Universidade Federal
do Rio de Janeiro
Escola Politécnica

MONITORAMENTO DE VARIÁVEIS ELÉTRICAS DE UM SISTEMA FOTOVOLTAICO COM ARDUINO E ANDROID

Gustavo Pacheco Epifanio

Projeto de Graduação apresentado ao Corpo Docente do Departamento de Engenharia Elétrica da Escola Politécnica da Universidade Federal do Rio de Janeiro, como parte dos requisitos necessários à obtenção do título de Engenheiro Eletricista.

Orientador: Luís Guilherme Barbosa Rolim

Rio de Janeiro
Março de 2015

MONITORAMENTO DE VARIÁVEIS ELÉTRICAS DE UM SISTEMA FOTOVOLTAICO COM ARDUINO E ANDROID

Gustavo Pacheco Epifanio

PROJETO DE GRADUAÇÃO SUBMETIDO AO CORPO DOCENTE DO DEPARTAMENTO DE ENGENHARIA ELÉTRICA DA ESCOLA POLITÉCNICA DA UNIVERSIDADE FEDERAL DO RIO DE JANEIRO COMO PARTE DOS REQUISITOS NECESSÁRIOS PARA A OBTENÇÃO DO GRAU DE ENGENHEIRO ELETRICISTA.

Examinado por:

Prof. Luís Guilherme Barbosa Rolim, D.Ing.

Prof. Sergio Sami Hazan, Ph.D.

Rafael de Oliveira Rodruigues, Eng.

RIO DE JANEIRO, RJ – BRASIL
MARÇO DE 2015

Epifanio, Gustavo Pacheco

Monitoramento de variáveis elétricas de um sistema fotovoltaico com Arduino e Android / Gustavo Pacheco Epifanio. – Rio de Janeiro: UFRJ/Escola Politécnica, 2015.

XI, 59 p.: il.; 29, 7cm.

Orientador: Luís Guilherme Barbosa Rolim

Projeto de Graduação – UFRJ/Escola Politécnica/ Departamento de Engenharia Elétrica, 2015.

Referências Bibliográficas: p. 34 – 35.

1. Sistemas Fotovoltaicos. 2. Qualidade de sinal. 3. Arduino. 4. Android. I. Rolim, Luís Guilherme Barbosa. II. Universidade Federal do Rio de Janeiro, Escola Politécnica, Departamento de Engenharia Elétrica. III. Monitoramento de variáveis elétricas de um sistema fotovoltaico com Arduino e Android.

Caminhando...

Agradecimentos

Foram muitos anos, e muitos me ajudaram a chegar até aqui. Não é possível citar todos mas com certeza se não fossem por eles não estaria aqui agora.

Em especial queria agradecer a sala de estudos LASP, por ceder este espaço para estudarmos, foram longas tardes, noites e madrugadas passadas por lá, eu e o grupo da elétrica estudando juntos para aprendermos.

Agradecimento as minhas amigas, que entraram na mesma turma que eu, Camila que foi inúmeras vezes incomodada por mim para me ajudar com vários problemas de matérias, com ajuda técnica e na revisão do projeto final, e conselhos, Helena que desde o começo uma grande amiga, trocando ideias, fazendo matérias junto e me aconselhando, Isabela que mesmo nos abandonando da engenharia elétrica e nos encontrando pouco, sem duvida marcou sua passagem.

Resumo do Projeto de Graduação apresentado à Escola Politécnica/UFRJ como parte dos requisitos necessários para a obtenção do grau de Engenheiro Eletricista

MONITORAMENTO DE VARIÁVEIS ELÉTRICAS DE UM SISTEMA FOTOVOLTAICO COM ARDUINO E ANDROID

Gustavo Pacheco Epifanio

Março/2015

Orientador: Luís Guilherme Barbosa Rolim

Departamento: Engenharia Elétrica

Apresenta-se neste trabalho, um arranjo simples e de baixo custo para analisar a corrente de saída do inversor de um sistema fotovoltaico. Para alcançar esse objetivo, a plataforma Arduino foi utilizada para amostrar e calcular os parâmetros necessários a um baixo custo, e um aplicativo para Android foi feito para apresentar os resultados, sendo uma solução simples e acessível. O sistema desenvolvido é flexível, podendo ser modificado com instruções simples para atender as necessidades do usuário. Este sistema mostrou resultados satisfatórios quando comparado com o analisador de sinal usado como referência. Portanto, podemos concluir que o sistema desenvolvido é confiável o suficiente para ser utilizado.

Abstract of Graduation Project presented to POLI/UFRJ as a partial fulfillment of the requirements for the degree of Electrical Engineer

ELECTRIC VARIABLE MONITORING OF A PHOTOVOLTAIC SYSTEM WITH ARDUINO AND ANDROID

Gustavo Pacheco Epifanio

March/2015

Advisor: Luís Guilherme Barbosa Rolim

Department: Electrical Engineering

In this work, we develop a simple and inexpensive way to analyse the output current of the inverter from a photovoltaic system. To achieve that, the Arduino platform was used to sample and calculate the parameters needed in a inexpensive way, and a Android application was made to display the results, being a simple and accessible solution. The system developed is flexible and can be modified with simple instructions to fit the user needs. This system has shown satisfactory results when compared with the signal analyser used as reference. Hence, we can conclude that the system developed is reliable enough to be used.

Sumário

Lista de Figuras

Capítulo 1

Introdução

Devido ao seu grande potencial com geração de eletricidade com baixo impacto ambiental, sistemas fotovoltaicos vêm sendo mais pesquisados, ao mesmo tempo que a adoção por usuários também vem crescendo. O uso residencial é comumente do tipo mostrado na Figura 1.1.

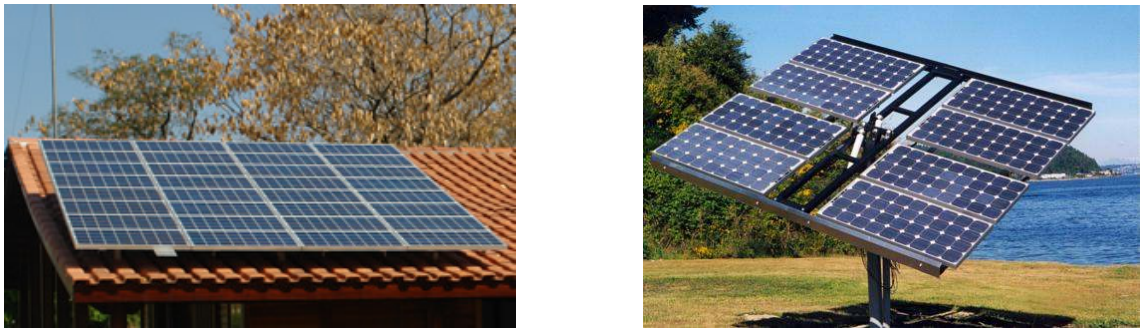
O sistema de energia elétrica do Brasil opera a $60Hz$ e os aparelhos domésticos e industriais também devem seguir esse padrão. A energia de sistemas fotovoltaicos é gerada em corrente contínua, e para ser utilizada em cargas comuns é necessário que exista uma conversão de corrente contínua para corrente alternada a $60Hz$. Isto é feito por circuito de eletrônica de potência, denominado inversor. Para avaliar a qualidade da conversão, é preciso que medições sejam feitas, pois tanto a tensão de saída do inversor quanto a corrente drenada pelas cargas que ele alimenta podem conter harmônicos em excesso.

Existem excelentes sistemas e equipamentos que fazem essa medição, porém, em geral têm um custo elevado, que acabam se tornando inviáveis para o uso em sistemas fotovoltaicos de pequeno porte e baixo orçamento. Para resolver este problema é indispensável que exista uma solução de baixo custo que esteja de acordo com especificações de sistemas de pequeno porte.

Este projeto tem por finalidade preencher esta lacuna, sendo capaz de fazer medições do sinal desejado, e de forma simples apresentar os resultados para serem analisados.

1.1 Funcionamento do Sistema Fotovoltaico

Este projeto será aplicado a um sistema fotovoltaico já existente que foi instalado para operar isolado da rede elétrica de distribuição. Quando os painéis são utilizados de maneira isolada para alimentar as cargas existentes, é necessário utilizar, além do inversor que condiciona a energia de CC para CA, um sistema de armazenamento



(a) (b)

Figura 1.1: Painel ono telhado de casa (a) e painel no terreno (b).

de energia. Um diagrama esquemático deste tipo de conexão é mostrado na Figura 1.2 e seus componentes serão descritos brevemente a seguir.

Figura 1.2: Diagrama do sistema fotovoltaico utilizado

Painéis Fotovoltaicos

Formado por várias células, que são constituídas de semicondutores, que por efeito fotovoltaico geram potência elétrica na forma de corrente contínua.

Carregador de Bateria/MPPT

Carregadores de bateria são dispositivos que servem para controlar a tensão e a corrente nos terminais do banco de baterias, de modo a evitar cargas ou descargas excessivas do mesmo.

Maximum Power Point Tracking (MPPT) são técnicas aplicadas em sistemas fotovoltaicos para ter máximo aproveitamento da potência de saída do painel, que depende da irradiação solar e da temperatura do ambiente [4]. Tais técnicas são usadas em controladores de carga mais sofisticados, ajustando continuamente a tensão de operação dos painéis fotovoltaicos e aumentando assim a eficiência do processo de carregamento do banco de baterias.

Banco de baterias

As baterias têm por finalidade evitar as interrupções sofridas pelo painel, por ações de chuva e falta de sol por exemplo, a fim de ter uma fonte de energia mais confiável.

Inversor

Tem por finalidade converter a corrente contínua proveniente da bateria em corrente alternada para a carga.

Carga CA

Carga em corrente alternada que será alimentada pelo sistema fotovoltaico.

O foco deste trabalho é o monitoramento na transferência que ocorre entre os dois últimos elementos, o inversor e a carga.

1.2 Relação entre Inversor e Carga

Os inversores podem ser divididos em três grupos em relação a forma de onda do sinal convertido, os puramente senoidal, quase senoidal, e quadrada. Com relação ao custo, os inversores senoidais são mais caros, pois precisam ter circuitos de controle e filtros mais complexos para produzirem formas de onda mais próximas do ideal. Algumas cargas são mais ou menos sensíveis a distorções na forma de onda, como por exemplo computadores ou lâmpadas. As diferentes formas de onda podem ser vistas na Figura 1.3 [5].

Inversores não são como a rede elétrica que idealmente podem fornecer correntes muito maiores que a corrente nominal de uma dada carga. Dessa forma se uma carga com uma alta corrente de partida for acionada, como um motor de indução, o comportamento do inversor será diferente da rede elétrica, podendo reduzir fortemente sua tensão de saída, demorando mais tempo para acionar a carga, ou mesmo desligar por sobrecorrente.

As diferentes formas de onda do inversor geram diferentes níveis de harmônicos, e esses harmônicos podem ser prejudiciais à carga, como em motores que podem aquecer mais, e dependendo de qual harmônico está presente, pode gerar ruídos audíveis na máquina elétrica [6]. Alguns tipos de medidores também podem sofrer com harmônicos caso tenham níveis suficientemente elevados, e se um medidor não medir um valor certo, este perde seu propósito.

O princípio de funcionamento de um inversor fonte de tensão monofásico, com carga indutiva e saída de onda quadrada, pode ser visto na Figura 1.4 [7]. Neste caso quando T1 e T4 conduzem, tensão positiva é aplicada à carga, e tensão negativa se dá quando T2 e T3 estão conduzindo. O papel dos diodos é garantir uma rota de retorno da corrente à fonte caso a carga tenha características indutivas. Na Figura 1.5 são mostrados os harmônicos que compõem a onda quadrada. Outros circuitos, elementos e disparos diferentes podem caracterizar outra forma de onda.

Uma carga linear alimentada por tensão senoidal implica uma corrente também senoidal. Porém, mesmo para as cargas lineares, quando alimentadas com tensões

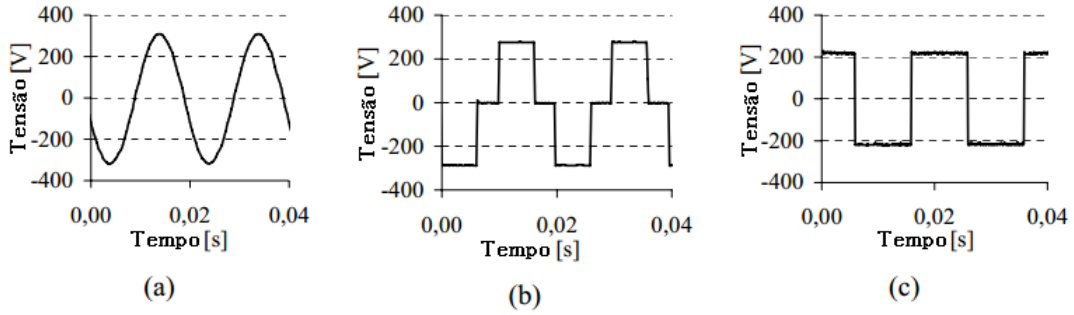


Figura 1.3: Diferentes formas de onda da tensão para os diferentes tipos de inversor. (a) Puramente Senoidal. (b) Quase Senoidal. (c) Quadrada.

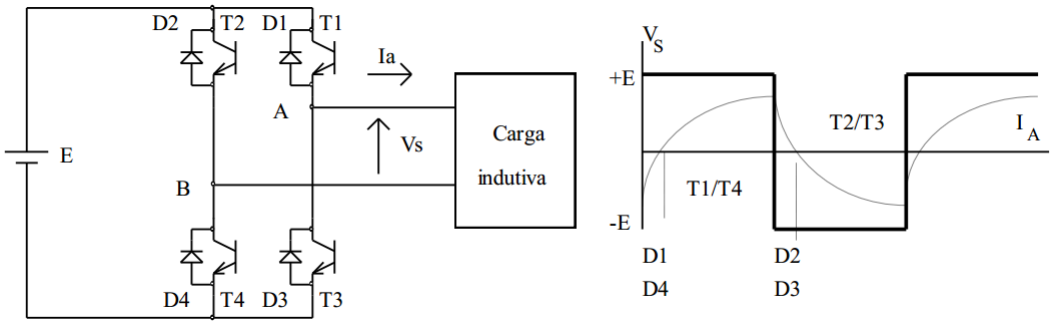


Figura 1.4: Circuito de um inversor monofásico com forma de onda quadrada

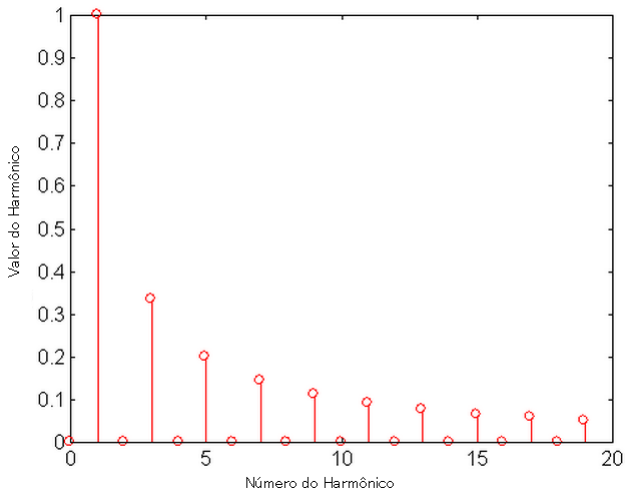


Figura 1.5: Espectro de frequência da onda quadrada.

não senoidais, aparecem correntes não senoidais, ou seja, com harmônicos. Isto pode ser mostrado observando o gráfico da Figura 1.4, que para uma tensão de alimentação quadrada a corrente toma uma forma diferente. Já para cargas não lineares, mesmo com fonte de tensão senoidal a corrente pode tomar outra forma como se pode observar na Figura 1.6. Uma mesma carga sendo alimentada pela tensão senoidal da rede ou por um inversor com forma de onda quase senoidal, a forma de onda da corrente não é senoidal, possuindo harmônicos.

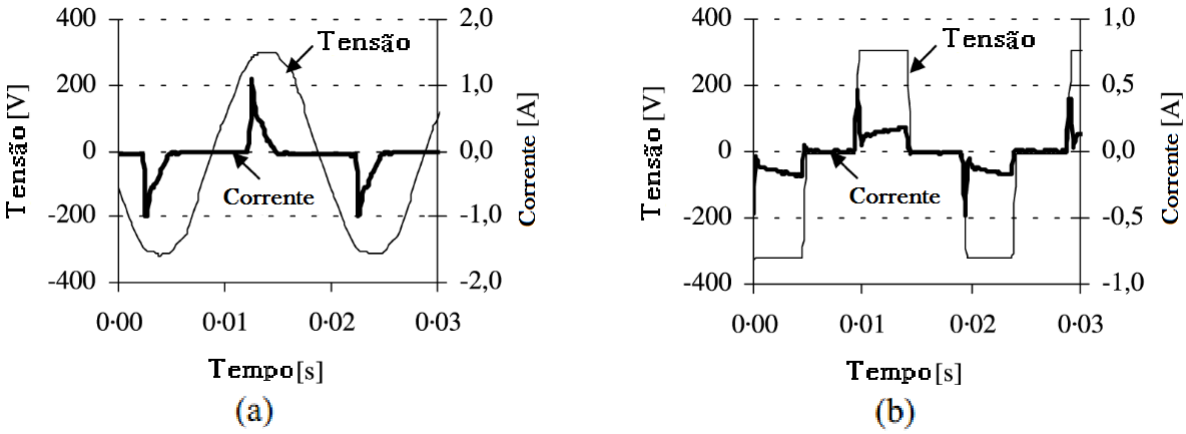


Figura 1.6: Formas de onda de uma mesma carga para diferentes fontes de alimentação. (a) Alimentação senoidal da rede. (b) Alimentação por um inversor de onda quase-senoidal

Existe um tipo de carga não linear, as que fazem uso de retificador próprio, como por exemplo computadores que internamente funcionam em CC mas recebem energia em CA. Em geral os retificadores destas cargas são não controlados e com filtro capacitivo. Desta forma diminuindo o custo de produção, por outro lado acabando por inserir mais harmônicos de corrente na rede, pela falta de tratamento na onda.

1.3 Objetivos

O LAFAE possuí um projeto em uma escola de Cabo Frio que carece de energia elétrica da rede, que consiste em um sistema fotovoltaico que alimenta duas salas da mesma escola. Porém algumas vezes acontecem situações de desligamento do sistema, mesmo o inversor tendo potência suficiente para alimentar as cargas. Algumas suspeitas foram levantadas quanto a razão disto, mas nada se pôde confirmar, para maiores certeza é preciso fazer um diagnóstico do que está acontecendo, sendo necessário um sistema de medição. Para suprir essa necessidade o sistema de monitoramento deste projeto foi inicialmente pensado

Este projeto tem por finalidade analisar a qualidade do sinal gerado por um inversor, alimentando cargas de corrente alternada. Para isso deve ser analisada a saída de corrente do inversor do sistema, conforme foi brevemente explicado na Seção 1.1. Com foco em sistemas fotovoltaicos de pequeno porte, o sistema de medição proposto neste trabalho não poderá ter custo elevado.

O escopo do projeto consiste em projetar e implementar um protótipo de uma sistema de medição para analisar um sinal alternado, o que envolve medir e calcular dados relevantes sobre o mesmo, sendo capaz de apresentar os resultados, de uma

forma simples, que possa ser interpretada pelo usuário de forma direta.

Não existe limitação técnica para que o projeto seja aplicado somente para análise de medição de qualidade de conversão de inversores, sendo assim traçado um objetivo secundário, com um mínimo de alteração, modificar o projeto para ser utilizado em medição de outros sinais, tornando o sistema de medição proposto mais versátil.

De forma simplificada, os objetivos traçados para o desenvolvimento deste projeto se resumem em:

- Medição de sinais periódicos.
- Simples visualização dos dados.
- Baixo custo.
- Fazer uso de tecnologias difundidas e de fácil acesso.
- De fácil adaptação para novas funcionalidades.

1.4 Trabalhos Anteriores

Existem na literatura trabalhos com objetivos semelhantes, utilizando a plataforma Arduino e sistemas fotovoltaicos, alguns com foco no monitoramento e outros com foco em otimização.

Pode-se citar o exemplo de um trabalho com foco em monitoramento que é feito em Goiânia, onde foi desenvolvido um projeto [8] para monitoramento remoto de um sistema fotovoltaico com foco em baixo custo e *Open Source*. Outros trabalhos nessa área também são encontrados como monitoramento de curvas corrente x tensão de sistema fotovoltaico [9].

É possível encontrar também outros tipo de estudo envolvendo Arduino e sistemas fotovoltaicos, em otimização, como exemplo pode ser citado um trabalho com foco em sistemas autônomos, com bom custo-benefício, para rastreamento do ponto de máxima potência [10].

Apesar desses tópicos, não foi encontrado nenhum trabalho que tenha uma maior semelhança com o que será desenvolvido. Sendo assim este trabalho pode servir como referência para projetos futuros.

1.5 Organização

Este trabalho está organizado em 5 capítulos, sendo o primeiro este, de introdução, onde foram apresentadas ideias gerais que envolvem o projeto.

No Capítulo 2 será apresentada a estrutura do projeto. Quais elementos que compõem o sistema e como se comunicam, e como foi feita a calibração do sensor.

O Capítulo 3 apresenta as metodologias e algoritmos de cálculo utilizados, e será explicado quais ferramentas e o ambiente de desenvolvimento utilizado.

O resultado e outras discussões são mostradas no Capítulo 4. Neste capítulo é mostrada a versão final do projeto, e justificativa da escolha da comunicação utilizada.

Por fim, o Capítulo 5 apresenta a conclusão do projeto, pontos positivos e negativos, e sugestão de trabalhos futuros.

Capítulo 2

Composição do Sistema

2.1 Introdução

Tendo em vista os objetivos traçado na Seção 1.3, foi estruturado um sistema baseado em Arduino, uma plataforma bem conhecida e de fácil acesso, e Android, que é o sistema operacional para dispositivos móveis mais utilizado no mundo.

Neste capítulo será abordado como foi realizada a composição dos sistema, quais elementos foram utilizados para alcançar os objetivos. Cada parte do sistema será explicada em mais detalhes a seguir.

2.2 Estrutura

O projeto tem duas grandes estruturas e a comunicação entre elas. A primeira e principal estrutura será responsável por realizar as medições do sinal a ser analisado, e fazer todos os tipos de cálculos necessários para obter os dados desejados, e os salvar em um arquivo, o Arduino foi a plataforma escolhida para realizar essas funções. A segunda estrutura será responsável por ler o arquivo da saída da primeira, e o traduzir graficamente de forma a facilitar a compreensão do usuário.

A medição do sinal do inversor é feita por meio de um sensor de corrente, que é digitalizado pelo Arduino. Para a análise em questão é preciso armazenar alguns dados característicos da onda como por exemplo, o valor RMS e amplitude do fundamental, esses dados são calculados por meio da série de Fourier.

Um resumo esquemático do funcionamento do sistema como um todo, e que operações fazem parte de cada estrutura, pode ser observado na Figura 2.1.

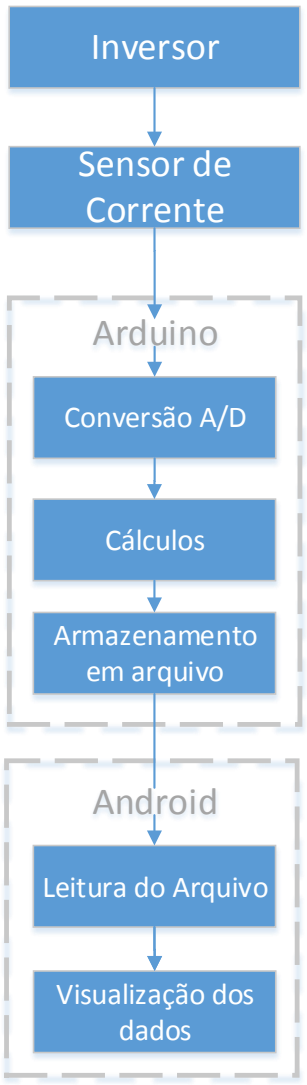


Figura 2.1: Diagrama da composição do sistema

2.3 Arduino

O papel do Arduino é ser responsável por adquirir os dados por meio do sensor de corrente, calcular os resultados e armazenar os valores obtidos.

Para adquirir os dados, foi utilizado um sensor de corrente da Allegro, modelo ACS712, com o módulo preparado para ser acoplado ao Arduino. A resolução máxima suportada pelo Arduino é de 10 bits, e como o limite de corrente deste sensor é de $-20\ A$ até $+20\ A$ temos uma resolução de 39 mA. O sensor será explicado em mais detalhes no Capítulo 3.

É necessário o tratamento dos dados após a aquisição dos mesmos, para que seja possível obter os valores necessários. Para este fim foi utilizada a série de Fourier. A escolha deste método para a resolução do problema foi principalmente

pelo baixo consumo de memória deste algoritmo no Arduino, métodos mais rápidos como FFT (*Fast Fourier Transform*) tinham o consumo de memória mais elevado que a capacidade do armazenamento do microprocessador. Com este método serão calculados no total 14 valores, sendo eles listado abaixo.

- Componente contínua
- Do primeiro ao décimo primeiro hormônico
- Valor RMS (*Root Mean Square*)
- Valor THD (*Total Harmonic Distortion*)

Um meio de armazenamento dos dados calculados é necessário para dar continuidade ao projeto. Para este fim foi utilizado um módulo para o Arduino de leitura e gravação, com armazenamento feito por meio de um cartão SD. Após um período ser amostrado e ser analisado com a série de Fourier, os valores obtidos são armazenados em um arquivo, de modo que cada linha representa os dados de um período de onda.

Um exemplo da forma final do arquivo de armazenamento pode ser visto na Figura 2.2. Neste formato os dados são salvos na ordem que foram apresentados na lista anterior, por exemplo o primeiro é a componente contínua, e o último é o THD. O importante deste arquivo é que o programa no Android seja capaz de interpretá-lo.

```
0.01 3.30 0.05 0.04 0.06 0.18 0.03 0.08 0.05 0.05 0.08 0.03 2.35 0.126
0.00 3.32 0.05 0.04 0.02 0.17 0.04 0.10 0.05 0.08 0.05 0.10 2.38 0.173
0.01 3.32 0.06 0.06 0.08 0.16 0.05 0.12 0.08 0.07 0.10 0.03 2.38 0.164
0.02 3.24 0.01 0.01 0.02 0.23 0.04 0.06 0.04 0.05 0.04 0.09 2.33 0.179
0.03 3.31 0.02 0.01 0.02 0.20 0.04 0.11 0.04 0.02 0.03 0.08 2.36 0.131
0.06 3.26 0.04 0.02 0.04 0.21 0.09 0.08 0.06 0.06 0.09 0.06 2.34 0.180
0.05 3.27 0.01 0.04 0.02 0.18 0.03 0.06 0.02 0.05 0.02 0.03 2.33 0.134
0.04 3.30 0.02 0.04 0.02 0.20 0.01 0.10 0.03 0.04 0.03 0.06 2.36 0.172
0.03 3.29 0.02 0.02 0.07 0.21 0.03 0.09 0.06 0.03 0.10 0.06 2.35 0.155
0.03 3.27 0.02 0.06 0.03 0.20 0.01 0.08 0.01 0.07 0.03 0.06 2.34 0.156
0.03 3.29 0.02 0.02 0.08 0.21 0.06 0.08 0.06 0.06 0.11 0.06 2.35 0.162
0.03 3.33 0.04 0.05 0.05 0.22 0.03 0.16 0.06 0.10 0.06 0.12 2.38 0.173
```

Figura 2.2: Exemplo de um arquivo salvo no cartão SD, cada linha é um amostra

Um resumo de como funciona a estrutura do Arduino pode ser visto na Figura 2.3. O programa em Android será responsável pela visualização dos dados, porém para ter uma noção do que se esperar deste programa, foi desenvolvido um script com ajuda do programa gnuplot para fazer este papel. O resultado deste script pode ser observado na Figura 2.4.

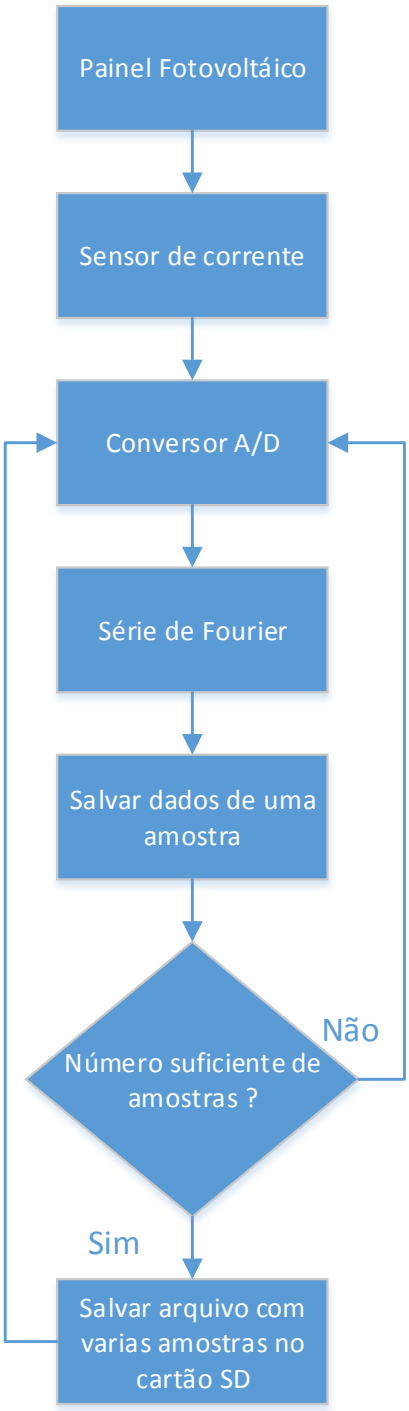


Figura 2.3: Fluxograma do programa no Arduino

2.3.1 Calibração do Sensor de Corrente

Os sensores precisam ser calibrados em algum momento, o que foi utilizado não é exceção, para isso foi utilizado como referência o analisador de sinal PowerPad® modelo 8335 da AEMC Instruments. Esse instrumento está disponível no LAFAE

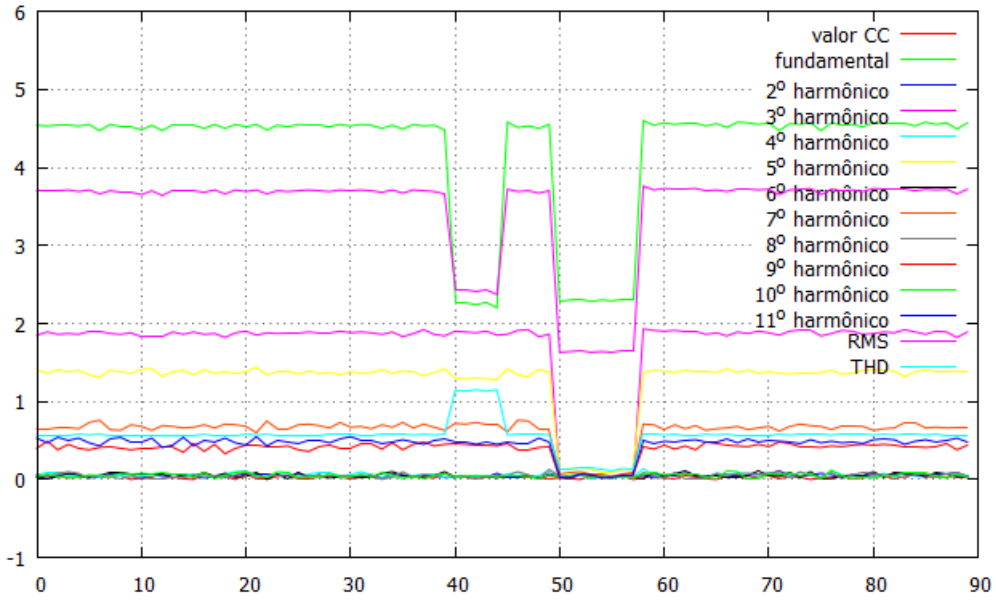


Figura 2.4: Exemplo ilustrativo de como podem ser apresentados os dados no Android, feito no gnuplot

(Laboratório de Fontes Alternativas de Energia da UFRJ) e é mostrado na Figura 2.5 [1].

Antes é preciso saber em que nível está a calibragem de fábrica do sensor, para isto é feita a medição em um mesmo estado do sistema alimentado pelo inversor, para o analisador e para o sensor de corrente, em seguida são repetidas as medições para 3 estados diferentes do sistema. A comparação pode ser vista na 2.6. Da observação da figura é possível perceber que a calibração de fábrica não está de acordo com o instrumento de referência, logo é necessário fazer a calibração do sensor.

Com a necessidade do processo de calibragem sendo constatada, discute-se a seguir como foi realizado este processo.

Para comparar os sinais do analisador com o sinal do Arduino ambos têm que ser colocados em um mesmo formato. Para o analisador, que exporta os pontos amostrados do sinal, foi calculada a série de Fourier, com o mesmo algoritmo utilizado no Arduino que calcula até o décimo primeiro harmônico. Em seguida foi reconstruído o sinal truncado no décimo primeiro harmônico. Isto funciona como um filtro para diminuir o ruído que poderia atrapalhar no processo de calibração. Para o Arduino a etapa de calcular a série de Fourier foi feita internamente no microprocessador, em seguida foi apenas reconstruído o sinal.

Dessa forma, obtemos sinais contínuos, que então foram discretizados, obtendo o mesmo número de amostras. Em seguida, multiplicando o sinal do Arduino por um fator k, esse valor foi sendo alterado para que obtivesse o menor somatório das diferenças quadráticas entre os pontos do sinal analisador e do Arduino. Enfim

[1] Imagem retirada de `http://www.aemc.com/`

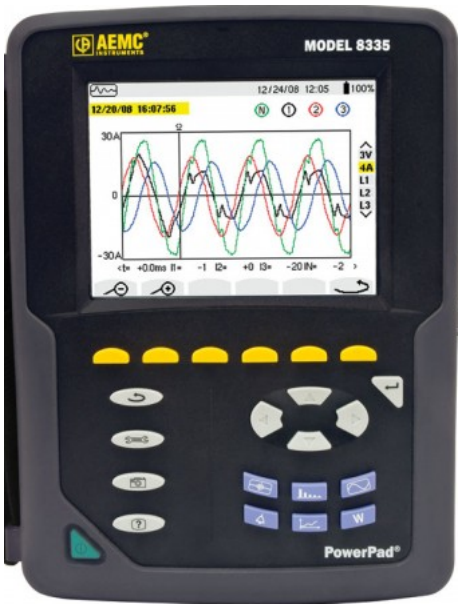


Figura 2.5: Analisador de sinal do LAFAE, usado como referência

chegou-se a um fator com 2 casas decimais de precisão, $k = 1,28$. Para demonstrar o resultado foi aplicado o fator k no sinal do Arduino da Figura 2.6 obtendo o resultado da Figura 2.7

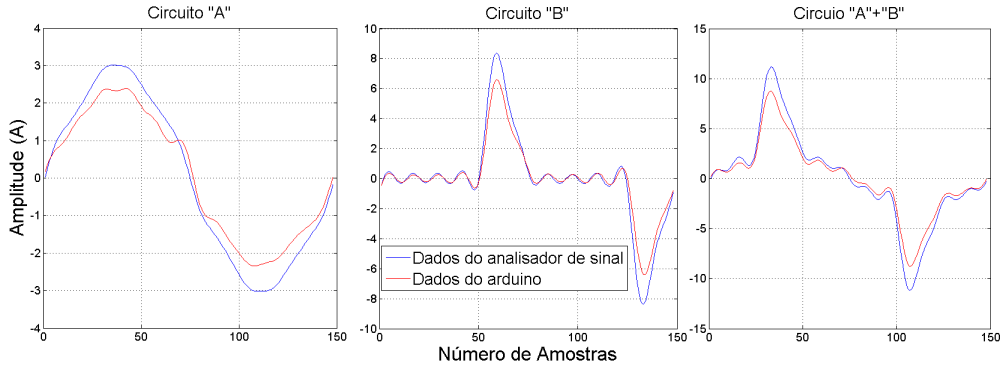


Figura 2.6: Comparação entre os dados medidos com o analisador de sinal e a medição feita com o sensor de corrente com o Arduino

2.4 Android

O aplicativo de Android é responsável pela visualização dos dados obtidos pelo Arduino. Por meio desse aplicativo será possível selecionar o arquivo e os dados que se deseja visualizar, de uma forma simples e clara.

Este programa deverá ser capaz de selecionar o arquivo proveniente do Arduino, a escolha do usuário. Este arquivo contém várias informações sobre o sinal original, mas nem sempre há interesse em analisar todos esses valores, e caso todos fossem mostrados em tela, iria ficar confuso e seria difícil obter a informação desejada.

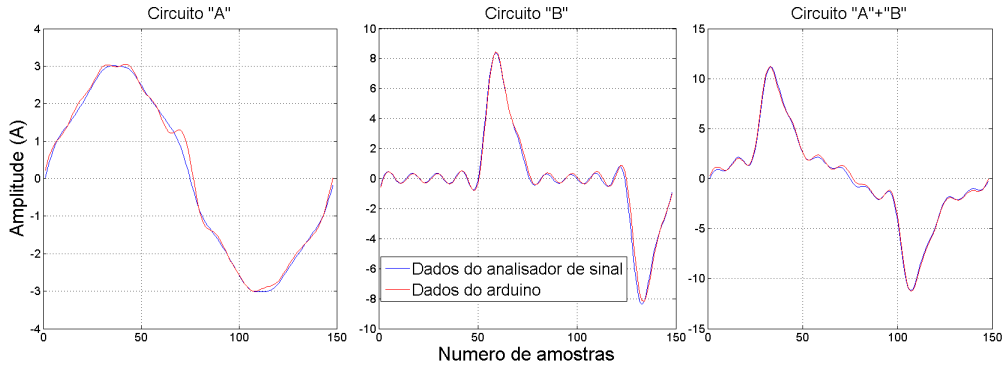


Figura 2.7: Comparação entre os dados medidos com o analisador de sinal e a medição feita com o sensor de corrente com o Arduino calibrado

Para contornar esse possível problema, o aplicativo deverá ter opções de selecionar quais valores se deseja visualizar.

No Capítulo 4, será mostrado o aplicativo completo com todas as suas funcionalidades.

A interface entre Arduino e Android será feita por meio de um cartão SD que será retirado do Arduino e colocado no dispositivo Android. O motivo da escolha dessa interface bem como outras opções serão abordados no Capítulo 4.

Capítulo 3

Método e Ferramentas

3.1 Introdução

Neste capítulo serão apresentados os métodos e algoritmos utilizados para desenvolver o sistema. Também será explorado o ambiente de desenvolvimento utilizado.

3.2 Amostragem

Amostragem é o processo utilizado para obter uma forma discretizada de um sinal contínuo, coletando amostras do sinal original em intervalos de tempo constantes ou não. Abordaremos a amostragem com intervalo constante, e veremos como esse processo interfere nas características do sinal original.

Um sinal amostrado, na teoria, pode estar associado a infinitos sinais contínuos que teriam dado origem a ele. Por exemplo na Figura 3.1, vemos três sinais que, com intervalo de amostragem T, têm a mesma representação discreta. Mas caso o sinal tenha largura de banda limitada, e caso o intervalo de amostragem seja suficientemente pequeno, podemos reconstruir o sinal analógico original perfeitamente [11]. O sinal pode ser recuperado caso a frequência de amostragem seja no mínimo duas vezes maior do que a maior frequência presente no sinal original.

Dessa forma, como é objetivo deste projeto analisar um número de harmônicos definido, devemos ter, teoricamente, a frequência de amostragem como sendo pelo menos o dobro da frequência do maior harmônico de interesse. No caso abordado neste projeto, há interesse em analisar até o décimo primeiro harmônico de um sinal, que tem frequência de $11 * 60Hz = 660Hz$. Como esse valor deve ser metade da frequência de amostragem, então esta não pode ser menor que $1220Hz$.

O arduino utilizado tem velocidade do clock de $16MHz$, e por padrão tem um *prescale*[2] de 128, nessa configuração tem resolução de $10bits$. Na interface do arduino

[2] divisor de frequência para os periféricos internos

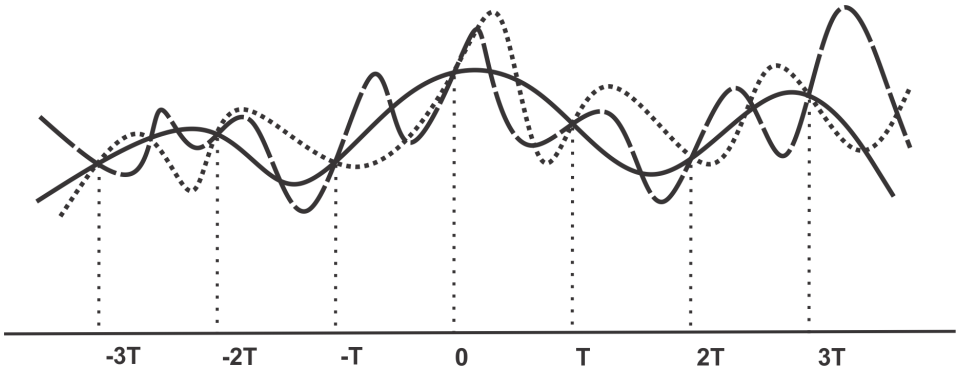


Figura 3.1: Sinais contínuos com mesmo tempo de amostragem T e mesma discretização

as funções estão bastante encapsuladas, levando um total de 13 ciclos de clock para amostrar e converter o sinal, temos por fim uma frequência de amostragem teórica do arduino de $F_a = 16M/(128 * 13) = 9615KHz$.

Na prática foi observado que esse valor teórico é próximo do valor observado, sendo possível obter 148 amostras a cada período de $60Hz$ ou seja, uma frequência de $8880Hz$, que é aproximadamente 4 vezes o valor mínimo aceitável, então não teremos problemas nesse sentido. Caso se esteja interessado em saber mais harmônicos ou analisar sinais com frequência maior, é recomendado diminuir o prescale a custo de resolução, ou então utilizar um microprocessador mais apropriado.

É interessante apontar que essa análise foi feita com base no maior harmônico, mas sabe-se que o sinal original tem inúmeras outras frequências, como por exemplo ruídos em forma de altas frequências. Como temos frequência de amostragem igual a $8880Hz$, as componentes de frequência maiores que a metade disso não serão identificados pelo conversor, sendo recomendável prevenir erros de amostragem (*alias*) empregando na entrada analógica um filtro passa baixa.

3.3 Método de cálculo utilizado

Nesta seção serão apresentados os métodos de cálculo para os algoritmos utilizados para calcular os harmônicos, THD, e RMS da onda original.

Para todos os cálculos foi utilizado um algoritmo de série de Fourier. Para as limitações de hardware existentes, este método foi o que mais se adequava, principalmente no quesito de memória que foi um ponto crítico que será abordado melhor no Capítulo 5.

O Algoritmo 1 foi o algoritmo utilizado para calcular a série de Fourier, apenas foram feitas as modificações necessárias para se adequar a linguagem utilizada no arduino. Esse algoritmo foi testado e comprovado que é válido, um exemplo pode ser visto na Figura 3.2, que mostra um sinal discretizado pelo arduino e esse mesmo

sinal sendo analisado pelo Algoritmo 1.

Algorithm 1: Série de Fourier

Result: Obter a parcela de cada harmônico que compões o sinal.

inicialização;

N = numero de harmônicos;

$N_{amostras}$ = numero de amostras em um período;

$T_{amostras}$ = tempo de amostragem;

$sinal[N_{amostras}]$ = sinal amostrado;

$\omega = 2\pi f$;

$z[N]$ = vetor com o módulo dos harmônicos, resultado do algoritmo;

for *i=0* **to** *(N-1)* **do**

$a = 0$; //parte real

$b = 0$; //parte imaginária

for *j=0* **to** $(N_{amostras} - 1)$ **do**

$angulo = N\omega\ i\ T_{amostra}$;

$a = a + sinal[j+1]\cos(angulo)$;

$b = b + sinal[j+1]\sin(angulo)$;

end

$z[i] = 2(1/N_{amostra})\ \sqrt{a^2 + b^2}$;

$z[0] = z[0]/2$; //valor CC

end

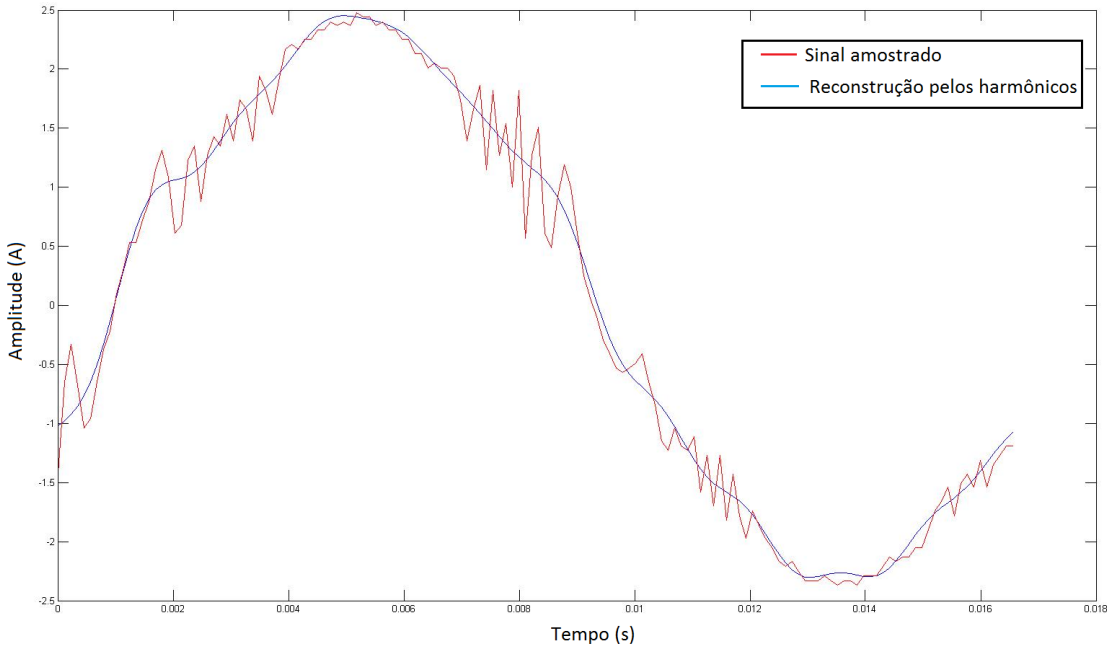


Figura 3.2: Sinal original amostrado e sinal reconstruído pelos harmônicos.

O valor eficaz da onda é uma média quadrática, os cálculos foram feitos utilizando a fórmula da Equação 3.1. Para aproveitar processamento foi utilizado o primeiro loop da série de Fourier para calcular a soma quadrática dos valores, em seguida foi só necessário dividir pelo número de amostras N e fazer a raiz quadrada desse valor.

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2} \tag{3.1}$$

O valor do THD (Total harmonic distortion), ou distorção harmônica total foi calculado de acordo com a Equação 3.2, que foi baseada em [12]. Porém é sempre melhor otimizar ao máximo os cálculos pois o algorítimo será embarcado em um microprocessador, que precisa fazer operações rápidas, por isso esta equação foi simplificada para a da Equação 3.3. Desta forma espera-se um melhor desempenho do arduino.

$$THD = \frac{\sqrt{V^2 - V_1^2}}{V_1} \tag{3.2}$$

$$THD = \sqrt{\left(\frac{V}{V_1}\right)^2 - 1} \tag{3.3}$$

Pelas equações, precisa-se do valor eficaz da onda completa, referenciada apenas por V, cuja forma de cálculo já foi demostrada, e o valor eficaz do primeiro harmônico, referenciado por V_1, que foi calculado após obtermos os coeficientes da série de Fourier. Os valores calculados são valores de pico, então foi necessário apenas dividir o valor do primeiro harmônico encontrado por $\sqrt{2}$, visto que os harmônicos são ondas senoidais puras.

3.4 Ambiente de desenvolvimento

Será abordado neste capítulo qual foi o ambiente de desenvolvimento utilizado para realizar o projeto do Arduino e Android. O computador utilizado para fazer todas as etapas de programação e depuração é um PC com processador Intel Core i7-2670QM e sistema operacional Windows 7 professional de 64bits.

3.4.1 Android

O sistema operacional android é um sistema focado para dispositivos móveis, principalmente smartphones. De acordo com [13], sua participação no mercado em 2014 foi de $84,7\%$ ou seja, é o sistema operacional móvel mais utilizado no mundo. É atualmente desenvolvido pela Google, e o código do sistema está sob a licença de

código aberto.

No início do projeto, o meio recomendado para desenvolvimento na plataforma, era utilizando a IDE eclipse na versão Luna. Na fase final do projeto a Google desenvolveu sua própria plataforma oficial, o chamado Android Studio [14], porém no projeto nada foi alterado pois estava em estágio final.

O ambiente de desenvolvimento por fim, consistiu em utilizar a IDE Eclipse Luna com plug-in para desenvolvimento do android chamado ADT (Android Developer Tools) na versão 23.0.4. A tela inicial pode ser vista na Figura 3.3, esta IDE possui ferramentas de ajuda muito poderosas, como busca por função pai, *refactoring*, encontrar declaração de função dentro do projeto, entre inúmeras outras. A ferramenta da depuração é muito completa, podendo verificar valores de variáveis quando em um *breakpoint*, marca valores que foram alterados de uma instrução para outra, e muitas outras funções que facilitam no desenvolvimento neste ambiente.

Existem duas opções para executar o aplicativo desenvolvido, um é por meio de uma AVD (*Android Virtual Device*) que é uma máquina virtual rodando android no computador. Apesar de ser simples utilização, por se tratar de uma máquina virtual muitas lentidões e travamentos são observadas, por isso optou-se por usar um aparelho real rodando android. A própria IDE faz o papel de instalar o aplicativo no dispositivo, porém também existe a opção de exportar um executável para ser instalado em qualquer aparelho compatível.

Para este programa foram utilizadas duas bibliotecas, uma para seleção de arquivo e outra para fazer os gráficos.

A versão do aplicativo desenvolvida foi feita com API mínima de 14 [3], que corresponde à versão do android 4.0 de codinome Ice Cream Sandwich, e tem como alvo a API 21, que corresponde à versão do android 5.0 de codinome Lollipop.

O dispositivo que foi utilizado para testes foi um Nexus 4 da fabricante LG de código E960 mostrado na Figura 3.4, que no começo do projeto estava rodando uma versão do android liberada diretamente pela Google, na versão 4.4 de code-nome KitKat com API 20, mas ao final do projeto, o dispositivo foi atualizado para a versão 5.0 com API 21, também com atualizações diretas da Google.

3.4.2 Arduino

Arduino é uma plataforma de hardware livre que tem por finalidade prototipagem eletrônica, e utiliza como base o microcontrolador Atmel. Utiliza uma linguagem padrão de programação, que é essencialmente C/C++. O objetivo desta plataforma é prover as pessoas que não têm muito conhecimento técnico um meio de realizar seus projetos, por isso o intuito é ser simples.

[3] Apenas dispositivos com no mínimo esta versão serão capazes de rodar o aplicativo

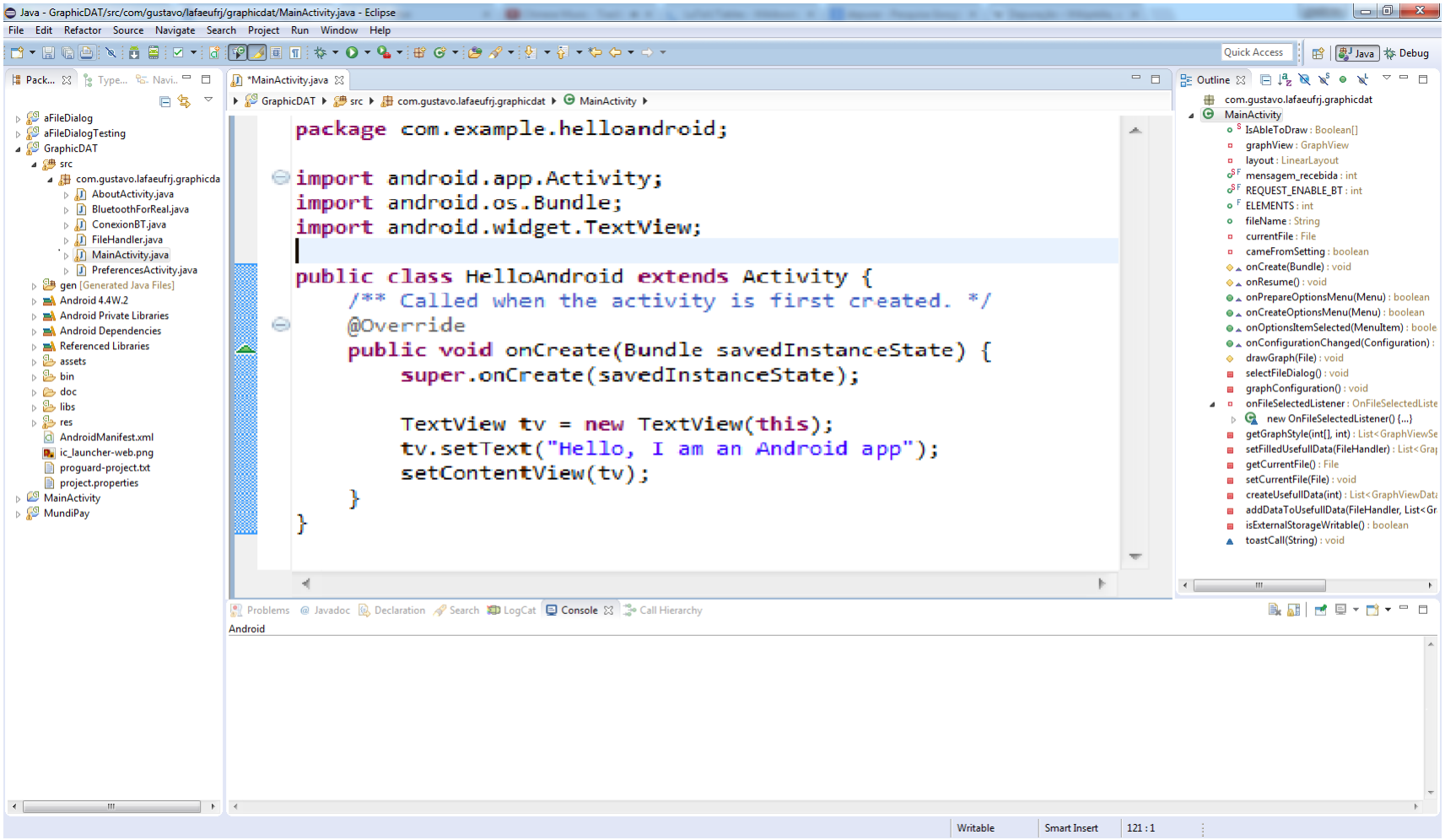


Figura 3.3: Tela inicial da IDE Eclipse versão Luna

Figura 3.4: Dispositivo utilizado para testes do aplicativo

Essencialmente é um microprocessador com interface simplificada e inúmeras bibliotecas e módulos de hardwares (shields) para facilitar o uso.

Existe uma IDE oficial para o Arduino porém foi escolhido utilizar uma modificação desta, chamada de Arduino ERW, na versão 1.0.5 [15], que traz uma série de melhorias, mas a principal delas é a possibilidade de ver a quantidade de memória que está sendo utilizada. Por meio desta funcionalidade pode-se identificar que o programa desenvolvido neste projeto estava consumindo quase que completamente a memória do Arduino que armazena variáveis.

Podemos ver na Figura 3.5 a tela inicial da IDE que foi utilizado para o desenvolvimento no Arduino. É uma interface propositalmente simples, ao final podemos observar a característica dita anteriormente, a utilização de memória. Com sua simplicidade também vem um ponto negativo, a falta de ferramentas de *debug*, o que dificulta a depuração, mas como em geral os programas são simples, este não é um problema tão grande. A única biblioteca utilizada foi a biblioteca nativa de leitura e gravação no cartão SD.

O dispositivo utilizado foi o Arduino UNO R3 da Figura 3.6, que é um dos modelos mais simples. A comunicação com o computador é feita por meio de um cabo USB, a IDE é responsável por compilar e fazer o programa rodar no arduino conectado.

3.5 Sensor de Corrente

O sensor de corrente utilizado foi o ACS712 da Allegro mostrado Figura 3.7. É um sensor que mede tanto correntes contínuas como alternadas. Internamente esse sensor consiste de um circuito de efeito HALL linear. A corrente flui pelo circuito

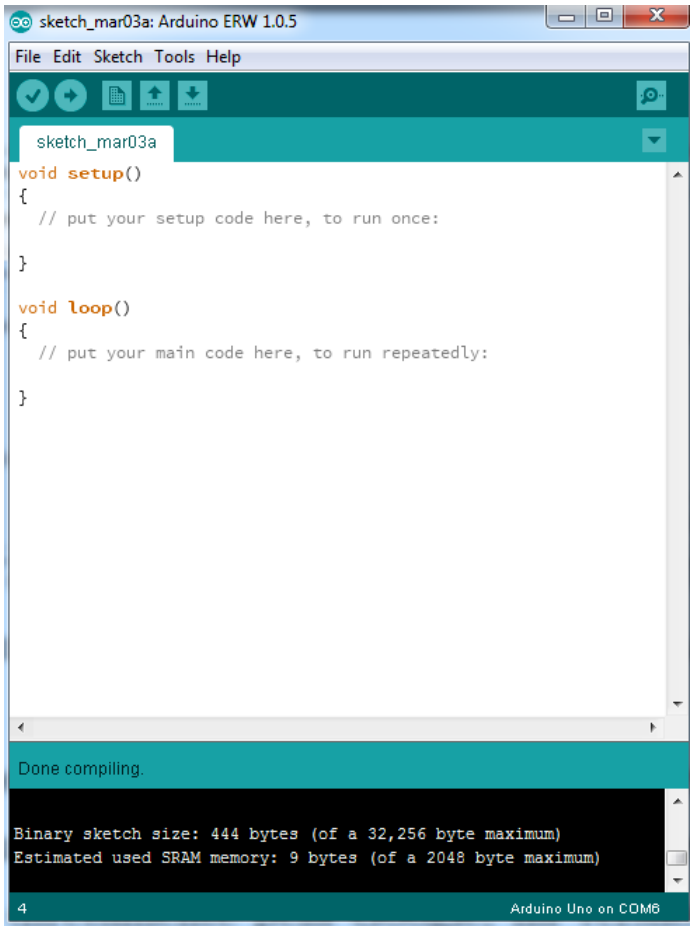


Figura 3.5: Tela inicial do Arduino ERW IDE

gerando um campo magnético que é convertido pelo circuito integrado em uma tensão proporcional.

Algumas de suas principais características de acordo com [16]:

- $5\mu s$ de resposta ao degrau de corrente na entrada;
- largura de banda de $80kHz$;
- Erro total de saída de 1.5% a temperatura de 25^oC;
- $1.2m\Omega$ de resistência interna;
- $2.1kVRMS$ de isolamento;
- Histerese magnética praticamente zero.

A curva característica do sensor pode ser vista na Figura 3.8 [16], pode ser observado como a tensão de saída varia com a corrente e temperatura.

No caso deste sensor, existe um módulo preparado para usar com o arduin. Esse módulo facilita a manipulação e configuração. Dessa forma o sensor possui apenas 3

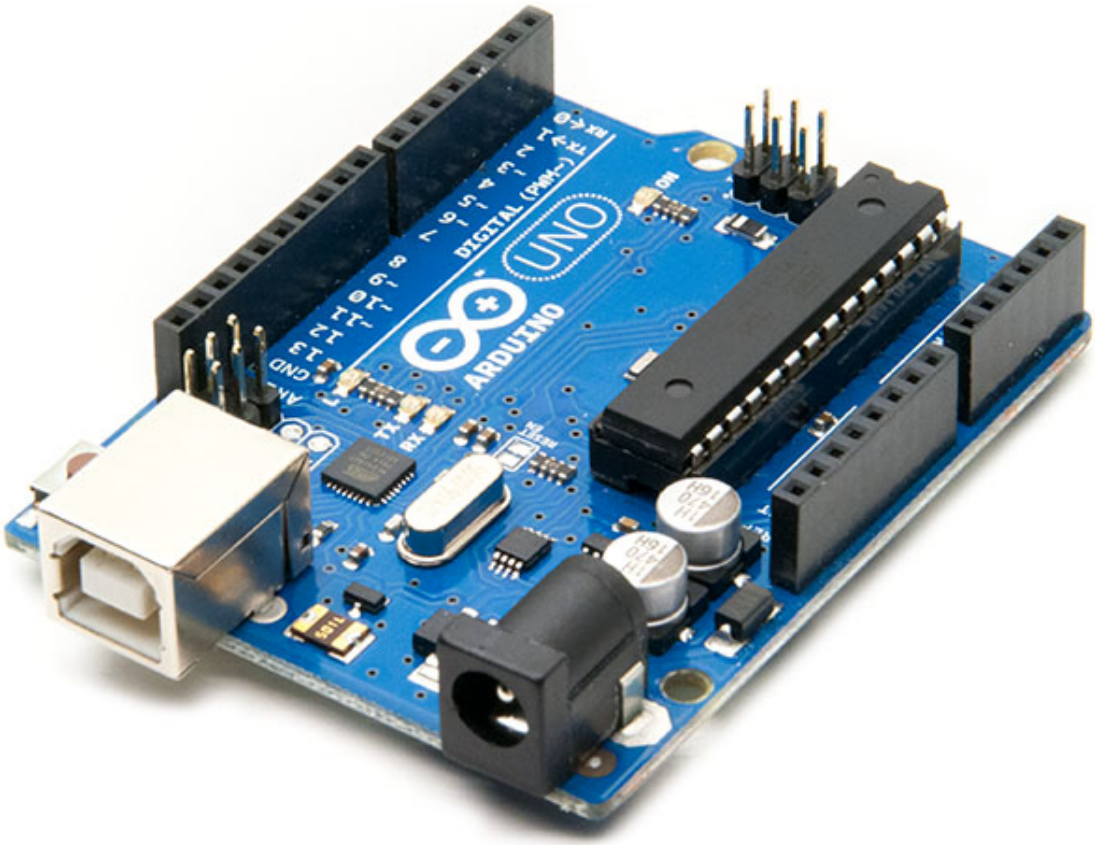


Figura 3.6: Arduino UNO R3

Figura 3.7: Chip ACS712

terminais, os de alimentação de $5,0V$ e GND, que são ligados diretamente nos pinos de alimentação do arduino, e um terminal de sinal, que é onde acontece a leitura, que é ligado ao pino de entrada analógica do arduino. Esse módulo pode ser visto na Figura 3.9.

O que foi utilizado para o projeto foi o módulo feito para o arduino, e por isso, a interface é bem simplificada, não é necessário fazer nenhuma configuração, apenas converter o resultado. No arduino é utilizado um comando de ler uma entrada

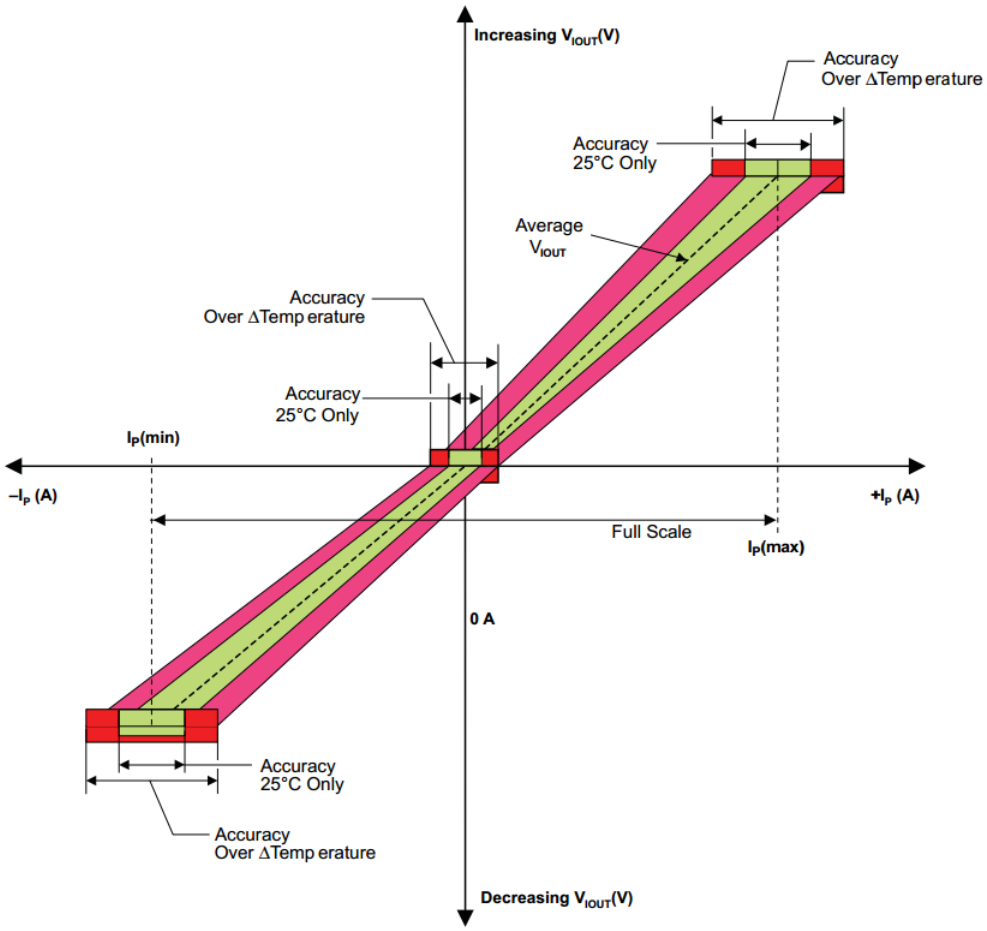


Figura 3.8: Tensão de saída em função da corrente de entrada e a temperatura

analógica, e como sua resolução é de 10bits temos 1024 valores possíveis para a leitura do sensor, ou seja enquanto o sensor varia a leitura de $-20A$ a $20A$ o módulo do arduino retorna entre 0 e 1023.

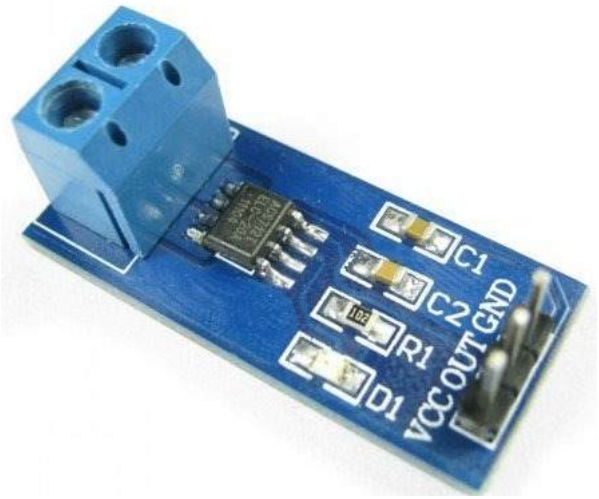


Figura 3.9: Módulo do arduino do sensor de corrente

Capítulo 4

Resultados e Discussões

4.1 Comunicação

Para a comunicação entre o Arduino e o Android foram pesquisados 3 formas mais viáveis para se fazer.

- Via cartão SD

  Consiste em salvar os dados em um cartão de memória SD no arduino, e quando for desejado, retirar esse cartão parando a operação de aquisição, e transferir os arquivos para o dispositivo com o aplicativo Android. Esse método é mais simples de se desenvolver e o usuário fica com um controle maior sobre os dados. A velocidade de transferência de dados nesse caso é a mais rápida.

- Via cabo serial

  Esse método precisa de um hardware específico que é o cabo que liga as duas portas seriais, além disso o dispositivo Android tem que ter suporte para isso e muitos não têm. Como a comunicação seria serial, toda uma lógica de selecionar arquivos abri-los e copia-los teria que ser desenvolvida. E por último a comunicação teria que ter uma forma de prever que aconteça perda de pacotes que é comum, isso tudo faria com que a transferência de arquivo fosse mais lenta.

- Via Bluetooth

  A comunicação bluetooth é uma comunicação sem fio, então tem a vantagem de não ter problema com conectores. Também precisaria de um hardware para o Arduino (Bluetooth shield), com isto enviar dados bluetooth é tão simples quanto escrever em uma saída serial. Como a comunicação é serial a mesma lógica de transferência que da comunicação por cabo teria que ser utilizada, com o agravante que no android não é simples tratar de comunicação bluetooth, e demandaria tempo excessivo.

Tendo-se as três opção supracitadas, e levando-se em consideração o tempo de projeto, a melhor solução encontrada foi fazer a comunicação via cartão SD, pois é uma solução em que um dispositivo não depende do outro, ambos podem funcionar independentes. Outra vantagem é que transferir arquivos pelo cartão é mais rápido do que via serial. Uma desvantagem é que ao retirar o cartão SD do Arduino é possível que aconteça algum tipo de falha que possa vir a corromper o arquivo, para prevenir esse fato foi implementado um botão que ao ser pressionado fecha o arquivo que esteja em uso. E devido ao prazo do projeto a comunicação via cabo e bluetooth se tornariam inviáveis. E como todas as 3 formas precisariam parar de aquisitar os dados ao longo da comunicação, esse ponto não foi relevante.

4.2 Android

O objetivo do programa no android é ter uma maneira fácil de visualizar os arquivos salvos do arduino, ou seja, a entrada do aplicativo é a saída do programa do arduino, logo os arquivos do arduino terão de ser salvos no dispositivo onde o aplicativo está instalado.

O aplicativo é simples de se usar, mas possui todos os requisito. A tela inicial mostra apenas o logotipo do Laboratório de Fontes Alternativas de Energia e um menu no canto superior direito como pode ser visto na Figura 4.1.

Ao pressionar este botão, abre-se um menu para escolher uma da três opções "Browse File", "Settings"e "About"como na Figura 4.2. A seguir uma explicação da funcionalidade de cada botão.

- *Browse File*

  Abre um *dialog* das pastas do dispositivo como na Figura 4.3, para selecionar o arquivo que se deseja visualizar.

- *Settings*

  Abre uma nova tela onde deverá ser selecionado os dados o usuário quer visualizar na tela, mostrado na Figura 4.4.

- *About*

  Esta última opção é apenas a tela que informa sobre as unidades de medida dos valores.

Apesar do nome do aplicativo "GraphicDAT"ele pode ler qualquer arquivo de texto puro, seguindo a mesma estrutura de arquivo salvo do arduino. Independente da orientação do dispositivo o gráfico se adapta, então um exemplo de como os

Figura 4.1: Tela inicial do aplicativo

dados são visualizados nesse aplicativo em modo retrato e paisagem podem ser vistos respectivamente na Figura 4.5 e na Figura 4.6.

É interessante apontar que assim que um gráfico é renderizado, o nome do arquivo aberto aparece no canto direito da barra superior, com uma fonte mais clara. Outro ponto interessante é que é possível dar zoom no eixo x do gráfico para melhorar a visualização. Os valores do eixo y se auto ajustam. É também possível, com um gesto de arrastar o dedo, rolar o gráfico para um lado ou para o outro.

Para fazer o *dialog* de seleção de arquivo, foi utilizada a biblioteca *aFilDialog* [17], tendo sido feitas apenas algumas modificações. A biblioteca *GraphView* [18] foi utilizada para desenhar os gráficos.

Qualquer aplicativo android possui um arquivo executável semelhante ao .exe do windows, mas de extensão .apk. As lojas de aplicativos instalam esse arquivo automaticamente no dispositivo, ou seja, para o aplicativo desenvolvido neste projeto ser utilizado, já que este não está disponível em nenhuma loja de aplicativos, basta transferir o arquivo .apk para o dispositivo e abri-lo com qualquer explorador de arquivo, apenas é preciso que o usuário dê permissão para que aplicativos externos

sejam instalados, isso é feito com uma opção de configuração.

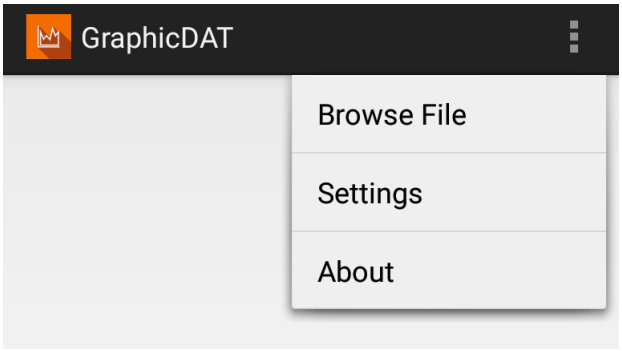


Figura 4.2: Menu principal do aplicativo

Figura 4.3: *Dialog* para selecionar o arquivo

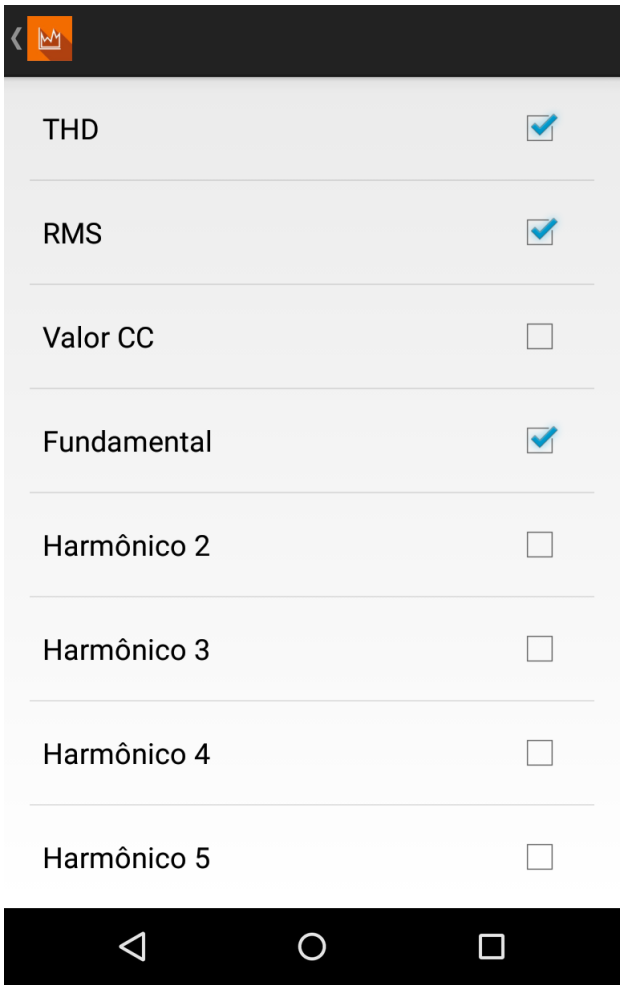


Figura 4.4: Menu para selecionar quais valores o usuário quer que apareça na tela

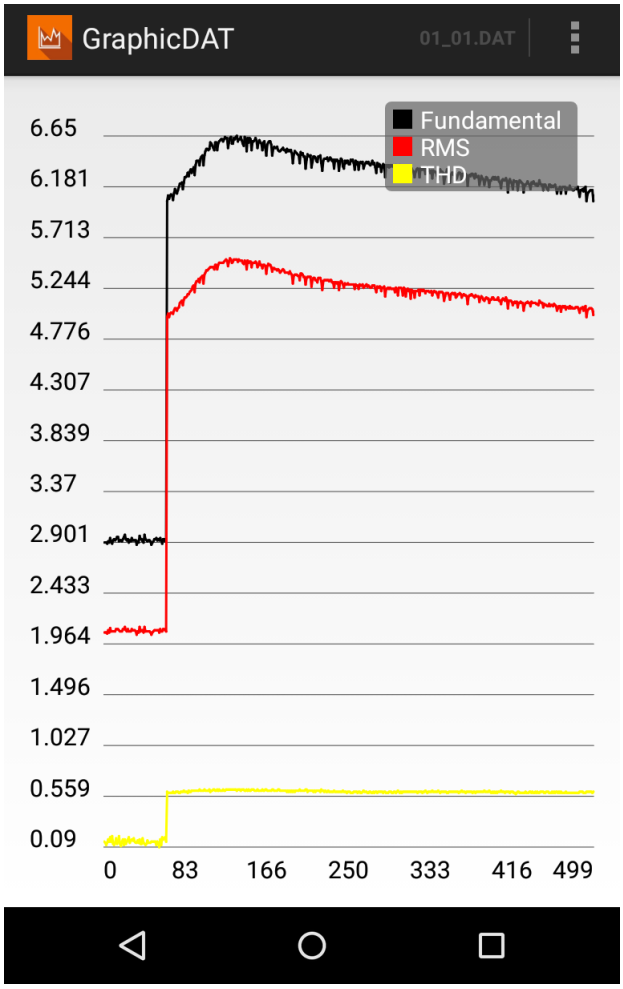


Figura 4.5: Gráfico sendo visualizado no aplicativo em modo retrato

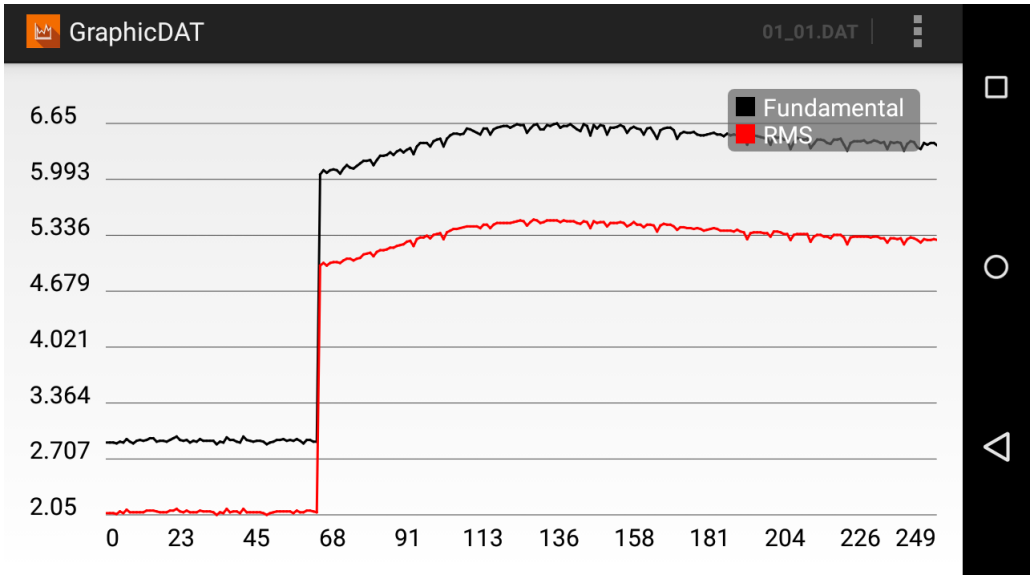


Figura 4.6: Gráfico sendo visualizado no aplicativo em modo paisagem

Capítulo 5

Conclusões e Trabalhos Futuros

5.1 Objetivos Atingidos

A proposta inicial era que o sistema fizesse a medição dos dados, e os apresentasse de uma forma simples, para o usuário final.

Ao longo do projeto esse objetivo não foi esquecido. Para o sistema ser utilizado é necessário apenas que o usuário conecte o sensor de corrente que está acoplado ao Arduino, ao sinal que deseja ser analisado. Para visualizar os dados é preciso que o arquivo de interesse seja transferido para o Android via cartão SD, em seguida abrir esse arquivo com o aplicativo desenvolvido.

Para se utilizar do sistema, é apenas necessário conectar o arduino onde se deseja medir o sinal, e quando for desejado, retirar o cartão SD, para armazenamento de dados, e transferir o arquivo escolhido para um dispositivo móvel que tenha o aplicativo, desenvolvido para este projeto instalado.

Este aplicativo por sua vez tem uma interface bem simples, como mostrado na Seção 4.2. Um resumo de como funciona o encapsulamento para o usuário final, pode ser visto no diagrama da Figura 5.1.

Outro objetivo do sistema era que fosse de fácil adaptação. Caso algo tenha que ser modificado, como a quantidade de harmônicos ou o sensor, essa modificação deverá ser facilitada. O código de todo o projeto foi escrito de maneira legível, para que seja de fácil compreensão.

Tomando um exemplo do arduino, caso queira trocar o sensor de corrente por um de tensão é necessário trocar no código, além da calibragem que é inevitável, apenas uma única instrução que converte os dados do conversor analógico digital, e caso os limites de tensão sejam numericamente iguais aos limites de corrente do projeto original, nem esse modificação é necessária.

Caso estejamos interessados em mais ou menos harmônicos, por exemplo, basta mudar uma variável no programa do arduino, no android só é preciso alterar esse

mesmo valor.

No aplicativo para android também não é difícil modificar algumas coisas, pois foram utilizados *design patterns* [19] que facilitam o entendimento. Também foi gerado um *javadoc* para as funções mais relevantes, este documento consiste em descrever, para as classes, os métodos que as compõem, e os parâmetros e retornos destes métodos.

Figura 5.1: Projeto encapsulado para usuário

5.2 Trabalhos Futuros

Para futuros trabalhos é primeiro recomendado que seja analisada mais a fundo a possibilidade de se fazer uma comunicação bluetooth para servir de mediador entre a passagem dos arquivos do arduino para o android. O projeto inicial já havia previsto que essa possibilidade existia, então o projeto foi desenvolvido com suporte para essa alteração.

Um ponto muito positivo que incentiva esse trabalho é que nada do que foi feito terá que ser modificado, apenas acrescentado. A estrutura do projeto se mantém. A principal diferença será no bloco de comunicação da Figura 5.1, que ao invés de ser feito pelo usuário, tirando o cartão SD do arduino e passando manualmente para o dispositivo móvel, isso passaria a ser feito por meio de comunicação bluetooth.

É possível desenvolver um meio para visualizar o gráfico em tempo real, já que a biblioteca utilizada tem suporte para isso. Com a comunicação bluetooth o dispositivo se conectaria com o arduino, e ao ser selecionada a opção, poderia começar a visualizar os dados desejados em tempo real.

Algo muito interessante de melhorar no projeto é a versão do arduino utilizado, o UNO, para algum outro com mais memória SRAM para armazenamento de variáveis. Por exemplo o arduino Mega que possui 8Kb dessa memória, que é muito superior aos 2kb do arduino UNO.

Dessa forma as otimizações que foram feitas para se adequar ao arduino UNO, ao custo de eficiência no tempo de processamento, podem então tornar o sistema

mais rápido, coletando mais amostras em menos tempo do que o atual.

O ciclo de análise do sistema [4] é relativamente alto para esse tipo de trabalho, atualmente gira em torno de 600 ms para analisar um período de uma onda de 60Hz. Esse tempo poderia ser melhorado com essas otimizações. Outra sugestão para a melhoria desse tempo é utilizar um outro arduino que além da maior capacidade de memória, tenha capacidade de processamento maior que os $16MHz$ do arduino UNO.

Outra alternativa seria usar um computador portátil como o Raspberry pi, que além de realizar as operações mais rápidas, seria bem mais simples de fazer a comunicação, tendo inclusive muito mais possibilidades como, fazer upload automático dos arquivos em um servidor para que seja acessado mais facilmente, nesse caso o aplicativo android também poderia estar conectado com esse servidor para obter os arquivos automaticamente.

Existem também melhorias que podem ser feitas na visualização dos dados além de observá-los em tempo real, por exemplo, é possível dar a opção a usuário de selecionar uma amostra e ser possível observar o histograma de harmônicos, a forma de onda, ou até mesmo qual tempo que foi amostrado.

Seria interessante também selecionar um determinado intervalo e ser capaz de visualizar alguns fatores estatísticos como por exemplo média e variância de um dado específico.

[4] Tempo entre amostrar e converter o sinal, calcular os dados desejados, e salvá-los no cartão SD.

Referências Bibliográficas

[1] HAYKIN, S. S., VEEN, B. V. *Sinais e Sistemas*. BOOKMAN COMPANHIA ED, 2001. ISBN: 9788573077414.

[2] PINTO, . T., GALDINO, M. A. *Manual de Engenharia para Sistemas Fotovoltaicos*. CEPEL, DTE, CRESESB, mar. 2014.

[3] ARDUINO. "Arduino - Home". Disponível em: <`http://arduino.cc/`>. Acessado em : 07-2014.

[4] LIU, F., DUAN, S., LIU, F., et al. "A Variable Step Size INC MPPT Method for PV Systems", *IEEE Transactions on Industrial Electrical*, v. 55, n. 7, 2008.

[5] MUÑOZ, J., LORENZO, E. "On the Specification and Testing of Inverters for Stand-alone PV Systems", *Wiley InterScience*, abr. 2005.

[6] DE SEIXAS, F. J. M., JR., D. P., JR., M. J. A. F. "Impacto da Utilização de Inversores em Sistemas de Geração Distribuída Sobre Equipamentos Rurais", maio 2002.

[7] POMILIO, J. A. "Eletrônica de Potência para Geração, Transmissão e Distribuição de Energia Elétrica". http://www.dsce.fee.unicamp.br/ antenor/pdffiles/it744/CAP4.pdf.

[8] TORELLI, G. D. A., MARTINS, A. M., ALVES, R. M., et al. "Open Source and Low Cost Remote Monitoring Prototype for Grid-Connected Photovoltaic Generation Systems", *Simpósio Brasileiro de Sistemas Elétricos*, maio 2012.

[9] PAASCH, K., NYMAND, M., HAASE, F. "Sensor System for Long-term Recording of Photovoltaic (PV) IVcurves", *kb*, 2013.

[10] ULAGANATHAN, M. K. D., SARAVANAN, C., CHITRANJAN, O. R. "Cost-effective Perturb and Observe MPPT Method using Arduino Microcontroller for a Standalone Photo Voltaic System", *International Journal of Engineering Trends and Technology (IJETT)*, v. 8, n. 1, fev. 2014.

[11] OPPENHEIM, A. V., WILLSKY, A. S. *Signals and Systems*. Prentice Hall PTR, 1996. ISBN: 9780138147570.

[12] SUEMITSU, W. I. "Retificadores Controlados". Slides de Aula, Eletrônica de Potência I, UFRJ, 2013.

[13] (IDC), I. D. C. "Worldwide Smartphone Shipments Edge Past 300 Million Units in the Second Quarter; Android and iOS Devices Account for 96Market, According to IDC". Disponível em: <`http://www.idc.com/getdoc.jsp?containerId=prUS25037214`>. Acessado em : 02-2015.

[14] GOOGLE. "Android Studio". . Disponível em: <`http://developer.android.com/sdk/index.html`>. Acessado em : 02-2015.

[15] ERIED. "Arduino Enhanced Release 1.0.5". Disponível em: <`http://forum.arduino.cc/index.php?topic=118440`>. Acessado em : 07-2014.

[16] *Fully Integrated, Hall Effect-Based Linear Current Sensor IC with 2.1 kVRMS Isolation and a Low-Resistance Current Conductor*. Allegro MicroSystems, LLC. Rev. 15.

[17] JFMDEV. "aFileDialog library". `https://code.google.com/p/afiledialog/`, 2013.

[18] GEHRING, J. "GraphView library". `http://www.android-graphview.org/`, 2014.

[19] GOOGLE. "Patterns — Android Developers". . Disponível em: <`https://developer.android.com/design/patterns/index.html`>. Acessado em : 10-2014.

Apêndice A

Códigos Fontes

A.1 Arduino

Segue abixo o código completo do Arduino.

```
/*
  SD card read/write

 This example shows how to read and write data to and from
   an SD card file
 The circuit:
 * SD card attached to SPI bus as follows:
 ** MOSI - pin 11
 ** MISO - pin 12
 ** CLK - pin 13
 ** CS - pin 4

 created   Nov 2010
 by David A. Mellis
 updated 2 Dec 2010
 by Tom Igoe

 This example code is in the public domain.

 */
#define PI 3.14159265358979323846
#define H 11 //quantidade de harmonicos desejado
#define SIZE 148 //tamanho do vetor em um periodo, depende
   da amostragem
```

```
#define SAMPLE_SIZE (1.0/(60*SIZE)) //INTERVALO DE
   AMOSTRAGEM
#define FUNDAENTAL_FREQUENCY (1.0/(SIZE*SAMPLE_SIZE))
#define ANGULAR_VELOCITY (2*PI*FUNDAENTAL_FREQUENCY)
#define INVSQRT 0.70710678  //1/sqrt(2)

#define IaD 500 //intercoes ao dia
#define DaM 30 //dias no mes



#include <SD.h>

// Define various ADC prescaler
const unsigned char PS_16 = (1 << ADPS2);
const unsigned char PS_32 = (1 << ADPS2) | (1 << ADPS0);
const unsigned char PS_64 = (1 << ADPS2) | (1 << ADPS1);
const unsigned char PS_128 = (1 << ADPS2) | (1 << ADPS1) |
   (1 << ADPS0);

SdFile root; //se this to get files in folder
int readSerial;
File sendFile;

File myFile;
int i;
int N; //indice do harmonico
float a[H+1]={0};
float b[H+1]={0};
float z[H+1]={0};
int sensorIn[SIZE];
float sensorValues[SIZE];
float sensorReal;
float angle;
float RMS;
float THD;
float RMSfundamental;
float calibragem;
```

```
int sensorPin = A0;

boolean isOpen;

long int iteracao=0;
long int day =1;
long int month =1;

String sDay;
String sMonth;
String fileName;
char buf[13];

int initTime;
int total;

void setup()
{

  calibragem=1.28; //obtido do analisador3
  Serial.begin(115200);
  delay(1000); // se nao colocar, no comeco da comunicacao
   aparece uns caracteres decodificados errados.

  // On the Ethernet Shield, CS is pin 4. It's set as an
   output by default.
  // Note that even if it's not used as the CS pin, the
   hardware SS pin
  // (10 on most Arduino boards, 53 on the Mega) must be
   left as an output
  // or the SD library functions will not work.
  pinMode(10, OUTPUT);


  //logica para o botao, para feixa
  pinMode(3,INPUT);
  pinMode(2,OUTPUT);
  digitalWrite(2,HIGH);
```

```
if (!SD.begin(4)) {
  //Serial.println("initialization failed!");
  //return; //vai pro loop
}

//criacao da pastas e arquivos
if(month<10){
  sMonth="0"; //adicionando um zero caso tenha um digito
 para filtrar por ordem
  sMonth+=month;
}
if(day<10){
  sDay="0";
  sDay+=day;
}
sMonth.toCharArray(buf,3);
SD.mkdir(buf);
Serial.println(buf);

fileName=(sMonth+"/"+sMonth+"_"+sDay+".dat");
Serial.println(fileName);
fileName.toCharArray(buf,13);
myFile = SD.open(buf, FILE_WRITE);



if(myFile){
  isOpen=true;
}
else {
  Serial.println("Erros opening testeEscrita.txt");
  isOpen=false;
}

// set up the ADC
ADCSRA &= ~PS_128;  // remove bits set by Arduino library

// you can choose a prescaler from above.
// PS_16, PS_32, PS_64 or PS_128
```

```
  ADCSRA |= PS_128;     // set our own prescaler to 128



}

void loop()
{

   //iniciando variaveis
  RMS=0.0;
  THD=0.0;
  RMSfundamental=0.0;
  memset(&a[0], 0, sizeof(a)); //zera todos os valores do
   arrey;
  memset(&b[0], 0, sizeof(b));
  memset(&z[0], 0, sizeof(z));
  //



  if(!isOpen)


  //logica de criar pastas e arquivos e nomealas
  if(isOpen){ //caso o botao de fechar seja apertado ele
   nao abre mais arquivo

    if(iteracao==IaD){
      iteracao=0;
      day++;
      myFile.close();
      if(day>DaM){ //condicional diferente pq iteracoao
   atualiza no final e dias no comeco
        day=1;
        month++;
        if(month<10){
          sMonth="0";
```

```
            sMonth+=month;
          }
          sMonth.toCharArray(buf,3);
          SD.mkdir(buf);

        }

        if(day<10){
          sDay="0";
          sDay+=day;
        }

        fileName=(sMonth+"/"+sMonth+"_"+sDay+".dat");
        fileName.toCharArray(buf,13);
        myFile = SD.open(buf, FILE_WRITE);
      }
}


//Amostragem
for(i=0;i<=SIZE-1;i++){
  sensorIn[i]=analogRead(sensorPin);
}
//


//calculo de harmonicos
for(N=0;N<=H;N++){

  for(i=0;i<SIZE;i++){
    sensorReal=calibragem*(sensorIn[i]*40.0/1023.0-20.0);
    angle=N*ANGULAR_VELOCITY*i*SAMPLE_SIZE;

    a[N]=a[N]+sensorReal*cos(angle);
    b[N]=b[N]+sensorReal*sin(angle);

    if(N==0){ //apesar de ficar confuso para otimizar foi
 aproveitado o loop, pois eh melhor fazer N*SIZE
 instrucoes de comparacao do que um for com operacoes;
```

```
        RMS=RMS+sensorReal*sensorReal;
      }
    }

    //o valor CC eh dividivo por 2
    if(N==0){

  z[N]=FUNDAENTAL_FREQUENCY*SAMPLE_SIZE*sqrt(a[N]*a[N]+b[N]*b[N]);
    }
    else{

  z[N]=2*FUNDAENTAL_FREQUENCY*SAMPLE_SIZE*sqrt(a[N]*a[N]+b[N]*b[N]);
    }


    myFile.print(z[N]);
    myFile.print(" ");

  }
  //RMS
  RMS=sqrt(RMS/((float) SIZE));

  myFile.print(RMS);
  myFile.print(" ");

  //THD
  RMSfundamental=z[1]*INVSQRT;
  THD=sqrt((RMS*RMS)/(RMSfundamental*RMSfundamental)-1.0);

  myFile.println(THD,3);

  if(digitalRead(3)==HIGH && isOpen){
    myFile.close();
    readSerial=Serial.read();

    while (myFile.available()) {
      Serial.write(myFile.read());
    }
```

```
        myFile.close();
        isOpen=false;
    }

    iteracao++;

}
```

A.2 Android

O código do Android será separado em suas classes, cada subseção é um classe.

A.2.1 MainActivity

```
package com.gustavo.lafaeufrj.graphicdat;


import java.io.BufferedReader;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.Arrays;
import java.util.List;

import android.app.Activity;
import android.app.Dialog;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Configuration;
import android.graphics.Color;
import android.os.Bundle;
import android.os.Environment;
import android.preference.PreferenceManager;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.widget.LinearLayout;
import android.widget.Toast;
import ar.com.daidalos.afiledialog.FileChooserDialog;
```

```
import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.GraphView.GraphViewData;
import com.jjoe64.graphview.GraphView.LegendAlign;
import com.jjoe64.graphview.GraphViewSeries;
import
   com.jjoe64.graphview.GraphViewSeries.GraphViewSeriesStyle;
import com.jjoe64.graphview.GraphViewStyle.GridStyle;
import com.jjoe64.graphview.LineGraphView;

public class MainActivity extends Activity {

        public static Boolean IsAbleToDraw[]=new
   Boolean[14];
        private GraphView graphView;// = new LineGraphView(
   getApplicationContext(), "GraphViewDemo");
        private LinearLayout layout;// = (LinearLayout)
   findViewById(R.id.graphLinearLayout);
        public static final int mensagem_recebida = 2;
     public static final int REQUEST_ENABLE_BT = 1;
     public final int ELEMENTS=14;
     public String fileName="";
     private File currentFile=null;

     private boolean cameFromSetting=false;

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

        }

        @Override
        protected void onResume() {
                super.onResume();
                if(this.cameFromSetting){
                        if(this.fileName!=""){
                                drawGraph(this.currentFile);
```

```
                }
            }
            this.cameFromSetting=false;
        }

        @Override
        public boolean onPrepareOptionsMenu (Menu menu){

menu.findItem(R.id.action_draw).setTitle(this.fileName);
            return true;
        }

        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            getMenuInflater().inflate(R.menu.main,
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();

            switch (id){
            case R.id.action_draw:
                    //Just to show fileName on ActionBar
                    return true;

            case R.id.action_browse:
                    selectFileDialog();
                    return true;
```

```
                case R.id.action_settings:
                        startActivity(new
Intent(this,PreferencesActivity.class));
                        cameFromSetting=true;
                        return true;

                case R.id.action_about:
                        startActivity(new
Intent(this,AboutActivity.class));
                        return true;

                }

                return super.onOptionsItemSelected(item);
        }

        @Override
        public void onConfigurationChanged(Configuration
newConfig) {
                super.onConfigurationChanged(newConfig);

                if(this.fileName!=""){
                        drawGraph(this.currentFile);
                }
        }



        protected void drawGraph(File fileToDraw){
                FileHandler file = new
FileHandler(fileToDraw);

                graphView = new LineGraphView(
getApplicationContext(), " "); //Eh tipo uma Figure do
matalab
                layout = (LinearLayout)
findViewById(R.id.graphLinearLayout);
                try{
                        graphView.removeAllSeries();
```

```
                        layout.removeAllViews();

                        //initializing
                        int i=0;
                        Arrays.fill(IsAbleToDraw,
Boolean.FALSE);
                        String[]
harmonicList={getString(R.string.ValorCC),
getString(R.string.Fundamental),
getString(R.string.Harmonic2),
getString(R.string.Harmonic3),
getString(R.string.Harmonic4),
getString(R.string.Harmonic5),
getString(R.string.Harmonic6),
getString(R.string.Harmonic7),
getString(R.string.Harmonic8),
getString(R.string.Harmonic9),
getString(R.string.Harmonic10),
getString(R.string.Harmonic11), getString(R.string.RMS),
getString(R.string.THD)};
                        int[] curvesColors={Color.BLUE,
Color.BLACK, Color.BLUE, Color.GREEN, Color.BLUE,
Color.DKGRAY, Color.BLUE, Color.CYAN, Color.BLUE,
Color.MAGENTA, Color.BLUE, Color.GRAY, Color.RED,
Color.YELLOW};
                        int thickness=3;
                        SharedPreferences preferences =
PreferenceManager.getDefaultSharedPreferences(this);
                        for(i=0;i<=(ELEMENTS-1);i++){

IsAbleToDraw[i]=preferences.getBoolean(harmonicList[i],
false);
                        }

                        List<GraphViewSeriesStyle>
graphStyle=this.getGraphStyle(curvesColors, thickness);
                        List<GraphViewData[]> usefullData =
this.setFilledUsefullData(file);
```

```
                        // add data to graphview
                        i=0;
                        for(i=0;i<=(ELEMENTS-1);i++){
                                if(IsAbleToDraw[i]){

graphView.addSeries(new GraphViewSeries(harmonicList[i],
graphStyle.get(i), usefullData.get(i)));
                                }
                        }

                        this.graphConfiguration();
                        layout.addView(graphView);



                }
                catch(Exception e){
                        Log.e("drawGraph",e.toString());
                }


        }

        private void selectFileDialog(){

                if(isExternalStorageWritable()){
                        FileChooserDialog dialog = new
FileChooserDialog(this);
                        // Assign listener for the select
event.

dialog.addListener(MainActivity.this.onFileSelectedListener);
                dialog.show();
                }
                else{
                        toastCall("External Storage Not
Avalible");
                        Log.e("External Storage Not
Avalible","but continuing program, error");
```

```
            }
        }

        /**
         * configurations of graphView
         */
        private void graphConfiguration(){

                this.graphView.setScrollable(true);
                this.graphView.setShowLegend(true);

this.graphView.setLegendAlign(LegendAlign.TOP);

this.graphView.getGraphViewStyle().setLegendWidth(240);

this.graphView.getGraphViewStyle().setHorizontalLabelsColor(
Color.BLACK);

this.graphView.getGraphViewStyle().setVerticalLabelsColor(
Color.BLACK);

this.graphView.getGraphViewStyle().setGridStyle(
GridStyle.VERTICAL);
                this.graphView.setScalable(true);

        }

        private FileChooserDialog.OnFileSelectedListener
onFileSelectedListener = new
FileChooserDialog.OnFileSelectedListener() {
                //open file
                @Override
                public void onFileSelected(Dialog source,
File file) {

MainActivity.this.fileName=file.getName();

MainActivity.this.invalidateOptionsMenu(); //refaz o
menu, e quando refaz atualiza o nome
```

```
                        source.hide();

MainActivity.this.setCurrentFile(file);
                        MainActivity.this.drawGraph(
MainActivity.this.getCurrentFile());
                }
                //create file
                @Override
                public void onFileSelected(Dialog source,
File folder, String name) {
                        source.hide();
                        Toast toast =
Toast.makeText(MainActivity.this, "File created: " +
folder.getName() + "/" + name, Toast.LENGTH_LONG);
                        toast.show();
                }
        };


        /*Order of  the data, used for example in "colors"
         * ccGraph
         * fundamentalGraph
         * secondHarmonicGraph
         * thirdHarmonicGraph
         * fourthHarmonicGraph
         * fifthHarmonicGraph
         * sixthHarmonicGraph
         * seventhHarmonicGraph
         * eighthHarmonicGraph
         * ninthHarmonicGraph
         * tenthHarmonicGraph
         * eleventhHarmonicGraph
         * rmsGraph
         * THDGraph
         */

        /**
         * set the graph style, color and thickness
```

```
         * @param colors
         * @param thickness
         * @return graphStyle
         */
        private List<GraphViewSeriesStyle>
getGraphStyle(int[] colors,int thickness){
                int i=0;
                List<GraphViewSeriesStyle> graphStyle =new
ArrayList<GraphViewSeriesStyle>();

                for(i=0;i<=(ELEMENTS-1);i++){
                        graphStyle.add(new
GraphViewSeriesStyle( colors[i],thickness));
                }

                return graphStyle;
        }

        /**
         * same as call createUsefullData +
addDataToUsefullData, return the usedullData with the
real data, ready to use
         * @param file
         * @return UsefullData
         * @throws NumberFormatException
         * @throws IOException
         */
        private List<GraphViewData[]> setFilledUsefullData(
FileHandler file) throws NumberFormatException,
IOException{
                return this.addDataToUsefullData(file,
this.createUsefullData( file.getNumberOfLinesInFile()));
        }

        private File getCurrentFile() {
                return this.currentFile;
        }

        private void setCurrentFile(File currentFile) {
```

```
            this.currentFile = currentFile;
        }

        /**
         * cretae empty usefull data
         * @param NumberOfLinesInFile
         * @return usefullData
         */
        private List<GraphViewData[]> createUsefullData(int
NumberOfLinesInFile){
            int i=0;
            List<GraphViewData[]> usefullData = new
ArrayList <GraphViewData[]>();

            for(i=0;i<=(ELEMENTS-1);i++){
                usefullData.add(new GraphViewData[
NumberOfLinesInFile]);
            }
            return usefullData;
        }
        /**
         * add data to the usufullData
         * @param file
         * @return
         * @throws IOException
         * @throws NumberFormatException
         */
        private List<GraphViewData[]> addDataToUsefullData(
FileHandler file, List<GraphViewData[]> usefullData)
throws NumberFormatException, IOException{
            BufferedReader reader =
file.getFileBufferedReader();
            int j = 0;
            int i = 0;
            String line=null;
            String[] valuesOfLine;

            //adding data to List
            while((line=reader.readLine()) != null){
```

```
                        valuesOfLine=line.split(" ");

                        for(i=0;i<=(ELEMENTS-1);i++){
                                if(IsAbleToDraw[i]){

    usefullData.get(i)[j]=new GraphViewData( j,
    Double.valueOf(valuesOfLine[i]));
                                }
                        }

                        j++;
                }
                reader.close();
                return usefullData;
        }

        private boolean isExternalStorageWritable() {
            String state =
    Environment.getExternalStorageState();
            if (Environment.MEDIA_MOUNTED.equals(state)) {
                return true;
            }
            return false;
        }


        void toastCall(String msg){
                Toast.makeText(getApplicationContext(),
    msg, Toast.LENGTH_LONG).show();
        }

}
```

A.2.2 FileHandler

```
package com.gustavo.lafaeufrj.graphicdat;

import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.File;
```

```
import java.io.FileReader;
import java.io.IOException;

import android.os.Environment;
import android.util.Log;

public class FileHandler {

	private static String pathToGetFile =
   Environment.DIRECTORY_DOWNLOADS;
	private String fileName;
	private int numberOfLinesInFile;
	private BufferedWriter fileBufferedWriter;
	private BufferedReader fileBufferedReader;



	private File pathOfFile;
     private File userFile;

	/**
	 * Nao faz sentido criar objeto fazio, mantendo o
   construtor vazio por boas praticas
	 */
	private FileHandler() {
		// TODO Auto-generated constructor stub
	}

	/**
	 * create FileHanler from 'fileName' in the default
   file path, download directory
	 * @param fileName Name of the file, with extension
	 */
	public FileHandler(String fileName){
		this(fileName, FileHandler.pathToGetFile);

	}



	/**
```

```
        * If file do not exists it will create one, and
Log.i it
        * @param fileName Name of the file, with extension
        * @param pathToGetFile Path of the 'fileName' file
        */
       public FileHandler(String fileName, String
pathToGetFile){
                FileHandler.pathToGetFile=pathToGetFile;
                this.fileName=fileName;
                this.pathOfFile =
Environment.getExternalStoragePublicDirectory(
pathToGetFile);
                this.userFile=new
File(this.pathOfFile,this.fileName);
                if(this.userFile.exists()){
                        Log.i("FileHanler","File '"+
this.pathOfFile+ this.fileName+"' exist");
                }
                else{
                        Log.i("FileHanlder","File '"
+this.pathOfFile+this.fileName+ "' do not exist,
creating one");
                }
       }

       public FileHandler(File userFile){
                this.userFile=userFile;
       }

       /**
        * @return pathToGetFile
        */
       public String getPathToGetFile() {
                return pathToGetFile;
       }

       /**
        * get Buffered Reader from File
        * @return fileBufferedReader
```

```
     * @throws IOException
     */
    public BufferedReader getFileBufferedReader()
throws IOException {
            this.fileBufferedReader=null;
            this.fileBufferedReader= new
BufferedReader(new FileReader(this.userFile));
            String line=null;
            int i=0;

while((line=this.fileBufferedReader.readLine()) != null){
                    i++;
            }
            setNumberOfLinesInFile(i);
            this.fileBufferedReader=null; //como nao
sei se o ponteiro zero no bufferedReadar, refiz por vias
das duvidas
            this.fileBufferedReader=new BufferedReader(
new FileReader(this.userFile));
            return this.fileBufferedReader;
    }

    /**
     * call getFileBufferedReader to know the number of
lines
     * @return numberOFLinesInFile
     */
    public int getNumberOfLinesInFile() {
            if(this.numberOfLinesInFile==0){
                    try {
                            getFileBufferedReader();
                    } catch (IOException e) {
                            // TODO Auto-generated
catch block
                            e.printStackTrace();
                    }
            }
            return this.numberOfLinesInFile;
```

```
	}

	private void setNumberOfLinesInFile(int
numberOfLinesInFile) {
		this.numberOfLinesInFile =
numberOfLinesInFile;
	}

	/**
	 * not implemented yet
	 * @return
	 */
	public BufferedWriter getFileBufferedWriter() {
		return fileBufferedWriter;
	}

	/**
	 * Not implemented yet <br>
	 * if file does not exist it will create a empty one
	 */
	private void createFileBufferedWriter() {

	}

	/**
	 * Not Implemented yet <br>
	 * if file does not exist it will create a empty one
	 */
	public BufferedWriter createFile(){
		this.createFileBufferedWriter();
		return null;
	}

	public void close() throws IOException{
		this.fileBufferedReader.close();
		this.fileBufferedWriter.close();
	}
```

```
}
```

A.2.3 PreferencesActivity

```
package com.gustavo.lafaeufrj.graphicdat;

import android.app.ActionBar;
import android.os.Bundle;
import android.preference.PreferenceActivity;
import android.preference.PreferenceFragment;

public class PreferencesActivity extends PreferenceActivity
   {

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

   getFragmentManager().beginTransaction().replace(android.R.id.content,
   new PreferencesActivityFragment()).commit();

                ActionBar actionBar=getActionBar();
                actionBar.setDisplayShowTitleEnabled(false);
                actionBar.setDisplayHomeAsUpEnabled(true);


        }


        public static class PreferencesActivityFragment
   extends PreferenceFragment
    {
        @Override
        public void onCreate(final Bundle
   savedInstanceState)
        {
            super.onCreate(savedInstanceState);
            addPreferencesFromResource(R.xml.pref_settings);
        }
```

```
    }
}
```

A.2.4 AboutActivity

```
package com.gustavo.lafaeufrj.graphicdat;

import android.app.ActionBar;
import android.app.Activity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;

public class AboutActivity extends Activity {

        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_about);

                ActionBar actionBar=getActionBar();
                actionBar.setDisplayShowTitleEnabled(false);
                actionBar.setDisplayHomeAsUpEnabled(true);
        }
}
```