UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL

ESCOLA DE ENGENHARIA

TRABALHO DE CONCLUSÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Utilização de dispositivo móvel na mensuração da Vibração de Corpo Inteiro (VCI)

Autor: Heittor Wermann

Orientador: Tiago Becker

Porto Alegre, dezembro de 2015

Sumário

Resumo iv

Lista de ilustrações v

Lista de Tabelas vi

Lista de Abreviaturas e Siglas vii

1 Introdução 1

2 Revisão Bibliográfica 2

2.1 Vibrações de Corpo Inteiro (VCI) 2

2.2 Normas e exposição à VCI. 2

2.3 Android, plataforma e aplicação. 3

3 Materiais e Métodos 7

3.1 O Aplicativo (versão 0.1) 11

3.1.1 Requisitos 11

3.1.2 Layout e funcionamento 11

3.2 O Aplicativo (versão 1.1) 12

3.2.1 Requisitos 14

3.2.2 Layout e funcionamento 14

3.2.3 Implementação 15

3.2.4 Manifesto 16

3.2.5 SplashActivity 16

3.2.6 HelpActivity 17

3.2.7 MainActivity 18

3.2.8 Diagrama de classes 20

3.3 O servidor 20

3.4 Instrumentação e calibração 21

3.4.1 Transdutor 21

3.4.2 Condicionador de sinal 22

3.4.3 Placa de aquisição de dados 22

3.4.4 Software para o tratamento de dados 23

4 Resultados. 24

5 Conclusões e Trabalhos futuros. 32

6 Referências 33

7 Anexos 34

APÊNDICE I – O aplicativo. 41

Resumo

Existe uma atualização de norma no Brasil (o anexo 8 da NR-15, 2014) que deverá aumentar o interesse pelo monitoramento da exposição de trabalhadores à Vibração de Corpo Inteiro (VCI). Devido ao alto custo dos equipamentos utilizados neste tipo de monitoramento, surgiu a ideia de verificar a viabilidade da utilização do acelerômetro embarcado em um smartphone comum para fazer avaliações prévias desse tipo de exposição.

Este trabalho propõe verificar se existe a possibilidade do uso dos sistemas embarcados em dispositivos móveis para a mensuração preliminar de acelerações sofridas pelo corpo humano para o caso específico da VCI.

Para isso, foi criado um aplicativo, tendo como base a plataforma Android, que utiliza o acelerômetro do dispositivo móvel (*Smartphone*), que tem por objetivo realizar uma mensuração preliminar da VCI, a um custo inferior dos atuais equipamentos no mercado, tornando viável essa pesquisa preliminar com uma aproximação de resultado satisfatória. Lembrando que um dispositivo móvel não possui características para realizar uma medida apta, pois segundo normas da ISO 8041, os equipamentos de medição de vibração devem ser combinações entre transdutores, amplificadores e detector-indicador de sinal com características metrológicas controladas.

Palavras-chaves: *Smartphone. Android. Vibração de Corpo Inteiro. ISO 2631. NR – 15.*

Lista de ilustrações

Figura 1 Zonas de exposição à vibrações. 3
Figura 2 Fluxograma Android. 5
Figura 3 Sensor KXTJ9. 7
Figura 4 ASIC do sensor. 7
Figura 5 Accelerometer Frequency. 8
Figura 6 CTRL_REG1 KXTJ9. 9
Figura 7 DATA_CTRL_REG KXTJ9. 10
Figura 8 Arquitetura do sistema Android. 11
Figura 9 (a,b) Aplicativo (versão 0.1). 12
Figura 10 *Matlab Mobile*. 13
Figura 11 Frequência de amostragem Matlab Mobile. 13
Figura 12 Aplicativo (versão 1.1). 14
Figura 13 Diagrama de Sequência. 15
Figura 14 Explorador de pacotes. 16
Figura 15 activity_splash layout. 17
Figura 16 help_activity layout. 18
Figura 17 main_activity layout. 18
Figura 18 Método de envio de dados. 19
Figura 19 Diagrama de Classes. 20
Figura 20 Transdutor 352C33. 21
Figura 21 ENDEVCO Modal 102. 22
Figura 22 USB 1208FS. 23
Figura 23 Ruído de fundo KXTJ9. 24
Figura 24 Plataforma Vibratória. 25
Figura 25 FFT *6Hz.* 26
Figura 26 FFT *9Hz.* 27
Figura 27 Gráfico Smartphone X Transdutor. 28
Figura 28 FFT 15Hz. 29
Figura 29 FFT onda triangular. 30
Figura 30 FFT Aliasing. 30

Lista de Tabelas

Tabela 1 Tabela de frequências de amostragem Asus Zenfone 5. 8
Tabela 2 Taxa de dados de saída. 9
Tabela 3 Registrador KXTJ9. 9
Tabela 4 Taxa de amostragem e filtro passa-baixa. 10
Tabela 5 Característica transdutor 352C33. 22
Tabela 6 Ruído de fundo KXTJ9. 24
Tabela 7 Aceleração em RMS, frequência em torno de 6*Hz.* 26
Tabela 8 Aceleração em RMS, frequência em torno de 9*Hz.* 27
Tabela 9 Valore de aceleração RMS para onda senoidal 15*Hz.* 28
Tabela 10 Valores aceleração RMS para onda triangular. 29

Lista de Abreviaturas e Siglas

A/D	Analógico Digital
ACK	Acknowledge
ADT	Android Developer Tools
ASCII	Código Padrão Americano para o Intercâmbio de Informação
ASIC	Application Specific Integrated Circuits
CVS	Sistema de Versões Concorrentes
HTML	HyperText Markup Language
IDE	Integrated Development Environment
I^2C	Inter-Integrated Circuit
ISO	Organização Internacional para Padronização
NR	Norma regulamentadora
ODR	Output Data Rate
OSI	Open Systems Interconnection
RAM	Random Access Memory
RMS	Root mean square
SDK	Software Development Kit
SNR	Signal-to-noise
TCP/IP	Transmition Control Protocol / Internet Protocol
UML	Unified Modeling Language
USB	Universal Serial Bus
VCI	Vibração de Corpo Inteiro
VIC	Vibração intensa corporal
VMB	Vibração no segmento mão-braço
XML	eXtensible Markup Language

1 Introdução

O objetivo deste trabalho é realizar uma analogia preliminar entre acelerômetros comerciais e acelerômetros embarcados em dispositivos móveis , tentando demonstrar se existe a possibilidade do uso do mesmo para mensurar a exposição de trabalhadores à Vibração de Corpo Inteiro (VCI). (Junior e Mendes, 1994) apontam para doenças do sistema musculoesquelético como principal motivo para afastamento do serviço, segundo o autor, é compreensível que a Vibração de Corpo Inteiro afete os motoristas, pois os mesmos estão sujeitos as trepidações provenientes das imperfeições na via pública, vibração sofrida decorrente do próprio motor do veículo bem como da falta de estabilidade do mesmo.

É possível encontrar níveis elevados de vibração em diversos setores grandes da economia, como transporte rodoviário (carga e passageiros), agricultura, mineração, construção (pavimentação de estradas, por exemplo) e indústria (empilhadeiras, por exemplo). Em cada setor, o nível de vibração em uma situação específica depende de muitos fatores, além do tipo de atividade, como as condições da via de circulação, velocidade de deslocamento, modelo e estado de conservação do veículo ou equipamento e da forma de conduzir e características físicas do motorista/operador, entre outros. Assim, monitorar a exposição à VCI no Brasil pode exigir uma enorme quantidade medições. Como os equipamentos para realizar este tipo de monitoramento apresentam um custo muito elevado (da ordem de R$ 50.000,00) pode ser útil, em muitas situações, dispor de uma forma de realizar uma avaliação prévia da exposição à VCI de trabalhadores.

A questão é: Aparelhos móveis podem ser utilizado para mensurar de forma preliminar a exposição de um trabalhador à VCI? Quais as limitações desse tipo de dispositivo frente a transdutores utilizados especificamente para estas aplicações?

Para isso foi desenvolvido um aplicativo que garante uma leitura eficaz dos sinais emitidos pelo sensor do *Smartphone*, em posse dos resultados, será realizada uma comparação com os resultados do acelerômetro comercial.

O trabalho é dividido em 03 etapas: na primeira parte são definidas as características esperadas dos sinais de aceleração em condições usuais de exposição, com base nas definições da norma (ISO-2631-1, 1997). Neste capitulo também é abordado a revisão bibliográfica das ferramentas utilizadas na criação do referido aplicativo: Java, Android, Eclipse e Protocolo TCP-IP.

Na segunda etapa é demonstrada as funcionalidades do *Smartphone*: *hardware* e *software*, e abordado a primeira versão do aplicativo com suas características e suas limitações.

A terceira e última etapa apresenta a segunda versão do aplicativo criado para resolver os problemas da primeira versão, juntamente com testes de mensuração e comparação das vibrações junto a um transdutor.

2 Revisão Bibliográfica

2.1 Vibrações de Corpo Inteiro (VCI)

A exposição de trabalhadores à VCI, prevista como agente de risco físico para a saúde de trabalhadores na NR – 15 há mais de trinta anos por uma série de fatores (Tiago Becker, 2011).

As formas para quantificar a exposição à vibração estão baseadas no tempo de exposição com algumas medidas das amplitudes da aceleração do movimento. Este conceito vale para Vibração de Mão e Braço (VMB) quanto para VCI. Outro conceito importante, relacionado com a quantificação das amplitudes da vibração, é o de que, além do tempo de exposição e do nível de aceleração, as frequências presentes no movimento também são importantes.

2.2 Normas e exposição à VCI.

A ISO 2631 foi desenvolvida tendo em vista fatores que determinam a resposta do corpo humano às vibrações. Os limites propostos na ISO 2631 levam em conta tipos de exposição especifica, tais como vibrações transmitidas simultaneamente para toda superfície do corpo ou grande parte do mesmo, vibrações transmitidas para todo o corpo através de superfícies de sustentação e vibrações aplicadas em partes especificas do corpo. Para ambas as situações, a norma internacional define valores numéricos limítrofes de exposição do corpo humano às vibrações entre frequências 1 Hertz à 80 Hertz.

Os limites aplicados na norma se baseiam em vibrações aplicadas no ponto de entrada do corpo humano, ou seja, a mensuração da vibração tem que ser feita exatamente no ponto de contato, ou tão perto quanto possível, do corpo com a superfície pela qual a vibração é imposta ao mesmo (João Candido Fernandes, 1978).

Para avaliar a resposta musculoesquelética à vibração, é necessário o conhecimento de quatro fatores físicos: intensidade, frequência, direção e tempo de exposição à vibração.

Assumindo que as respostas estão relacionadas à energia, duas exposições diferentes de vibração diária são equivalentes quando (Alessandro L. Cargnelutti):

$$a_{RMS1} * \sqrt{T_1} = a_{RMS2} * \sqrt{T_2} \qquad (1)$$

a_{RMS1} e a_{RMS2} são os valores, em RMS, da aceleração em que o corpo está exposto no primeiro e segundo período, respectivamente.

T_1 e T_2 são relacionado ao primeiro e segundo período de exposição à vibração.

Outros estudos relatam que as respostas às vibrações diferentes são equivalentes quando (Alessandro L. Cargnelutti):

$$a_{RMS1} * \sqrt[4]{T_1} = a_{RMS2} * \sqrt[4]{T_2} \qquad (2)$$

Para ambos os casos, são traçados gráficos de limite de exposição à vibração (Figura 1).

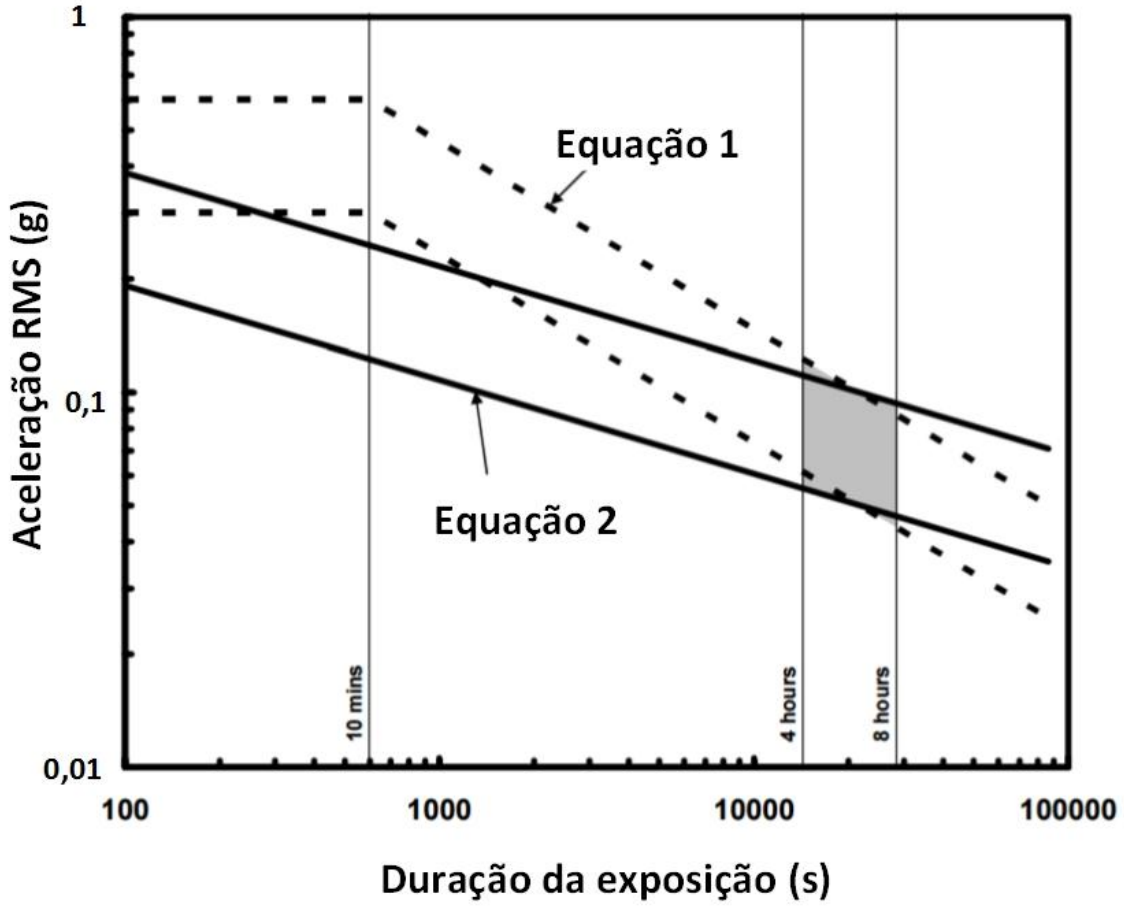


Figura 1 Zonas de exposição à vibrações.

Em exposições a vibrações abaixo da zona, os efeitos à saúde não estão bem claros ou documentados, exposições dentro da zona sugerem riscos para a saúde e exposições acima da zona são de extremo risco a saúde do indivíduo, podendo causar graves problemas para o corpo humano. Conforme a Figura 1, os valores de aceleração em RMS a serem explorados estão entre $0{,}0433g$ a $0{,}6g$, aplicando a Equação (3), o sensor deve possuir uma faixa de no mínimo $\pm 0{,}8485g$ pico-a-pico.

$$Aceleração_{RMS} = \frac{\sqrt{2}}{2} Aceleração_{pico-a-pico} \quad (3)$$

2.3 Android, plataforma e aplicação.

Um aplicativo para *Smartphone* pode ser desenvolvido em diversos sistemas operacionais, atualmente existem vários desenvolvedores de sistemas, por exemplo, Apple, Microsoft, LG, Samsung e assim por diante. Para o desenvolvimento deste trabalho em especial, o sistema operacional escolhido foi o Android, por concentrar a maior parte do mercado de sistemas operacionais móveis e principalmente por ser livre, também chamado de *open source* (João Bosco Monteiro, 2012).

Atualmente o Android é o sistema operacional para *Smartphones* mais utilizado do mundo, segundo *Dan Rowinki*, o mercado de *Smartphone* global, em 2013, chegou ao número de 1,2 bilhões de aparelhos vendidos.

Baseado no Linux, o sistema operacional Android teve seu desenvolvimento iniciado em 2003 pela empresa Android Inc. A sua estrutura de programas e classes é baseada em Java, e segue a organização de linguagens tradicionais como C e C++. Algumas linguagens permitem que funções e variáveis existam em diversos pontos de um programa, como se estivessem desatrelados de qualquer estrutura. Em Java, todas as variáveis e métodos devem estar localizados dentro das classes, forçando o uso de orientação a objeto até mesmo em tarefas simples, "Java é obrigatoriamente orientada a objetos" (Rafael Santos, 2001)

O Google, que hoje lidera o desenvolvimento sistema operacional Android, adquiriu os direitos sobre o mesmo em 2005. Em 2007 foi anunciada a *Open Handset Alliance* (Aliança de Telefonia Móvel Aberta) e a Plataforma Android, lançando a primeira versão *beta* do Kit de Desenvolvimento de Software (SDK). A Aliança integra softwares e outras propriedades intelectuais contribuídas pelas empresas participantes e disponibiliza-os para todos os desenvolvedores da plataforma, como foi explicitado em comunicado coletivo à imprensa:

"Esta Aliança partilha uma meta comum de inovação em dispositivos moveis e de fornecimento aos consumidores de uma experiência de usuário muito superior à de muitos produtos disponíveis em plataformas moveis da atualidade. Fornecendo aos desenvolvedores um novo nível de abertura que permite um trabalho mais colaborativo, o Android acelerará o ritmo em que novos serviços movem e competitivos serão disponibilizados aos consumidores. "

E assim surgiu a primeira plataforma de aplicação *open source* do mundo, que em outras palavras, os desenvolvedores têm total liberdade para desenvolver aplicativos, de forma independente, para comercialização ou não. A licença *Apache V2* (*http://www.apache.org/licenses/LICENSE-2.0.txt*) dá a garantia que os fabricantes de dispositivos possam usar o código Android, modifica-lo conforme seja necessário e então mantê-lo como código do proprietário ou libera-lo novamente para a comunidade *open source* (Ricardo R. Lecheta, 2012).

Um aplicativo Android possui, basicamente, quatro tipos básicos de componentes que são definidos em sua arquitetura. A Atividade (Activity), são trechos de código executáveis que vão e voltam no tempo. Uma grande limitação imposta pelo *Smartphone* é sua Memória de Acesso Aleatório, *Random Access Memory* (RAM), (muitas vezes de poucos gigabytes) e a bateria, então o Android foi projetado para preservar esses recursos, basicamente operando em "ciclo de vida", e isso define os estados ou eventos que uma Atividade atravessa desde o momento em que é criada até o término de sua execução. Quando não estiver em execução, uma atividade pode ser eliminada pelo sistema operacional para economizar memória. A maior parte do código executável reservado para as interações com o usuário são executadas no contexto de uma Atividade. Basicamente uma Atividade funciona segundo o fluxograma (Figura 2):

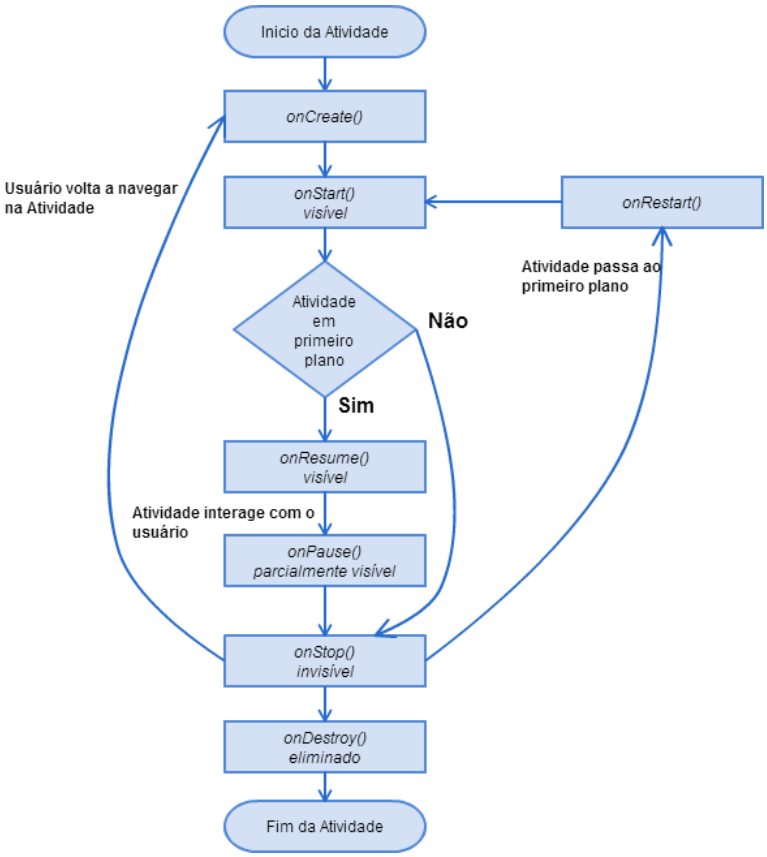


Figura 2 Fluxograma Android.

A Atividade corresponde à interface com o usuário e juntamente com ela operam os Serviços que são trechos executáveis de longa duração, geralmente são executados em segundo plano e não demandam iteração com o usuário. Provedores de Conteúdo são criados para compartilhar dados com outras Atividades ou Serviços, os dados de uma aplicação Android são de acesso exclusivo à aplicação e o único meio de compartilhar esses dados é através do Provedor de Conteúdo. Por último, os Receptores de Broadcast correspondem aos avisos de broadcast, ou seja, a um aviso global de eventos gerados pelo sistema operacional (ex. BATERY_LOW) ou em qualquer evento nas aplicações.

Assim como algumas linguagens de programação, o Android pode ser desenvolvido em um Ambiente de Desenvolvimento Integrado, ou *Integrated Development Environment* (IDE), para este caso, foi escolhido o Eclipse que é uma IDE de programação desenvolvida pela IBM®, e é administrado por um consórcio de nove empresas.

O Eclipse em si fornece apenas o ambiente integrado para a execução dos plug-ins e uns poucos plug-ins básicos, como editor de texto ASCII, sistema de ajuda e integração aos CVS (Edson Gonçalves, 2006). Seu *layout* apresenta uma pequena *runtime*, *workbench, workspace, help* e uma variedade de componentes.

Dentro do aplicativo será utilizado um protocolo de comunicação via *Socket*, que nada mais é que a maneira mais popular de utilizar as funcionalidades de comunicação TCP/IP. A arquitetura TCP/IP é largamente utilizada nas interconexões de sistemas operacionais, por ser um dos protocolos de comunicação mais antigos. Era a melhor opção frente aos protocolos de propriedade privada, e assim, acabou tornando-se um padrão no mercado (Carlos Alberto Goldani, 1997).

A arquitetura TCP/IP é organizada em camadas, não existindo uma estrutura formalmente evidenciada como a definida pelo modelo de Interconexão de Sistemas Abertos, ou Open Systems Interconnection (OSI). Por ser uma arquitetura que tem como objetivo a simplicidade e a funcionalidade, a arquitetura TCP/IP é composta essencialmente por dois protocolos principais. IP é responsável pelo transporte do pacote de dados através de diversas sub-redes desde a máquina de origem até a máquina de destino. TCP é um protocolo de transporte orientado à conexão, diferente do protocolo UDP que não utiliza mecanismos de reconhecimento para assegurar que as mensagens transmitidas cheguem ao seu destino, portanto, o conjunto TCP/IP oferece uma conexão de alta qualidade e confiabilidade.

A aplicação do processo TCP/IP é localizado através de protocolo de transporte, que é basicamente formado pelo endereço de IP da máquina dentro da rede onde ela está inserida (ethernet, internet...) associada a uma porta de serviço TCP, onde será feita a aplicação. Portanto, uma porta quando associada a um endereço de IP, constitui um "socket".

Existem vários níveis de demultiplexação no TCP/IP e a informação necessária para este procedimento está contida dentro de uma série de cabeçalhos (headers). Um *header* é um número extra de octetos colocado no início do datagrama por um protocolo para transferir informações de controle junto com os dados. Uma transmissão TCP/IP agrega vários *headers* (Carlos Alberto Goldani, 1997).

Para a transferência de dados, o protocolo TCP utiliza uma técnica denominada janela deslizante (sliding window) com alocação de crédito para exercer controle de fluxo sobre a taxa de dados trocados entre duas estações. Cada sinal acknowledge (*ACK*) especifica quantos créditos foram recebidos e quantos créditos o receptor está preparado para receber. Conforme o receptor vai guardando os dados dentro em um *buffer*, o emissor recebe a informação "*window*" para enviar uma janela de dados maior ou menor. Caso os dados cheguem fora de ordem para o receptor, os mesmos serão reorganizados dentro do próprio receptor, pois cada segmento está associado a um número dentro da sequência de envio, de modo a ordenar os segmentos recebidos fora de ordem e eliminar segmentos duplicados.

Para cada segmento de dado transmitido há a inserção do *checksum*, que por sua vez é verificado no receptor. Caso seja detectado algum erro, o segmento é descartado e o receptor não envia um sinal de *ACK*, e por estouro na espera, o transmissor se obriga a reenviar o mesmo segmento. O conjunto de endereço, porta, sequencia, *window*, *ACK* e *checksum* formam os cabeçalhos (headers) de transmissão.

3 Materiais e Métodos

O aparelho *Smartphone* utilizado nos testes é um *Asus Zenfone 5 A501* com processador Intel *Cover Trail Plus 1.6Ghz*, *Android 4.4.2* (KitKat) e *2 Gb* de memória RAM.

Esse aparelho é equipado com um sensor KXTJ9 (Figura 3) de três eixos fabricado pela Kionix®, a faixa de atuação é de $\pm 2g$, $\pm 4g$ ou $\pm 8g$. O sensor é fabricado utilizando um processo de miniaturização desenvolvido pela Kionix chamado de *microusinagem.*

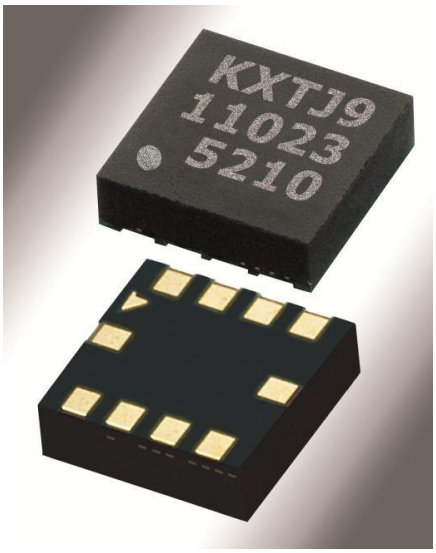


Figura 3 Sensor KXTJ9.

A mensuração de aceleração é baseada no princípio de uma capacitância diferencial resultante de uma aceleração diferencial entre os elementos do sensor, e utiliza ligação de modo comum para minimizar os erros de variação de processo, temperatura e desgaste. Um pacote de dispositivos *Application Specific Integrated Circuits* (ASIC) acoplados separadamente é utilizado para o condicionamento dos sinais e conversão A/D (Figura 4). Há o uso de reguladores de tensão para manter as tensões de operação interna constante, pois a taxa de amostragem do sensor é função da corrente de entrada (Tabela 2), isto resulta em características de operação estáveis mesmo que ocorram variações na tensão de entrada. O protocolo de comunicação I²C é utilizado para configurar o sensor e os dados de saída.

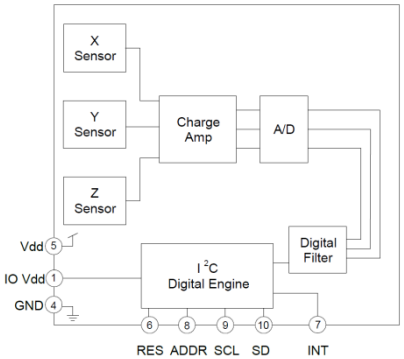


Figura 4 ASIC do sensor.

Através de um aplicativo chamado *Accelerometer Frequency* (Figura 5), que serve para adquirir informações sobre o acelerômetro embarcado no *Smartphone* e demonstrar eventos do mesmo, como, taxa de amostragem, resolução, modelo do sensor e corrente de alimentação. Conforme a informação, o sensor está configurado para rodar com uma faixa de $\pm 2g$ e resolução de $0{,}013629 m/s^2$ ($\sim 0{,}001390 g$), conforme Equação (4), é assumido que o sensor opera em $12\ bits$.

$$Resolução = \frac{Amplitude}{2^{número\ de\ bits}} \quad (4)$$

Figura 5 Accelerometer Frequency.

É possível perceber que na Figura 5 na opção *FREQUENCY*, está selecionado *SENSOR_DELAY_FASTEST*, na programação *Android*, *SENSOR_DELAY_FASTEST* é um inteiro de valor 0 (0x00000000) que é utilizado dentro do método *registerListener()*, no caso dessa configuração o sistema está registrando os dados do sensor o mais rápido possível. Foram testadas para outras taxas de amostragem, conforme Tabela 1.

Tabela 1 Tabela de frequências de amostragem Asus Zenfone 5.

	Mínima frequência de amostragem (Hz)	**Máxima frequência de amostragem (Hz)**
SENSOR_DELAY_UI	*5,894*	*14,893*
SENSOR_DELAY_NORMAL	21,564	26,235
SENSOR_DELAY_GAME	46,840	51,616
SENSOR_DELAY_FASTEST	46,877	51,696

A taxa de amostragem do sensor depende da corrente de suprimento, neste caso (Figura 5), a corrente é de $0{,}5700mA$ o que nos dá uma ODR (taxa de dados de saída) de 200 Hertz (Tabela 2). Ao passar pelo conversor A/D (Figura 4) e pelo Teorema de Nyquist essa taxa é limitada em 100 Hertz.

Tabela 2 Taxa de dados de saída.

KXTJ9-1007 Representative Current Profile		
ODR (Hz)	**RES**	**Current (µA)**
0	Disabled	0.9
0.781	0	1.7
1.563	0	2
3.125	0	2.2
6.25	0	3.3
12.5	0	5
25	0	9
50	0	16
100	0	29
200	0	57
400	0	120
800	0	120
1600	0	120
All Rates	1	120

A faixa de atuação do sensor é selecionado via protocolo I²C. Este protocolo foi criado pela *Philips* para atender periféricos em sistemas embarcados, por ser simples de utilizar, é bastante popular, possui comunicação em 8 *bits* e pode operar em até $5Mhz$, é usado para se comunicar com o microcontrolador e configurar suas saídas. Seguindo esta ideia, para alterar a faixa de atuação do sensor, deve-se alterar os valores dos *bits GSEL1* e *GESL0* conforme (Tabela 3), estes *bits* estão contidos dentro do registrador *CTRL_REG1* (Figura 6) através dos *bit4* e *bit3* respectivamente.

Tabela 3 Registrador KXTJ9.

Sensitivity (12-bit)[1]	GSEL1=0, GSEL0=0 (± 2g)
	GSEL1=0, GSEL0=1 (± 4g)
	GSEL1=1, GSEL0=0 (± 8g)

Quando é inicializado, neste caso, o sensor está configurado para atender uma faixa menor, mas com uma melhor resolução, conforme Equação **(4)**.

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
PC1	RES	DRDYE	GSEL1	GSEL0	0	WUFE	0	Reset Value
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	00000000
					I²C Address: 0x1Bh			

Figura 6 CTRL_REG1 KXTJ9.

Dentro do circuito integrado do sensor (Figura 4) existe um filtro passa-baixa,cujo objetivo de limitar a frequência do sinal de forma a evitar erros de *Aliasing.* Este filtro é controlado via registrador DATA_CTRL_REG (Figura 7), através dos bits 0 à 3. Em sua configuração inicial, o filtro possui uma banda de passagem de até 50 Hertz (Tabela 4).

Como visto na Figura 5, e agora demonstrado pela Tabela 4, a taxa de amostragem máxima possível para este aparelho é de 50 Hertz, sendo assim, para que não exista perda de informação, frequência máxima para aquisição de dados é de 25 Hertz. Alguns aparelhos possuem acelerômetros que operam a frequências maiores que 100 Hertz, portanto, atendendo um espectro maior de frequência.

R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
0	0	0	0	OSAA	OSAB	OSAC	OSAD	Reset Value
Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0	00000010
					I^2C Address: 0x21h			

Figura 7 DATA_CTRL_REG KXTJ9.

Tabela 4 Taxa de amostragem e filtro passa-baixa.

OSAA	OSAB	OSAC	OSAD	Output Data Rate	LPF Roll-Off
1	0	0	0	0.781Hz	0.3905Hz
1	0	0	1	1.563Hz	0.781Hz
1	0	1	0	3.125Hz	1.563Hz
1	0	1	1	6.25Hz	3.125Hz
0	0	0	0	12.5Hz	6.25Hz
0	0	0	1	25Hz	12.5Hz
0	0	1	0	50Hz	25Hz
0	0	1	1	100Hz	50Hz
0	1	0	0	200Hz	100Hz
0	1	0	1	400Hz	200Hz
0	1	1	0	800Hz	400Hz
0	1	1	1	1600Hz	800Hz

Estes valores são alterados via kernel. O kernel é o núcleo do sistema operacional, mais precisamente, é o nível zero da arquitetura de programação (Figura 8) em torno do qual todos os programas são escritos. Embora o kernel do Android seja baseado na versão 3.3 do kernel Linux, ele diferencia de outros tipos de kernel por utilizar códigos específicos do Android.

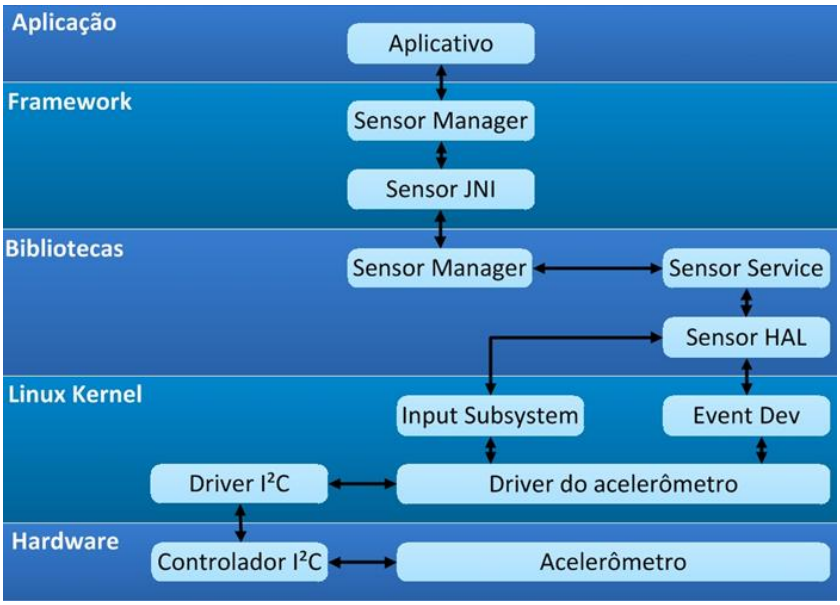


Figura 8 Arquitetura do sistema Android.

3.1 O Aplicativo (versão 0.1)

Heittor TCC é a primeira versão do aplicativo criado neste trabalho que permite executar a leitura do sensor acelerômetro para o sistema operacional Android. Com ele é possível realizar a leitura dos sinais do acelerômetro com uma taxa de amostragem de até 50 amostras por segundo, como foi visto anteriormente.

3.1.1 Requisitos

Esta versão do aplicativo foi desenvolvida em cima de uma biblioteca de gráfico *AChartEngine* (www.achartengine.org), que é compatível com Android *SDK 1.6* ou superior, ou seja, de grande abrangência dentro do mercado de *Smartphone*. A escolha desta biblioteca se fez pelas características gráficas, um *layout* de fácil acesso e compreensão.

3.1.2 Layout e funcionamento

A primeira inicialização do programa exige que o usuário acione o botão "*Start*" na tela inicial (Figura 9 a). Esse comando inicializará a leitura do *status* do acelerômetro, toda vez que os valores do sensor mudam, a função *onSensorChanged()* é chamada, a tela permanecerá em branco até que o usuário acione o botão "*Show*".

Depois disso, o programa exibe sua tela principal os gráficos referentes às leituras realizadas pelo acelerômetro no intervalo de tempo entre o acionamento do botão "*Start*" e o botão "*Show*" (Figura 9 (a,b)), sendo um sensor com três eixos as leituras são referentes a cada eixo do mesmo (*x*, *y* e *z*).

E por fim o usuário pode exportar esse gráfico em forma de matriz acionando o botão "*Convert to .txt*", esse comando irá gerar um arquivo *.txt* dentro da pasta "*Câmera*" do Smartphone.

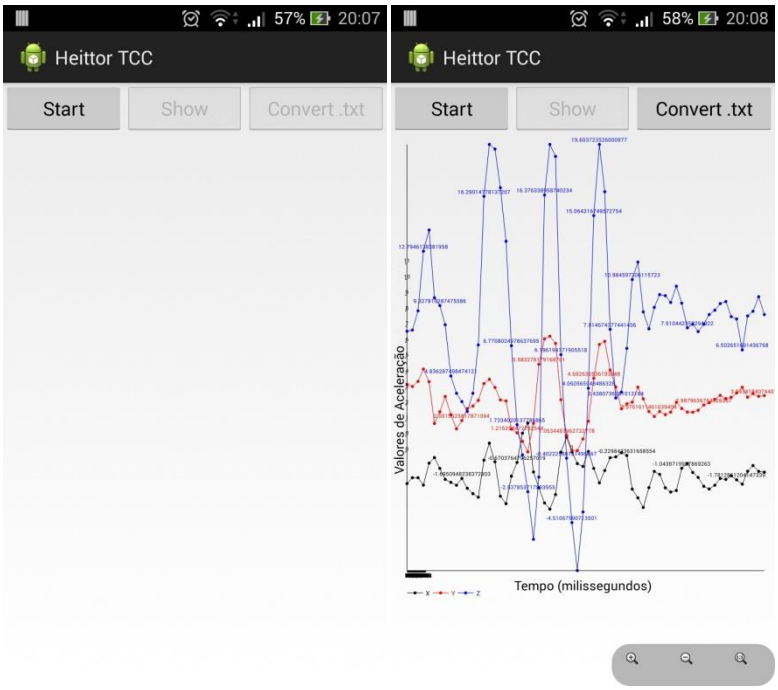


Figura 9 (a,b) Aplicativo (versão 0.1).

Um dos problemas encontrados para esse *layout* de aplicação, é que as informações correntes do sensor ficam disponibilizadas somente dentro da memória interna do aparelho.

Para salvar dados do acelerômetro na própria memória interna, é necessário que o aparelho guarde 4 tipos de dados, qual correspondem as acelerações discretas dos eixos x, y e z e também o tempo ao qual essas acelerações foram amostradas. No o caso, são necessários três vaiáveis do tipo *float* e um *long*, dentro da programação, um *float* possui 32 bits, e um *long* possui 64 bits, sendo que 1 byte é constituído por 8 bits, um *buffer* de dados (*float* + *float* + *float* + *long*) possui 20 bytes. A taxa máximia de amostragem do sensor é de 50 amostras por segundo, o que equivale a 1000 bytes por segundo, supondo que seja necessário realizar uma medida durante 1 hora, ao final teremos que alocar uma memória de 3,6 Mbytes para os dados provenientes do acelerômetro.

Levando em conta o tamanho da memória interna dos dispositivos atuais, salvar o *buffer* não seria um fator limitante, mas para uma maior agilidade no tratamento dos dados, foi escolhido enviá-los diretamente para um dispositivo com maior capacidade de processamento.

3.2 O Aplicativo (versão 1.1)

Conforme abordado anteriormente, optou-se por criar uma nova versão do aplicativo, com o objetivo de utilizar a menor quantidade de memória e processamento possível do *Smartphone* e realocar a parte de tratamento de dados exclusivamente para um *software* de linguagem e computação técnica.

Portanto, este novo projeto é baseado em comunicação via protocolo TCP/IP. Basicamente, o programa envia dados do dispositivo *Android* para um servidor criado dentro do *Matlab* através de um arquivo *ACELEROMETRO_TCP.m,* criado para receber e tratar os dados enviados pelo aplicativo.

A própria *MathWorks*, desenvolvedora de software de computação matemática, disponibiliza via *Google play* o *MATLAB Mobile™* (Figura 10), este aplicativo é capaz de se conectar remotamente à um programa *Matlab* via protocolo TCP/IP. A partir de um dispositivo *Android* é possível executar *script*, analisar resultados e transferir dados de sensores magnéticos, GPS e acelerômetros.

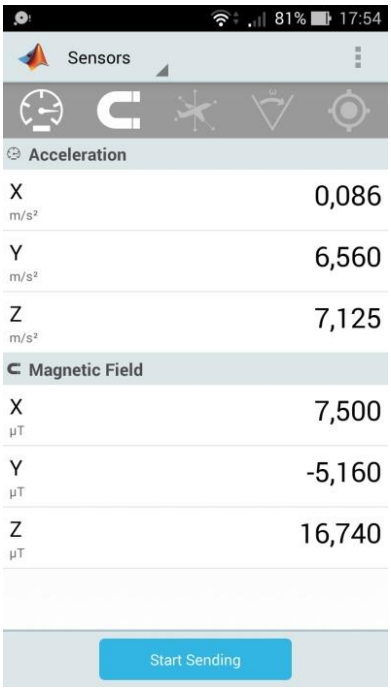


Figura 10 *Matlab Mobile*.

O problema em utilizar este programa é que não é possível alterar a frequência de amostragem do sensor, por exemplo, dentro do método *registerListener()* é permitido escolher a taxa de amostragem do sensor, através do SENSOR_DELAY, ao todo são 4 opções: SENSOR_DELAY_UI, SENSOR_DELAY_NORMAL, SENSOR_DELAY_GAME e SENSOR_DELAY_FASTEST. Através de testes foi constatado que o programa opera com uma taxa de amostragem padrão, possivelmente SENSOR_DELAY_UI, pois foram realizados testes com o *MATLAB Mobile™* e constatou-se uma frequência de amostragem por volta de 13 Hertz (Figura 11).

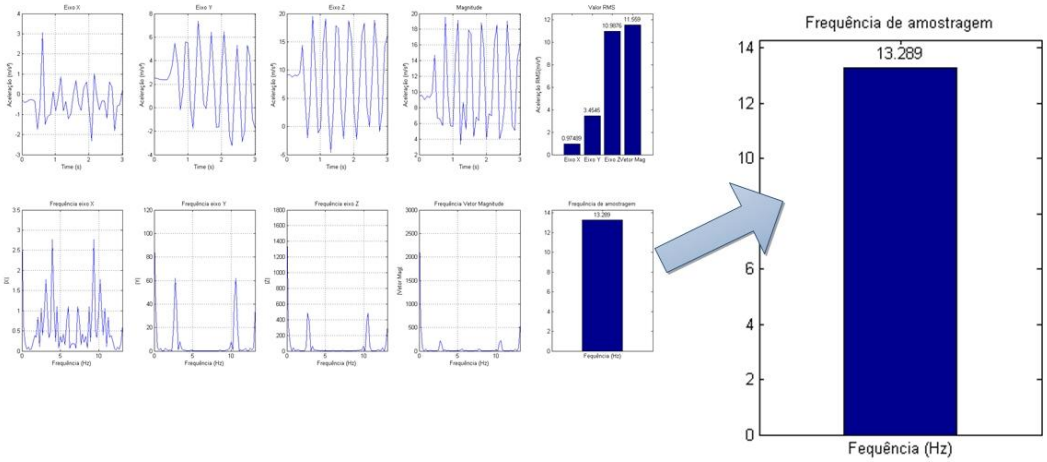


Figura 11 Frequência de amostragem Matlab Mobile.

Conhecendo limitação do aplicativo criado pela *MathWorks*, pois o mesmo pode ser utilizado em processos com, no máximo, 6,5 Hertz (no caso do Asus Zenfone 5). Isso sugere também que, para qualquer outro aparelho, o aplicativo não exige um desempenho máximo do *hardware*.

3.2.1 Requisitos

O protótipo da aplicação foi desenvolvido para funcionar em *Smartphones* com plataforma *Android* junto com um computador que possua o programa *Matlab* devidamente instalado. Este computador será utilizado para receber os dados enviados via protocolo TCP/IP.

Ambos os aparelhos (*Smartphone* e computador) devem estar conectados na mesma rede, esta conexão é necessária, pois a partir dela vamos ter o registro de IP alocado para o computador e os dados serão transmitidos dentro desta mesma rede. Caso não haja uma rede implementada, o próprio aparelho é capaz de criar e gerenciar uma rede Wi-Fi.

3.2.2 Layout e funcionamento

O protótipo do aplicativo é composto por três telas distintas (Figura 12), cada uma delas representa um componente *Activity*, a primeira tela é exibida somente por um tempo determinado, também conhecida como *Splash Screen*, esse tipo de método é utilizado como tela de *launcher* ou tela de inicialização. A segunda tela é basicamente uma tela de *First Step* ou *Help Screen*, é exatamente a tradução destes termos, nela estão presentes informações sobre como utilizar o aplicativo junto com suportes para que o usuário não tenha dúvidas e possa fazer um uso correto. A terceira tela é a tela principal da aplicação, é normalmente chamada de *MainActivity* ou atividade principal, é onde o usuário irá permanece a maior parte do tempo e onde serão realizadas as principais funções do aplicativo. Todos estes componentes são declarados dentro do arquivo manifesto (Anexo 1).

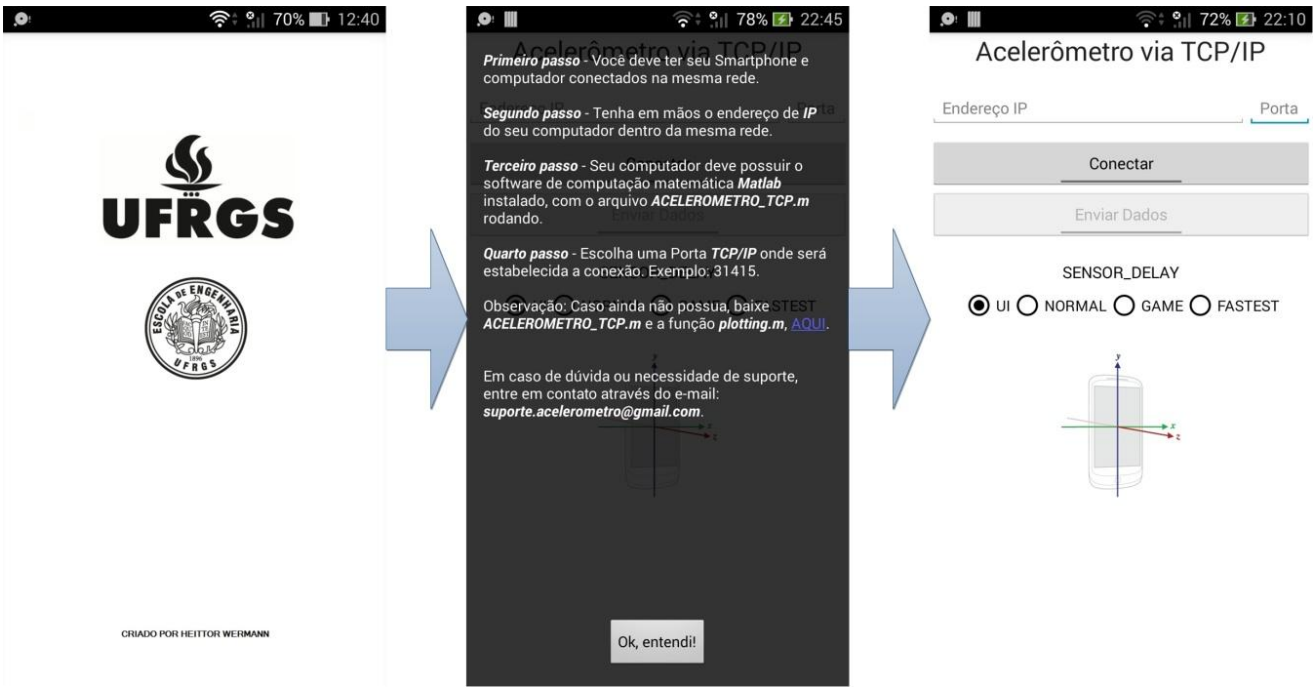


Figura 12 Aplicativo (versão 1.1).

Para uma melhor compreensão do funcionamento e sequenciamento do aplicativo é demonstrado o diagrama de sequência. O Diagrama de Sequência é um diagrama usado em UML (Unified Modeling Languange), que basicamente permite que desenvolvedores visualizem os produtos de seus trabalhos em diagrama padronizado. O diagrama tem o objetivo de demonstrar como são as iterações entre os objetos dentro programa, como são sequenciadas as mensagens com ênfase na perspectiva temporal.

No diagrama (Figura 13) é possível verificar a sequência de interações encontradas na aplicação *Acelerômetro TCP*.

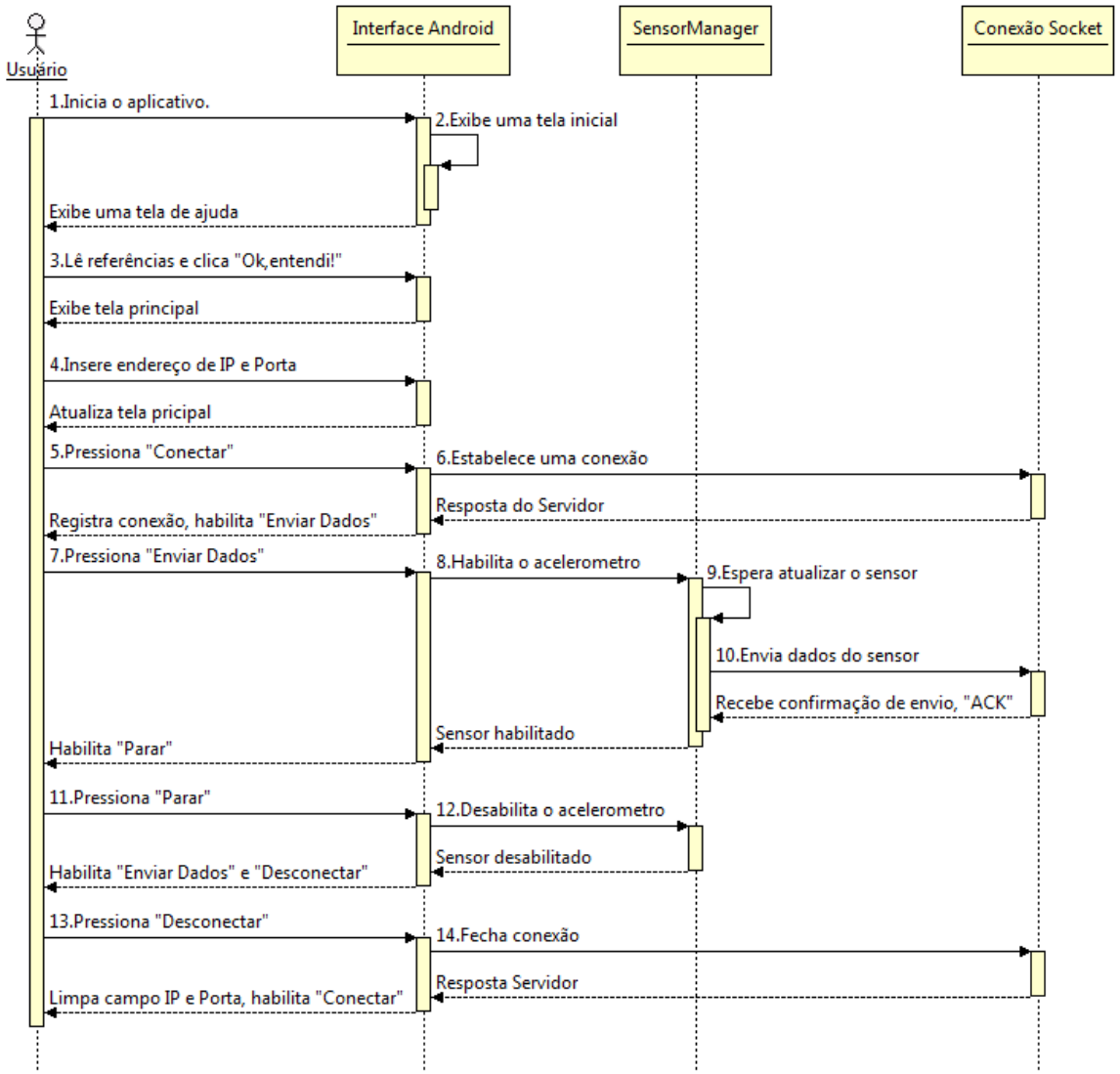


Figura 13 Diagrama de Sequência.

3.2.3 Implementação

Para o desenvolvimento foi utilizada a IDE *Eclipse Luna* juntamente com um *plug-in* ADT. Apesar da plataforma *Android* possibilitar a codificação de interfaces de forma procedural, permitindo codificar os elementos diretamente nas classes, é recomendado a declaração de interfaces de forma declarativa, utilizando arquivos de *layout* (Figura 14) do tipo XML, pois além de tornar o código de fácil visualização, facilita manutenções futuras do aplicativo (Júlio Cesar Gonçalves, 2011).

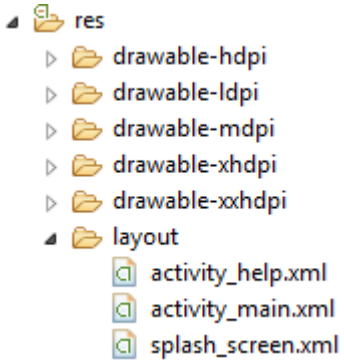


Figura 14 Explorador de pacotes.

O aplicativo possui quatro classes distintas *MainActivity*, *SplashActivity*, *HelpActivity* e *CommsThread*. Cada classe está associada a um determinado arquivo descritor XML que define os componentes a serem exibidos na tela, cada componente é identificado por um *id* especifico que é vinculado no código da classe. O vínculo pode ser demonstrado pelos anexos Anexo 2, Anexo 3 e Anexo 4 dentro da função *onCreate()* através do método *setContentView()* sendo os arquivos *activity_splash.xml*, *activity_help.xml* e *activity_main.xml* "amarrados" dentro de suas respectivas atividades.

3.2.4 Manifesto

O arquivo manifesto (Anexo 1) apresenta informações essenciais sobre o aplicativo para o sistema *Android*, no caso, as três *Activity* (tela inicial, tela de ajuda e tela principal) estão declaradas dentro do manifesto como sendo: *SplashActivity*, *HelpActivity* e *MainActivity* respectivamente. Dentro dele também estão informações sobre permissões que o aplicativo deve ter para acessar áreas protegidas da API, no caso, o aplicativo necessita de permissões para acessar a internet, status da rede local, status da rede sem fio e ter acesso para habilitar ou não a rede sem fio.

3.2.5 SplashActivity

Esta atividade é uma atividade temporal, ou seja, ela é executada por um curto espaço de tempo, é essencialmente uma tela informativa, o usuário não possui nenhum controle sobre seu funcionamento.

Como visto anteriormente, esta atividade é declarada como um *launcher*, sendo assim, a atividade é executada antes da atividade principal (*MainActivity*), o tempo de execução é declarado dentro método *postDelayed()* (Anexo 2) que nada mais é que uma função de espera, que mantém o programa contando até 3000 milissegundos, durante esse tempo o usuário visualiza mostrada na Figura 15, depois o método *startActivity()* encaminha para a atividade principal e finaliza a atividade corrente.

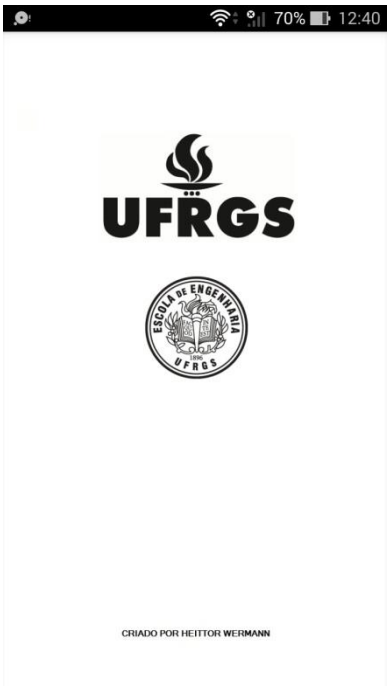


Figura 15 activity_splash layout.

3.2.6 HelpActivity

Se for exemplificar as atividades conforme elas fossem chamadas ao decorrer do tempo, a *HelpActivity* (Anexo 3) estaria em terceiro lugar, mas para o usuário que utiliza a aplicação, visualiza-a logo após a *SplashActivity*, isso acontece pois essa atividade é chamada dentro da *MainActivity* e assim como a atividade anterior, ela possui caráter informativo.

Dentro da atividade são expressas informações sobre o "passo a passo" de como utilizar a aplicação, assim como, informações sobre arquivos adjacentes e contatos para suporte. Para criar um texto mais "suave" e de melhor visualização, é utilizado uma biblioteca *android.text.Html*, esta biblioteca tem o objetivo de formatar textos utilizando funções HTML, para isso, além de "amarrar" o *layout activity_help.xml* (Figura 16), foi necessário criar um arquivo para formatar o texto (*help_string.xml*), neste arquivo o texto visualizado pelo *layout* é devidamente formatado com as funções HTML.

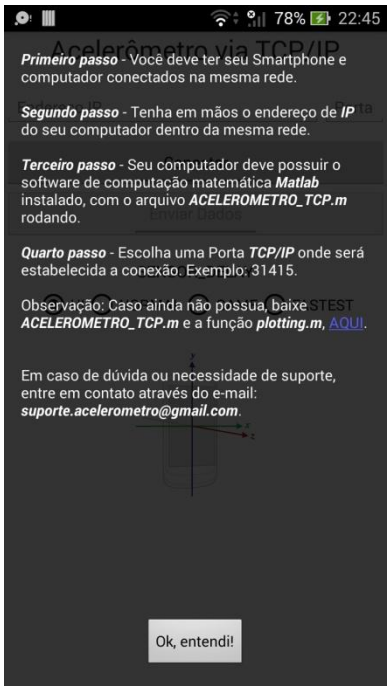


Figura 16 help_activity layout.

3.2.7 MainActivity

Finalmente a atividade principal, a *MainActivity* é responsável pelas principais funções da aplicação, é ela quem "chama" a *HelpActivity*, que realiza a conexão via protocolo TCP/IP e gerencia o acelerômetro do aparelho. Seu *layout* é o *activity_main.xml* (Figura 17), nele é possível visualizar campos de texto (*EditText*), botões (*ToggleButton*) e escopo de múltipla exclusão (*RadioGroup*).

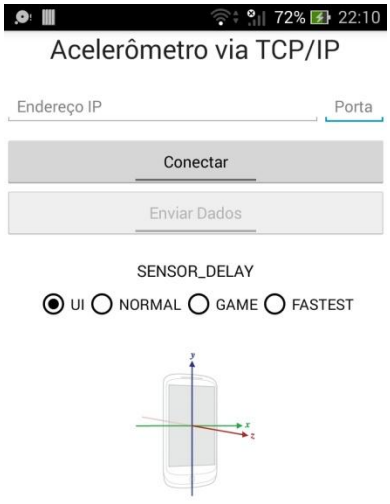


Figura 17 main_activity layout.

Assim como em todas as atividade, os periféricos de controle são declarados dentro da função *onCreate()* (Anexo 4) através do método *findViewById()* que tem por objetivo "amarrar", neste caso, os botões, textos e opções de escolha às variáveis que serão utilizadas dentro da atividade, para isso utiliza uma classe *R.java* que é responsável por fazer essa ligação entre as atividades e seus respectivos *layout*. (Observação: *R.java* não deve ser editada pelo programador, a própria IDE realiza essa edição)

É possível perceber que o sensor tem sua configuração inicial programada dentro da mesma função, além de ser inicialmente implementado dentro da classe ele é administrado pela biblioteca *android.hardware.SensorManager* e configurado pelo método *getDefaultSensor()*, no caso, o sensor escolhido é do tipo acelerômetro.

A conexão via protocolo TCP/IP é gerenciada pela biblioteca *java.net.Socket*, a conexão inicial é realizada pelo construtor *Socket()* (Anexo 5), este construtor é responsável por criar uma conexão TCP do lado do cliente, portanto suas entradas são *editTextIp* que nada mais é que o endereço de *IP* fornecido no campo do texto e pela *port* que representa um inteiro referente a porta em que se habilitara a comunicação.

A partir do momento que a conexão for estabelecida o aplicativo torna o botão "*Enviar Dados"* visível o que possibilita que o usuário inicie o envio dos dados mensurados pelo acelerômetro. Após acionado, o programa registra qual é o *SENSOR_DELAY* escolhido pelo usuário e habilita através do método *registerListener()* o início do funcionamento do sensor assim como sua taxa de amostragem, como visto no diagrama de sequência (Figura 13), toda vez que os valores do sensor muda, a função *onSensorChanged()* (Anexo 6) é chamada e dentro dela os valores do sensor são atribuídos a variáveis flutuantes (*x, y* e *z*), após executar esta função, o programa é direcionado para a função *onResume()*.

Dentro de *onResume()* (Anexo 7) o programa passa por uma condição através da variável booleana *start*, esta variável utilizada é alterada pelos botões de envio de dados, portanto se o botão "*Enviar Dados"* estiver pressionado a condição é satisfeita e assim segue para uma classe *myCommsThread* localizada dentro da própria *MainActivity*.

A classe *myCommsThread* é utilizada para enviar os dados atribuídos pelo sensor às variáveis *x*, *y* e *z* junto com uma variável *currentTime* responsável por contar o tempo entre o primeiro envio de dados e o próximo envio. As variáveis *x*, *y*, *z* e *currentTime* enviadas dentro do mesmo buffer (Figura 18), são convertidas para ASCII e enviadas junto com o caractere *#*, que no programa executado pelo servidor tem o objetivo de diferenciar cada variável, sabendo que o código ASCII possui oito bits para cada caractere, o buffer possui no mínimo *168 bits*, sem contar que a variável *currentTime* aumenta seu tamanho ao decorrer do tempo.

```
PrintWriter out = new PrintWriter(new BufferedWriter(new OutputStreamWriter(client.getOutputStream())),true);
out.printf("%.0f# %3.2f# %3.2f# %3.2f#",currentTime,x,y,z);
```

Figura 18 Método de envio de dados.

Sendo a capacidade teórica padrão Wi-Fi (IEEE 802.11a), de transmitir até 2Mbytes de dados por segundo, o mesmo sensor poderia operar a uma frequência de 95 MHz que ainda seria possível o envio de todos os dados.

3.2.8 Diagrama de classes

O diagrama de classes (Figura 19) descreve informações de estruturas utilizadas na aplicação, apresenta as classes implementadas, seus atributos, métodos e relação entre as classes. Diferente do diagrama de sequencias, a ênfase deste diagrama é nos métodos utilizados e não possui perspectiva temporal.

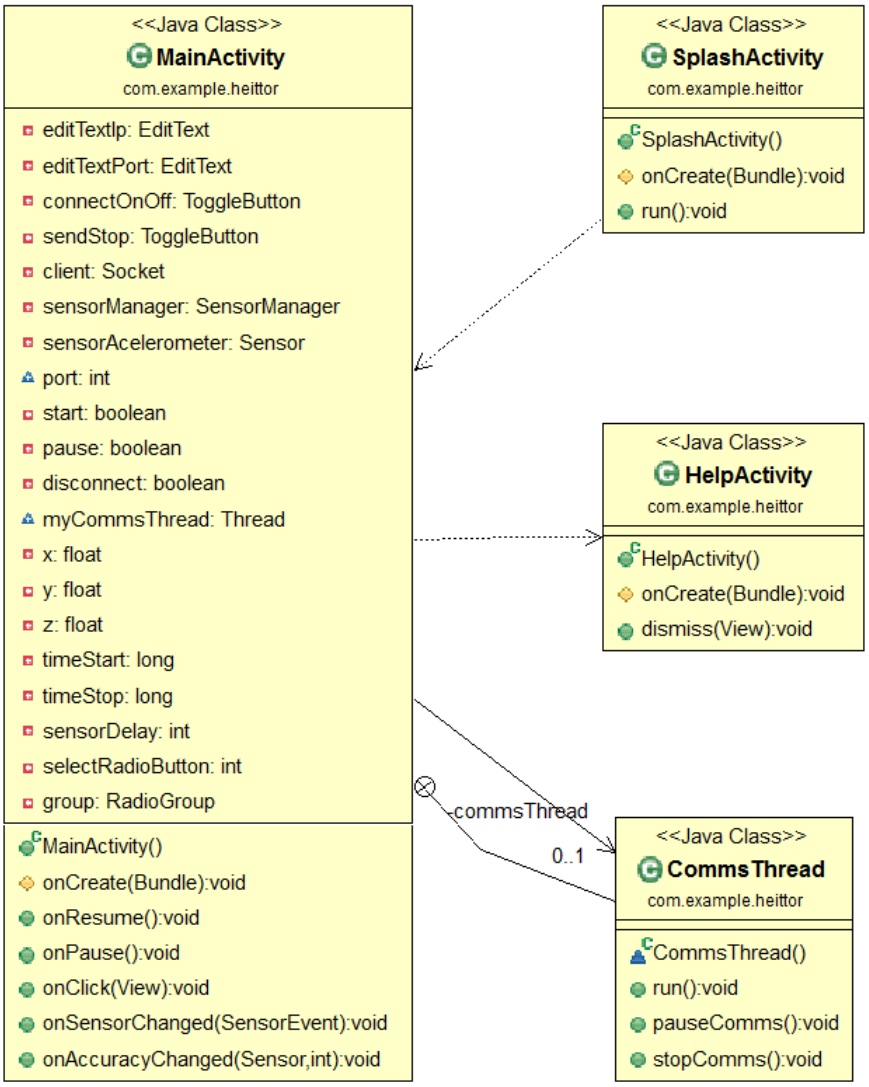


Figura 19 Diagrama de Classes.

3.3 O servidor

O servidor é criado utilizando bibliotecas Java do Matlab, através das bibliotecas java.net.* e java.io.* é possível criar um servidor utilizando o método ServerSocket, este método tem o objetivo de criar um servidor via socket. É necessário que o usuário escolha uma porta livre em sua conexão e a partir desta escolha é criado um servidor TCP/IP anexos Anexo 8 e Anexo 9. Então o programa fica ouvindo a rede, à espera de uma conexão. Quando a conexão for estabelecida, o programa entra em um laço While que tem por objetivo manter o servidor ouvindo qualquer dado enviado pelo cliente, dentro

deste laço é criado um laço "*For*" capaz de ler todo o *buffer* enviado e tratar cada *byte.* Como foi abordado, o aplicativo envia cada atualização de estado do acelerômetro seguido de um caractere "#" esse caractere tem o objetivo de diferenciar cada sinal dentro do servidor. Cada sinal é guardado em um *buffer* especifico, *dadoAcelX()*, *dadoAcelY()*, *dadoAcelZ()* e *dadoTempo()* guardam sinais dos eixos x, y, z (referentes do acelerômetro do dispositivo móvel) e o tempo (*currentTime*) em que cada dado foi adquirido, respectivamente.

Dentro do próprio arquivo existe uma função *plotting* que é responsável por tratar cada *buffer* de dados e criar gráficos a partir deles.

3.4 Instrumentação e calibração

Conforme a ISO-2631, o equipamento de medida de vibrações deve, geralmente, consistir de um transdutor, um dispositivo amplificador, ou que realize algum tratamento do sinal enviado pelo transdutor, e um indicador de nível ou registrador. Se for conveniente pode ser incluído circuitos para limitar a amplitude de frequência do equipamento e aplicar a avaliação de frequência recomendada ao sinal de absorção. Não é necessário contar com medições imediatas, portanto, o equipamento pode gravar os sinais transmitidos pelo transdutor e assim obter registros representativos para análise subsequente.

Os equipamentos disponibilizados são de propriedade do departamento de Engenharia Mecânica da Universidade Federal do Rio Grande do Sul

3.4.1 Transdutor

Um transdutor normalmente utilizado na captação de uma vibração é o acelerômetro piezoeléctrico, que possui boa linearidade e uma banda dinâmica maior em comparação a outros acelerômetros. Não necessitam de fonte de alimentação, não possuem partes móveis e geram um sinal proporcional à aceleração. (PCB Piezotrinics)

O transdutor utilizado é o modelo 352C33 (Figura 20) fabricado pela PCB Piezotronics, possui um grau de liberdade e oferece uma enorme versatilidade para medições de choque e vibrações. Suas características podem ser visualizadas na Tabela 5.

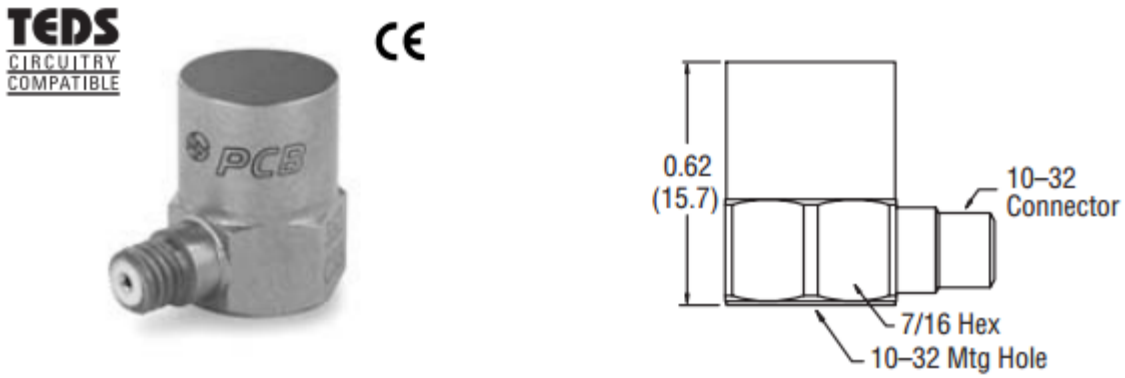


Figura 20 Transdutor 352C33.

Tabela 5 Característica transdutor 352C33.

Sensibilidade (mV/g)	**Frequência** (Hz)	**Amplitude** (pico-a-pico)	**Resolução** (*RMS*)	**Temperatura** (^oC)	**Peso** (*Gramas*)
100	0.3 à 15k	$\pm$50g	0.00015g	-35 à $+93$	5.8

3.4.2 Condicionador de sinal

Para o condicionamento do sinal emitido pelo transdutor, foi utilizado um condicionador ENDEVCO Modal 102 Signal Conditioner (Figura 21). É um condicionador de sinal de dois canais projetado para o uso de conversores de carga remotos (transdutores piezoeléctricos). O condicionador proporciona uma tensão de saída alternada e a tensão de entrada é proporcional a aceleração do transdutor conectado. Este modelo possui um filtro que permite passa alta, passa-baixa e passa-banda.

Figura 21 ENDEVCO Modal 102.

3.4.3 Placa de aquisição de dados

A placa USB 1208FS (Figura 22) é um dispositivo de aquisição de dados analógico e saída digital, possui oito canais de entrada analógica podendo ser selecionado para oito saídas de 11 bits de entrada simples ou quatro entradas diferenciais com saídas de 12 bits. É conectada via USB e possui uma alimentação de 5 volts do próprio conector USB.

Figura 22 USB 1208FS.

3.4.4 Software para o tratamento de dados

A plataforma VEE, da Agilent, consiste em uma linguagem gráfica de programação voltada para testes e medições com recursos avançados de tratamento e análise de sinais, incluindo suporte a algoritmos e funções desenvolvidas em MATLAB

O software VEE da Agilent é um ambiente visual de engenharia que permite ao programador criar "diagramas de blocos" intuitivos. Objetos selecionados de menus são editados e conectados a outros de forma a especificar a sequência de tarefas que se deseja executar. O usuário pode criar, testar e atualizar códigos, já que o programa permite a execução e a manutenção durante a operação através dos botões *Start, Stop, e Pause*, além das funções *Watch Window e Output Window* (Anexo 10).

O software foi utilizado para o tratamento do sinal da placa de aquisição de dados, a partir dos dados de entradas foram construídas funções de filtro passa baixa, além de funções para cálculo das transformadas de Fourier e RMS.

4 Resultados.

Por ser um dispositivo de certa forma, lacrado, e com mecanismos de auto compensação o Smartphone não pode ser calibrado da mesma forma que tansdutor. Um meio de compensar a resposta do mesmo é realizar a análise do ruído de fundo.

O ruído eletrônico é gerado pelo circuito amplificado, é definido como sendo de banda larga ou de espectro (indicado em frequência especifica). Os níveis de ruído foram representado em *g* (Tabela 6). Normalmente diminui à medida que a frequência aumenta. Isso significa que o ruído em baixa frequência é mais problemático.

Tabela 6 Ruído de fundo KXTJ9.

	Mínimo (g)	Máximo (g)	RMS (g)	Incerteza tipo A (g)
Eixo X	-0,0082	0,0071	0,00256	0,0000269
Eixo Y	-0,0041	0,0041	0,00164	0,0000176
Eixo Z	-0,0082	0,0112	0,00330	0,0000351

Para medir o ruído, foi necessário isolar o aparelho telefônico em um quarto fechado, de modo a garantir a melhor isolação possível, o mesmo foi envolto por camadas de isolantes sintéticos (fibras de vidro e nylon), os testes foram realizados no período da madrugada (4:00 horas, horário de Brasília), o tempo de amostragem do teste foi de 250 segundos (Figura 23).

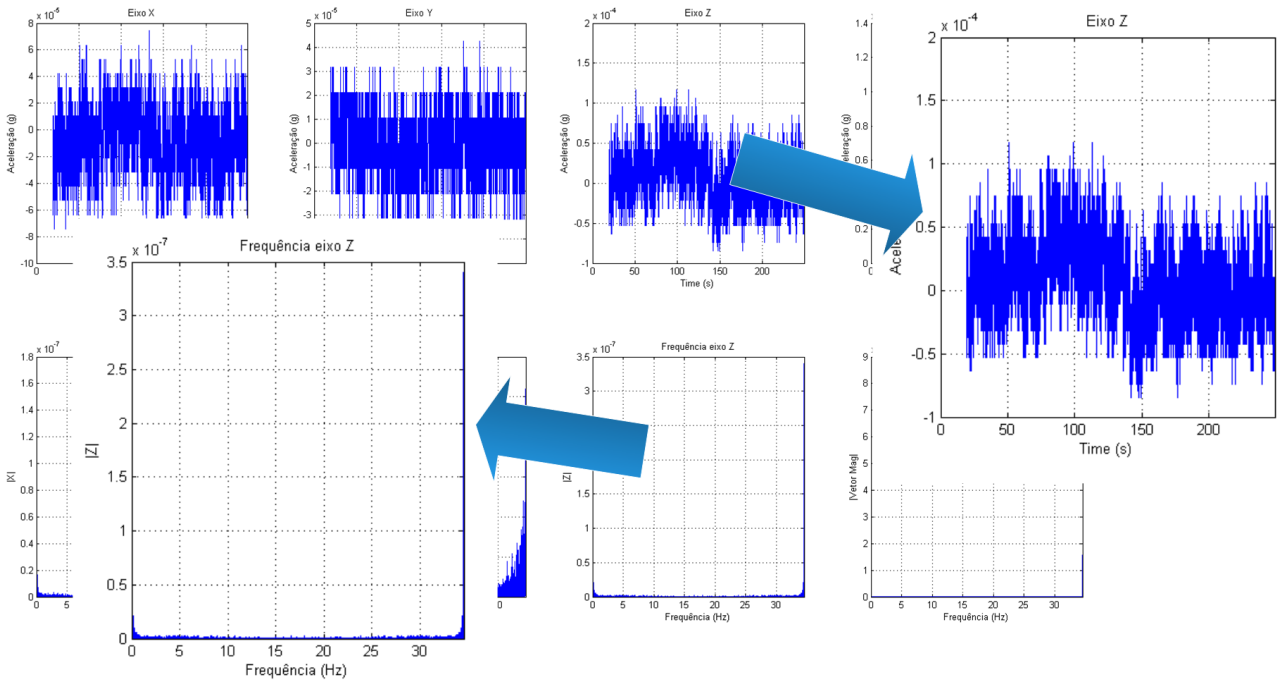


Figura 23 Ruído de fundo KXTJ9.

De posse destas informações é possível prosseguir com a comparação entre os dispositivos. Os testes foram realizados em uma mesa vibratória (Figura 24), nela é possível gerar vários tipos de onda (senoidal, quadrada, triangular...).

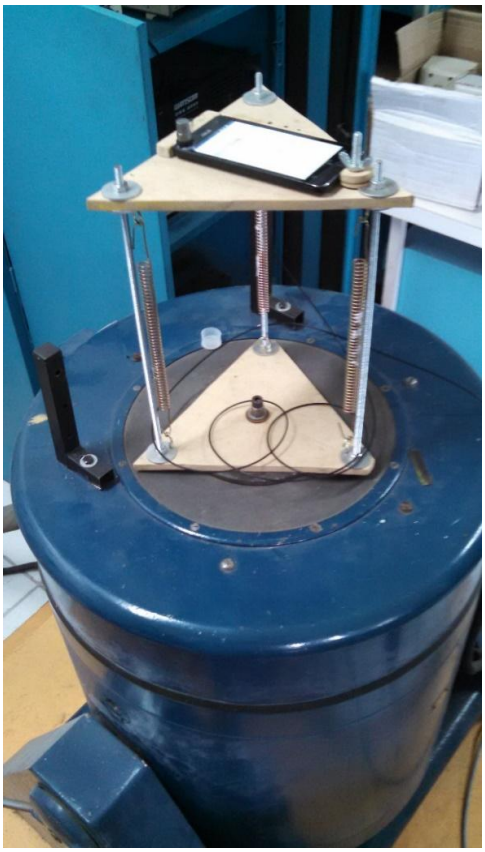


Figura 24 Plataforma Vibratória.

Por ser uma plataforma magnética e aparelho *Smartphone* ser sensível a determinados níveis de magnetismo, foi necessária a confecção de uma plataforma com o objetivo de afastar o aparelho do núcleo magnético da plataforma (Figura 24), existe uma espécie de extensão de 3 pontos, feita de madeira e apoios de metal, no topo desta plataforma é colocado o aparelho juntamente com o transdutor. Para a aquisição dos dados, o aparelho celular é fixado na plataforma e o acelerômetro piezoeléctrico é colado na parte superior do *Smartphone*.

Foram realizados inicialmente cinco testes distintos, os dois primeiros testes têm o objetivo de analisar a tendência de repetitividade para duas frequências distintas, o terceiro realiza uma comparação entre o transdutor comercial e o acelerômetro embarcado para uma onda senoidal definida, mas alterando a amplitude e o quarto teste faz essa mesma comparação, mas com uma onda triangular.

Em nenhum teste a amplitude máxima da vibração ultrapassou $1g$, lembrando que para a mensuração efetiva da VCI as máximas amplitudes não ultrapassam $\pm 0{,}8485g$ e o próprio sensor do celular, que tem uma faixa de $\pm 2g$ em ambos os eixos, o eixo z não é contabilizado a gravidade da Terra, para essa posição de medida, o que limita a faixa da medida em $+1g$ e $-3g$.

Sabendo que, para evitar *Aliasing* é necessária uma taxa de aquisição de dados à uma taxa mínima de 2 vezes a frequência de corte do filtro passa baixa. (Teorema de Nyquist), os testes serão realizados com frequência até 25 Hertz. No primeiro teste foi escolhida uma onda senoidal com frequência de 6 Hertz (Figura 25), a partir do momento em que a mesa vibratória é acionada e os parâmetros são ajustados, os testes são iniciados. Cada teste tem duração de, aproximadamente, 10 segundos. O período inicial e final dos

gráficos são descartados, visto que, quando o usuário habilita o envio de dados para o servidor, ele tem que entrar em contato físico com o aparelho, sendo assim, causando um momento de transiente na medida.

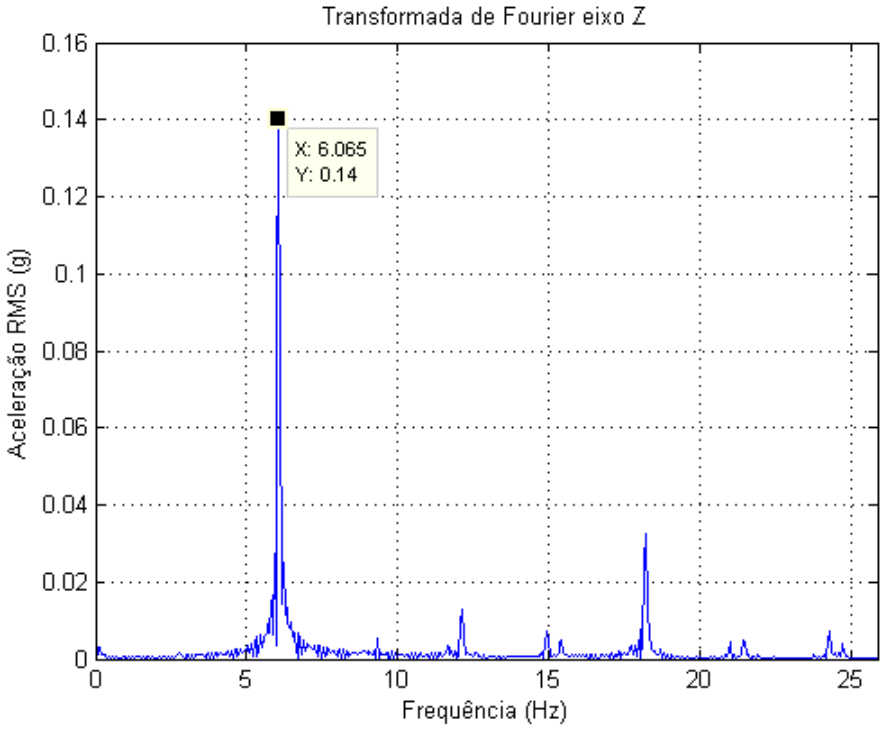


Figura 25 FFT *6Hz.*

Conforme a proposta foi analisada a tendência repetitiva do aparelho e chegou-se a um resultado de: $Aceleração_{RMS} = 0{,}19343g \pm 0{,}000119g$, vide Tabela 7. No mesmo período o transdutor teve uma média de: $Aceleração_{RMS} = 0{,}1902g$. Juntando com a incerteza do ruído de fundo temos:

$$Aceleração_{RMS} = 0{,}19343g \pm 0{,}000154g$$

Tabela 7 Aceleração em RMS, frequência em torno de 6*Hz.*

	Aceleração (*g*)
Teste 1	0,1927
Teste 2	0,1938
Teste 3	0,1934
Teste 4	0,1930
Teste 5	0,1933
Teste 6	0,1934
Teste 7	0,1937
Teste 8	0,1941
Teste 9	0,1936
Teste 10	0,1934
Média	0,19343
Desvio padrão	0,000377
Incerteza tipo A	0,000119

Para o segundo teste, foi realizado o mesmo procedimento, mas com uma frequência de 9 Hertz (Figura 26).

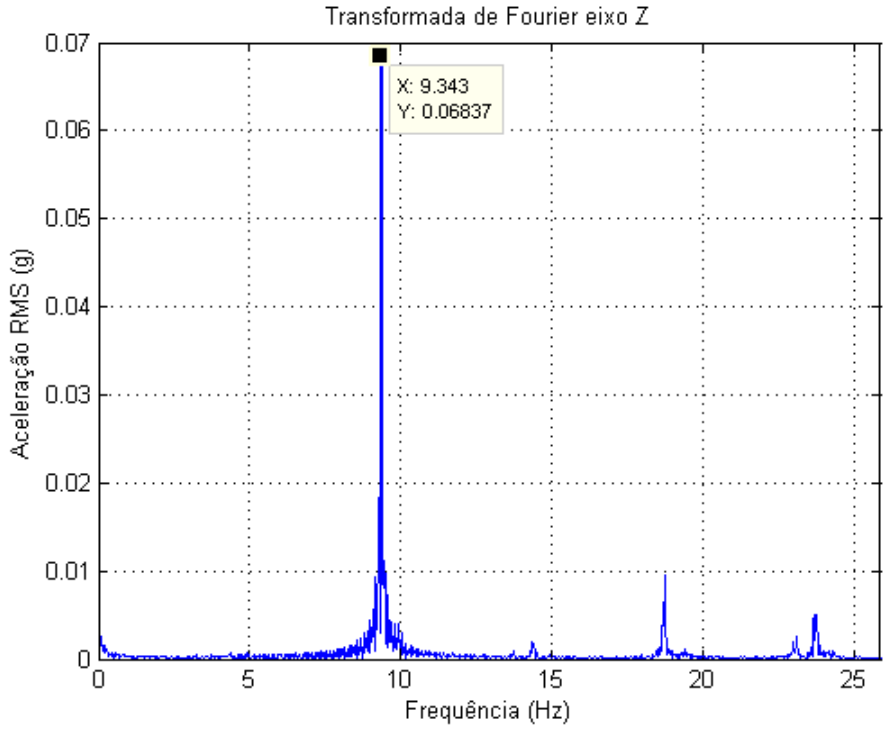


Figura 26 FFT *9Hz.*

A segunda análise de tendência teve como resposta: $Aceleração_{RMS} = 0{,}0980g \pm 0{,}00017g$, e o transdutor marcou: $Aceleração_{RMS} = 0{,}095g$. Juntando com a incerteza do ruído temos:

$$Aceleração_{RMS} = 0{,}0980g \pm 0{,}000205g$$

Em ambos os caso a incerteza padrão não ultrapassou 0,1%. O que demonstra um sistema com repetitividade elevada.

Tabela 8 Aceleração em RMS, frequência em torno de 9*Hz.*

	Aceleração (*g*)
Teste 1	0,0986
Teste 2	0,0988
Teste 3	0,0979
Teste 4	0,0978
Teste 5	0,0975
Teste 6	0,0988
Teste 7	0,0983
Teste 8	0,0975
Teste 9	0,0975
Teste 10	0,0973
Média	0,0980
Desvio padrão	0,00055
Incerteza tipo A (g)	0,00017

A variação da temperatura devido a mudanças no ambiente e auto aquecimento pode afetar a estabilidade do sensor. Nesse estudo não foi avaliada a influência da variação da temperatura dentro de condições controladas. No entanto, como o desvio padrão não foi significante nos dados coletados, e segundo o fabricante, o sensor possui ligações de modo comum com o objetivo de reduzir esse tipo de erro, foi assumido que variações de temperatura são compensadas internamente.

Os próximos testes têm por objetivo realizar uma comparação direta entre o acelerômetro embarcado e o de referência. Para isso será escolhida uma frequência dentro da faixa de atuação dos mesmos e a partir disso, será alterada a amplitude da onda. Os valores RMS dos mesmos são expostos na Tabela 9 e comparados através da Figura 27.

Tabela 9 Valore de aceleração RMS para onda senoidal 15*Hz.*

	Smartphone (Asus Zenfone 5)	**Transdutor (352C33)**
Teste 1	0,1166g	0,12g
Teste 2	0,2150g	0,235g
Teste 3	0,3669g	0,393g
Teste 4	0,4555g	0,489g

Percebe-se que para amplitudes maiores, os valores absolutos vão divergindo, mas não ultrapassam de 10% de diferença entre o valor maior e menor, comparativamente.

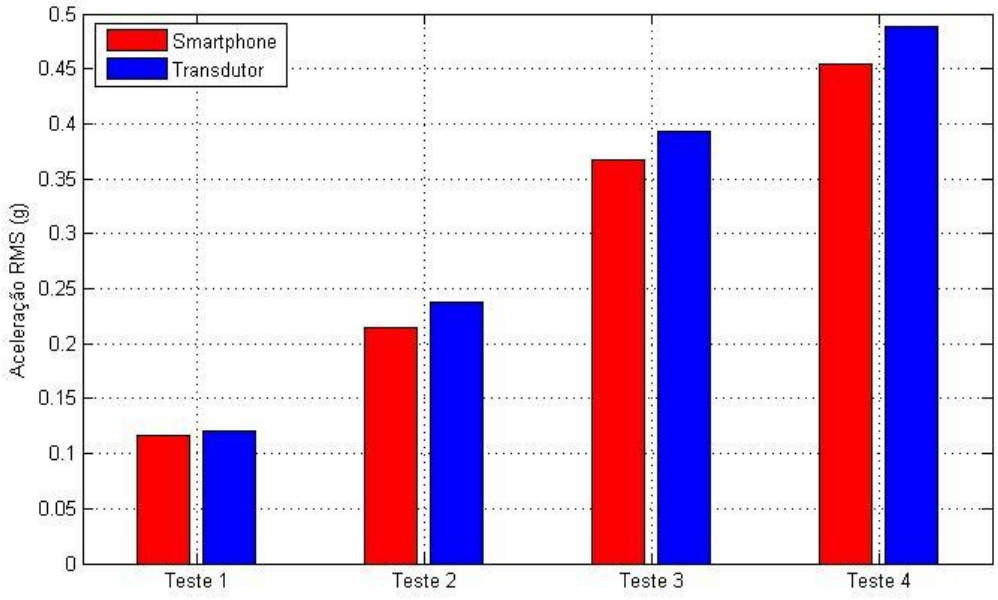


Figura 27 Gráfico Smartphone X Transdutor.

A amplitude escolhida para este teste foi de, aproximadamente, 15 Hertz (Figura 28).

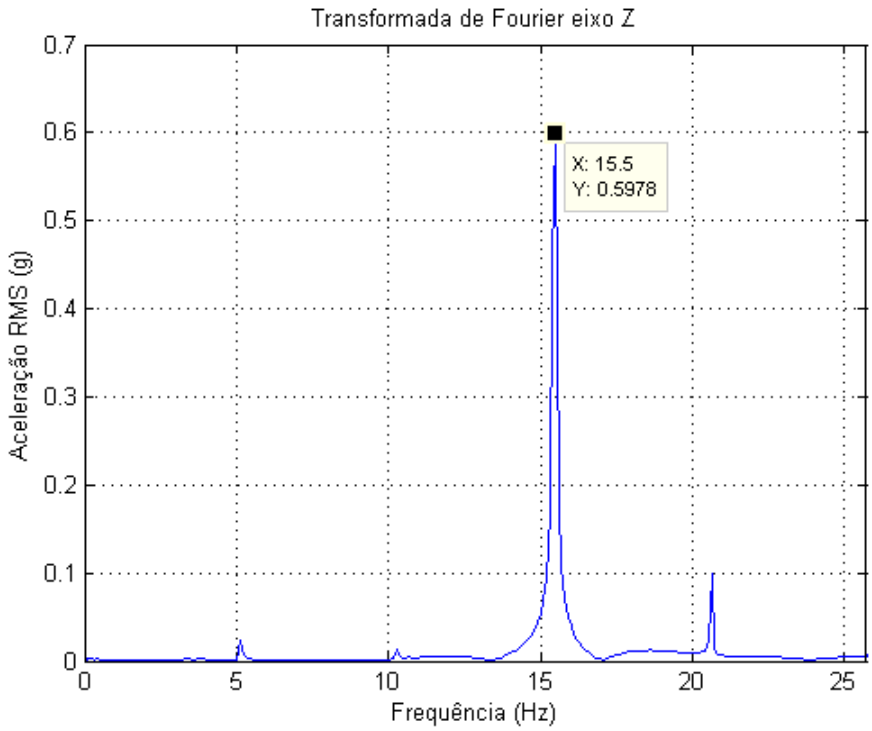


Figura 28 FFT 15Hz.

Para o próximo caso é mantida praticamente a mesma relação entre os resultados, os valores RMS de cada sensor estão relativamente próximos, o que leva a crer que o funcionamento está dentro do esperado.

Tabela 10 Valores aceleração RMS para onda triangular.

	Smartphone (KXTJ9)	**Transdutor (352C33)**
Teste 5	0,2942g	0,278g
Teste 6	0,3612g	0,362g

A ideia inicial era gerar uma onda que simulasse uma Delta de Dirac, este tipo de onda é um choque mecânico, ou seja, uma força aplicada em um curtíssimo espaço de tempo. Pode-se tratar um delta de Dirac como sendo um triangulo com a base infinitamente estreita e com uma área igual a 1, portanto, matematicamente falando, a delta de Dirac possui uma altura que tende ao infinito e uma base que tende a zero. Mecanicamente a simulação desta função é impossível de se conseguir. No espectro de frequência, uma onda triangular é representada, agora sim, por deltas de Dirac espaçadas por harmônicas da sua onda original, no caso, a onda simulada possui uma frequência aproximada de 7 Hertz, como é possível analisar, as harmônicas na Figura 29 estão bem definidas, mas existem alguns ruído inesperados, quando analisada, a resposta do sensor no tempo mostra uma onda “limpa” sem ruídos perceptíveis. Esses ruídos mostram o espelhamento das frequências acima da banda de passagem do sensor.

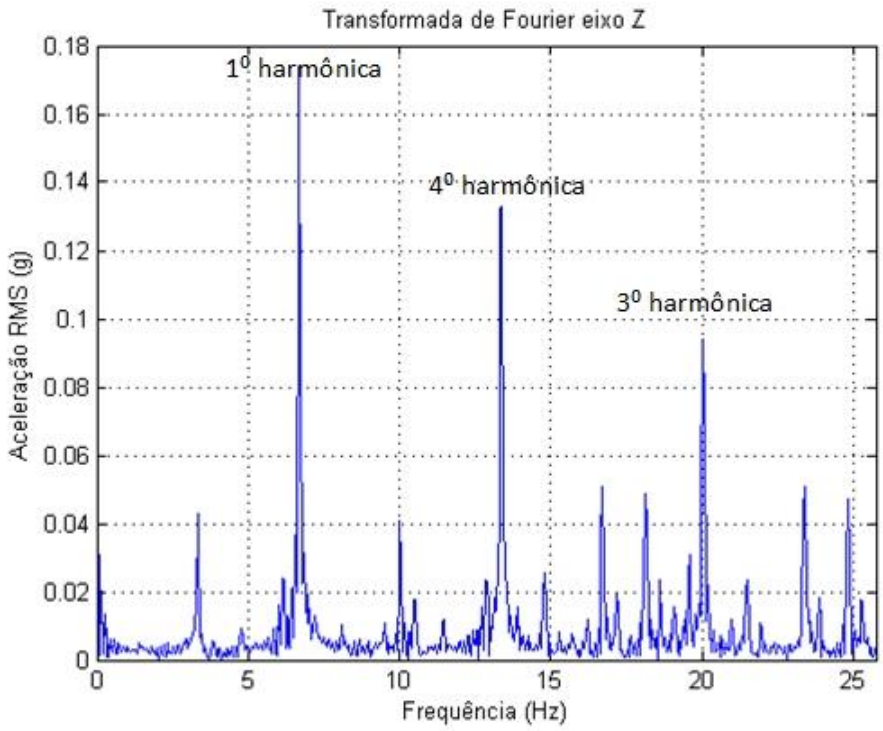


Figura 29 FFT onda triangular.

Outro teste foi realizado, o Smartphone foi excitado com uma onda senoidal de frequência 35 Hertz, maior que a máxima frequência de amostragem possível pelo aparelho. Foi encontrado um indício de aliasing (Figura 30 FFT Aliasing).

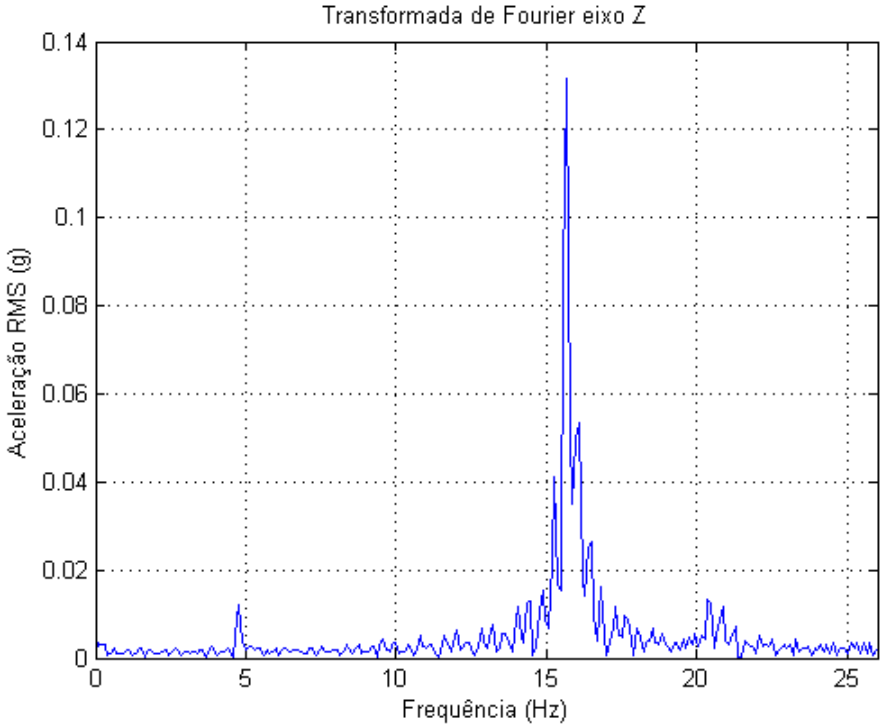


Figura 30 FFT Aliasing.

Segundo a Tabela 4 e a Figura 7, o sensor possui um filtro passa-baixa, sendo este filtro controlado pelo registrador DATA_CTRL_REG, e que por *default* está setado para uma frequência máxima de banda de 25 Hertz. Mas não é possível perceber uma grande atenuação do sinal do mesmo, o que demostra a atuação irrisória do filtro para frequências maiores de 25 Hertz.

O nível de influência de um ruído nos sistemas eletrônicos é apresentado de várias formas. Uma das mais importantes é a razão entre os valore RMS do sinal desejado e o valor RMS do ruído, simplesmente chamado de razão sinal/ruído (SNR). Uma outra forma é a caracterização de um sistema e não de um sinal, chamada de faixa dinâmica (DR). Essas duas grandezas são geralmente representadas em dB. Considerando-se a informação do ruído, na forma de aceleração, tem-se:

$$SNR_{dB} = 20log_{10}\left(\frac{A_{sinal}}{A_{ruido}}\right) \quad (5)$$

Onde A_{sinal} e A_{ruido} são expressos em valores RMS.

Por fim, é realizado uma relação sinal-ruído entre o ruído de fundo medido e os valores do limite inferior da zona de cautela da ISO 2631 (Figura 1). É estabelecida que para oito horas de exposição o limite de vibração mínima é de aproximadamente 0,0441g, com base nesse limite, é calculado o nível de influência do ruído para o caso. O valor encontrado foi de:

$$SNR_{dB} = 22{,}51dB$$

Ou seja, a energia do sinal é 13.36 vezes maior que a energia do ruído para p caso.

5 Conclusões e Trabalhos futuros.

Com base nos experimentos realizados e com o conhecimento adquirido em relação ao sensor embarcado nos aparelhos móveis, comparativamente ao que é exigido pela ISO 2631, é possível afirmar que para o caso de uma verificação preliminar da Vibração de Corpo Inteiro, a principal limitação dos smartphones é a faixa de frequência.

Em relação ao sinal, foi possível demonstrar que o aparelho possui um alto grau de concordância entre os resultados de medição sucessivas de um mesmo mensurando, a incerteza padrão combinada não passou de 0,1% de variação em relação a média das medidas. Para a norma, amplitudes de vibração maiores que $\pm 0{,}8485g$ e menores que $\pm 0{,}0612g$ estão fora das zonas de exposição à vibrações, portanto os valores da amplitude das amostras, relação sinal ruído e a resolução do sensor não se caracterizam como fatores que limitam a aquisição dos dados.

A frequência acabou se tornando um valor limitante neste caso, o espectro possível em que o aparelho está apto a realizar medidas não abrange o espectro de frequência proposta pela ISO 2631, mas no final, não acaba sendo um valor excludente, visto que, em muitas aplicações, a maior parte da energia do movimento está contida em frequências abaixo dos 25 Hz, que é o limite do *Smarphone* utilizado neste trabalho. Vale lembrar, ainda, que existe a possibilidade de configurar diretamente o kernel do sistema e assim obter uma faixa maior de amostragens por segundo.

O tempo de exposição a ser mensurado está diretamente vinculado a capacidade de armazenamento do dispositivo, para o caso de armazenamento de dados internos. Aparelho com maior frequência de amostragem, necessita de maior memória interna livre.

O fato dos aparelhos móveis conterem sensores triaxiais possibilita a realização das medidas em qualquer orientação, sendo diferenciado o eixo vertical por uma simples comparação com a gravidade, neste trabalho somente o $eixo\ z$ foi abordado, mas em relação ao sensor, ambos os eixos possuem o mesmo tratamento de sinal. Programas posteriores podem abordar os resultados medidos e trata-los conforme a norma, comparando os resultados com as zonas de exposição às vibrações, descartando a necessidade de análise dos dados por técnicos capacitados. Não é necessária uma conexão de rede para o tratamento de dados, os mesmos podem ser guardados na memória interna no aparelho sem interferir significativamente no funcionamento do mesmo, facilitando a aquisição das medidas em campo.

É possível realizar uma medida preliminar da VCI, trabalhos futuros podem se basear no fato que é possível realizar essa medida preliminar de vibração e que basta um bom projeto de programação para que esse processo de mensuração se torne uma ferramenta realmente eficaz para a segurança do trabalho.

6 Referências

ALESSANDRO LIMBERGER CARGNELUTT. Et al. Análise do nível de vibração em plataformas vibratórias para condicionamento físico frente à norma iso 2631. Universidade federal do Rio Grande do Sul. Porto Alegre, v. 13, n. 25, p. 42-43, 2012.

CARLOS ALBERTO GOLDANI. TCP/IP: Protocolos Internet. Lajeado: Univates, 1997.

EDSON GONÇALVES. Dominando Eclipse: Tudo que o desenvolvedor Java Precisa para Criar Aplicativos para Desktop, da criação do aplicativo ao desenvolvimento de relatório. Rio de janeiro: Editora Ciência Moderna Ltda., 2006.

JOÃO BOSCO MONTEIRO. Google Android: Crie aplicações para celulares e tablets. Casa do Código.

JOÃO CANDIDO FERNANDES. Et al. Guia para avaliação de exposição humana à vibrações de corpo inteiro. Universidade Estadual Paulista. São Paulo, 1978

JULIO CESAR GONÇALVES. *Uso da plataforma android em um protótipo de aplicativo coletor de consumo de gás natural*. 2011. 63 f. Monografia de Especialização - Universidade Tecnológica Federal do Paraná, Curitiba, 2011.

JUNIOR, E.; MENDES, R. Estudo das condições de trabalho e saúde de motoristas de ônibus urbano de Belo Horizonte –MG. Fundacentro, v. 25, n. 95/96, p. 131 -141, 1994.

PCB PIEZOTRINICS. Model 352c33, Platinum Stock Produts; Hight sensitivity, ceramic shear ICP® accel. Datasheet.

RAFAEL SANTOS. Introdução à Programação Orientada a Objetos usando Java. São José dos Campos: UNIVAP, 2001.

RICARDO R. LECHETA. Google Android para tablets: Aprenda a desenvolver aplicações para Android de smartphones a tablets. São Paulo: Novatec Editora, 2012.

SOTRON CONDITIONER. Endevco 102 / 109 Isotron Conditioner and Power Supply. Datasheet.

TIAGO BECKER. Comparação entre critérios de exposição à VCI. *Relações do Trabalho*, Porto Alegre, v. 1, n. 1, p. 1, 2015.

7 Anexos

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.heittor"
    android:versionCode="1"
    android:versionName="1.0" >
    <uses-permission android:name="android.permission.INTERNET"/>
    <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"/>
    <uses-permission android:name="android.permission.ACCESS_WIFI_STATE"></uses-permission>
    <uses-permission android:name="android.permission.CHANGE_WIFI_STATE"></uses-permission>
    <uses-sdk
        android:minSdkVersion="8"
        android:targetSdkVersion="21" />
    <application
        android:allowBackup="true"
        android:icon="@drawable/ic_launcher"
        android:label="@string/app_name"
        android:theme="@style/AppTheme">
        <activity
            android:name="SplashActivity"
            android:label="@string/app_name">

            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity
        android:name="HelpActivity"
        android:theme="@android:style/Theme.Translucent.NoTitleBar" >
        </activity>
        <activity
            android:name="MainActivity"
            android:label="@string/app_name" >
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

Anexo 1 Manifesto

```
public class SplashActivity extends Activity implements Runnable{
    protected void onCreate(Bundle savedInstanceState){
        super.onCreate(savedInstanceState);
        setContentView(R.layout.splash_screen);

        Handler handler = new Handler();
        handler.postDelayed(this, 3000);

    }

    @Override
    public void run() {
        // TODO Auto-generated method stub
        startActivity(new Intent(this, MainActivity.class));
        finish();

    }

}
```

Anexo 2 Classe SplashActivity

```
public class HelpActivity extends FragmentActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_help);
        TextView htmlTextView = (TextView)findViewById(R.id.html_text);
        htmlTextView.setMovementMethod (LinkMovementMethod.getInstance());
        htmlTextView.setText (Html.fromHtml (getString (R.string.help_text)));
    }

    public void dismiss(View v) {
        finish();
    }

}
```

Anexo 3 Classe HelpActivity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    startActivity(new Intent(getApplicationContext(), HelpActivity.class));

    sensorManager = (SensorManager)getSystemService(Context.SENSOR_SERVICE);
    sensorAcelerometer = sensorManager.getDefaultSensor(Sensor.TYPE_ACCELEROMETER);

    editTextIp=(EditText)findViewById(R.id.ip);
    editTextPort=(EditText)findViewById(R.id.port);
    connectOnOff=(ToggleButton)findViewById(R.id.connectT);
    sendStop=(ToggleButton)findViewById(R.id.sendStopTog);
    group =(RadioGroup)findViewById(R.id.radioGroup);

    sendStop.setOnClickListener(this);
    connectOnOff.setOnClickListener(this);

    sendStop.setEnabled(false);
    connectOnOff.setEnabled(true);

    sensorDelay=0;
    start = false;
    disconnect = false;
    pause = false;

}
```

Anexo 4 Função onCreate

```
try
{
    client = new Socket(editTextIp.getText().toString(),port);


}catch(UnknownHostException e){
    e.printStackTrace();
}catch(IOException e){
    e.printStackTrace();
}
```

Anexo 5 - Conexão Cliente

```
    public void onSensorChanged(SensorEvent event) {
        // TODO Auto-generated method stub
        Sensor meuSensor = event.sensor;
        if(meuSensor.getType()==Sensor.TYPE_ACCELEROMETER){
            x = event.values[0];
            y = event.values[1];
            z = event.values[2];


           onResume();

        }

    }
```

Anexo 6 Função onSensorChanged

```
public void onResume(){
    super.onResume();
    if(start){
        sensorManager.registerListener(this, sensorAcelerometer, sensorDelay);
        this.commsThread = new CommsThread();
        this.myCommsThread = new Thread(this.commsThread);
        this.myCommsThread.start();
    }

}
```

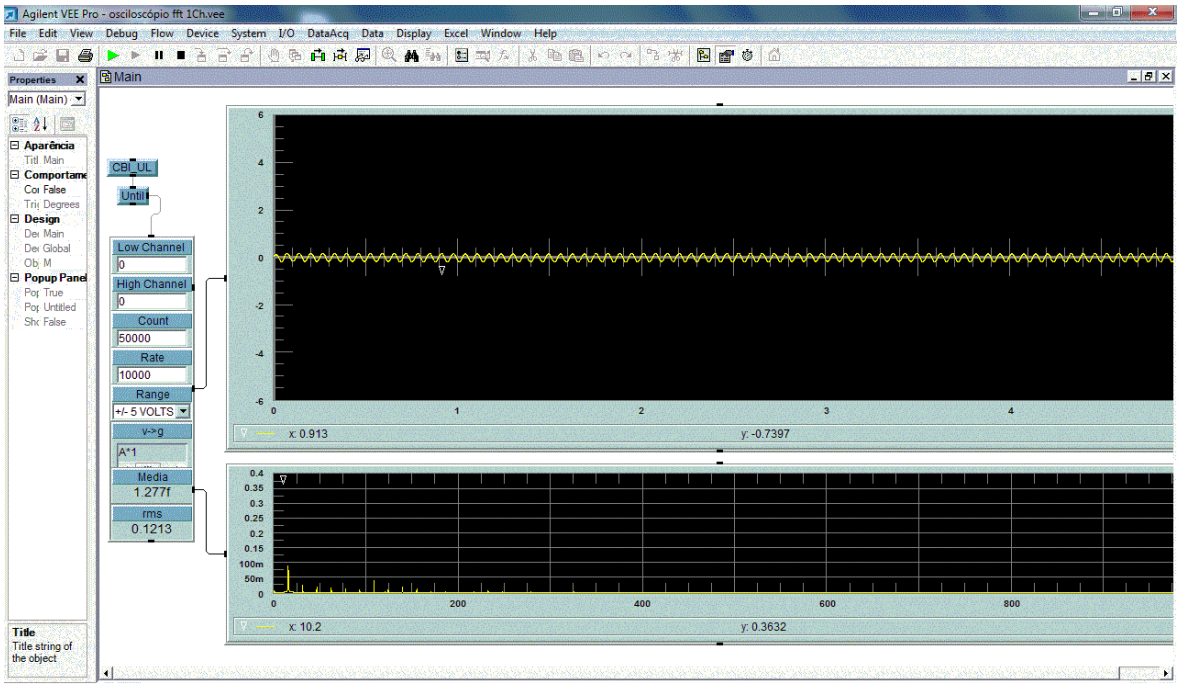
Anexo 7 Função onResume

```
clear all;
close all;
import java.net.*;
import java.io.*
%selecionando porta
porta=input('Escolha uma porta de acesso para o cliente: ');
servidor= ServerSocket(porta);
disp('Esperando conexão')
conectado=servidor.accept;
entrada=conectado.getInputStream;
disp('Conexão estabelecida!')
menssagem='';
nCont=0;
yCont=0;
zCont=0;
xCont=0;
tCont=0;
txCont=0;
tempo=0;
testeEntrada=0;
b=zeros();
a=zeros();
dadosTempoSeg=zeros();
dadosTempo=zeros();
dadosAcelX=zeros();
dadosAcelY=zeros();
dadosAcelZ=zeros();
txByte=zeros();

while isempty(strfind(menssagem,'!q'))
    entrada=conectado.getInputStream;
    while ~(entrada.available)
       %Espera uma entrada
       initialTimeByte=clock;
    end
    ent=entrada.available;
    for i=1:entrada.available
     b(i)=entrada.read();%lê byte a byte
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%
     if b(i)== 33
         menssagem = '!q';  %espera sinal para encerrar conexão
     end
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%
     if b(i)== 63
         dadosTempoSeg=dadosTempo/1000;
         plotting(dadosAcelX, dadosAcelY, dadosAcelZ, dadosTempoSeg,
txByte);
         dadosTempoSeg=zeros();
         dadosTempo=zeros();
         dadosAcelX=zeros();
         dadosAcelY=zeros();
         dadosAcelZ=zeros();
         txByte=zeros();
         yCont=0;
         zCont=0;
         xCont=0;
         tCont=0;
         txCont=0;
         testeEntrada=0;
     end
     %%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

Anexo 8 Servidor

```
      if b(i)==35 %compara se o byte é == #
          testeEntrada=testeEntrada+1;
          if testeEntrada==1
              tempo=str2double(char(a));
              tCont=tCont+1;
              dadosTempo(tCont)=tempo;
           end
          if testeEntrada==2
              newa=strrep(char(a),',','.');%pradroniza dados
              acelerometro=str2double(newa);%converte string em numero//
atribuo o dado recebido como aceleração
              xCont=xCont+1;
              dadosAcelX(xCont)=acelerometro;
           end
          if testeEntrada==3
              newa=strrep(char(a),',','.');%pradroniza dados
              acelerometro=str2double(newa);%converte string em numero//
atribuo o dado recebido como aceleração
              disp(acelerometro);

              yCont=yCont+1;
              dadosAcelY(yCont)=acelerometro;
          end
          if testeEntrada==4
              newa=strrep(char(a),',','.');%pradroniza dados
              acelerometro=str2double(newa);%converte string em numero//
atribuo o dado recebido como aceleração
              disp(acelerometro);
              zCont=zCont+1;
              dadosAcelZ(zCont)=acelerometro;
              testeEntrada=0;

          end
          nCont=0;
          a=zeros();
      else
          nCont=nCont+1;
          a(nCont)=b(i);%converte byte em array
      end
     end
         ms = round(etime(clock,initialTimeByte)*1000);
         txCont=txCont+1;
         txByte(txCont)=ent/ms;
end

disp('Conexão encerrada!');

pause(1)
conectado.close;
```

Anexo 9 Servidor



Anexo 10 Agilent VEE Pro, software de aquisição de dados.

APÊNDICE I – O aplicativo.

O aplicativo está disponível via *Google Drive*, pelo *link* (6), na forma de um arquivo de extensão (*.apk*), o nome é *Aceletrometro.apk*, o mesmo pode rodar em qualquer plataforma Android de atualização 1,6 à 6,0. Para instala-lo é simples, basta baixa-lo em seu aparelho *Smartphone* e quando for clicar para instalar, na caixa de diálogo, habilite a instalação de aplicativos de fontes desconhecias, depois é só clicar em "Instalar" e o aplicativo já pode rodar.

Junto com o mesmo arquivo, ainda existem mais dois arquivos de extensão (*.m*) que são executáveis via *Matlab*. Ambos arquivos, *ACELEROMETRO_TCP.m* e *plottin.m*, são arquivos referentes ao servidor, e juntos eles habilitam um *ServerSocket*, após adiciona-los a pasta "*Current folder*" do Matlab, basta clicar em "*Run*" e escolher uma porta.

Lembrando que o *Smartphone* e computador devem estar conectados na mesma rede e é preciso ter o endereço de IP do computador dentro desta rede.

https://drive.google.com/folderview?id=0B0drRYJI3suFWmh0aC00enhSSWM&usp=sharing

(6)