocarmos para baixo (ou para cima, afastando-o do utilizador) o valor da aceleração de y, vai ser positivo. Por outro lado, se elevarmos o equipamento com uma aceleração de A m/s^2, o valor da aceleração de Z vai ser igual a A+9,81, que corresponde á aceleração do equipamento menos a força da gravidade.

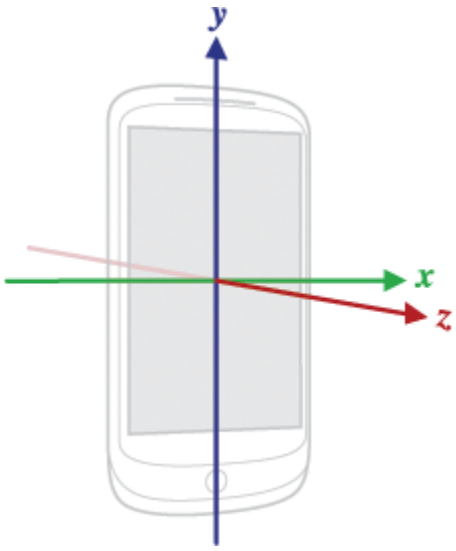


Figura 8: Sistema de Eixos [4]

Se o equipamento estiver sem movimentos, por parte do utilizador, vai existir uma aceleração de valor +9,81, que corresponde á aceleração do equipamento (0 m/s^2 menos a força da gravidade).

Quase todos os equipamentos android (Smartphones e tabletes) possuem um acelerómetro como sensor, dado que necessita de menos dez vezes energia que outros sensores de movimento.

g. Sensor – Magnético

Este sensor mede os valores do campo magnético ao redor do dispositivo relativo aos seus três eixos (x, y e z). Para tal usa-se a seguinte instância:

```
mSensorM = mSensorManagerM.getDefaultSensor(Sensor.TYPE_MAGNETIC_FIELD);

x = event.values[0];

y = event.values[1];

z = event.values[2];
```

A precisão da medida deste sensor é extremamente afetada por diversas interferências magnéticas provocadas pela aproximação de outros campos magnéticos ou materiais ferrosos. Para além desta desvantagem, existe outra que se prende com o facto do magnetómetro, após um movimento brusco, demorar algum tempo até estabilizar os seus registos novamente.

h. Classes Sensores

No projeto foram utilizadas as seguintes classes associadas aos sensores

Sensor - Classe que representa um sensor;

SensorManager - Classe que permite o acesso aos sensores do dispositivo, associa os *listeners* a eventos e fornece as constantes, por exemplo: a gravidade, o campo magnético entre outros;

SensorEvent - Classe que representa a leitura de um sensor, mostrando o *timestamp* do registro, o tipo do sensor e medições.

SensorEventListener - Classe que mostra as mudanças nos valores de leitura dos sensores.

i. Obtenção de uma instância de um Sensor

Primeiro temos que obter o SensorManager. Podemos fazê-lo chamando o método getSystemService() e passando o parâmetro Context.SENSOR_SERVICE.

O momento seguinte é através do SensorManager em que se obtém a instância do sensor executando o SensorManager.getDefaultSensor(), adquirindo o sensor que pretendemos, por exemplo para o nosso projeto, o Sensor.TYPE_ACCELEROMETER.

Exemplo de código de como se obtém uma instância de um Sensor;

```
private SensorManager mSensorManager ;

private Sensor mSensor ;

mSensorManager = ( SensorManager ) getSystemService ( Context .
SENSOR_SERVICE );

mSensor = mSensorManager . getDefaultSensor ( Sensor . TYPE_ACCELEROMETER );
```

j. Receber eventos do Sensor

Para receber os eventos de um sensor, vamos associar uma instância que implemente o SensorEventListener (utilizando o SensorManager) ao sensor que se pretende que receba os dados indicando também no parâmetro a frequência de recolha das informações. Esta associação é realizada através do método SensorManager.registerListener().

Assim o *listener* deve ser implementado com dois métodos:

SensorEventListener.onSensorChanged() – Quando existir mudança de valores no sensores, o método é chamado.

SensorEventListener.onAccuracyChanged() - Quando existir mudança de precisão no sensor, o método é chamado.

Um exemplo do código associado é:

```
public class SensorTest extends Activity implements SensorEventListener {

private SensorManager mSensorManager ;

private Sensor mSensor ;
```

```
protected void onCreate () {

super . onCreate ();

 ...

 mSensorManager = ( SensorManager ) getSystemService ( Context .
SENSOR_SERVICE );

 mSensor = mSensorManager . getDefaultSensor ( Sensor . TYPE_ACCELEROMETER );

 mSensorManager . registerListener (this , mSensor , SensorManager .
SENSOR_DELAY_NORMAL );

 }



 public void onSensorChanged ( SensorEvent event ){

 // utilização dos dados lidos do sensor

 }



public void onAccuracyChanged ( SensorEvent event , int accuracy ){

 // resposta à modificação de precisão

 }

}
```

A *framework* já possui intervalos pré-definidos em constantes: SENSOR_DELAY_NORMAL, SENSOR_DELAY_UI, SENSOR_DELAY_GAME, ou SENSOR_DELAY_FASTEST.

k. Captura de som

O som é uma vibração das moléculas do ar pelo que faz sentido o seu registo para o nosso projeto. Sendo a captura do som a tarefa primordial de qualquer telefone possui uma classe distinta dos outros sensores.

O Android possui uma API para o desenvolvimento de aplicações para reprodução e gravação de áudio.

Com esta framework do Android é possível reproduzir os mais comuns tipos de média. É possível a sua reprodução no estado puro ou codificado a partir de ficheiros ou em *streaming* da internet.

l. Classes Áudio

A *framework* do Android permite a gravação de áudio nos formatos mais usados. A classe mais importante é a MediaRecorder.

Para tal é necessário no manifesto o pedido de permissão para gravação de áudio.

```
<uses - permission android:name =" android . permission . RECORD_AUDIO " />
```

Para além disso são ainda necessárias as classes

MediaPlayer – Classe principal para a reprodução de som e vídeo.

AudioManager – Com esta classe podemos manipular as entradas e saídas do áudio.

Uma instância da classe MediaPlayer tem a capacidade de obter, reproduzir e descodificar ficheiros de áudio.

A classe MediaPlayer tem com principais funções:

MediaPlayer.setDataSource() – Indica o caminho do ficheiro, localmente ou remotamente, a ser reproduzida;

MediaPlayer.prepare() – Prepara o tipo de áudio, transformando em áudio puro, pronto a ser reproduzido. Tem a capacidade ainda de descodificar e pré-armazenar o ficheiro.

MediaPlayer.start() - Inicia a reprodução do áudio;

MediaPlayer.pause() – Faz a pausa na reprodução do áudio;

MediaPlayer.stop() – Termina a reprodução do áudio;

mediaPlayer.setAudioStreamType() – Identifica o tipo de áudio que vai ser reproduzido. Para arquivos de música o parâmetro será a constante AudioManager.STREAM_MUSIC.

m. Áudio pelo microfone

Quando se pretende recolher áudio pelo microfone é necessário efetuar alguns passos:

1. Temos que criar uma instância de android.media.MediaRecorder.

2. Utilizamos o método MediaRecorder.setAudioSource() atribuindo a constante MediaRecorder.AudioSource.MIC, dado que é o microfone que pretendemos.

3. Temos que configurar o tipo de saída com o MediaRecorder.setOutputFormat() passando o formato que pretendemos

Formatos possiveis:

AAC_ADTS - AAC ADTS formato de ficheiro

AMR_NB - AMR NB formato de ficheiro

AMR_WB - AMR WB formato de ficheiro

DEFAULT – Formato Padrão

MPEG_4 - MPEG4 formato de ficheiro media

THREE_GPP - 3GPP formato de ficheiro media

4. De seguida atribuímos um nome ao arquivo para a saída através do método MediaRecorder.setOutputFile() passando uma s*tring* que vai identificar o arquivo de saída, como por exemplo:

```
mRecorder . setOutputFile ( fileName );
```

5. Definimos o tipo de *encoder* através do método MediaRecorder.setAudioEncoder().

Opções possíveis:

AAC - AAC Low Complexity (AAC-LC) audio codec

AAC_ELD - Enhanced Low Delay AAC (AAC-ELD) audio codec

AMR_NB - AMR (Narrowband) audio codec

AMR_WB - AMR (Wideband) audio codec

DEFAULT - Tipo padrão

HE_AAC - High Efficiency AAC (HE-AAC) audio codec

6. Depois destas informações definidas podemos chamar o método MediaRecorder.prepare().

7. Quando o gravador (recorder) estiver preparado podemos iniciar assim a gravação com o método MediaRecorder.start().

8. Para parar a gravação chamamos o método MediaRecorder.stop().

9. Após a conclusão da gravação temos que chamar o método MediaRecorder.release() para libertação dos recursos ocupados pelo gravador.

O código de exemplo para o efeito é:

```
MediaRecorder recorder = new MediaRecorder ();

recorder . setAudioSource ( MediaRecorder . AudioSource . MIC );

recorder . setAudioEncoder ( MediaRecorder . AudioEncoder . AMR_NB );

recorder . setOutputFormat ( MediaRecorder . OutputFormat . THREE_GPP );

recorder . setOutputFile ( PATH_NAME );

recorder . prepare ();

recorder . start (); // inicia a gravação

...

recorder . stop ();

recorder . release (); // Liberta os recursos
```

O MediaRecorder possui estados que, dependendo do estado em que o gravador está, alguns comandos podem ou não ser realizados.

De seguida mostra-se o mapa completo dos estados do MediaRecorder.

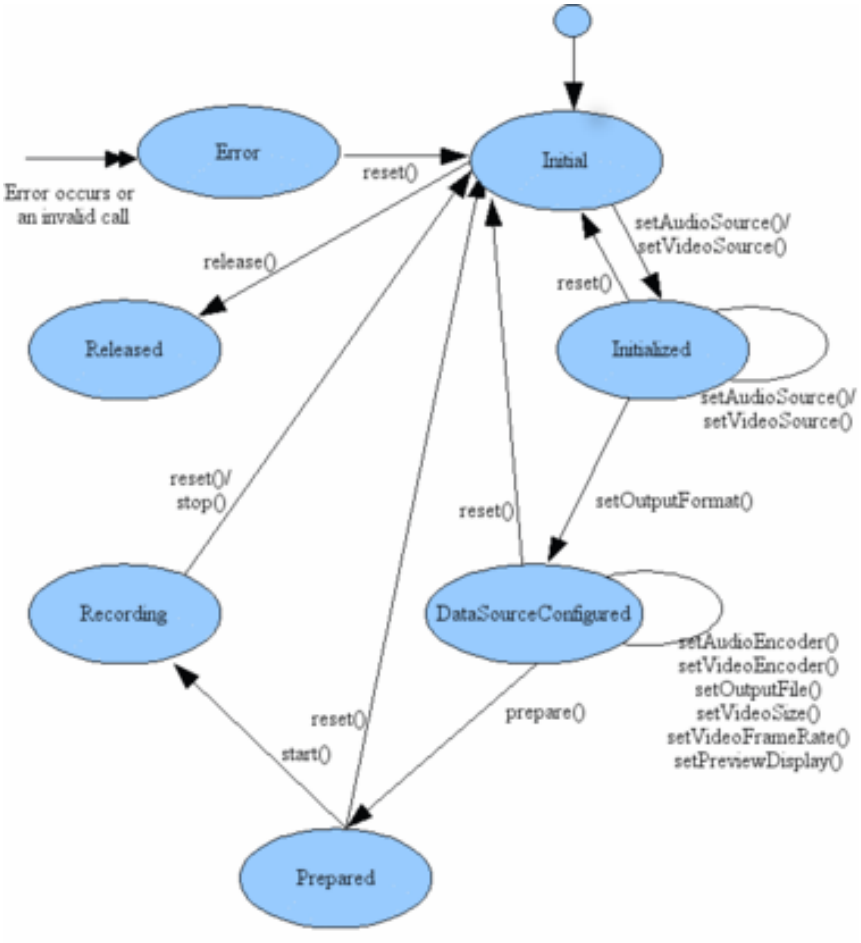


Figura 9: Diagrama de Estados MediaRecorder [4]

4- Vibroid (Prints)

Esta é a nossa página principal da nossa APP, onde podemos escolher qual o sensor a utilizar ou a aquisição do som.

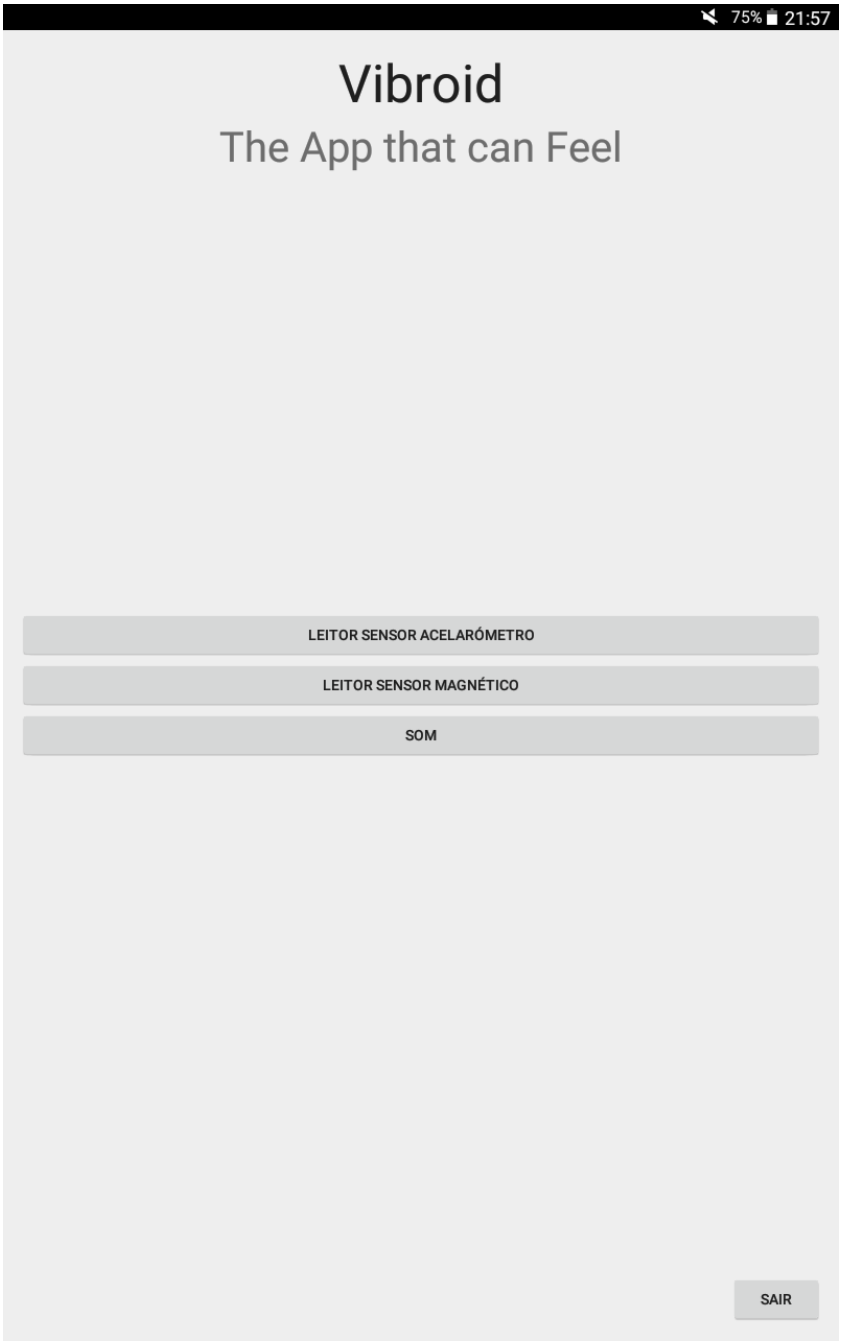


Figura 10: Página Inicial

Este é o menu do sensor acelarometro e do sensor magnétio

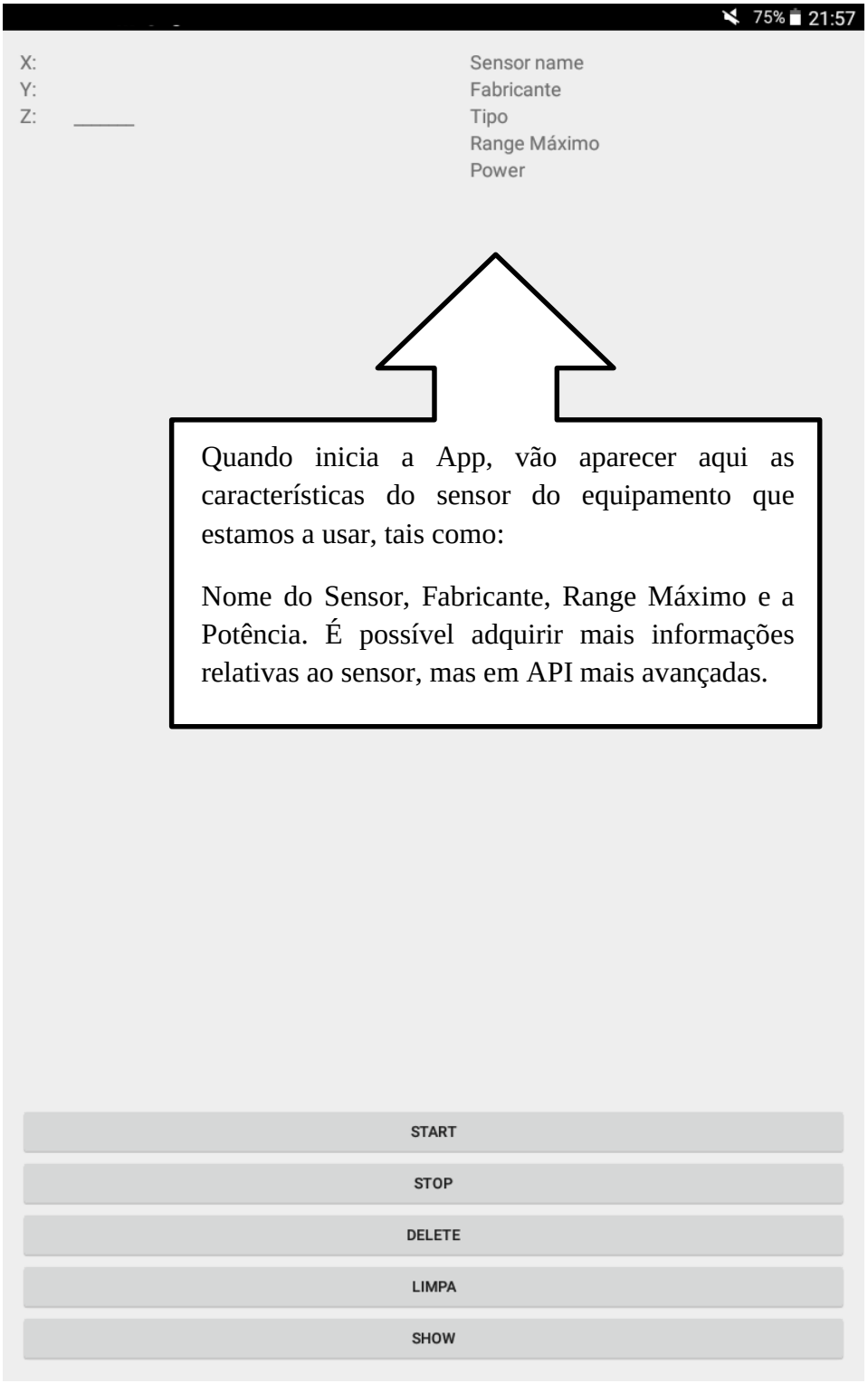


Figura 11: Ecrã Inicial dos sensores

Neste ecrâ podemos adquirir o som. Com o butão Start Recording e Start Playing.

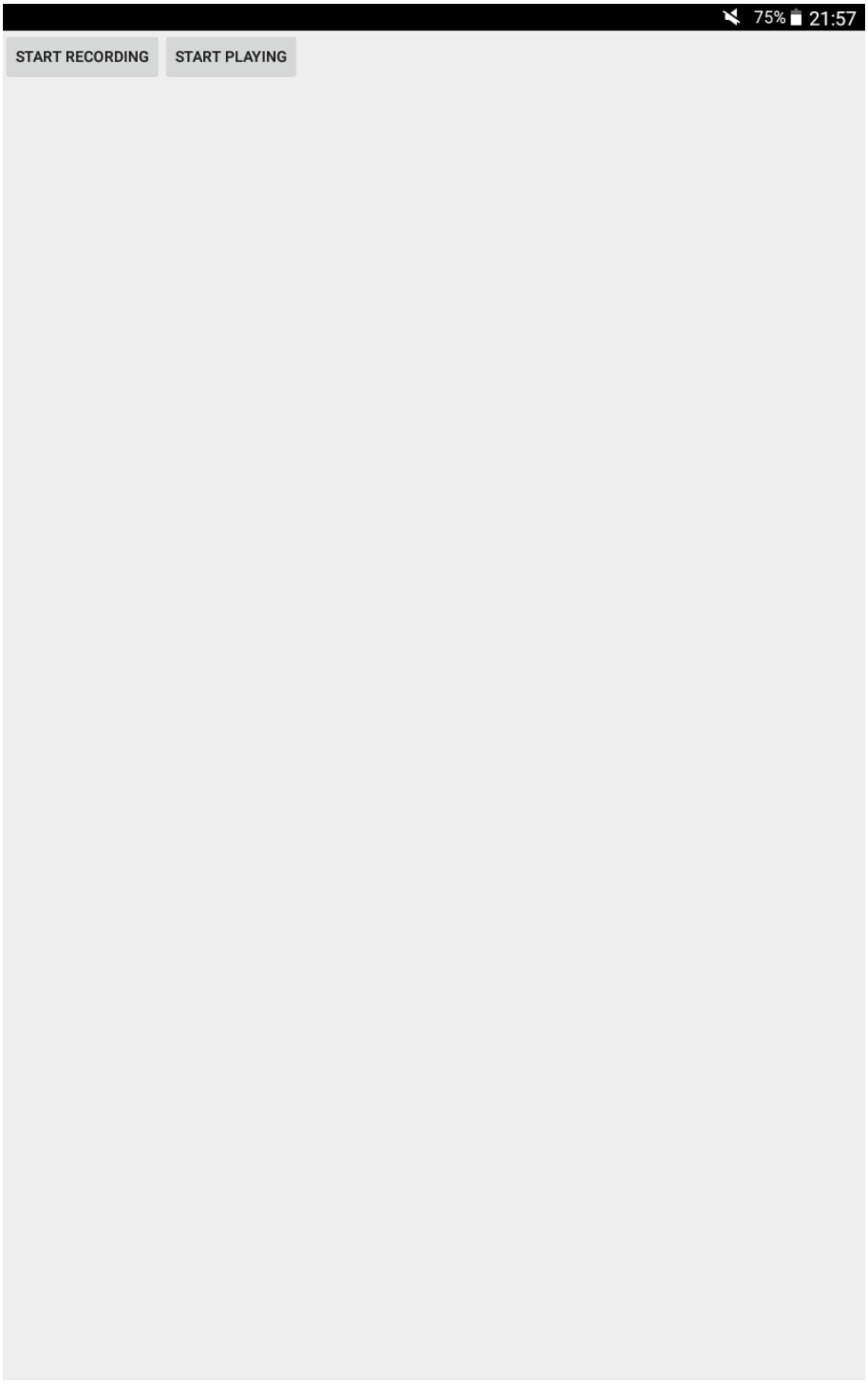


Figura 12: Ecrã de Aquisição de Som

Neste ecrã, mostra os resultados obtidos.

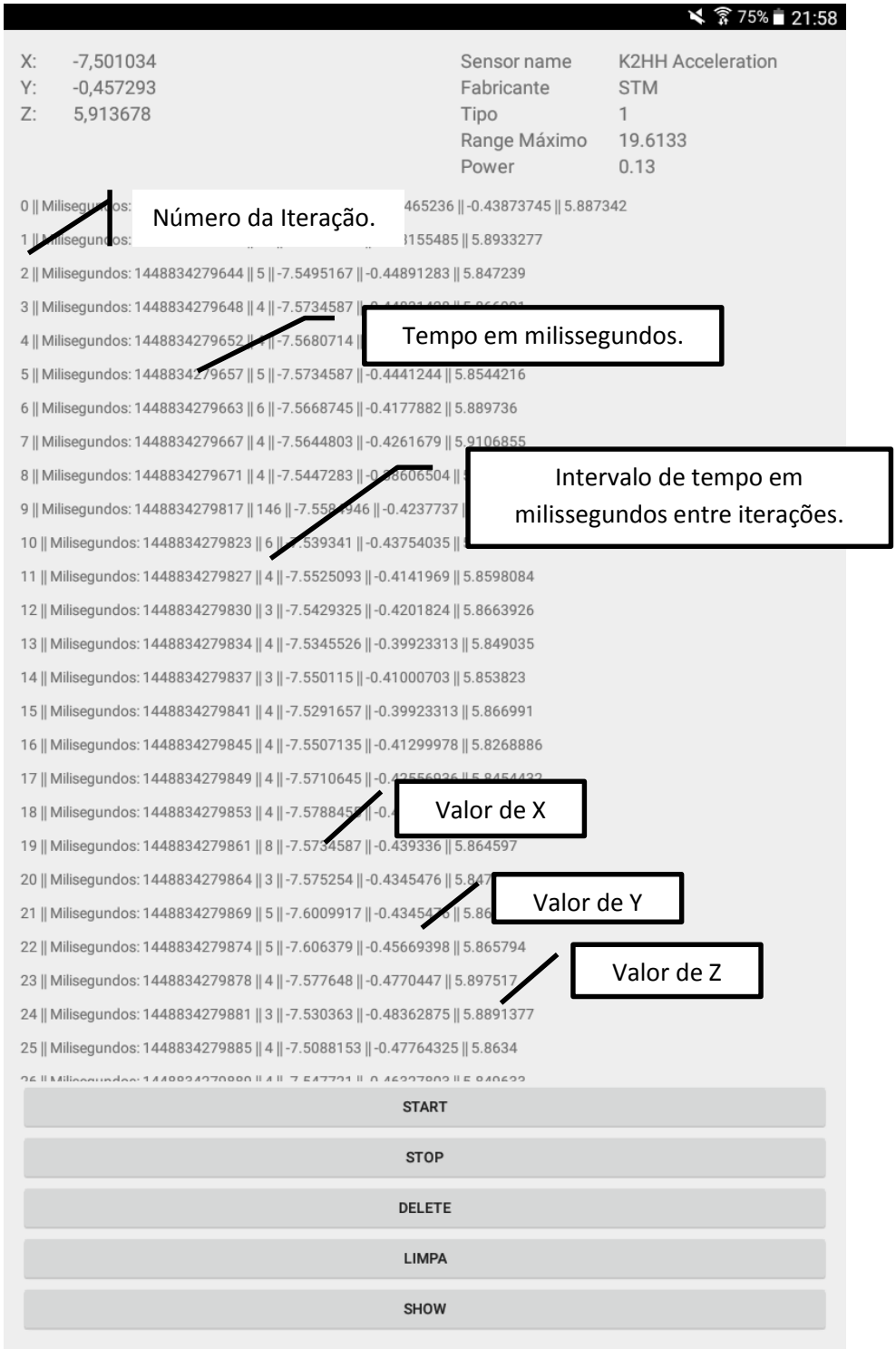


Figura 13: Ecrã de Aquisição de Som

5- Conclusão

Este trabalho permitiu desenvolver uma aplicação preliminar para dispositivos Android, para aquisição dos dados relativos ao acelerómetro e magnetómetro, com o intuito futuro de poder aplicar um Smartphone/Tablet como meio inicial de diagnóstico de falhas e avarias em sistemas eletromecânicos, nomeadamente motores elétricos. Face à inexistência de *background* em termos de programação de dispositivos móveis e técnicas de diagnóstico deste tipo de sistemas, a tarefa proposta revelou-se desafiadora. Por outro lado a evolução constante dos sistemas operativos Android e ferramentas de programação tornou ainda mais laboriosa a obtenção dos objetivos pretendidos. Como resultado do projeto foi possível recolher a informação pretendida e apresentá-la numa forma de tabela permitindo a sua leitura direta.

Para dar continuidade ao desenvolvimento deste trabalho será necessário proceder à análise da informação, usando ferramentas como a FFT (normalmente usada neste tipo de situações) e extrair resultados que permitam concluir sobre o estado de funcionamento dos sistemas eletromecânicos. Para aplicação da FFT é importante que o intervalo entre amostras seja o menor possível.

Para potenciar a fiabilidade da solução idealizada deverá ser feita a fusão da informação recolhida pelos sensores já referidos. Esta informação deverá ser analisada tendo em conta os conhecimentos já existentes sobre o comportamento deste tipo de sistemas em diversas situações.

O uso de um Smartphone/Tablet como ferramenta auxiliar de diagnóstico ao funcionamento deste tipo de sistemas mecânicos, começa a dar os primeiros passos, mas atendendo às capacidades crescentes de processamento que estes dispositivos têm, é expetável que esta seja mais um área de aplicação daqueles, que evoluíram de simples telefones portáteis para computadores de bolso com capacidade de aquisição de dados do mundo físico.

Bibliografia / Web grafia

- [1] Jungchan Cho, Inhwan Hwang, Songhwai Oh, "Vibration-Based Surface Recognition for Smartphones", Proceedings of IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, 2012;
- [2] De Dominicis, C.M., Depari, A., Flammini, A., Sisinni, E., more authors, "Performance Assessment of Vibration Sensing using Smartdevices", Proceedings of IEEE International Conference on Instrumentation and Measurement Technology, 2014;
- [3] António Ricardo Teixeira Pessanha Carvalho, "Sistema móvel de telemetria para automóveis", Dissertação de Mestrado, ISEP, http://recipp.ipp.pt/bitstream/10400.22/4620/1/DM_AntonioCarvalho_2013_MEI.pdf, (acedido a 30/07/2015);
- Greg Milette, Adam Stroud (2012 - Wrox) *Professional Android Sensor Programming*;
- Reto Meier (2012 - Wrox) *Professional Android 4 Application Development;*
- Sayed Hashimi & Satya Komatineni & Dave MacLean (2010 - Apress) *Pro Android 2;*
- Ian F. Darwin (2012 - O´ Reilly Media) *Android Cookbook;*
- *[4] Android Studio Overview* - http://developer.android.com/tools/studio/index.html, (acedido a 30/07/2015);

ANEXO

Código Sensor Acelerómetro

```
package graph.app.com.vibroid_kappa;
import android.annotation.TargetApi;
import android.app.Activity;
import android.content.Context;
import android.hardware.Sensor;
import android.hardware.SensorEvent;
import android.hardware.SensorEventListener;
import android.hardware.SensorManager;
import android.os.Bundle;
import android.os.Environment;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.TextView;
import android.widget.Toast;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.Objects;
import java.io.OutputStream;


public class sA extends Activity implements SensorEventListener {

    public static final String DATA_COLLECTION_FILE = ("Valores_Acelarometro.txt");
    public static File myDataCollection = null;
    TextView dataX;
    TextView dataY;
    TextView dataZ;
    TextView SensorData;
    TextView tFabricante;
    TextView tTipo;
    TextView tRangeMaximo;
    TextView tPower;
    TextView LoadText;
    private long millprevious = 0;
    private SensorManager mSensorManager;
    private Sensor mSensor;
    private int Scount = 0;
```

```
  @Override
  protected void onResume() {
    super.onResume();

  }

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //Cria a pasta no sistema

    String newFolder = "/Vibroid";
    String extStorageDirectory = Environment.getExternalStorageDirectory().toString();
    File myNewFolder = new File(extStorageDirectory + newFolder);
    myNewFolder.mkdir();

    myDataCollection = new File(extStorageDirectory + newFolder + "/" +
DATA_COLLECTION_FILE);
    try {
      if (!myDataCollection.exists()) {

        myDataCollection.createNewFile();
      }
    } catch (IOException ioExp) {
      Log.d("AndroidSensorList::", "Erro na criação do ficheiro");
    }

    setContentView(R.layout.activity_sa);

    mSensorManager = (SensorManager) getSystemService(Context.SENSOR_SERVICE);
    mSensor = mSensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);
    SensorData=(TextView)findViewById(R.id.SensorData);
    tFabricante=(TextView)findViewById(R.id.tFabricante);
    tTipo=(TextView)findViewById(R.id.tTipo);
    tRangeMaximo=(TextView)findViewById(R.id.tRangeMaximo);
    tPower=(TextView)findViewById(R.id.tPower);
    dataX=(TextView)findViewById(R.id.dataX);
    dataY=(TextView)findViewById(R.id.dataY);
    dataZ=(TextView)findViewById(R.id.dataZ);
  }

  public void buttonOnClickStop(View v) {

    mSensorManager.unregisterListener(this);
    Toast.makeText(getBaseContext(),
```

```
            "Gravação parada",
            Toast.LENGTH_SHORT).show();
    }

    public void buttonOnClickSart(View v) {

        if (myDataCollection.exists())
            myDataCollection.delete();

        mSensorManager.registerListener(this, mSensor,
SensorManager.SENSOR_DELAY_FASTEST);
        Toast.makeText(getBaseContext(),
            "A criar ficheiro",
            Toast.LENGTH_SHORT).show();
    }


    public void buttonOnClickDelete(View v) {

        myDataCollection.delete();
        Toast.makeText(getBaseContext(),
            "Ficheiro Apagado",
            Toast.LENGTH_SHORT).show();


    }

    public void LimpaTexto(View v) {

        LoadText.setText("");
    }

    public void ButtonTextLoad(View v) {
        LoadText = (TextView) findViewById(R.id.LoadText);
        //Escreve valores na caixa de texto
        try {
            File myFile = new File("/sdcard/Vibroid/" + DATA_COLLECTION_FILE);
            FileInputStream fIn = new FileInputStream(myFile);
            BufferedReader myReader = new BufferedReader(
                new InputStreamReader(fIn));
            String aDataRow = "";
            String aBuffer = "";
            while ((aDataRow = myReader.readLine()) != null) {
                aBuffer += aDataRow + "\n";
            }
            LoadText.setText(aBuffer);
            myReader.close();
            Toast.makeText(getBaseContext(),
```

```
            "Done reading SD 'Vibroid.txt'",
            Toast.LENGTH_SHORT).show();
    } catch (Exception e) {
        Toast.makeText(getBaseContext(), e.getMessage(),
            Toast.LENGTH_SHORT).show();
    }
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {

}

@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    getMenuInflater().inflate(R.menu.menu_sa, menu);
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle action bar item clicks here. The action bar will
    // automatically handle clicks on the Home/Up button, so long
    // as you specify a parent activity in AndroidManifest.xml.
    int id = item.getItemId();

    //noinspection SimplifiableIfStatement
    if (id == R.id.action_settings) {
        return true;
    }

    return super.onOptionsItemSelected(item);
}
@TargetApi(19)
@Override
public void onSensorChanged(SensorEvent event) {

    dataX.setText(String.format("%.6f",event.values[0]));
    dataY.setText(String.format("%.6f",event.values[1]));
    dataZ.setText(String.format("%.6f",event.values[2]));
    SensorData.setText(String.format(""+event.sensor.getName() ));
    tFabricante.setText(String.format("" +event.sensor.getVendor() ));
    tTipo.setText(String.format("" +event.sensor.getType() ));
    tRangeMaximo.setText(String.format("" +event.sensor.getMaximumRange() ));
    tPower.setText(String.format("" +event.sensor.getPower()));
```

```
    /*+"Delay minimo :"  +event.sensor.getMinDelay()+"Delay Maximo :"
+event.sensor.getMaxDelay()*/


    System.out.println("+++++++++++++++++Dentro do onSensorChanged()
++++++++++++++++++++++");
    System.out.println("event.sensor.getName():" + event.sensor.getName());
    float x, y, z;

    x = event.values[0];
    y = event.values[1];
    z = event.values[2];
    writeDataTofile(event.sensor.getName(), x, y, z);

  }


    @TargetApi(19)
  public void writeDataTofile(String sensorsName, float x, float y, float z) {


    System.out.println(sensorsName + "::" + "X=" + x + "Y=" + y + "Z=" + z);


    long mill_diference;
    long mill = System.currentTimeMillis();
    mill_diference = mill - millprevious;
    millprevious = mill;
    String millstr = String.valueOf(mill_diference);
    String mills = Objects.toString(mill);
    //String Stime = DateFormat.getDateTimeInstance().format(new Date());
    String strScount = String.valueOf(Scount);
    Scount = Scount + 1;
    String xVal = String.valueOf(x);
    String yVal = String.valueOf(y);
    String zVal = String.valueOf(z);
    byte[] bcount = strScount.getBytes();
    byte[] bX_Value = xVal.getBytes();
    byte[] bY_Value = yVal.getBytes();
    byte[] bZ_Value = zVal.getBytes();
    String newLine = "\n";
    byte[] bnewLine = newLine.getBytes();
    byte[] bmill = millstr.getBytes();
    String sSeparator = " || ";
    String STimeMillis = "Milisegundos: ";
    byte[] STimeMillisZ = STimeMillis.getBytes();
    byte[] bSeparator = sSeparator.getBytes();
    //byte[] bSensorName = sensorsName.getBytes();
```

```
        byte[] Stimex = mills.getBytes();


        try {
            OutputStream fo = new FileOutputStream(myDataCollection, true);
            fo.write(bnewLine);
            fo.write(bcount);
            fo.write(bSeparator);
            fo.write(STimeMillisZ);
            fo.write(Stimex);
            fo.write(bSeparator);
            fo.write(bmill);
            //fo.write(bSensorName);
            fo.write(bSeparator);
            fo.write(bX_Value);
            fo.write(bSeparator);
            fo.write(bY_Value);
            fo.write(bSeparator);
            fo.write(bZ_Value);
            fo.write(bnewLine);
            fo.close();

        } catch (IOException e) {
            Log.e("AndroidSensorList::", "File write failed: " + e.toString());
        }
    }

    @Override
    protected void onPause() {
        super.onPause();
        mSensorManager.unregisterListener(this);
    }

}
```