Programa Institucional de Bolsas de Iniciação Científica – PIBIC
RELATÓRIO FINAL

IDENTIFICAÇÃO DO PROJETO

Título do Projeto: Automação De Sistemas Elétricos De Potência Com Dispositivos Eletrônicos Inteligentes

Local de Realização (Unidade/Instituto/Departamento/Laboratório): Escola de Engenharia / Universidade Federal Fluminense / Departamento de Engenharia Elétrica / Laboratório de Eletrônica

Endereço: Rua Passo da Pátria, 156 – Bloco E, Sala 420A

Bairro: São Domingos Cidade: Niterói UF: RJ CEP:24210-240

DADOS DO ORIENTADOR

Nome: Rainer Zanghi

Matrícula Siape: 2393115 CPF:034205097-40

Endereço: Rua Passo da Pátria, 156 – Bloco D, Sala 421

Bairro: São Domingos Cidade: Niterói

UF: RJ CEP: 24210-240 E-mail: rzanghi@id.uff.br

Telefone 1: (21) 2629-5471 Telefone 2: (_____)

DADOS DO BOLSISTA

Nome: Lucas Abdalla Menezes

Matrícula: 116.038.019 CPF:171.509.847-11 CR:6.9

Curso/Departamento/Instituto: Engenharia Elétrica/ Departamento de Engenharia Elétrica – TEE/ Universidade Federal Fluminense - UFF

Endereço: Rua Santa Rosa 91, Apto. 906

Bairro: Santa Rosa

Cidade:Niterói UF:RJ CEP:24240225

Email:lucasabdalla@id.uff.br Telefone1:(21)971475033

1. TÍTULO

AUTOMAÇÃO DE SISTEMAS ELÉTRICOS DE POTÊNCIA COM DISPOSITIVOS ELETRÔNICOS INTELIGENTES

2. INTRODUÇÃO

Non-Intrusive Load Monitoring (NILM) ou Desagregação de Cargas — do inglês Load Disaggregation —, é o nome dado ao reconhecimento das cargas operantes de um circuito, por meio da análise das curvas de corrente e tensão, que são entregues a ele [1].

As discussões acerca de métodos para a conscientização e redução do desperdício de energia elétrica, são crescentes no contexto atual. Com o foco na área de eficiência energética e estudo de Smart Grids (redes elétricas inteligentes), esse projeto estuda a possibilidade de implementação de tecnologias de baixo custo, que sejam capazes de monitorar a rede de maneira não intrusiva. Esta pesquisa, diferentemente de [2], [3] trata-se de um estudo que foca na capacidade de dispositivos computacionais econômicos (normalmente microcontroladores), de reconhecer e agrupar diferentes sinais presentes na rede.

Como mostrado por [4], os consumidores finais, mostram-se mais sensíveis a mudanças no hábito de consumo, quando conseguem visualizar seus gastos em tempo real, ao invés da maneira tradicional, a cada 30 dias. Por esses motivos, o uso de técnicas de NILM, mostra-se adequado para essa visualização de dados.

Segundo [5], existem diversos algoritmos e abordagens para realizar a desagregação das cargas, e esses podem ser divididos em quatro etapas. A primeira, consiste na detecção dos eventos. Com base nas variações da corrente que alimenta o circuito, pode-se identificar quando uma carga troca de estado Ligado para Desligado, ou vice-versa. Na segunda etapa, após a identificação de um novo evento, o sinal de corrente começa a ser armazenado e avaliado, até o término do evento. Considera-se o fim do evento quando é atingido um novo estado de regime permanente. Na terceira, é feita uma triagem desses eventos; onde aqueles com características similares, enquadram-se em um grupo, e eventos do mesmo grupo, correspondem à mesma carga. Por fim, é mostrado para o consumidor a energia consumida por cada carga em tempo real.

Nesse contexto, na seção de metodologia desse relatório, serão detalhadas as etapas de elaboração do ambiente de testes que será usado para as medições; criação do esquema do circuito para medição da corrente instantânea da rede; desenvolvimento de um código para cálculo de seu valor com disponibilização em rede Ethernet; criação de um servidor que colete esses valores; elaboração de uma base de dados com as amostras capturadas automaticamente pelo servidor e classificadas manualmente; treinamento de uma rede neural para classificação das cargas; e implementação e teste da rede neural em um dispositivo embarcado de baixo custo.

3. METODOLOGIA

Etapa 1:

A primeira etapa do projeto foi destinada ao estudo de bibliografias recomendadas pelo orientador, sobre alguns aspectos iniciais que viriam a ser encontrados durante o desenvolvimento da pesquisa. Foram explorados os seguintes temas em artigos científicos, Teses e trabalhos de conclusão de curso de graduação: Técnicas para medição de corrente de maneira não intrusiva em sistemas de distribuição de energia elétrica e tipos de algoritmos da inteligência computacional para identificação de padrões. Após uma pesquisa feita em portais de publicações científicas (Springer Link, Science Direct, IEEE Xplore e Periódicos Capes), foram encontrados 814 artigos de conferências e revistas sobre a Desagregação de Cargas, publicados desde 1992. Mesmo com esse número expressivo de publicações sobre o tema, pouco foi discutido sobre a implementação de algoritmos e técnicas NILM para sistemas embarcados. Usando essas mesmas plataformas de pesquisa, foram encontrados apenas 117 trabalhos com as palavras chave "NILM" e "embedded" no contexto das Engenharias (filtrando com a palavra chave "-cytology"). Sendo os artigos [1] – [6], os mais relevantes para essa parte inicial do estudo.

Etapa 2:

Nessa etapa, foi iniciado o desenvolvimento de um relatório sobre o uso de sistemas embarcados de baixo custo para o monitoramento não intrusivo de cargas ou NILM (do inglês Non-Intrusive Load Monitoring). Além do estudo de outras bibliografias para investigar os requisitos mínimos de software e hardware do sistema para o tratamento dos dados, de maneira que a proposta de reconhecimento de cargas não fosse afetada.

Etapa 3:

Na terceira etapa, foram estudadas plataformas computacionais de baixo custo disponíveis no mercado e sua aderência aos requisitos mínimos de software e hardware escolhidos na etapa anterior. Foi feito também, o projeto do circuito para medições de corrente, incluindo a seleção de componentes para montagem do circuito.

Etapa 4:

Nessa etapa, foi feita a montagem de um protótipo do circuito projetado, e iniciada a elaboração de um código que possa ser reaproveitado para outros dispositivos que poderão ser testados futuramente. Esse código é responsável por fazer a aferição de corrente de uma malha do circuito, e então faz o cálculo do seu valor RMS para cada instante, dado um certo tempo de amostragem.

Etapa 5:

Essa etapa do projeto foi destinada ao estudo de bibliografias e técnicas de como seria feita a criação da base de dados para o treinamento do algoritmo de classificação [18]-[25]. O estudo tomou como norte, os parâmetros e técnicas de modelagem usadas por algumas bases de dados de código aberto já existentes [25]. O algoritmo de classificação que foi citado, será introduzido no código do ESP32. O algoritmo em questão será utilizado recorrentemente para garantir que o reconhecimento das cargas aconteça periodicamente.

Etapa 6:

Nessa etapa, foi iniciado o desenvolvimento de um aplicativo para celular e uma alternativa para Web, que façam consultas periódicas ao ESP 32 e preparem as informações para o armazenamento na base de dados. As duas implementações foram feitas visando uma fácil interação com o usuário facilitando o processo de captura de valores de corrente. Essas implementações serão responsáveis de gerar arquivos de tabelas permitindo uma melhor integração com diferentes sistemas e ferramentas.

Etapa 7:

Na terceira etapa, foram estudados modelos de classificadores e formas de treinamentos desses para a identificação das cargas testadas. Modelagens como RNN, RCNN e MLP foram estudadas, a fim de selecionar a mais adequada e que respeite as limitações do dispositivo embarcado. Além disso, foram estudadas as formas que os dados seriam passados para o classificador, e a topologia do algoritmo pensando ainda nos recursos computacionais oferecidos pelo microcontrolador.

Etapa 8:

Nessa etapa, foi criado um classificador que é responsável pela identificação de cargas. Esse classificador foi criado utilizando uma biblioteca em Python específica para aprendizado de máquina, Scikit-Learn. A base de dados foi criada analisando seus parâmetros pelo software de mineração de dados e aprendizado de máquina, Weka. E por fim o classificador gerado foi replicado para C++ e então implementado no microcontrolador.

4.RESULTADOS

4.1 Elaboração do ambiente de testes.

Após estudados métodos de extração dos valores de corrente, o primeiro passo foi montar uma esquemática do ambiente de testes. Na Figura 1, está ilustrado um diagrama com os elementos utilizados no ambiente de testes.

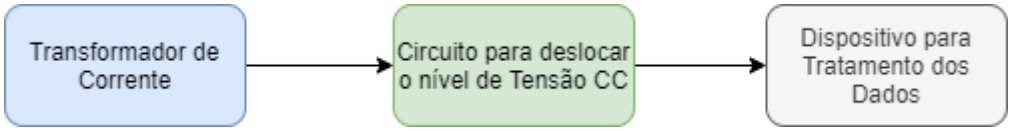

Figura 1: Diagrama do ambiente de testes

O esquema funciona da seguinte forma: Uma corrente atravessa o Transformador de Corrente (TC), que por sua vez, produz uma tensão no seu enrolamento secundário, que é proporcional ao fluxo enlaçado gerado por essa corrente. Essa tensão está contida no intervalo de [-1,1] V. O microcontrolador utilizado suporta apenas valores de tensão positivos entre 0 e +3,3V na sua porta de leitura para o conversor analógico -digital (ADC). Portanto, foi criado um circuito, que fosse capaz de produzir um deslocamento (offset) do nível de tensão CC, que garanta que essa tensão seja estritamente positiva, e dentro destes limites. Nesse caso, considerando a maior variação possível do sensor, ficará entre +0,65 e +2,65 V. Depois disso, foi dado início ao desenvolvimento de um código que fosse capaz de ler a tensão na porta do dispositivo computacional (porta 34, no nosso caso), e que fizesse a conversão desse valor para equivalência em Amperes RMS que atravessa o TC. A descrição dos três elementos será feita nas seções a seguir.

4.1.1 Um sensor de corrente não-intrusivo - transformador de corrente (TC) tipo janela.

O TC tipo janela utilizado no protótipo está ilustrado na *Figura 2*. Com este instrumento, é possível medir a corrente elétrica em um circuito de maneira não-intrusiva, evitando a interrupção do circuito para instalação do medidor.

Figura 2: TC tipo janela não intrusivo (disponível em [13])

O esquema de funcionamento deste TC é ilustrado na Figura 3.

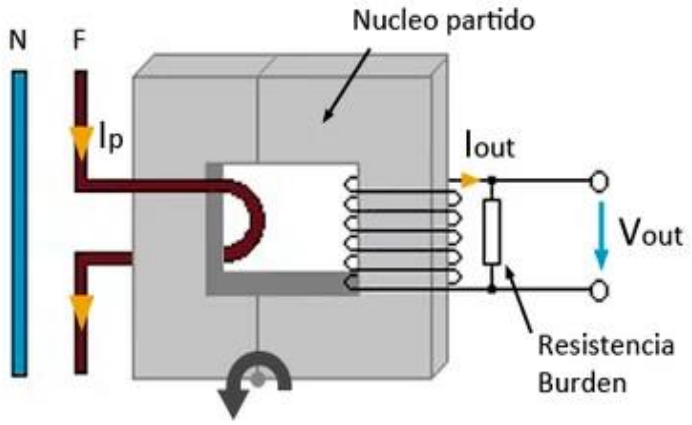

Figura 3: Esquema de funcionamento do TC (disponível em [13])

A relação de transformação do TC, pode ser descrita por (1).

$$\frac{I_r}{I_P} = \frac{V_p}{V_s} = \frac{N_p}{N_s} \qquad (1)$$

Onde,

- I_r é a corrente produzida no enrolamento secundário;
- I_p é a corrente correspondente ao número de condutores que atravessam o núcleo magnético do TC;
- V_P é a tensão no terminal primário;
- V_S é a tensão produzida na Resistência Burden;
- N_P/N_s, representa a relação de transformação;

A tensão de saída fornecida pelo TC, que é proporcional à corrente que o atravessa, é entregue por meio de um conector P2 de 3,5 mm, que é muito comum no campo de áudio [13]. Onde a tensão de saída é dada pela diferença de potencial entregue pelos contatos, "Ponta" e "Capa" do conector ($V_{(3)} - V_{(1)}$), representada pela Figura 4.

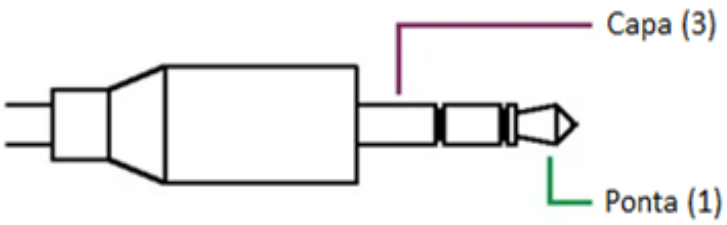

Figura 4: Conector P2 de 3,5 mm (disponível em [13])

4.1.2 Circuito de condicionamento do sinal do sensor – deslocamento de nível CC (Corrente Contínua).

Uma vez que a maioria dos microcontroladores só suporta tensão CC na sua entrada, para a criação do circuito, é necessário saber qual a corrente máxima que o TC é capaz de suportar sem que a sua saturação magnética seja atingida, a fim de tornar o sinal de saída estritamente positivo. Com os dados nominais do TC, temos que para uma corrente máxima de 30A, esse produz uma tensão de 1V. Dessa forma, foi montado um circuito que gera, pelo menos, 1Volt de Offset do sinal de entrada, já que essa seria a tensão máxima que o TC consegue reproduzir.

Seguindo [13] e [14], foi montado o circuito da Figura 5.

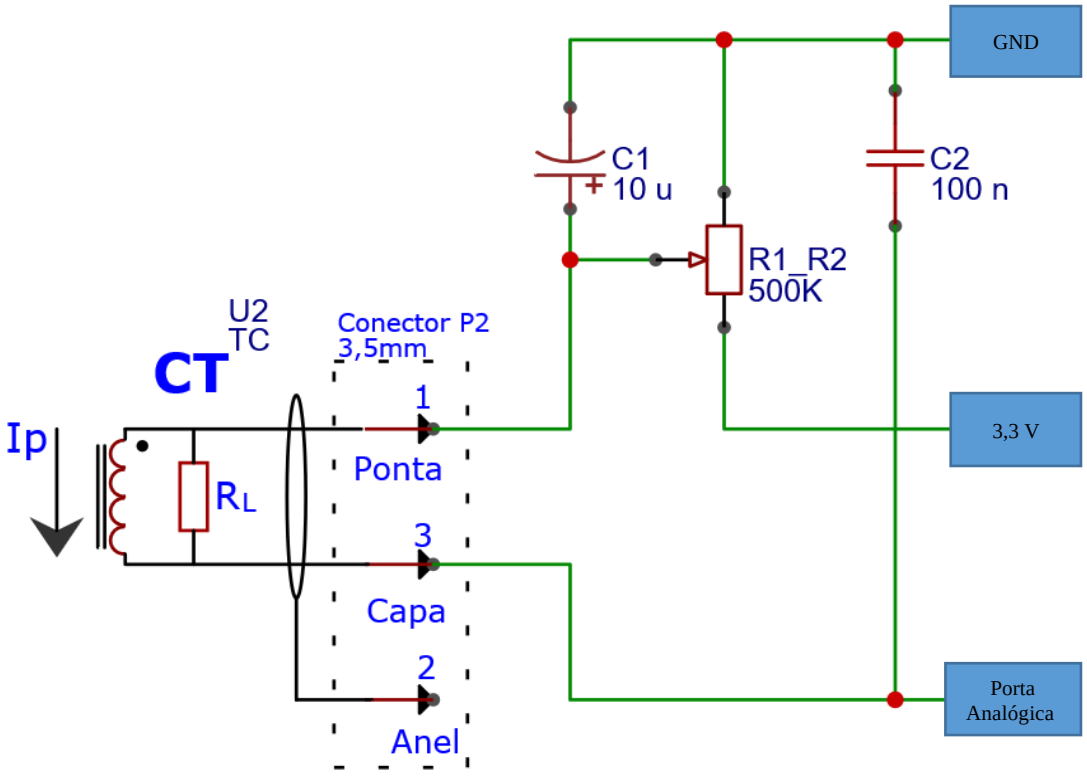

Figura 5: Esquema do circuito para o deslocamento CC do Sinal

O circuito foi alimentado com uma tensão de (3,3 V c.c.) para um offset de 1,65V (3,3 /2 V). A tensão de saída V_tc é exatamente a tensão produzida pelo secundário do TC, deslocada de 1,65 V positivamente. Essa portanto, assume valores de (1,65 – 1) V até (1,65V +1) V, definindo o intervalo entre 0,65 V e 2,65 V. O valor de tensão na entrada do ADC dependerá do valor instantâneo de corrente demandado pela carga.

Os componentes utilizados neste circuito são:

- 1 Trimpot de 500kOhms, separada igualmente entre R1 e R2;
- 1 Capacitor de 10 uF / 50 V para filtrar sinais de alta frequência que chegam no circuito;
- 1 Capacitor de 100 nF / 250 V para filtrar sinais de alta frequência que chegam no dispositivo computacional;
- 1 Conector P2 para receber o sinal de saída do TC;
- 1 Fonte CC de 3,3 V proveniente da própria placa de desenvolvimento do dispositivo;

O circuito projetado foi montado em um protótipo, utilizando uma matriz de contatos (protoboard) e cabos conectorizados (jumpers) para as conexões. A Figura 6 ilustra o protótipo do circuito de condicionamento de sinais montado.

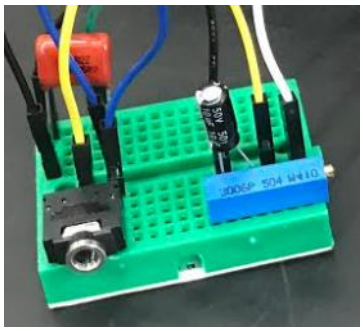

Figura 6: Circuito para o deslocamento CC do Sinal

4.1.3 Uma plataforma computacional de baixo custo que será responsável por receber e tratar o sinal de saída do circuito montado.

Para início da bancada de testes, foi utilizado um ESP32 [15]. pelas suas características um pouco mais robustas que outros sistemas computacionais de baixo custo na mesma faixa de preço como o Arduino UNO [16]. Para as principais diferenças entre os dispositivos, tendo como base o Datasheet dos fabricantes [15] e [16], apresenta-se uma comparação entre as características mais relevantes na Tabela 1.

Tabela 1: Comparação entre Arduino - ESP32

Descrição	Arduino UNO	ESP32
Pinos de I/O	23 pinos com 6PWM	34 pinos com 16 PWM
Memória RAM/SRAM	2 kB	520 kB
Temporizadores	3 timers de 16-bits e dois de 8bits	4 timers de 64-bits
Memória Flash	32 kB	4 MB
Memória ROM/EEPROM	4 kB	448 kB
Interface Ethernet Wi-fi	Não possui	Possui
Frequência de Operação	0 – 16 MHz	80 – 240 MHz
Preço (estimativa em fev. 2020)	R$70,00	R$40,00

Além da capacidade de processamento e memória, outro critério importante na escolha do ESP32 para esta solução foi a verificação nas referências bibliográficas [7]-[10] indicadas pelo professor, da inserção de dispositivos de IoT (Internet of Things) na automação de sistemas elétricos de potência. Segundo a norma IEC61850 [11], padrão utilizado para a automação de sistemas elétricos de potência, todos os dispositivos devem se comunicar através de uma interface Ethernet. Para a plataforma Arduino UNO, a interface Ethernet só é possível através de placas de circuito adicionais (shields).

Uma foto do dispositivo ESP32 utilizado, montado em uma placa de desenvolvimento ESP32 DEVKIT V4 [16] se encontra na Figura 7.

Figura 7: ESP32 DEVKIT V4

Na Figura 8, foram juntadas as três partes abordadas anteriormente, e postas num diagrama.

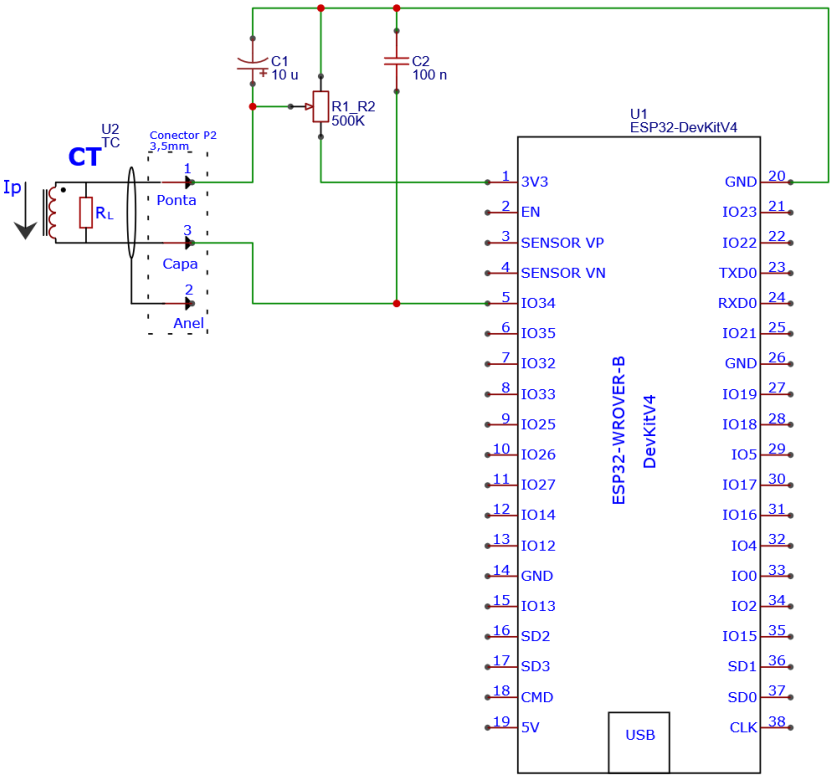

Figura 8: Diagrama do Ambiente de testes completo

Na Figura 9, é mostrada a montagem completa do diagrama descrito na Figura 8.

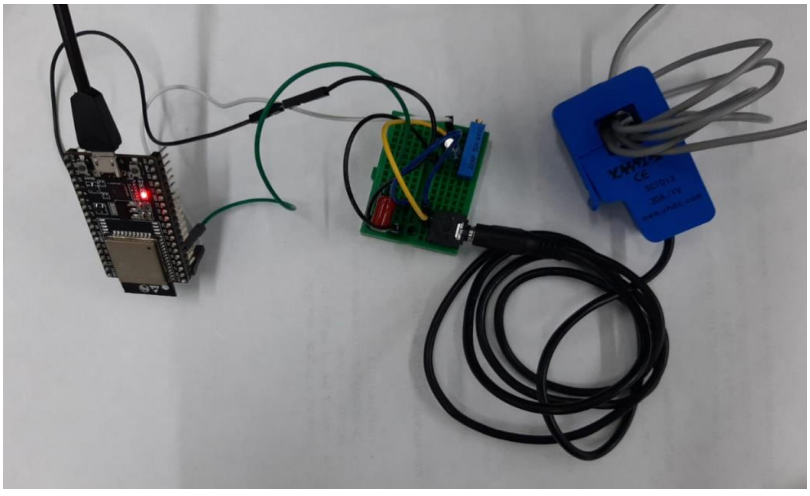

Figura 9 -Montagem do protótipo

4.2 Produção de um código para o ESP32 que faça a leitura da corrente que passa pelo TC.

4.2.1 Idealização do Código

Começando o desenvolvimento do código, o desafio se dá em identificar o quão rápido pode ser a frequência de amostragem, uma vez que o código seja escrito da forma mais otimizada possível.

O ESP32, consegue receber códigos de *Micro-Python* [12] ou *C/C++*, como os demais microcontroladores do mercado. Utilizando *Micro-Python*, foi verificado que esse era muito inconstante e pouco confiável para tempos exatos de amostragem. Um simples código como o descrito no pseudo-código a seguir, gerava valores muitos distintos para cada valor da variável TEMPO_DA_ITERACAO, variando de $[200, 500]\ us$. Isso ocorre, por falta de otimização na conversão de Micro-Python para C e pela ausência de implementação de interrupções para esta plataforma computacional em Micro-Python.

```
VALORES_LIDOS = [];
TEMPO_PERCORRIDO = 0;
TEMPO_DA_ITERACAO;
PARA I de 0 ATÉ 255 FAÇA PASSO 1
      VALORES_LIDOS[I] = LER_PORTA_34();
      TEMPO_DA_ITERACAO = TEMPO_ATUAL – TEMPO_PERCORRIDO;
      ESCREVER(TEMPO_DA_ITERACAO);
      TEMPO_PERCORRIDO = TEMPO_ATUAL;
FIM PARA;
```

O mesmo código anterior escrito em C, ainda que gerando grandes variações, mostrou-se mais eficiente, sempre entregando valores dentro do intervalo de $[50, 200]us$. Para a identificação das cargas, os intervalos entre amostras devem ser os mesmos, para que esses não interfiram no resultado. Dessa forma, foi necessário a utilização de outra ferramenta para aquisição dos dados, além de uma simples iteração usando FOR.

Para isso, foi utilizado um timer/cronômetro do próprio ESP32, o uso de Interrupção e Região Crítica do código. O timer é responsável simplesmente por armazenar o tempo decorrido desde que o dispositivo foi ligado. A interrupção faz com que o processador pause a sua tarefa atual, e a atenda o dispositivo que chamou a interrupção. Já a Região Crítica de um código, garante que apenas aquele trecho de código, seja executada naquele instante, tornando atômica a operação sobre recurso compartilhado. A programação foi feita na IDE Arduino, pois esta IDE possui todas as funções para ESP32, desenvolvidas pela própria ESPRESSIF (fabricante do ESP32), seguras para uso em interrupções.

Então, esse código funcionaria da seguinte forma: O ESP32 é ligado, o timer é disparado, e uma função de Interrupção é associada ao timer, para que a cada $100\ us$ essa função seja chamada. Quando essa é chamada, o código entra numa Região Crítica, e garante que apenas a leitura da porta ligada ao ADC seja executada naquele instante, otimizando o processo de leitura. Depois, que alguns valores fossem armazenados, uma função para o cálculo RMS dos valores obtidos seria chamada, e o timer reiniciado para recomeçar o processo. Além disso, o último valor RMS calculado fica disponível num servidor web, hospedado no próprio ESP32, e que pode ser acessado por protocolo HTTP.

Além disso, os valores gerados pelo conversor A/D do ESP32, variam de 0 a 4095, e esses, correspondem à uma tensão CC que também varia de 0 a 3,3 V. Dessa forma o nosso código, deve fazer essa conversão do valor em contagens fornecido pelo conversor A/D para Amperes. Utilizando a relação do transdutor de 0-30 A para 0-1 V e (1), obtemos (2) para o cálculo da corrente instantânea que atravessa o TC.

$$I_P = \left(\frac{Counts}{Counts_{AD_{CC}}} - 1\right).\left(\frac{I_{MÁX_{TC}}}{n}\right).V_{OFFSET_{CC}} \qquad (2)$$

Onde:

- Counts: É o valor lido pela entrada analógica e convertido pelo conversor A/D;

-Counts_ad_cc: É o valor convertido pelo conversor A/D para a corrente de entrada do TC em 0;

- I_máx_tc: É o valor máximo de corrente medido pelo TC, especificado por [13];

- n: Constante que representa o número de condutores no interior do TC. Este recurso deve ser utilizado somente nos casos onde a corrente máxima medida é muito menor que o valor máximo de corrente do TC;

-V_offset_cc: É o valor de Tensão fornecido pelo circuito de deslocamento de nível de tensão.

4.2.2 Produção do código

A fim de otimizar e permitir o bom funcionamento do código, os seguintes artifícios foram usados:

- Definir constantes com ***#define***, para reservar um espaço em memória que não precisa ser acessado.
- Declaração dos timers que serão utilizados. O timer0 é responsável por pegar as amostras a cada 100 us e armazená-las num vetor de 256 espaços. O timer1 é responsável por resetar o timer0, a cada 500 ms.
- Declaração das variáveis com o modificador ***volatile,*** para evitar otimizações indesejadas do compilador. E uso de **DRAM_ATTR** para garantir que a variável vá para a memória RAM, que tem acesso mais rápido que a Flash.
- Declaração de um semáforo que identifica se no momento está ocorrendo um evento de interrupção. Uso de **uint32_t**, para garantir 32 bits de memória para a variável.
- Associação de uma função ao **timer0** que é chamada a cada 100 us. Associando uma função ao **timer1** para disparar o **timer0**, que é chamada a cada 500 ms. O esquema fica como demonstrado na Figura 10.

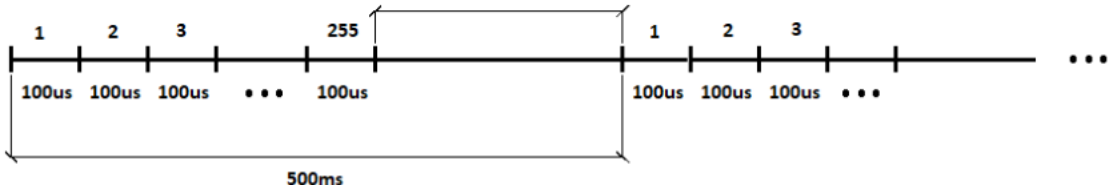

Figura 10: Funcionamento dos Timers para a leitura do sinal

- Criação de uma função que é chamada pela Interrupção, e entra em uma Região Crítica para fazer a leitura da porta 34 do controlador.
- Criação de uma função que é chamada pela Interrupção, a cada 500 ms para resetar o **timer0**.
- Função que retorna o último valor RMS calculado, ao acessar https://IP.DO.DISPOSITIIVO/i_rms.

Em termos de pseudocódigo para a leitura do último valor rms, temos:

```
importar  <WebServer>;

declarar timer0, timer1: hw_timer_t ;
declarar ADC_SAMPLES, I_RMS_VEC_SIZE: INT;
declarar isrCounter: uint32;
declarar i_rms_data: float;
declarar adc_data[]: uint32;
...
//função chamada pelo timer 0
funcao onTimer
    Entrada_na_zona_critica();
    adc_data[isrCounter] = leitura_porta_analogica(34);
    isrCounter++;
    Saida_da_zona_critica();
fim funcao
funcao onTimer1
    resetarTimer(timer0);
fim funcao

funcao calcularRms : float
...
retorna valorRms;
fim funcao
//Primeiro método chamado
funcao Setup
    Conectar.Wifi(login, senha);
    Definir.endpoints();
    AssociarFucoesAosEndpoints();
    Iniciar.servidor();
    Iniciar_timer0();
    Associar_leitura_a_interrupcao_do_timer0();
    Associar_chamada_do_timer0_ao_timer1();
fim Setup
funcao handleAcessoI_rms
    i_rms = calcularRms(buffer_adc_data[]);
    Servidor.enviar(i_rms);
fim funcao
//chamada após o Setup até o desligamento do dispositivo
funcao Loop
    se (timer_ja_disparado())
      entrar_em_zona_critica();
        isrCount = isrCounter++;
      sair_da_zona_critica();
    fim se
    se (isrCount>ADC_SAMPLES-1) E (timer0 ativo)
      parar_timer_0();
      entrar_em_zona_critica();
        isrCounter = 0;
        para (int i = 0; i < numero_maximo_de_amostras; i++)
          buffer_adc_data[i] = adc_data[i];
        fim para
      sair_da_zona_critica();
    fim se
verificar_requisicoes_pro_servidor();
fim Loop
```

Após inserido o código compilado no ESP32, ao acessar http://IP, obtemos a última janela de valores completa usada para o cálculo do valor RMS. Acessando http://IP/i_rms, é obtido o valor RMS, para uma janela de intervalo de $100 * 255\ us$. E esse valor é atualizado a cada $500\ ms$. Além disso, utilizando o endpoint http://IP/i_rms_data, é possível obter os últimos 512 valores, RMS calculados.

Foi produzido então, um código em OCTAVE, por conta da praticidade, para uma primeira visualização dos gráficos, e confirmar se as curvas obtidas condizem com a variação das cargas do circuito.

Para testar o bom funcionamento, o TC foi colocado em uma das fases no quadro geral, monitorando a corrente das cargas ligadas a ele.

Acessando http://IP, foram passados para um gráfico os valores obtidos, ficando como mostrado na Figura 11. Os valores presentes no eixo y representam as contagens fornecidas pelo conversor A/D para uma amostra realizada em laboratório e são mostradas na Figura 11 apenas para fins de ilustração.

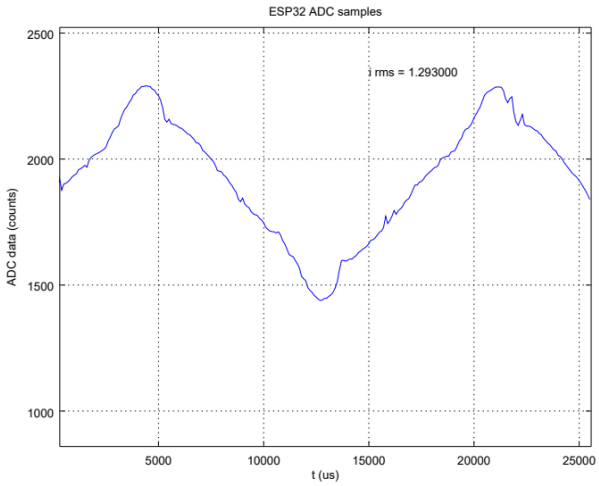

Figura 11: Exemplo de Curva de Corrente

Para testar a mudança de valores de corrente RMS com a variação de cargas residenciais, foram testados um micro-ondas e uma máquina de lavar em um circuito de distribuição monofásico de uma residência.

Primeiro, o endereço http://IP/i_rms_data foi acessado, com as cargas que serão monitoradas desligadas como na Figura 12.

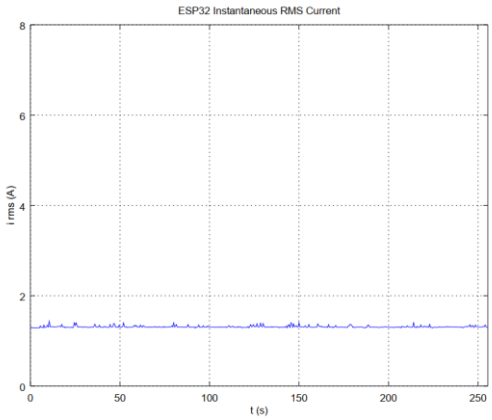

Figura 12: RMS no tempo para Micro-ondas e Máquina de lavar desligados

Logo em seguida foi iniciada a rotina de eventos de teste que teria as seguintes etapas e o tempo em que elas ocorreram:

1- Ligando a Máquina de lavar com o Micro-ondas desligado – t = 0 s;
2- Ligando a Micro-ondas – t = 60 s;
3- Desligando e Ligando o Micro-ondas rapidamente – t = 160 s;
4- Desligando e Ligando o Micro-ondas rapidamente – t = 240 s;
5- Desligando e Ligando o Micro-ondas – t = 50 s;
6- Desligando e Ligando o Micro-ondas – t = 120 s;
7- Desligando o Micro-ondas – t = 170 s;
8- Desligando a Máquina de Lavar – t = 190 s;

Os resultados encontrados entre 1 e 4 estão ilustrados no gráfico à esquerda da Figura 13. No gráfico à direita, da Figura 13, encontram-se os resultados de 5 a 8.

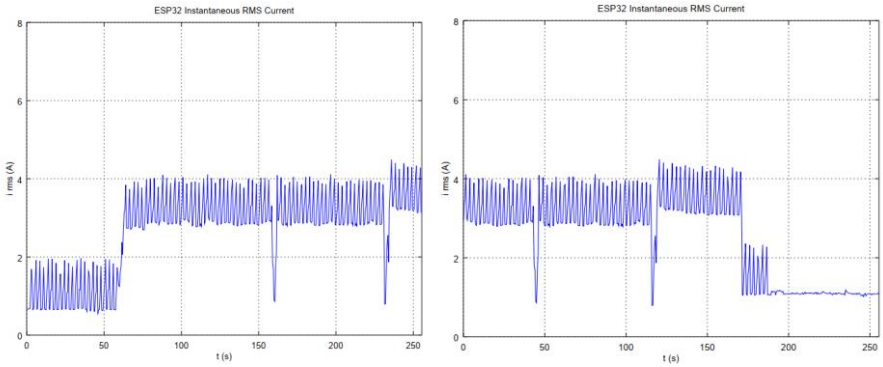

Figura 13: Rotina de testes

4.3 Produção de um código que faça a captura dos valores RMS calculados pelo ESP 32.

Após estudados os métodos de aquisição de dados para desagregação das cargas [18]-[25], foi montado na Figura 14 uma esquemática que relaciona as características extraídas do circuito analisado de acordo com a taxa de amostragem de dados utilizada. Este diagrama mostra algumas abordagens comuns em metodologias para NILM. A utilização do valor RMS da corrente é proposta no presente estudo e não foi encontrada na literatura para medições de baixa frequência.

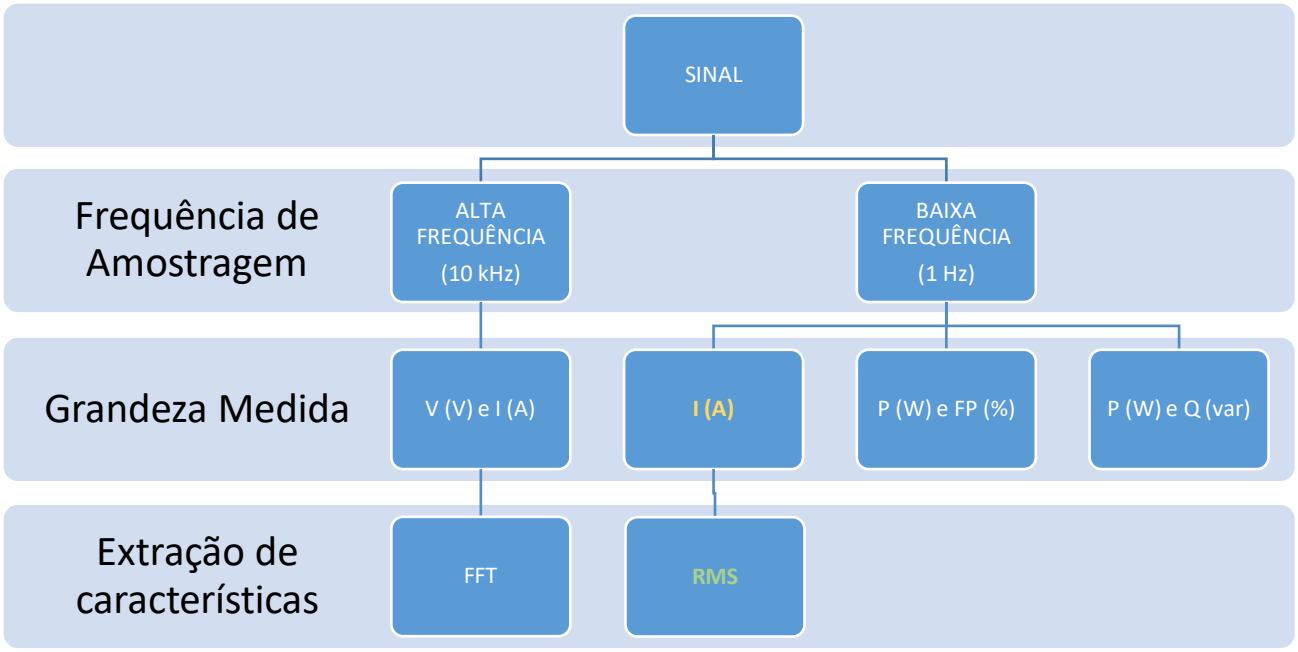

Figura 15: Caraterísticas extraídas do sinal por faixas de taxa de amostragem

A escolha desta abordagem, deu-se por limitações computacionais e por custo. Após calculado o RMS o classificador embarcado será responsável por fazer a identificação da carga. Para fazer a captura das amostras RMS geradas pelo ESP 32, o primeiro passo foi montar uma esquemática do funcionamento do ambiente de captura de valores. Na Figura 16, está ilustrado um diagrama de como o servidor de captura funcionaFigura 17: Diagrama para Captura de Valores RMS.

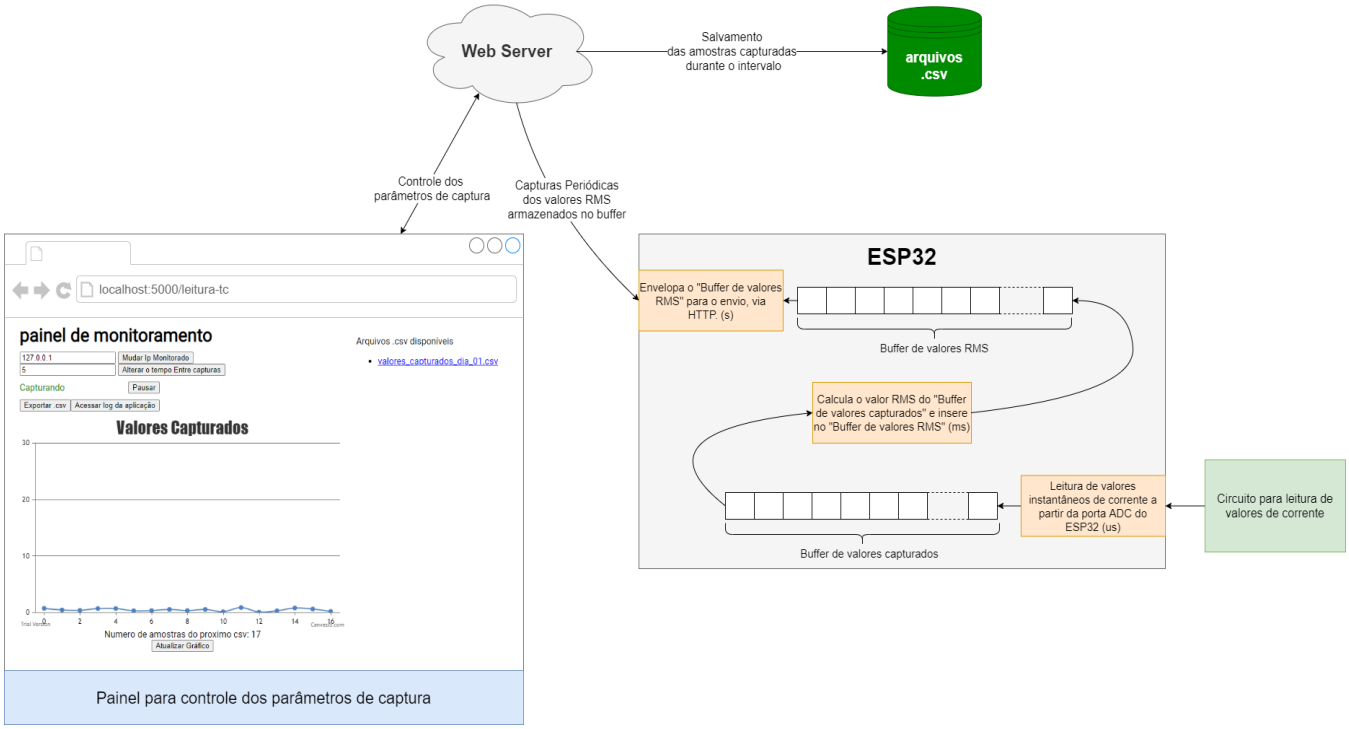

Figura 17: Diagrama para Captura de Valores RMS

Este web server deve ser responsável por obter as amostras calculadas pelo ESP 32 e armazená-las de forma confiável para a criação da base de dados de treinamento do classificador. O esquema funciona da seguinte forma: A cada 100 us o microcontrolador captura um valor instantâneo de corrente que atravessa o Transformador de Corrente. Após a aquisição de 256 amostras, é feito o cálculo do valor RMS de um ciclo completo desse vetor de amostras, considerando um período de 1/60 s. A captura de amostras é então reiniciada. Uma vez calculado o valor RMS para um vetor de amostras, este valor é inserido num vetor que contém os últimos 256 valores RMS calculados e fica disponível em: http://IP_DO_ESP_32/i_rms_data. A partir daí, o Web Server faz requisições periodicamente a esse endereço, recebendo os valores RMS armazenados no buffer e garantindo que não existam amostras faltantes nem duplicadas durante o processo. Esses dados capturados pela aplicação são armazenados em um arquivo .csv a cada 24 horas.

4.3.1 Elaboração do Aplicativo para captura

A primeira tentativa de captura dos dados gerados pelo ESP32, foi por meio da criação de um aplicativo para celular usando a biblioteca React Native, escrita em JavaScript. Na Figura 18: , temos, à esquerda, captura de tela do layout do aplicativo. Após alguns testes, foi verificado que a aplicação conseguia fazer a obtenção dos dados de maneira eficiente. Depois do período desejado, o usuário poderia exportar um arquivo .csv pelo próprio aplicativo, como ilustrado a direita na Figura 18: .

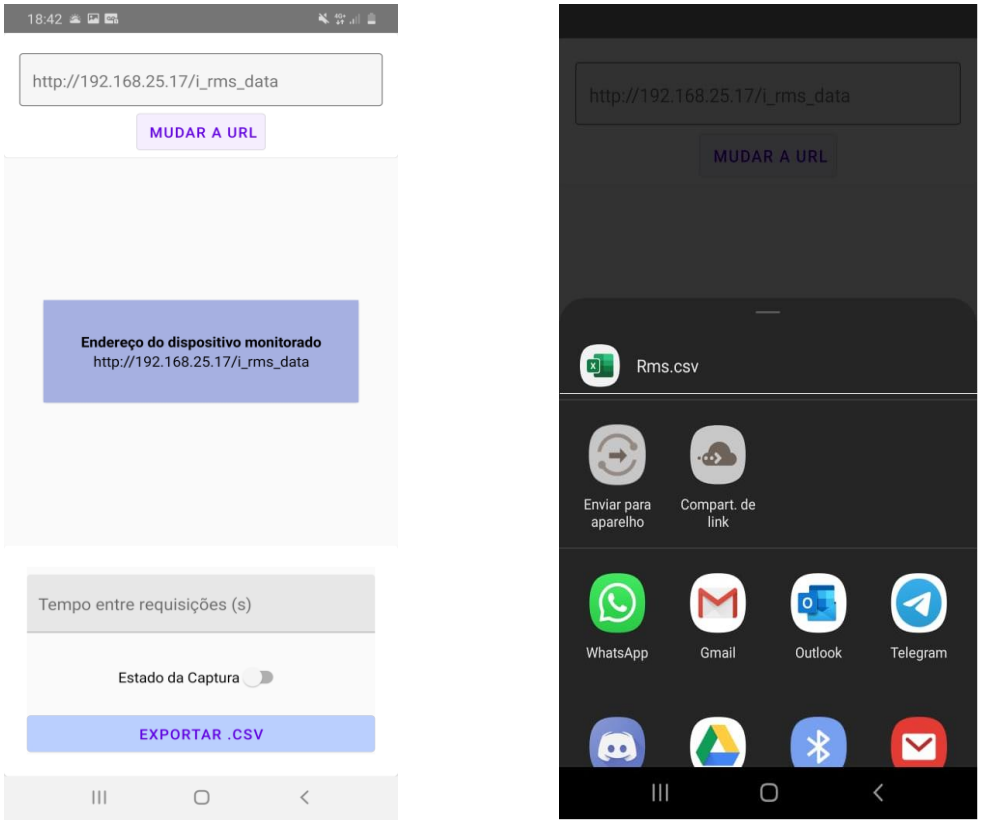

Figura 18: Telas do aplicativo de captura

O problema da utilização de uma solução para dispositivos móveis, se dá por conta de uma função nativa do celular. Quando a tela do celular é bloqueada, todos acessos a rede são “pausados” não ocorrendo de maneira convencional, ocorrendo apenas de tempos em tempos e esses intervalos são chamados de “período de manutenção”. Dessa forma, o uso do aplicativo mostrou-se inviável, mostrando a necessidade de uma solução mais robusta, que permitisse o controle dos intervalos de requisição do vetor de correntes RMS do ESP 32.

4.3.2. Elaboração do Web Server

O Web Server foi criado a fim suprir as deficiências do aplicativo anterior. A nova solução deveria atender os seguintes requisitos:

- Painel de configuração das requisições HTTP
- Possibilidade de realizar chamadas HTTP ao ESP 32 sem interrupções
- Geração de arquivos .csv a cada 24 horas ou após solicitado pelo usuário

Com o intuito de aproveitar a maior parte de código desenvolvido para o aplicativo, foi utilizado o framework Node.Js, que também é feito em JavaScript, para o desenvolvimento do Web Server.

Uma vez definidas as funções do nosso webserver, foram feitas algumas modificações no código do aplicativo para que ele se adequasse ao servidor. O vetor disponibilizado na rede pelo ESP 32 é uma reescrita do vetor circular de valores RMS calculados. Os valores mais recentes são sempre introduzidos ao final da lista, e ao final da lista de valores, é indicada a última posição escrita no vetor circular. Dessa forma, o algoritmo para captura das amostras deve ser capaz de selecionar apenas amostras válidas. Sendo consideradas válidas aquelas com valores não nulos e não repetidas. Assim, é possível garantir a integridade dos dados capturados em cada requisição. Um fluxograma do funcionamento do armazenamento dos dados para uma requisição, está descrito na **Erro! Fonte de referência não encontrada.**. No fluxograma, a variável “delta” é responsável por quantificar o número de índices com valores não repetidos do vetor circular.

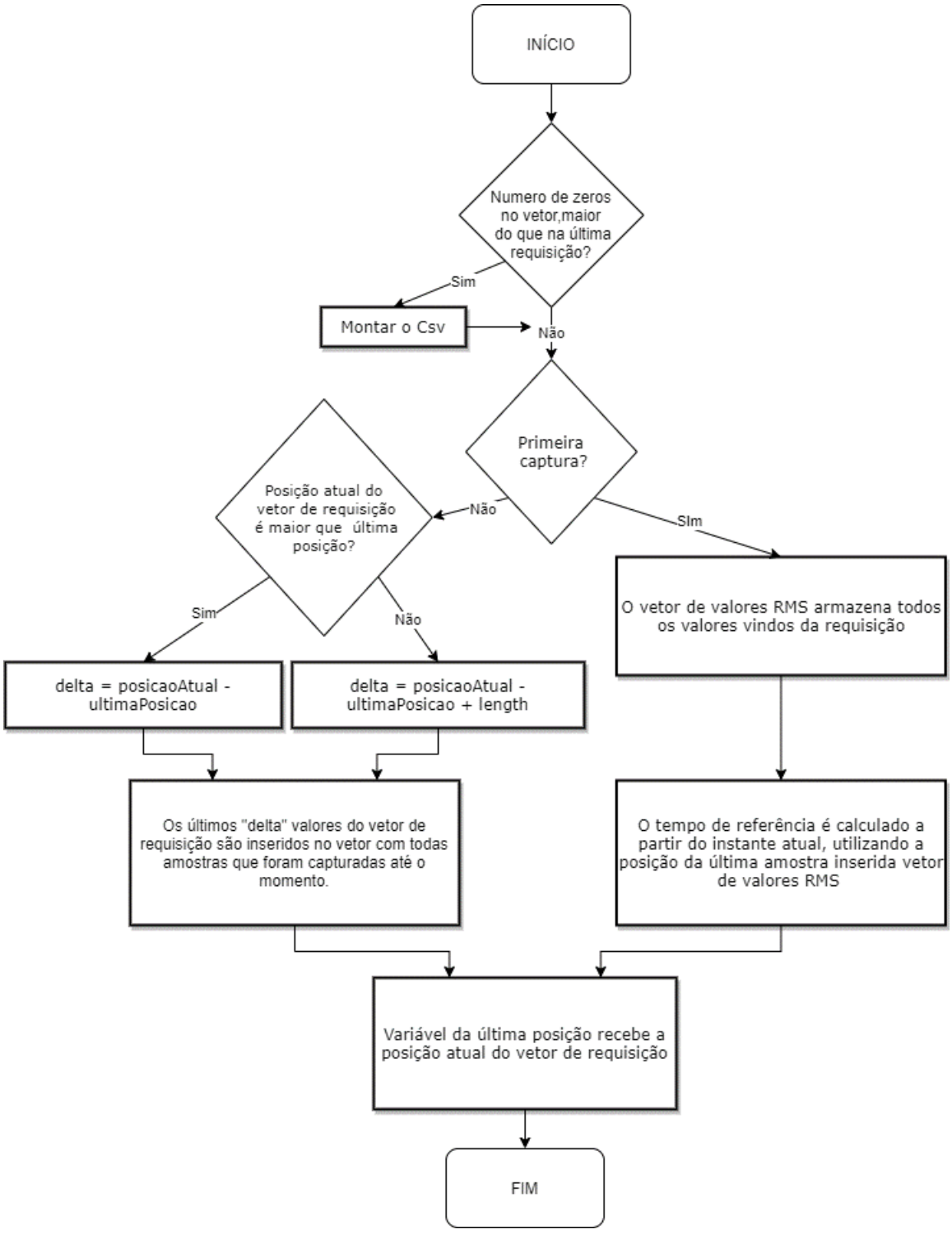

Figura 19: Fluxograma para captura de amostras

Atendendo as funcionalidades citadas em 4.3.2, foi construído um painel de monitoramento do ESP 32 para facilitar interação do usuário. A Figura 20 mostra uma captura de tela da interface criada.

Figura 20: Painel de Monitoramento

Além das funcionalidades já descritas anteriormente, o painel de monitoramento tem um gráfico para mostrar os valores das amostras que entrarão para o arquivo csv. No lado direito da tela fica disponível uma lista de todos arquivos já gerados, que podem ser baixados do servidor clicando no seu respectivo link. Por fim, caso haja qualquer exceção durante a captura dos dados, o log de erros gerados pode ser acessado.

Como mostrado na Figura 21, a configuração de parâmetros que não são alterados com tanta frequência pode ser feita por meio do arquivo **api-config.json** na pasta do servidor.

```
{
    "nodeServerIp": "localhost",
    "PORT": 5000,
    "pathBase": "/leitura-tc",
    "ipParaCaptura": "http://127.0.0.1",
    "apiPort": 80,
    "endPointCaptura": "i_rms_data",
    "tempoEntreCapturas": 5000,
    "caminhoArquivoDeLog": "web-api-out.log",
    "caminhoPastaArquivosRms": "rms"
}
```

Figura 21: Arquivo de configuração do servidor

Ele segue um formato padrão JSON de Chave-Valor, onde cada Chave representa:

- **nodeServerIp**: Endereço de IP onde rodará o Web-Server (por default usar localhost);
- **PORT**: É a porta usada pelo servidor que consumirá os dados do ESP32 (qualquer uma que já não esteja em uso);
- **pathBase**: Endereço base onde deseja acessar o Web-Server (caso não deseje utilizar nenhum caminho base, usar: “/”)
- **ipParaCaptura**: Endereço de IP que o ESP32 ocupa;
- **apiPort**: Porta usada pelo ESP32;
- **endPointCaptura**: EndPoint para captura dos dados no ESP32;
- **tempoEntreCapturas**: Tempo em milissegundos entre as consultas ao ESP32;
- **caminhoArquivoDeLog**: É o nome do arquivo de log (por padrão criará um arquivo no diretório do projeto chamado: “web-api-out.log”);
- **caminhoPastaArquivosRms:** É o nome da pasta que pasta que serão armazenados todos os arquivos .csv gerados.

4.4. Criação de padrão para salvamento dos arquivos csv.

Para um primeiro armazenamento dos valores capturados os arquivos csv são construídos com três colunas. São elas, “ValoresRms”, “TimeStamp” e “EPOCH”. Onde cada linha de registro da tabela tem um Valor RMS calculado e o momento da geração desse valor (TimeStamp). A coluna EPOCH é um valor inteiro que representa, em milissegundos, quanto tempo passou desde 01/01/1970 pelo horário de GreenWich. Na Figura 22: Exemplo de Tabela de valores RMS temos um exemplo de tabela gerada.

```
ValoresRms,TimeStamp,EPOCH
0.707,05/07/2020 18:08:59.754,1593983339754
0.715,05/07/2020 18:09:00.254,1593983340254
0.717,05/07/2020 18:09:00.754,1593983340754
0.703,05/07/2020 18:09:01.254,1593983341254
0.712,05/07/2020 18:09:01.754,1593983341754
0.716,05/07/2020 18:09:02.254,1593983342254
0.721,05/07/2020 18:09:02.754,1593983342754
0.711,05/07/2020 18:09:03.254,1593983343254
0.718,05/07/2020 18:09:03.754,1593983343754
0.712,05/07/2020 18:09:04.254,1593983344254
0.713,05/07/2020 18:09:04.754,1593983344754
0.703,05/07/2020 18:09:05.254,1593983345254
```

Figura 22: Exemplo de Tabela de valores RMS

4.5. Uso de Rede Neural para o Reconhecimento de Carga

4.5.1. Elaboração da Rede Neural

Após a geração dos arquivos csv com os valores RMS de corrente ao longo de dias de captura, o próximo passo foi definir qual tipo de algoritmo seria utilizado para o reconhecimento das cargas.

Para que uma carga seja reconhecida, é necessário que o classificador esteja apto a reconhecer dois tipos de comportamento: acionamento e desacionamento da carga. Portanto, dois eventos distintos. Segundo [6], uma janela de evento ocorre quando um aparelho sai de um estado de regime permanente, entrando em regime transitório, e então, atinge um novo estado de regime permanente.

Como as amostras RMS calculadas pelo ESP descrevem um sinal dinâmico, foram consideradas as seguintes abordagens para a classificação e detecção de eventos: Rede Neural Recursiva (RNN), Rede Neural Convolucional Recursiva (RCNN) e Perceptron Multicamadas (MLP) [18],[19] e [20]. Uma vez que o ESP 32 não tem grande folga computacional e considerando também uma abordagem mais simplificada, foi decidido utilizar o Perceptron Multicamadas.

Para o treinamento e teste de uma rede neural, faz-se necessário a utilização de uma base de dados que represente bem o domínio do problema [21]. Em um processo de aprendizado supervisionado, uma base de dados segue um padrão de tabela onde cada linha contém as informações de entrada e a resposta desejada da rede para essa entrada. Além disso, para se treinar uma rede de forma eficiente e conseguir extrair informações e estatísticas sobre ela, é feita uma divisão da base de dados entre: base de treinamento e base de testes.

4.5.2. Uso de Bases de Dados existentes

Atualmente já existem bases de dados bem desenvolvidas e abertas para uso na internet. Após uma pesquisa sobre elas foi verificado que as mais utilizadas são as BLUED (Building-Level fUlly labeled Electricity Disaggregation dataset) e REDD (The Reference Energy Disaggregation Data Set)[19] e [25]. As duas bases de dados são mais focadas para amostragens de alta frequência. Considerando que o cálculo do valor RMS por parte do ESP 32 ocorre a cada 0,5 segundos, torna a utilização dessas bases prontas inviável e complexa.

4.5.3. Definição de parâmetros utilizados

Uma rede neural necessita de parâmetros de entrada para que ela faça o processamento dos dados e gere uma resposta de saída para aquela entrada. Como o ESP 32 deve identificar em tempo real se houve ou não o acionamento ou desacionamento dos aparelhos “conhecidos”, a nossa rede ao ser treinada deve simular esse comportamento de input de dados e percorrer um processo de classificação do sinal a cada iteração do loop.

Então, foi feito um estudo bibliográfico de como seria a melhor forma de apresentar os valores de entrada para a rede e quais seriam os parâmetros [5], [6], [21], [22] e [23]. Os dados de entrada da rede devem descrever as variações de comportamento do sinal de corrente. Para um sistema monovariável —como o valor RMS de corrente gerado pelo ESP 32— algumas assinaturas e comportamentos do sinal podem ser extraídos e parametrizados a partir de suas curvas.

Considerando que no presente estudo deve-se observar as limitações de memória e de processamento, torna-se importante portanto, que o processamento da rede não seja muito custoso. Isso porque o tempo de processamento da rede é limitado e deve durar no máximo 400ms (dos 500ms do timer, 100ms são utilizados para obtenção de amostras). Alguns exemplos de parâmetros facilmente extraíveis são: média do sinal, variação do desvio padrão do sinal, valor máximo, valor mínimo, diferenciais entre as amostras, entre outras características.

Entretanto, extrair manualmente as características desejadas de um sinal torna-se preocupante por um principal motivo: as características escolhidas podem não ser tão essenciais para o aprendizado da rede fazendo com que existam resultados tendenciosos e que não representem a realidade.

Dessa forma, assim como feito em [5] e [6], foi decidido utilizar uma janela de tamanho **L** que contenha os últimos **L** valores RMS calculados. Todos esses valores servirão de entrada para a nossa rede. Ao fazermos isso, delegamos à nossa rede a necessidade de extração de características para o reconhecimento da carga. Uma vez que temos uma base de dados densa e uma rede suficiente grande e bem estruturada para que os neurônios não sejam saturados durante o aprendizado, busca-se que as características "escolhidas" pela rede convirjam para as características ótimas.

4.5.4. Criação da base de dados própria

Para a organização da base de dados, foi adotada a seguinte abordagem:
É utilizada uma janela de tamanho fixo que contém os últimos **L** valores calculados. Essa janela tem o formato de fila PEPS (Primeiro a Entrar Primeiro a Sair). A cada nova amostra, a mais antiga é descartada e todas as outras são deslocadas para a de um índice abaixo. Então, a nova amostra é inserida na última posição da fila.

Definido como será a estrutura da rede, é necessário agora fazer a criação da base de dados. Como citado no item **4.5.**1, cada linha da base deve conter a entrada e resposta da rede. Portanto, cada linha da base é algo como:

$$x_1, x_2, x_3, \dots, x_L, y \qquad (3)$$

Onde x_i representa os valores de entrada e y a saída esperada. Para início da etapa de testes foi assumido um valor arbitrário de **L** de cinco amostras. Além disso foi adicionada uma nova coluna chamada ClasseAparelho que representa os acionamentos e desacionamentos dos aparelhos escolhidos. Esses são feitos em pares. Caso existam dois aparelhos para serem detectados, o acionamento do aparelho 1 tem o acionamento e desacionamento identificados pelos índices 1 e 2. Já o aparelho o acionamento do aparelho 2 tem o índice 3, e o seu desacionamento, índice 4. O índice 0 fica dedicado para quando nenhum aparelho que deve ser "conhecido" pela rede é ligado ou desligado.

Devido à pandemia de COVID-19, o circuito completo do protótipo foi instalado na casa do orientador do projeto. Para fazer a identificação dos eventos de acionamento e desacionamento do microondas, foi combinado com todos da casa do professor que toda vez que o micro-ondas fosse ligado ou desligado (que é a nossa carga de estudo) esse evento seria registrado numa tabela informando a hora do dia em que ficou ligado. Dessa forma seria possível sinalizar os eventos no arquivo .csv criado pelo Web Server, permitindo o "janelamento" do arquivo para criação da base. A partir daí, o próximo passo foi extrair as janelas que existem nos csvs criados pelo Web Server.

Então, foi criado um script em Python para fazer o "janelamento" dos arquivos csv gerados, pelo Web Server. Um exemplo de arquivo antes e depois do janelamento, é mostrado mostrados nas Figuras 23 e 24 respectivamente.

```
ValoresRms,TimeStamp,EPOCH,ClasseAparelho
0.752,15/07/2020 13:06:27.522,1594829187522,0
0.751,15/07/2020 13:06:28.022,1594829188022,0
0.752,15/07/2020 13:06:28.522,1594829188522,0
0.761,15/07/2020 13:06:29.022,1594829189022,0
1.410,15/07/2020 13:06:29.522,1594829189522,1
1.403,15/07/2020 13:06:31.522,1594829191522,1
1.971,15/07/2020 13:06:32.022,1594829192022,1
2.857,15/07/2020 13:06:32.522,1594829192522,0
2.845,15/07/2020 13:06:33.022,1594829193022,0
2.867,15/07/2020 13:06:33.523,1594829193523,0
2.854,15/07/2020 13:06:34.023,1594829194023,0
```

Figura 25: Arquivo csv gerado pelo Web Server

```
x1,x2,x3,x4,x5,ClasseAparelho
0.752,0.751,0.752,0.761,1.410,1
0.751,0.752,0.761,1.410,1.403,1
0.752,0.761,1.410,1.403,1.971,1
0.761,1.410,1.403,1.971,2.857,1
1.410,1.403,1.971,2.857,2.845,1
1.403,1.971,2.857,2.845,2.867,1
1.971,2.857,2.845,2.867,2.854,1
```

Figura 26: Janelamento do arquivo csv

Uma vez feita a marcação do início dos eventos em cada arquivo csv gerado pelo Web Server, e passados esses arquivos um a um no script para janelamento, os novos arquivos janelados estão prontos para entrada na Rede Neural.

4.5.5. Análise das amostras de treinamento utilizando Weka

O Weka [26] é um Software de código aberto (*open source)* para Aprendizado de Máquina (*Machine Learning*) e é amplamente usado para pesquisas e aplicações industriais. Ele possui ferramentas gráficas para visualização de informações estatísticas sobre base de dados. E também, diversos classificadores e técnicas de aprendizado com interface gráfica. Neste projeto, este software não foi utilizado para modelagem do classificador, apenas para visualização de estatísticas descritivas dos dados da base.

Durante os testes para criação da rede, diversas versões da base de dados foram geradas, utilizadas e testadas no treinamento da rede. A Figura 27 ilustra a visualização da base de dados no software Weka, mostrando informações da última base utilizada para treinamento da rede.

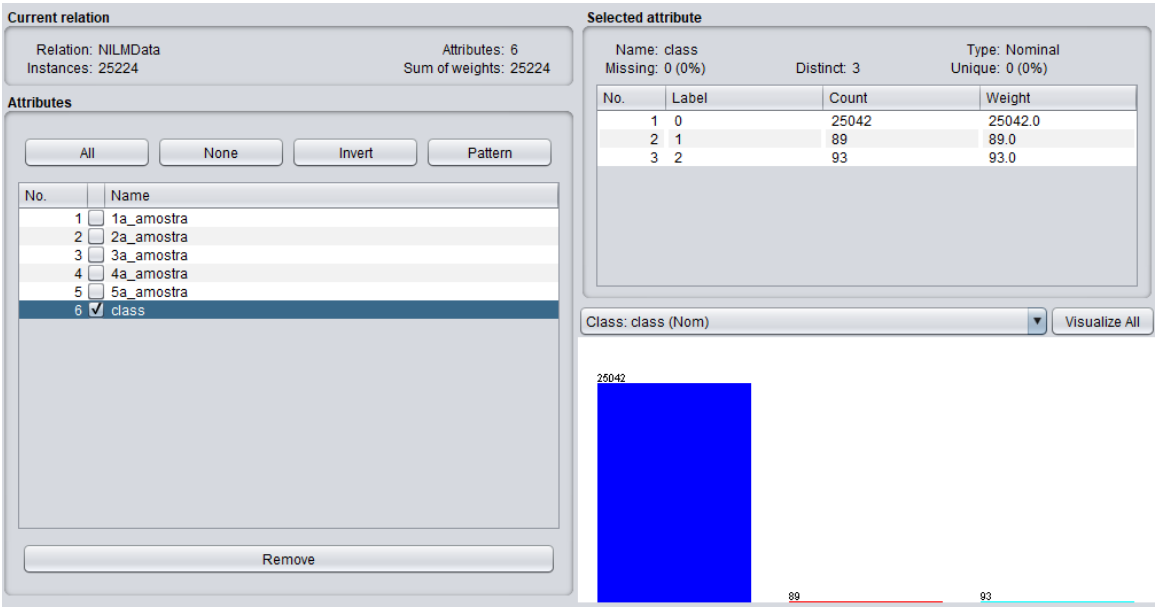

Figura 28: Proporção de amostras de treinamento

Para representar de maneira eficiente o domínio, foram escolhidas 25042 janelas onde nenhum evento ocorre, 89 janelas com eventos de acionamento da carga e 93 janelas com eventos de desacionamento da carga. Para fins de teste, a carga escolhida para monitoramento, foi um micro-ondas. Esta escolha se deu por limitações impostas pelo distanciamento social adotado durante a epidemia de COVID-19.

4.5.6. MLP usando Scikit-Learn

Para facilitar a modelagem e testes de topologias da Rede Neural, foi utilizada a biblioteca Scikit-Learn para Python. Essa, é uma ferramenta simples e eficiente para análise de dados e criação de modelos preditivos [27].

Como já discutido anteriormente, foram testadas diversas topologias de Rede Neural e o Scikit-Learn foi essencial. A simplicidade no processo de implementação de modelos e de personalização de características da rede, permitiram agilidade nas mudanças de topologia e maior facilidade para fazer alterações e correções na base de dados.

Como dito na seção **4.5.1.** foi utilizada uma rede do tipo MLP Perceptron Multicamadas (Perceptron Multicamadas). O MLP é um algoritmo de aprendizado supervisionado, que aprende uma função $f(.): R^m \rightarrow R^o$ treinando um conjunto de dados, onde $\boldsymbol{m}$ é o número de parâmetros de entrada (no presente caso 5), e $\boldsymbol{o}$ representa a dimensão das possíveis saídas do sistema. Neste projeto, testando apenas para o micro-ondas, existem três possíveis classificações: sem evento, ligando ou desligando. Estas três classificações foram representadas numericamente por 0, 1 e 2, respectivamente. A Figura 29 mostra uma topologia clássica de MLP. Pela documentação da biblioteca, podemos ver os parâmetros que podem ser alterados na rede e como é estruturada a rede padrão para esse classificador [28].

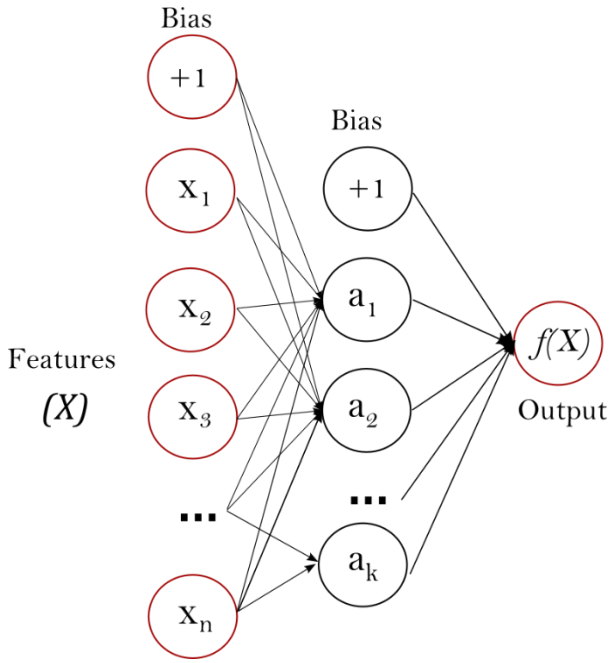

Figura 30: Topologia clássica MLP

Como descrito em [28], seja um conjunto de treinamento $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ onde $x_i \in \mathbb{R}^n$ e $y_i \in \{0,1\}$, um MLP de apenas uma camada escondida contendo apenas um neurônio. Esse neurônio utiliza a função $f(x) = W_2 g(W_1^T x + b_1) + b_2$ para aprendizado. Sendo $W_1 \in \mathbb{R}^m$ e $W_2, b_1, b_2 \in \mathbb{R}$ parâmetros do modelo. W_1 e W_2 representam o peso da camada de entrada e de saída respectivamente; e b_1 e b_2 representam seus vieses (*bias*).

Para as funções de ativação, todos os neurônios, exceto aqueles da camada de saída, utilizam a ReLU (*rectified linear unit*). Essa função retorna o máximo entre 0 e o valor de entrada passado. Para a função de ativação de saída, como esta pode assumir valores 0, 1 e 2, fez-se necessário a utilização da função SoftMax (4).

$$softmax(z)_i = \frac{e^{z_i}}{\sum_{l=1}^{k} e^{z_l}} \qquad (4)$$

Onde i representa o elemento de entrada da SoftMax, que corresponde a classe i (0,1 e 2 no nosso caso) e k é o número de classes. O resultado é um vetor com a probabilidades da amostra x pertencer a cada classe.

Foram analisadas diversas topologias de rede. A rede com a topologia de 5 neurônios de entrada, duas camadas escondidas com 8 neurônios cada e uma camada de saída com 3 neurônios (5, 8, 8, 3), apresentou melhores resultados na etapa de validação do treinamento. A rede MLP usa o método do SGD (Gradiente Descendente Estocástico) para o seu treinamento.

Para fazer as tomadas de decisão definindo se algum dos parâmetros da rede seria alterado, ou até mesmo a base de dados, foram utilizadas métricas estatísticas. Durante o treinamento da rede, a base de dados é separada em duas partes. A primeira, o conjunto de treinamento, fica responsável pelo treinamento da rede e fazer com que ela aprenda os padrões e comportamentos do que será classificado. A segunda, o conjunto de testes, fica responsável por fazer testes da rede já treinada com o primeiro conjunto. Uma vez concluída a rotina de testes, é possível extrair informações essenciais para identificar se a rede está com o comportamento adequado.

A principal "ferramenta" que foi utilizada para decidir as alterações que seriam feitas na rede, foi a matriz de confusão. Essa matriz permite visualizar de maneira objetiva como a rede está classificando os eventos. As linhas da matriz correspondem a predição feita pela rede, e as colunas aos valores verdadeiros. A Tabela 2 representa a matriz confusão da última versão da base de dados utilizada para a classificação.

Tabela 2: Matriz Confusão utilizando a base de dados de teste do classificador

		Predição		
		Classe 0	Classe 1	Classe 2
Esperado	Classe 0	496	0	0
	Classe 1	3	29	0
	Classe 2	0	0	12

De acordo com os resultados apresentados na Tabela 2, podemos concluir que: a Classe 0 apresentou 496 Verdadeiros Positivos e 0 Falsos Positivos, uma vez que não foi classificada nenhuma como Classe 1 ou 2. A Classe 1 apresentou 29 Verdadeiros Positivos e 3 falsos positivos. Já a Classe 2 teve 12 Verdadeiros Positivos e 0 Falsos negativos. Na Tabela 3 está descrita a acurácia detalhada por classe.

Tabela 3: Acurácia detalhada por classe

Razão VP	Razão FP	Precisão	Sensibilidade	Classe
1,000	0,000	1,000	1,000	0
0,906	0,103	0,906	0,906	1
1,000	0,000	1,000	1,000	2

Pelos resultados apresentados, percebe-se que a rede consegue identificar de maneira satisfatória quando nenhum evento ocorre. Além disso, se confundiu algumas vezes com eventos de ligamento, errando a classificação pelos de Classe 0. Os eventos de desligamento também estão sendo bem reconhecidos.

4.6. Implementação da rede no ESP 32

Após o treinamento da rede utilizando o Scikit-Learn, foram extraídos os valores dos pesos entre os neurônios e dos vieses da rede. Além disso, a função de ativação de cada neurônio é conhecida. Dessa forma, tendo mapeada a topologia da rede, é possível replicá-la.

Antes de fazer a implementação diretamente no ESP 32, como a linguagem que é compilada é C++, a fim de assegurar o comportamento adequado da rede, foi feito um código de testes. Os resultados obtidos pela rede recriada feita em C++ foram comparados com os resultados obtidos da rede gerada pelo Scikit-Learn. E então, o código do programa escrito em C++ foi adaptado para o ESP 32. O pseudo-código da implementação está descrito a seguir.

```
importar <WebServer>;
...
declarar tamanhoJanela 5;
declarar biasCamadas[][];
declarar pesosCamadas[][];
declarar vetorNeurôniosCamadas[][][];
declarar entradaRede [tamanhoJanela];
declarar vetorEventosDetectados [256];
...
//Primeiro método chamado
funcao Setup
...
funcao aplicarReLU(): float;
...
funcao aplicarSoftMax(): int;
...
funcao propagarEntradaParaProximaCamada(entrada): float
      para (i = 0; i <tamanhoEntrada; i++)
            para (j=0; j<tamanhoSaida; j++)
                  saida [j] = entrada[i]*pesos[i][j]
            fim para
      fim para

      para (i = 0; i < tamanhoSaida; i++)
            saida [i] = saída[i] + biasCamada[i];
      fim para
      retornar saida
fim funcao
...
//Chamada após o Setup até o desligamento do dispositivo
funcao Loop
    ...
    //Prepara o vetor de entrada da rede neural
    //com tamanhoJanela amostras
    entradaRede = preparaEntradaDoClassificador();

    //Inicia a classificação após ter uma janela de amostras completa
    se (rmsCalculados >= tamanhoJanela)
      resposta = entradaRede;
      para (int i = 0; i < numeroDeCamadasInternas; i++)
            //Utiliza ReLU como função de ativação
            resposta = propagarEntradaParaProximaCamada(resposta, i);
            resposta = aplicarReLU(resposta);
      fim para
      //Utiliza SoftMax como função de ativação
      resposta = propagarEntradaParaProximaCamada(resposta);
      resultadoClassificador = aplicarSoftMax(resposta);

      //Registra apenas eventos de ligar (1) e desligar (2)
      se (resultadoClassificador > 0)
            vetorEventosDetectados.Adicionar(resultadoClassificador);
      fim se

    fim se
    ...
fim Loop
```

O novo trecho implementado de código, correspondente ao classificador, funciona da seguinte forma: a cada nova amostra RMS calculada pelo ESP 32, essa é adicionada ao fim de uma lista de tamanho fixo (janela de cinco valores). Essa janela de valores é então passada para o classificador, e caso a resposta seja 1 ou 2 (ligamento e desligamento respectivamente), o resultado é adicionado a um vetor circular dos últimos 256 eventos detectados juntamente com o tempo EPOCH de quando esse evento ocorreu.

4.7 Resultados Encontrados após a implementação no ESP 32

Após feita a implementação da rede neural no ESP 32, foi criada uma página adicional (http://ipdoesp32/events) no microcontrolador que mostra a listas dos últimos 256 tipos de eventos detectados (resposta 1 ou 2 do classificador) e os horários em que ocorreram. Uma rede "perfeita" deve ser capaz de detectar todos eventos de ligamento de desligamento da carga detectada, sem falsos positivos e negativos.

Foi iniciado então uma fase de testes da rede neural. Essa fase consiste em deixar o ESP 32 conectado ininterruptamente ligado por um dia fazendo as classificações de cargas. Enquanto o classificador trabalha para fazer as classificações em tempo real, o Web Server foi acionado em um outro computador. Assim como na fase de criação da base de dados de teste para o classificador, para toda ativação e desativação da carga (micro-ondas), foi registrado manualmente o horário do evento utilizando um relógio externo, sincronizado por GPS. O Web Server foi responsável por fazer a captura dos valores RMS que foram gerados pelo ESP 32.

Após a conclusão do período de 24 horas, foi criado um script em Python para a visualização dos dados. A Figura 31 mostra o gráfico gerado indicando a variação dos valores RMS ao longo do dia, os eventos que foram detectados pelo ESP 32 bem como os eventos registrados manualmente.

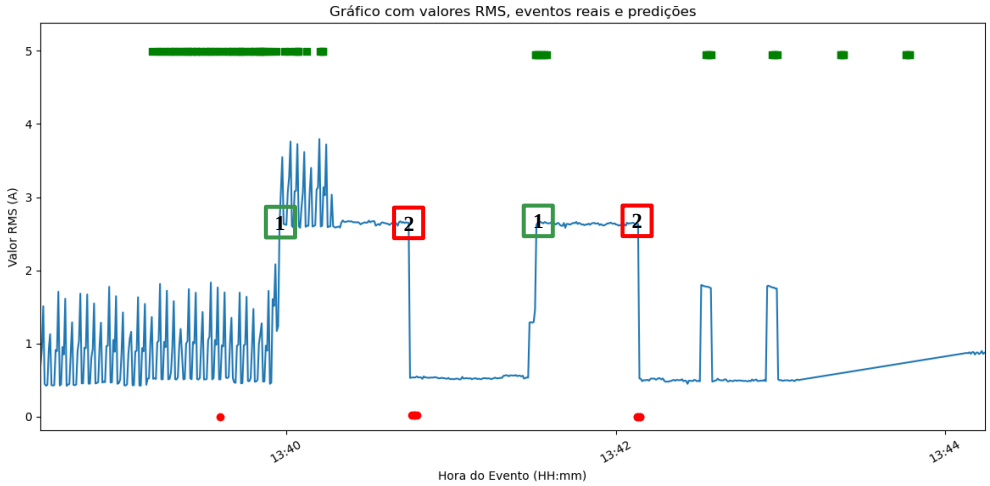

Figura 32: Gráfico com a corrente RMS e detecção de eventos durante o dia

Na Figura 33, foram ilustrados reais acionamentos feitos pelo usuário e as identificações feitas pelo classificador embarcado. O gráfico traçado em azul mostra a variação do valor RMS de corrente medida pelo ESP 32 em um dado intervalo de tempo. Os quadrados com os números 1 e 2, representam, respectivamente, os eventos de ligamento e desligamento do micro-ondas registrados manualmente. Os quadrados em verde na parte superior e os circulos vermelhos na parte inferior, representam, respectivamente, as janelas que foram classificadas pelo ESP 32 como eventos de ligamento e desligamento.

Pelos resultados apresentados pelo gráfico da figura 32, é possível perceber que o ESP 32 foi capaz de reconhecer corretamente os quatro (2 de ligamento + 2 de desligamento) eventos apresentados na figura. Além disso, o trecho anterior ao primeiro acionamento do micro-ondas mostra que a rede ainda está sensível à ruídos, e não consegue identificar cargas onde as variações de valores RMS assemelham-se ao do acionamento do micro-ondas. Para classificação dos eventos de desligamento, foi observado um número menor de falsos positivos comparado aos eventos de ligamento.

Vale ressaltar que as condições desfavoráveis do ambiente de teste. Isso ocorreu pela falta de acesso a um ambiente de teste controlado, isolado pelas medidas restritivas durante a epidemia de COVID-19. Uma validação mais detalhada do classificador será realizada em uma etapa posterior da pesquisa.

5. PRODUÇÃO TÉCNICO-CIENTÍFICA

Devido às restrições impostas pela pandemia de COVID-19 no acesso a um ambiente de teste controlado, não foi possível concluir o artigo científico planejado. O levantamento bibliográfico e a descrição da metodologia presentes neste relatório serão insumos para um artigo científico e um Trabalho de Conclusão de Curso. Futuramente, quando for possível validar a implementação do classificador, os resultados serão incluídos no artigo para submissão em congressos nacionais.

6. CONCLUSÕES

O objetivo da primeira parte do projeto, era de levantamento bibliográfico sobre métodos de medição de corrente não intrusivos, levantamento das possíveis plataformas disponíveis no mercado, leitura de sinais em micro controladores, e métodos para o tratamento de dados, considerando as limitações técnicas dos dispositivos.

Para a medição de corrente foi necessária a criação de um circuito para a leitura dos dados pelo ESP32. Logo no início da elaboração do código que seria inserido no dispositivo, foram encontradas limitações e dificuldades de processamento e acesso à memória, que dispositivos maiores, como computadores de mesa, não teriam.

Encontrados esses problemas, foi necessário ser feito mais estudos, sobre técnicas que otimizassem o processo de leitura, tornando-a eficiente e confiável. Após a conclusão da montagem do circuito para o deslocamento de sinal CC, que viria do TC, e finalização do código para o dispositivo, foi criado um pequeno script em OCTAVE, para visualização dos gráficos de corrente, e a variação seu valor RMS no tempo. Concluídas essas etapas, é esperado que o sinal que chega ao dispositivo seja confiável.

A partir daí, foi criado um servidor web responsável por fazer a captura dessas amostras disponibilizadas pelo ESP 32 periodicamente. Após a sua criação, as amostras foram armazenadas em arquivos .csv que facilitam a integração do sistema com outras ferramentas.

Depois disso, foram separados os arquivos gerados e os eventos classificados manualmente pelo professor foram inseridos na tabela gerada pelo servidor. Então, essas tabelas passaram por um script que faria o “janelamento” das amostras coletadas. A junção desses arquivos gerou a base de dados bruta que serviu para treinamento da rede neural. Alguns parâmetros estatísticos para interpretação da base foram extraídos utilizando o software Weka.

Por fim com a base de dados já criada, foi utilizado uma biblioteca em Python (Scikit-Learn) que foi responsável por treinar a rede neural. Essa ferramenta facilitou a remodelagem e ajustes de parâmetros referentes à rede por conta da simplicidade. Após isso a rede foi implementada no ESP 32, apresentando eficiência e funcionalidade parcialmente verificada, devido às restrições de acesso a um ambiente de teste controlado, impostas pela pandemia da COVID-19. Os resultados preliminares indicaram que o classificador foi capaz de reconhecer os eventos gerados, mas ainda classifica falsos positivos.

7. REFERÊNCIAS BIBLIOGRÁFICAS

[1] **Hart, G. W.** (1992). "Nonintrusive appliance load monitoring," in Proceedings of the IEEE, vol. 80, no. 12, pp. 1870-1891, doi: 10.1109/5.192069

[2] **Pereira, E. L., C. Cavalieri, D. e Resende, C. Z.** (2017). Método de identificação de cargas elétricas as técnicas Short-Time Fourier e Kernel PCA, XIII Simpósio Brasileiro de Automação Inteligente

[3] **Wild, B., Barsim, K. S. e Yang, B.** (2015). "A new unsupervised event detector for non-intrusive load monitoring," 2015 IEEE Global Conference on Signal and Information Processing (GlobalSIP), Orlando, FL, pp. 73-77, doi: 10.1109/GlobalSIP.2015.7418159.

[4] **Darby, S.** (2006). "The Effectiveness of Feedback on Energy Consumption: A Review for DEFRA of the Literature on Metering, Billing and Direct Displays," Environmental Change Institute, University of Oxford, Oxford

[5] **Azzini, H. A. D., Torquato, R. and da Silva, L.C. P.** (2014). "Event detection methods for nonintrusive load monitoring," 2014 IEEE PES General Meeting | Conference & Exposition, National Harbor, MD, pp. 1-5, doi: 10.1109/PESGM.2014.6939797.

[6] **Meziane, M. N., Ravier, P., Lamarque, G., Bune-tel, J.-C. L. and Raingeaud, Y.** (2017). "High accuracy event detection for Non-Intrusive Load Monitoring," 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, pp. 2452-2456, doi: 10.1109/ICASSP.2017.7952597.

[7] **JPEMBEDDED**, "Integrating IoT with IEC 61850", disponível em:
https://www.jpembedded.eu/en/integrating-iot-with-iec-61850
(Acessado em 07 de fevereiro de 2020)

[8] **Kok Tong Lee, Hou Kit Mun** (2019) K. T. Lee and H. K. Mun, "Real-time power monitoring using field-programmable gate array with IoT technology," in IET Science, Measurement & Technology, vol. 13, no. 6, pp. 931-935, doi: 10.1049/iet-smt.2018.5692.

[9] **Dragan Mlakić, Srete Nikolovski, Hamid Reza Baghaee** (2019) D. Mlakić, S. Nikolovski and H. R. Baghaee, "An Open-Source Hardware/Software IED based on IoT and IEC 61850 Standard," 2019 2nd International Colloquium on Smart Grid Metrology (SMAGRIMET), Split, Croatia, 2019, pp. 1-6, doi: 10.23919/SMAGRIMET.2019.8720361.

[10] **Markel Iglesias-Urkia, Diego Casado-Mansilla, Simon Mayer, Josu Bilbao, and Aitor Urbieta** (2019) "Integrating Electrical Substations Within the IoT Using IEC 61850, CoAP, and CBOR," in IEEE Internet of Things Journal, vol. 6, no. 5, pp. 7437-7449, doi: 10.1109/JIOT.2019.2903344.

[11] **IEC 61850-SER** (2018), "The IEC 61850 standard — Communication networks and automation systems from an electrical engineering point of view," 2016 19th International Symposium on Electrical Apparatus and Technologies (SIELA), Bourgas, pp. 1-4, doi: 10.1109/SIELA.2016.7543038.

[12] **MicroPython ORG**, "Getting started with MicroPython on the ESP32", disponível em:

https://docs.micropython.org/en/latest/esp32/tutorial/intro.html
(Acessado em 07 de fevereiro de 2020)
[13] **YHDC**, "SENSOR NÃO INVASOR: YHDC SCT013-000 CT USADO COM ARDUINO. (SCT-013)", disponível em:
https://www.poweruc.pl/blogs/news/non-invasive-sensor-yhdc-sct013-000-ct-used-with-arduino-sct-013.
(Acessado em 07 de fevereiro de 2020)

[14] **LEARN OPEN ENERGY MONITOR**, "CT Sensors - Interfacing with an Arduino", disponível em:
https://learn.openenergymonitor.org/electricity-monitoring/ct-sensors/interface-with-arduino.
(Acessado em 9 de fevereiro de 2020)
[15] **ESPRESSIF**, "ESP32 DEVKIT V4 Hardware Overview", disponível em:
https://www.espressif.com/en/products/hardware/esp32/overview.
(Acessado em 9 de fevereiro de 2020)
[16] **Arduino**, "Arduino UNO specs overview", disponível em:
https://www.farnell.com/datasheets/1682209.pdf . (Acessado em 10 de fevereiro de 2020)
[17] **IEEE XPlore**, disponível em:
https://ieeexplore.ieee.org/Xplore/home.jsp
(Acessado em 12 de fevereiro de 2020)
[17] **IEEE XPlore**, disponível em:
https://ieeexplore.ieee.org/Xplore/home.jsp
(Acessado em 12 de fevereiro de 2020)
[18] **Darío Baptista, (2018)** Baptista, D.; Mostafa, S.S.; Pereira, L.; Sousa, L.; Morgado-Dias, F. Implementation Strategy of Convolution Neural Networks on Field Programmable Gate Arrays for Appliance Classification Using the Voltage and Current (V-I) Trajectory. Energies 11, 2460.
[19] **J. Zico Kolter, (2011)** REDD: A Public Data Set for Energy Disaggregation Research, Proceedings of the SustKDD workshop on Data Mining Applications in Sustainability
Kolter, J. Z., & Johnson, M. J.
[20] **A.G. Ruzzelli, C. Nicolas, A. Schoofs, (2010)** "Real-Time Recognition and Profiling of Appliances through a Single Electricity Sensor," 2010 7th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), Boston, MA, pp. 1-9, doi: 10.1109/SECON.2010.5508244.
[21] **Antonio Ruano, (2019)** NILM Techniques for Intelligent Home Energy Management and Ambient Assisted Living: A Review. Energies 12, 2203.

[22] **John Esquiagola, Laisa Costa, Pablo Calcina, Geovane Fedrecheski and Marcelo Zuffo, (2017)** Performance Testing of an Internet of Things Platform.In Proceedings of the 2nd International Conference on Internet of Things, Big Data and Security - Volume 1: IoTBDS, ISBN 978-989-758-245-5, pages 309-314. DOI: 10.5220/0006304503090314

[23] **Christoph Klemenjak, (2018)** "On performance evaluation and machine learning approaches in non-intrusive load monitoring." Energy Informatics, vol. 1, no. Suppl 1

[24] **Ahmed Zoha, Alexander Gluhak (2012)** Non-Intrusive Load Monitoring Approaches for Disaggregated Energy Sensing: A Survey. Sensors 12, 16838-16866.

[25] **NILM Toolkit,** disponível em:
https://nilmtk.github.io/
(Acessado em 15 de julho de 2020)

[26] **Weka,** disponível em:
https://www.cs.waikato.ac.nz/ml/weka//
(Acessado em 15 de julho de 2020)

[27] **Scikit-Learn,** disponível em:
https://scikit-learn.org/stable/
(Acessado em 15 de julho de 2020)

[28] **Scikit-Learn MLP,** disponível em:
https://scikit-learn.org/stable/modules/neural_networks_supervised.html
(Acessado em 15 de julho de 2020)

8. AUTO-AVALIAÇÃO DO ALUNO

Com o aumento da relevância de sistemas de automação de redes elétricas de potência, a demanda de dispositivos responsáveis pela supervisão inteligente da rede, também aumentou. É interessante participar de um projeto que estuda a aplicabilidade de micro controladores, para o monitoramento da rede. Esse projeto traz para o meu contexto de graduando, a usabilidade de ferramentas computacionais para solução de problemas de Engenharia Elétrica. Foi possível também entender melhor conceitos de medição, funcionamento de circuitos integrados, e utilização de modelos computacionais focados em classificação. Além disso, o projeto foi importante para mostrar o funcionamento de metodologias científicas e como deve ser feito o planejamento para a realização da pesquisa. Por fim, pude ter contato com alguns temas que são vistos muito rápido durante o curso de graduação em Engenharia Elétrica, e tive a oportunidade aplicar alguns dos conceitos que já foram aprendidos nas cadeiras cursadas.