UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO
CENTRO TECNOLÓGICO
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROJETO DE GRADUAÇÃO

FELIX LUIS GALEANO QUEIROZ

MEDIÇÃO DE CONSUMO DE ENERGIA E ÁGUA COM SISTEMA SUPERVISÓRIO

VITÓRIA-ES

MARÇO/2022

Felix Luis Galeano Queiroz

MEDIÇÃO DE CONSUMO DE ENERGIA E ÁGUA COM SISTEMA SUPERVISÓRIO

Parte manuscrita do Projeto de Graduação do aluno Felix Luis Galeano Queiroz, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Vitória-ES

Março/2022

Felix Luis Galeano Queiroz

MEDIÇÃO DE CONSUMO DE ENERGIA E ÁGUA COM SISTEMA SUPERVISÓRIO

Parte manuscrita do Projeto de Graduação do aluno Felix Luis Galeano Queiroz, apresentado ao Departamento de Engenharia Elétrica do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do grau de Engenheiro Eletricista.

Aprovado em 25 de março de 2022.

COMISSÃO EXAMINADORA:

Antônio Manoel Ferreira Frasson
Universidade Federal do Espírito Santo
Orientador

Prof. Dr. André Ferreira
Universidade Federal do Espírito Santo
Examinador

Prof. Dr. José Denti Filho
Universidade Federal do Espírito Santo
Examinador

Vitória-ES

Março/2022

RESUMO

A indústria 4.0 é o conceito utilizado para descrever as constantes e rápidas mudanças nos processos tecnológicos e industriais, mudanças impulsionadas por avanços na área da computação, produção de *microchips* e tecnologias de sensoriamento. Um dos pilares dessa revolução tecnológica é a internet das coisas (IoT), conceito que descreve a interligação de dispostivos capazes de realizar medições e atuações em processos, com o uso de sensores e atuadores adequados. O conceito também engloba a aquisição e registro desses dados para posterior análise, realizada através de uma infraestrutura de rede adequada para a aplicação. Apesar de existirem diversas opções de dispostivos com sensores ou entradas para sensores de terceiros no mercado, essas soluções não são capazes de suprir cenários muito específicos, exigindo assim uma estrutura *tailor-made*, sob medida para a aplicação. Além disso, soluções prontas disponíveis no mercado podem prender o usuário no ecossistema de determinado fabricante através do uso de tecnologias proprietárias, impossibilitando a interligação com dispositivos e sistemas de terceiros. Pensando nesse cenário, o objetivo desse projeto foi a elaboração de dois medidores, um para consumo de água e outro para a medição de dados de potência e consumo de energia elétrica, com a possibilidade de leitura desses dados a partir dos protocolos de comunicação MODBUS TCP e HTTP através de uma API REST. O trabalho também realizou a aquisição desses dados através de script em Python 3 agendado para ser executado através de um *cron job*, que pode ser realizado em sistemas Linux ou MacOS, assim como o registro das informações em um banco de dados SQLite3. A visualização desses dados é realizada através da biblioteca Matplotlib para Python 3.

Palavras-chave: Internet das Coisas; Medidor de Energia; Medidor de Água.

ABSTRACT

The industry 4.0 is the conceptualization of the constant changes on technology and industrial processes that are happening on the 21st century. Those changes are possible because of the advances on computer engineering, large supply of microchips and the diversity of sensors available on the market. A key concept that enables the industry 4.0 is the internet of things (IoT). It describes the interaction and connectivity between devices capable of acquiring data and act on processes when connected to the proper sensors and actuators. It also describes the management of the data generated through those devices using network technology. There are devices and sensors available in the market, but the available solution may not suit some applications, requiring a tailor-made solution. Also, when using proprietary technology you may have to deal with closed-source software, restricting your technology options when scaling. This project proposes a flow meter and a energy meter, both capable of communicating using two standard protocols: MODBUS TCP and HTTP with a REST API. It's also proposed a scheduled data acquisition using cron job on Linux and MacOS systems with a Python 3 script. This data is stored in a SQLite3 database. The acquired data can later be visualized with Matplotlib, a Python 3 library.

Keywords: Internet of Things; Energy Meter; Water Meter.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analog-to-Digital Converter*
API	*Application Programming Interface*
ASCII	*American Standard Code for Information Interchange*
DNS	*Domain Name System*
HART	*Highway Addressable Remote Transducer*
HTTP	*Hypertext Transfer Protocol*
IDE	*Integrated Development Environment*
IoT	*Internet of Things*
IO	*Input Output*
I2C	*Inter-Integrated Circuit*
JSON	*JavaScript Object Notation*
LAN	*Local Area Network*
LSB	*Least Significant Bit*
MCU	*Microcontroller Unit*
MSB	*Most Significant Bit*
M2M	*Machine to Machine*
OSI	*Open Systems Interconnection*
PLC	*Programmable Logic Controller*
PWM	*Pulse Width Modulation*
REST	*Representational State Transfer*
RTU	*Remote Terminal Unit*
SCADA	*Supervisory Control and Data Acquisition*
SoC	*System on Chip*

SPI	*Serial Peripheral Interface*
TCP	*Transmission Control Protocol*
WiFi	*Wireless Fidelity*
XML	*Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

1.1 Apresentação

Uma revolução industrial é caracterizada por incorporações de tecnologias e seus impactos nos diversos setores. Em sua primeira forma, a primeira revolução industrial se inicia na Inglaterra no século XVIII com a invenção do tear mecânico e das máquinas a vapor. Esse processo iniciado no setor têxtil eventualmente se estendeu para outros setores (BRITO, 2017), impactando a forma do homem produzir. A segunda revolução industrial, no século XIX, é caracterizada pelo uso extensivo da eletricidade, possibilitando a substituição de máquinas movidas a vapor por máquinas movidas a energia elétrica. Já no século XX, após a segunda guerra mundial, o uso de tecnologias como robótica, informática e telecomunicações foi crescente, processo conhecido como a terceira revolução industrial. Agora, estamos vivendo a sua quarta etapa, que apesar de poder ser chamada de quarta revolução industrial, foi batizada durante a feira de Hannover em 2011 de indústria 4.0. A proposta dessa revolução envolve fábricas mais inteligentes, flexíveis e dinâmicas com seus elementos devidamente conectados. Tecnologias como inteligência artificial, internet das coisas, big data e computação na nuvem são usadas a fim de expandir a capacidade educacional e tecnológica num processo contínuo acompanhando o surgimento e a atualização de tecnologias (SCHWAB, 2016). Essa conexão e integração proposta se dá por meio da aplicação de larga escala de comunicação M2M (máquina a máquina, do inglês *machine-to-machine*).

De forma concisa, o processo revolucionário da indústria 4.0 pode ser identificado pela existência de fontes inteligentes, uso extensivo de robótica, sistemas confiáveis de processamento e armazenamento de dados e o monitoramento e controle de operações em tempo real (MAZZAFERRO, 2018). Dentro desse contexto, um elemento central desse processo é o conceito de internet das coisas (termo adaptado do inglês *internet of things*, também conhecida como IoT). Ele é definido como uma nova visão para a internet, em que a internet passa a abarcar não só computadores, como, também, objetos do dia a dia (FILHO, 2016).

Os circuitos integrados de silício surgiram em 1954. Em 1965, esse tipo de *microchip* continha por volta de 2000 transistores. Já em 2001, O Pentium 4 produzido pela Intel possuía 42 milhões de transistores (CHELIKOWSKY, 2004). A miniaturização dos chips devido a redução do tamanho dos transistores e o seu custo de produção permitiram o surgimento de MCUs (*microcontroller units*, em português, microcontroladores), diferenciados de processadores usuais por possuírem conexões de I/O (*input/output*, em português, entrada/saída) ao invés de conexões de dados e endereçamento (GRIDLING;

WEISS, 2007). Tais conexões são utilizadas para controle de processos e aquisição de dados, tornando-o um elemento crucial para a indústria 4.0 pela capacidade de integração de processos. Esse tipo de dispositivo é encontrado em lojas de eletrônicos em sua forma SoC (*System on a chip*, em português, sistema em um chip) onde todos os elementos necessários para o seu funcionamento estão integrados na placa de circuito, com preços acessíveis. O *Raspberry Pi Pico*, incluindo um processador de dois cores, tem o preço sugerido de 4 dólares pelo fabricante (em 2021). Existem modelos como o ESP32, incluindo as tecnologias Wi-Fi e Bluetooth, tornando-o uma ferramenta ideal para a interligação de processos, que também são encontrados por preços baixos, principalmente se comprados em maior escala, cenário usual para a indústria. Seguindo a ideia de baixo custo e baixo consumo de energia, o microcontrolador MSP430 é uma opção usual para a indústria por possuir uma arquitetura já muito estudada e conhecida, e também por possuir todas as interfaces usualmente encontradas, como PWM (*Pulse-width modulation*), UART (*Universal asynchronous receiver-transmitter*), SPI (*Serial Peripheral Interface*), ADC (*Analog-to-digital converter*) e I^2C (*Inter-Integrated Circuit*). Além disso, é possível expandir a conectabilidade desses dispositivos através de módulos expansivos ou conversores de sinal, como módulos que permitem uma conexão *ethernet*. Um exemplo de protocolo implementado sobre essas interfaces muito utilizado no meio industrial por questões de compatibilidade é o protocolo Modbus, que possui tanto a sua forma sobre *ethernet*/TCP quanto por conectores RS-232 ou RS-485, conhecido como Modbus RTU. O protocolo permite a leitura e escrita de registros, especificados de acordo com a funcionalidade desejada através do mapeamento fornecido pelo fabricante. Assim, através da programação dos microcontroladores é possível adequar qualquer equipamento que o utilize a aplicação desejada.

Neste cenário de ampla disponibilidade de microprocessadores por preços acessíveis e diversos protocolos de comunicação bem documentados e de uso consolidado, torna-se realizável a elaboração de um sistema de medição de energia e água e o armazenamento das informações geradas por tais medidores. Além disso, o ano de 2021 foi caracterizado por uma crise energética mundial, e em especial, essa crise no Brasil foi intensificada por outro problema: a crise hídrica (MALAR, 2021). Assim, a proposta desse projeto foi a montagem de dois medidores e também da aquisição, armazenamento e visualização dos dados gerados por estes. O primeiro medidor, para a medição de consumo de água, utiliza um microcontrolador ESP8266 NodeMCU para a contagem de pulsos de um sensor de fluxo de água modelo FS300A. O outro medidor, de consumo de energia elétrica, realiza a medição da rede elétrica através de um microcontrolador Arduino Nano, que comunica-se com um ESP8266 através de uma conexão serial, sendo este responsável pela comunicação com a rede. O ESP8266 em ambos os medidores conecta-se a uma rede local através de Wi-Fi e disponibiliza as informações através do protocolo Modbus TCP, com as devidas variáveis de interesse acessíveis em registradores específicados no

projeto e também por HTTP (*Hypertext Transfer Protocol*), que entrega um arquivo JSON (*JavaScript Object Notation*) por uma API (*Interface de Programação de Aplicações, do inglês Application Programming Interface*) conhecida como REST (*Representational State Transfer*). Utilizando a API REST, um *script* em linguagem de programação Python na versão 3 realiza a aquisição dos dados de forma periódica, com execução agendada por um *cron job*, funcionalidade disponível em sistemas operacionais do tipo Unix. Os dados são armazenados em um banco de dados com a tecnologia SQLite3. A visualização dos dados também é realizada através de Python, com a biblioteca Matplotlib.

1.2 Objeto e escopo do trabalho

1.2.1 Objeto

O objeto do presente trabalho é a elaboração de medidores de produção própria para leitura de consumo de água e energia capazes de se comunicar via Wi-Fi com duas opções de protocolos de comunicação usuais para aplicações IoT, podendo ser utilizado em outras aplicações, e também o desenvolvimento de um sistema capaz de ler e armazenar os dados medidos por ambos, assim como uma ferramenta capaz de visualizar tais dados de acordo com o período específicado pelo usuário.

1.2.2 Escopo

O escopo é delimitado pela confecção dos medidores, programação de *scripts* e configuração de rede necessária para integração. Para validação, o sistema foi instalado e configurado na casa do meu professor orientador, Antônio Manoel Frasson. Ao final do projeto, serão discutidas formas de expansão e implementação em diferentes casos de instalação.

1.3 Trabalhos relacionados

Existem diversas soluções prontas no mercado para aquisição de dados por sensores. Muitas vezes esses sistemas podem utilizar protocolos de comunicação proprietário ou não ter o seu protocolo detalhado pela expectativa do fabricante do usuário final utilizar o seu *software* também proprietário. Esse cenário dificulta a integração de medidores e sensores de diferentes origens, prejudicando a expansão e manutenção do sistema e aumentando

o custo de implementação. Devido a essa situação e a ampla disponibilidade de sensores no mercado, há um interesse de pesquisadores em desenvolver suas próprias soluções. A integração de sensores e MCUs é simplificada pela facilidade de programação de tais dispositivos, permitindo o usuário customizar a leitura, o armazenamento de dados e o tratamento de erros e alarmes de acordo com o seu objetivo, tornando a solução altamente adaptativa.

Os microcontroladores da série AtMega, usualmente encontrados em placas *Arduino*, é usualmente escolhido tanto por entusiastas como no meio acadêmico por ser uma opção *open source* e de fácil programação, baseada na linguagem C/C++. Ramos e Andrade optaram pelo uso do *Arduino* no desenvolvimento de uma central de monitoramento de consumo de energia e água, justificando o seu uso por ser um sistema *open source* e de baixo custo (RAMOS; ANDRADE, 2016). Para o sensoriamento, optaram por um sensor de corrente elétrica SCT-013-000, que possui uma saída de corrente de 50 mA e um sensor de fluxo de água de meia polegada com saída de pulso. A solução proposta no trabalho difere pela proporção das grandezas medidas e pelo foco no sistema supervisório, ao contrário do trabalho citado, que teve como o seu foco a criação dessa unidade medidora e a calibração dos sensores. Como será medido valores nas entradas dos prédios os sensores devem ser adequados para tal tarefa. Também difere porque a proposta atual engloba a elaboração de uma rede para coleta de dados, agrupando os dados de diversas unidades medidoras em uma unidade de processamento central que executa um *software* supervisório que também será elaborado.

Com a motivação de integrar dispositivos e realizar o seu controle através de variáveis negligenciadas pelos sistemas de controle tradicionais, Urzêda propôs o desenvolvimento de um sistema "inteligente" (URZEDA, 2006). A proposta foi realizar o controle dos aparelhos de ar-condicionado da empresa visando o conforto térmica, mas levando em consideração a temperatura externa do aparelho e a incidência solar. Seu ambiente de validação foi a sede da empresa Spin Engenharia de Automação Ltda. Diferente do trabalho a ser proposto, seu trabalho teve foco além do sistema supervisório na modelagem do sistema de controle, tendo as leituras e medições realizadas como uma ferramenta para a automação. Além disso, utilizou o *software* SCADA (*supervisory control and data acquisition*, no português, sistema de supervisão e aquisição de dados) *ActionView*, sendo que a proposta deste trabalho é o desenvolvimento de um sistema supervisório próprio.

A revista brasileira de mecatrônica mostrou um sistema para leitura de vazão de água *wireless*, utilizando um MCU PIC 18F4550 com o sensor de fluxo YF-S201 e módulo *wireless* HC-12, que opera na faixa de 433 MHz (SOUZA; AIROLDI, 2018). Com o módulo de comunicação o sistema aquisitava dados em um supervisório remoto, que computava o

consumo de água.

Apesar de haver diversos trabalhos na área, cada aplicação citada atende um nicho específico, de acordo com os requisitos próprios do cenário de estudo. Essa é a principal motivação do desenvolvimento de um supervisório ao invés de utilizar opções disponíveis no mercado. Ele será desenvolvido visando a possibilidade de expansão e o atendimento de demandas específicas.

1.4 Objetivos

Objetivo geral

- Desenvolvimento de solução de medição, transmissão, armazenamento e visualização de dados de consumo de energia e água com capacidade de escalar em instalações com múltiplas entradas de energia e água.

Objetivos específicos

- Prototipar e programar medidores de água e energia;
- Instalação dos medidores em cenário residencial;
- Implementar *script* para leitura periódica dos dispositivos medidores;
- Implementar ferramenta de visualização das variáveis armazenadas em banco de dados;
- Elaboração de instruções de uso do sistema.

1.5 Estrutura do texto

O presente trabalho está estruturado da seguinte maneira:

- **Introdução**: o capítulo atual, que realiza a contextualização, a apresentação inicial da proposta e a visão geral do projeto;

- **Referencial teórico**: neste capítulo são apresentadas as tecnologias utilizadas;
- **Metodologia e etapas de desenvolvimento**: apresentação do modelo de desenvolvimento do projeto e as etapas que compõem a sua elaboração. O capítulo é finalizado com a proposta definida dentro do cenário estudado, justificando as escolhas de tecnologia e *design*;
- **Resultados**: apresentação do protótipo final e sua integração no sistema proposto;
- **Conclusão**: considerações finais sobre o projeto e perspectivas futuras para ampliação e incrementação do sistema desenvolvido.

2 REFERENCIAL TEÓRICO

2.1 Microcontroladores

2.1.1 Arduino Nano

O Arduino Nano é um microcontrolador de placa única baseado no MCU (do inglês, *microcontroller unit*) ATmega328, desenvolvido pela empresa Arduino, responsável pela família Arduino de *hardware open-source*. Uma opção para a sua programação é a IDE (ambiente de desenvolvimento integrado, do inglês integrated development environment) *open-source* Arduino IDE, disponibilizada pela empresa Arduino. É programado através de arquivos do tipo .INO utilizando linguagem própria. Essa linguagem é um superconjunto de C++, isto é, é escrita da mesma forma mas possui a adição de funcionalidades extras (MORANDINI, 1996). O *hardware* possui portas para leitura digital, analógica e PWM. Suas especificações de *hardware* estão dispostas na tabela 1.

Tabela 1 – Especificações do Arduino Nano.

Característica	**Valor**
Microcontrolador	ATmega328 8-bit
Cores	1
Clock	16 MHz
Tensão de operação	5V
Portas digitais	22 (sendo 6 PWM)
Portas analógicas	8

Fonte: (ARDUINO, 2022).

2.1.2 ESP8266 NodeMCU

O ESP8266 é um *System on Chip* do fabricante Espressif, que atua como microcontrolador e também possui conectividade Wi-Fi. Possui programação similar ao Arduino, podendo ser desenvolvido dentro da mesma IDE. O modelo ESP8266 NodeMCU apresenta o *microchip* da Espressif em um *hardware open-source* em forma de microcontrolador de placa única. Apesar de já oferecer conectividade via Wi-Fi sem necessidade de demais módulos expansivos, como no caso do Arduino Nano, possui limitação na quantidade de ADCs disponíveis, somente um, o que foi incrementado no seu sucessor ESP32. O modelo sucessor possui 2 cores e 2 entradas ADCs. Suas especificações de hardware estão dispostas na tabela 2.

Tabela 2 – Especificações do ESP8266.

Característica	**Valor**
Microcontrolador	ESP-8266 32-bit
Cores	1
Clock	80 MHz
Tensão de operação	3,3V
Portas digitais	11 (sendo 4 PWM)
Portas analógicas	1

Fonte: (ESPRESSIF, 2020).

2.2 Biblioteca EmonLib

A biblioteca EmonLib (*Arduino Energy Monitoring Library*) é desenvolvida pela Open Energy Monitor com o intuito de funcionar com o medidor emonTx vendido pela própria empresa. Apesar de venderem o medidor pré-fabricado, a empresa tem o objetivo de propagar o aprendizado e entendimento sobre a energia elétrica, com a missão de explorar fontes de energia renováveis e de zero emissão de carbono. Por isso, a empresa disponibiliza todo o *hardware* e *software* elaborados por eles de forma *open-source*. Em seu site há uma aba destinada para aprendizado, contendo o passo a passo de um projeto de medidor para monofásico compatível com a biblioteca EmonLib.

Como o objetivo do uso dessa biblioteca no trabalho é a medição trifásica utilizando o Arduino Nano, que possui um único core, foram necessárias modificações na biblioteca para realizar a leitura das três fases simultaneamente. Essa modificação será detalhada no capítulo de metodologia e etapas de desenvolvimento. O algoritmo e a sua explicação monofásica continuarão válidos após a modificação, já que ela só acarreta na minimização do tempo sem leitura de uma determinada fase.

2.2.1 O algoritmo de medição de energia

Na etapa de *setup* na inicialização do microcontrolador os pinos ADCs para leitura de corrente e tensão devem ser especificados, assim como as variáveis de calibração para tensão, corrente e atraso de fase (V_{cal}, I_{cal} e ϕ_{cal}, respectivamente). As variáveis de calibração são responsáveis pela aquisição de valores confiáveis através da biblioteca e devem ser obtidos experimentalmente. A obtenção desses valores se dá utilizando um multímetro confiável como referência para medição de tensão e corrente e para a calibração da fase usa-se uma carga puramente resistiva. As variáveis de calibração estarão devidamente selecionadas quando os valores obtidos pela biblioteca forem iguais aos medidos pela referência, e o

fator de potência obtido for um para a carga resistiva.

O algoritmo de medição é implementado na função *calcVI* que possui duas entradas: o número de *crossings*, isto é, o número de vezes que a tensão alternada inverte o sinal, e o tempo para *timeout*, tempo que se extrapolado a função é encerrada sem êxito. Considerando a frequência da tensão de entrada, que é 60 Hz no caso do Brasil, e um número de *crossings* 30, o tempo de *timeout* especificado deve ser maior que 15 vezes o período relativo à 60 Hz, já que há cada dois cruzamentos se passa o tempo de um período da tensão de entrada. O tempo mínimo de *timeout* para o funcionamento correto da biblioteca é obtido pela equação 2.1.

$$T_{timeout} \geq \frac{Crossings}{2}\frac{1}{60Hz} \tag{2.1}$$

O algoritmo possui três etapas. Considera-se que qualquer etapa pode ser encerrada previamente caso o tempo de *timeout* seja extrapolado. Para realizar a medição de tensão da rede elétrica, essa tensão deve ser reduzida para níveis até a tensão de operação do microcontrolador (processo de *step down*). Como a tensão da rede pode assumir valores positivos e negativos, após o *step down* deve ser adicionado um *offset* para garantir somente valores positivos de tensão. Assim, a entrada ADC do microcontrolador é capaz de realizar leitura e mapear em valores digitais, discretizados em N_{MAX} níveis. No caso do Arduino, N_{MAX} corresponde a 1024 níveis. A corrente deve ser mapeada de forma similar, sempre considerando a tensão de operação do microcontrolador. O esquemático do circuito que realiza essa adequação do sinal para a entrada ADC será detalhado na metodologia. As operações no decorrer do algoritmo são realizadas utilizando os níveis, sendo convertidos para valores de tensão somente na terceira etapa.

- **Primeira etapa**, consiste em um *loop* que se encerra no momento que a leitura de tensão esteja na eminência de um *crossing*. O algoritmo considera 5% do ponto central, isto é, uma leitura entre $0,45N_{MAX}$ e $0,55N_{MAX}$.
- **Segunda etapa**, esse *loop* é iterado até o número de *crossings* ser atingido. Inicializa-se essa etapa com a medição de tensão e corrente. O valor de *offset* V_{offset} é o nível DC adicionado para obter somente valores positivos na entrada ADC. Uma solução mais precisa do que simplesmente subtrair $N_{MAX}/2$ para a remoção desse *offset* é utilizar um filtro passa-baixas para filtrar o nível DC, já que esse valor pode variar. Como o filtro depende do *offset* calculado no *loop* anterior, em sua primeira iteração ele é inicializado com $N_{MAX}/2$. O cálculo do *offset* é especificado na equação 2.2 e a tensão filtrada $V_filtrado$ a partir do *offset* é específicada na equação 2.3.

$$V_{offset}^{n} = V_{offset}^{n-1} + \frac{V_{amostra} - V_{offset}^{n-1}}{1024} \tag{2.2}$$

$$V_{filtrado} = V_{amostra} - V_{offset} \tag{2.3}$$

A mesma operação é realizada com a amostra de corrente. A demonstração de como a equação 2.2 atua como um filtro passa-baixas está no Apêndice A. Os valores filtrados de tensão e corrente são elevados ao quadrado e somados num acumulador que será utilizado na terceira etapa do algoritmo. Para o cálculo da potência instantânea, é necessário realizar a correção da fase da tensão em relação a corrente pelo tempo decorrido entre as medidas de tensão e corrente ADCs. Essa correção é realizada pela equação 2.5, onde é necessário ter a tensão filtrada (sem a correção de fase) do loop anterior.

$$V_{fase} = V_{filtrado}^{N-1} + \phi_{cal} * (V_{filtrado}^{N} - V_{filtrado}^{N-1}) \tag{2.4}$$

$$P_{instantânea} = V_{fase} * I_{filtrado} \tag{2.5}$$

A demonstração de como a equação 2.5 realiza a correção de fase está no Apêndice B. O valor de potência instantânea é somado num acumulador. Por fim, realiza-se a verificação se houve *crossing*. O processo é iterado até atingir os *crossings* definidos pelo usuário, que foi definido como 30.

- **Terceira etapa**, com os valores instantâneos acumulados de tensão ao quadrado, corrente ao quadrado e potência adquiridos na etapa anterior, realiza-se o cálculo dos valores de tensão e corrente eficazes, potência real e fator de potência. O número de amostras $N_{amostras}$ corresponde a quantidade de vezes que o *loop* da etapa anterior foi iterado até atingir os 30 *crossings* desejados. Na equação 2.6, X pode ser substituído pela corrente I e pela tensão V para o cálculo de seus valores eficazes. Vale ressaltar que k é um número inteiro.

$$X_{RMS} = X_{ratio}\sqrt{\frac{1}{kT}\int_{0}^{kT} X(t)^2\,dt} \approx X_{ratio}\sqrt{\frac{1}{N_{amostras}}\sum_{n=0}^{N_{amostras}} X[n]^2} \tag{2.6}$$

Com a aproximação pela discretização demonstrada na equação, nota-se a presença de um somatório da variável ao quadrado. Esse somatório na equação 2.6 equivale justamente ao valor quadrático acumulado na etapa anterior. Vale ressaltar que essa aproximação só é válida se a frequência de amostragem for muito maior que a frequência do sinal. A frequência de amostragem é o tempo decorrido entre duas leituras do ADC, que nesse cenário equivale ao tempo de uma iteração da etapa

anterior. Executando o algoritmo da EmonLib no Arduino Nano com a adição da função *millis* medindo o tempo de cada iteração, encontra-se uma frequência de amostragem de 834.93 Hz, valor 13,92 vezes maior que a frequência de 60 Hz da rede. A variável X_{ratio} é responsável pela conversão de valores de nível para valores de tensão entre 0 e a tensão de alimentação do microcontrolador através do termo X_{r1}, e também pela conversão para os valores de tensão reais através das variáveis de calibração X_{cal}, como mostra a equação 2.7.

$$X_{ratio} = X_{cal}X_{r1},\ com \quad X_{r1} = \frac{V_{in}/1000}{1024}\frac{[V]}{[níveis]} \tag{2.7}$$

A tensão de alimentação V_{in} é obtida através da função *readVcc* do Arduino. Através de uma tensão de referência interna de 1,1 V, a função calcula a tensão de alimentação do microcontrolador (MICROCHIP, 2020). Como os níveis de 0 a 1023 são mapeados de 0 a V_{in}, é possível obter uma conversão mais precisa com essa referência. A potência ativa P calculada a partir das amostras de potência instantânea p(t) acumuladas na segunda etapa é dada pela equação 2.8.

$$P = \frac{I_{ratio}V_{ratio}}{kT}\int_0^{kT} p(t)dt \approx \frac{I_{ratio}V_{ratio}}{N_{amostras}}\sum_{n=0}^{N_{amostras}} p[n] \tag{2.8}$$

Os valores eficazes de tensão e corrente são utilizados para calcular a potência aparente S, e o fator de potência FP é dado pela razão entre a potência real e a potência aparente recém obtidos, como descritos pelas equações 2.9 e 2.10, respectivamente.

$$S = V_{RMS}I_{RMS} \tag{2.9}$$

$$FP = \frac{P}{S} \tag{2.10}$$

A partir da potência real obtida pela biblioteca, é possível calcular a energia utilizando o tempo Δt equivalente ao tempo decorrido desde o último calculo de energia, como mostrado na figura 2.11

$$E[kWh] = \frac{P[w]\Delta t[ms]}{3,6 \cdot 10^9} \tag{2.11}$$

2.3 Comunicação

Os modelos de medidores de consumo de água e energia desenvolvidos nesse projeto tem a proposta de funcionar não somente no modelo de aquisição de dados desenvolvido aqui, mas também em outros cenários. Para isso, os medidores oferecem duas opções de conectividade. Uma opção é por Modbus TCP, protocolo que apesar de antigo ainda é amplamente utilizado por ter seu uso consolidado pelo tempo, oferecido até os dias de hoje em equipamentos recém-fabricados por questões de compatibilidade em plantas antigas. A segunda opção de comunicação é uma API REST, opção que funciona sobre o protocolo HTTP e apresenta diversas opções de *payload*. No caso, foi utilizado como *payload* um arquivo no formato JSON.

2.3.1 Modbus TCP

O protocolo Modbus foi criado pela Modicon, atual Schneider Electric, em 1979 com a finalidade de ser utilizado por PLCs (controladores lógico programável, do inglês *programmable logic controllers*). É possível utilizá-lo por camada de transporte serial assíncrona no formato RTU ou ASCII, e também em redes *ethernet* com o uso do formato TCP, por padrão na porta 502. Para o projeto, o dispositivo medidor funciona como um cliente Modbus, que pode ter seus registradores lidos por um servidor Modbus.

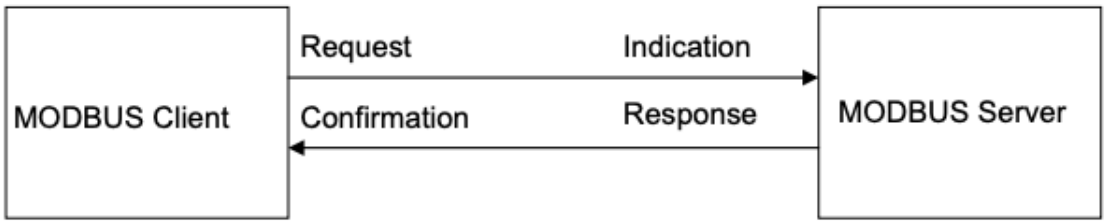


Figura 1 – Esquema de comunicação .

Fonte: (MODBUS ORGANIZATION, 2006)

Devido a época de desenvolvimento, o protocolo apresenta opção de registradores somente de 1 bit ou 16 bits, podendo ser variáveis disponíveis apenas para leitura (*Read-Only*) ou para leitura e escrita (*Read-Write*). Os tipos de variáveis disponíveis estão especificadas na tabela 3.

Para o registro de variáveis de 32 bits registra-se os bits do menos significativo (LSB) para o mais significativo (MSB) em dois registradores de 16 bits. O servidor MODBUS

Tabela 3 – Tipos de registradores Modbus.

Nome	Tipo de objeto	Tipo de acesso
Discretes Input	1 bit	Read-Only
Coils	1 bit	Read-Write
Input Registers	16 bits word	Read-Only
Holding Registers	16 bits word	Read-Write

Fonte: (MODBUS ORGANIZATION, 2006).

possui diversas funções de leitura e escrita de um único ou múltiplo registradores, como por exemplo a FC3, que realiza a leitura de múltiplos registradores do tipo *input register* ou *holding register*. Essa função pode ser utilizada para ler todas as variáveis de energia ou água registradas no ESP8266 em uma só requisição, fazendo o tratamento e interpretação dos dados recebidos no lado do servidor.

2.3.2 API REST

HTTP, *Hypertext Transfer Protocol*, é um protocolo da camada de aplicação do modelo OSI (Sistemas Abertos de Interconexão, do inglês Open System Interconnection) utilizado para transmissão de documentos. A API REST, *Representational state transfer*, define um conjunto de restrições sobre o protocolo HTTP, para descrever um estilo de arquitetura de *software* afim de padronizar serviços *web*. É possível enviar diferentes tipos de documentos pela API, podendo ser de diversos formatos como HTML, XML e, como no caso do projeto, JSON. Esses documentos são enviados através dos métodos definidos no protocolo HTTP, e no caso, usa-se o método GET, usado quando deseja-se somente requisitar dados (MOZILLA FOUNDATION, 2022).

O formato JSON utiliza um tipo de sintaxe específica para serializar variáveis do tipo *objects, numbers, strings, booleans* e *nulls*. É baseado nos tipos de variáveis disponíveis na linguagem de programação *Javascript*, mas há bibliotecas em demais linguagens capazes de ler arquivos .JSON e armazená-los de uma forma compatível com os tipos disponíveis. Em Python, há uma biblioteca nativa chamada JSON para ler, escrever, manipular e converter arquivos e variáveis desse tipo. O dicionário do Python possui a mesma sintaxe de um arquivo JSON, fator que facilita essa manipulação.

Figura 2 – Exemplo de arquivo JSON utilizado no projeto.

2.4 Banco de dados

Através da API REST oferecida nos medidores, foi desenvolvido um *script* em Python (versão 3) para aquisitar os dados. A linguagem foi escolhida por oferecer soluções nativas para realizar a leitura de arquivos JSON recebidos pela API (biblioteca *JSON*) e gerenciamento do banco de dados SQLite na versão 3 (biblioteca *sqlite3*). Para visualização dos dados graficamente, utiliza-se a bibliteca *Matplotlib*, que é mantida pela comunidade e livre para uso em publicações científicas (HUNTER, 2007). Já as requisições HTTP utilizam a biblioteca *requests*).

2.4.1 Python

Python é uma linguagem de programação interpretada, orientada a objetos e de tipagem dinâmica. Para a leitura periódica dos medidores e armazenamento no banco de dados, o *script* em Python desenvolvido nesse projeto deve ser usado em conjunto com o *Cron*, ferramenta disponível em sistemas do tipo *Unix*, agendando a sua execução a cada minuto.

Em um *Cron Job* indica-se através de números os minutos, horas, dias do mês, mês ou dias da semana que um comando especificado deve ser executado, podendo ser utilizado um asterico significando todos (por exemplo, asterisco em minutos indica todos os minutos). Na figura 3, utiliza-se o asterico para indicar que o comando explicitado deve ser executado em todo minuto, de toda hora, de todo dia, de todo mês de todo dia da semana. Na prática, isso implica que o comando será executado periodicamente a cada minuto.

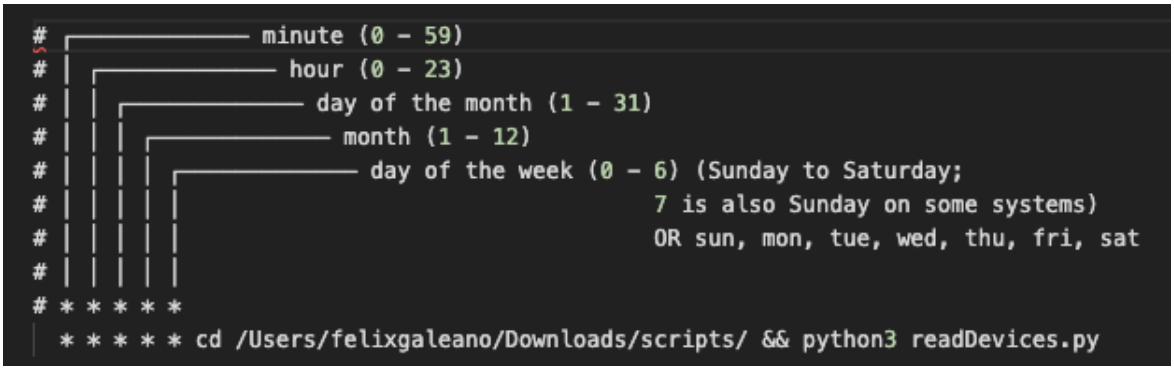


Figura 3 – Exemplo de Cron Job utilizado no projeto.

2.4.2 SQLite

O SQLite, atualmente na versão 3, é um implementador de banco de dados no modelo SQL (*Structured Query Language*) criado em 2004 e amplamente utilizado por ser de domínio público. Apesar de não averiguar o tipo das variáveis armazenadas, ele implementa majoritariamente as diretrizes definidas no padrão SQL-92.

2.4.3 Matplotlib

A biblioteca Matplotlib para Python é capaz de gerar gráficos pelos vetores do tipo lista, nativos de Python, ou de forma otimizada, utilizando vetores da biblioteca NumPy. Por ser livre para uso científico foi escolhida para a elaboração de um *script* de visualização. Esse *script* acessa o banco de dados, solicitando as informações armazenadas num período definido pelo usuário e através da bibloteca exibe esses valores graficamente.

3 METODOLOGIA E ETAPAS DE DESENVOLVIMENTO

3.1 Metodologia

O trabalho proposto, por se tratar de um projeto experimental, foi alterado durante sua construção de acordo com as adversidades encontradas, até chegar na versão final do protótipo. A proposta pode ser dividida em duas áreas independentes, que apesar de interagirem entre si podem ter aplicações individuais. A primeira é a construção dos medidores de água e energia, que foram revisados até atingirem resultados satisfatórios na aquisição de medidas, tendo precisão, constância no seu funcionamento em longo prazo, e verificada a ausência de falhas de operação no microcontrolador, além do funcionamento pleno da comunicação externa via MODBUS e API REST. A segunda área do projeto é o tratamento dos dados coletados dos medidores, que pode ser dividida vista por dois lados: primeiramente a coleta dos dados, dada pela requisição das informações ao microcontrolador pela rede, também envolvendo o registro no banco de dados desses dados, e o segundo lado como a leitura e visualização desses dados, que engloba o acesso ao banco de dados e o tratamento dessas informações para expô-las por gráficos. Para certificar que todo o processo é capaz de funcionar no longo prazo, foram observados os erros ocorridos durante a leitura em um período de testes e foram propostas formas de tratar tais adversidades.

3.2 Etapas de desenvolvimento

O projeto consistiu em quatro momentos de desenvolvimento.

- **Primeira etapa**, o protótipo de medição foi construído e testado individualmente, alterando-o quando necessário para garantir a confiabilidade de sua operação. Com ambos os medidores operando consistentemente, foi realizada a sua montagem em placa de circuito e inserção em uma carcaça para operação externa.
- **Segunda etapa**, com os medidores em pleno funcionamento, realizou-se a estruturação do banco de dados para registrar os dados dos medidores e a organização desses dados em tabelas distintas. Após a estruturação dos dados, foram testadas situações adversas de conexões e erros durante o processo, realizando tratamento e registro automático dessas adversidades em arquivo de *log*. Com o modelo de tratamento de dados proposto, foi testada a constância da coleta periódica no longo prazo.

- **Terceira etapa**, após aquisitar uma quantidade de dados satisfatória para visualização, foram elaboradas funções de acesso ao banco de dados a partir de variáveis de entrada solicitadas pelo usuário, e a sua devida exposição em forma de gráficos.
- **Quarta etapa**, por último, ocorreu a produção do manual de uso dos dispositivos medidores e dos *scripts*, anexado no Apêndice C.

3.3 Proposta

3.3.1 Estruturação da rede

Para o fluxo de dados ocorrer, propõe-se que todos os dispositivos envolvidos estejam ligados em uma rede LAN (Local Area Network). Apesar da coletada de dados poder ser realizada em qualquer sistema operacional do tipo *Unix*, sugere-se o uso de um servidor, já que a operação deve ocorrer de forma contínua e periódica. Usando uma máquina pessoal, a operação pode ser afetada ou interrompida por intervenção do usuário no decorrer do seu uso cotidiano. A figura 4 demonstra o fluxo e acesso dos dados.

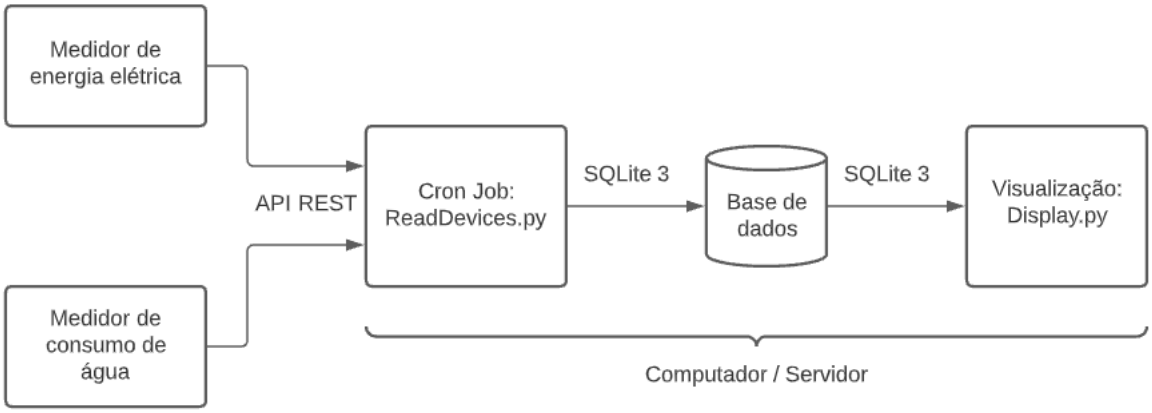


Figura 4 – Fluxo de dados no sistema.

3.3.2 Medidores

Para fornecer uma tensão de alimentação de 5 V ambos os medidores utilizam um circuito alimentado pela rede elétrica e também uma bateria para o caso de falhas na rede elétrica. O esquema de alimentação está apresentado na figura 5.

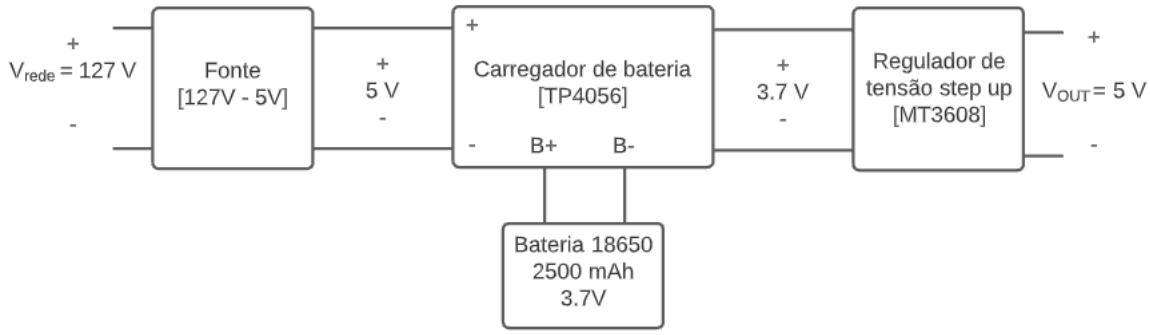


Figura 5 – Circuito regulador de tensão de alimentação.

O medidor de água utiliza um ESP8266 NodeMCU alimentado pelo circuito da figura 5 no pino V_{in}, que permite a alimentação com 5 V e realiza o processo de *step down* interno para 3,3 V, tensão de operação desse microcontrolador. Para a leitura de água usa-se um medidor de fluxo de água FS300A, com diâmetro de 3/4" e capacidade de leitura de 1 a 60 litros por minuto. Ele é ligado no circuito de alimentação e sua saída de pulso é conectada no pino 2 do ESP8266. No pino 12 há um botão de *reset*, que quando pressionado por 5 segundos reinicia o acumulador de litros. O esquema está ilustrado na figura 6.

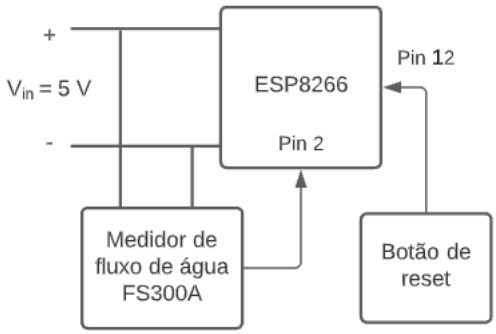


Figura 6 – Esquema de ligação do medidor de consumo de água.

A quantidade de água consumida L é computada em intervalos de 1 segundo a partir do número de pulsos N contados no decorrer desse intervalo. Há uma constante para adequação de unidade fornecida pelo fabricante do sensor no valor de C = 5,5. A operação para chegar no consumo em litros está na equação 3.1.

$$L = \frac{fluxo}{\Delta t} \quad com, \quad fluxo[\frac{L}{s}] = \frac{N}{C \cdot 60} \tag{3.1}$$

Apesar da operação ser executada periodicamente em intervalos de 1 segundo, calcula-se considerando o Δt, pois devido a frequência de amostragem do MCU e o tempo de execução do código, o cálculo do Δt pela função *millis* pode levar um tempo decorrido levemente maior do que 1 segundo.

O medidor de energia utiliza um TC (transformador de corrente) e um TP (transformador de potencial) em cada uma das três fases para realizar a medição. O TC utilizado é o modelo SCT013 de medição máxima de 100 A, com saída de 0 a 50 mA. Já o TP possui proporção do trafo de alta para baixa tensão de 127V para 9V.

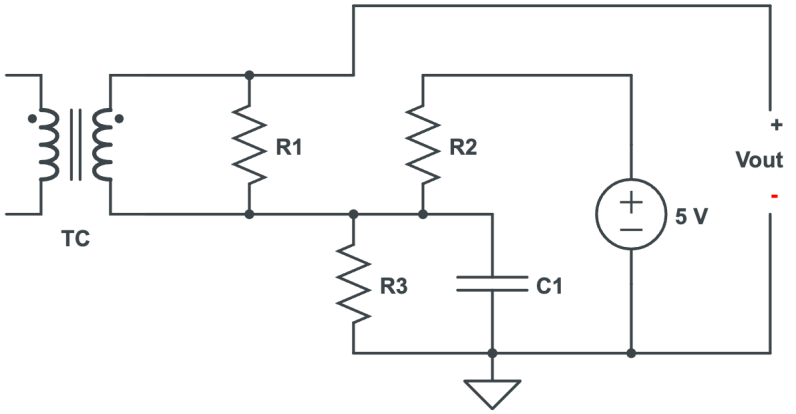


Figura 7 – Esquemático de ligação do TC.

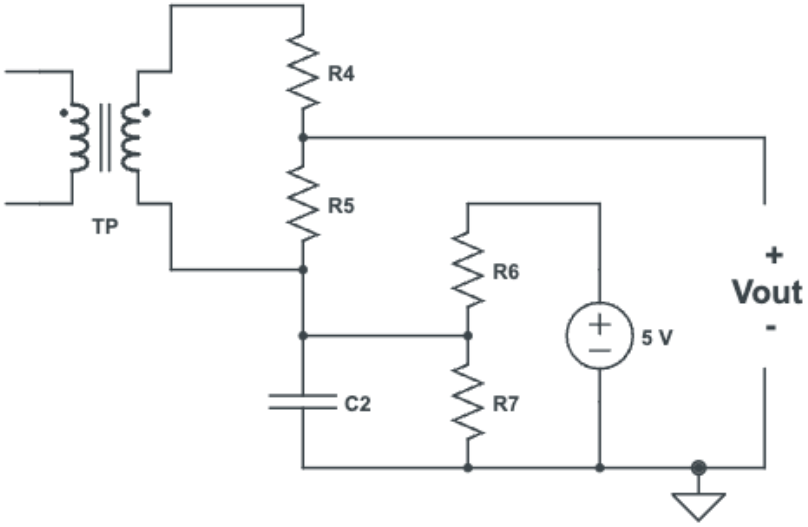


Figura 8 – Esquemático de ligação do TP.

Tabela 4 – Componentes.

Componente	Valor nominal
R1	100 Ω
R2=R3=R5=R6=R7	10 kΩ
R4	100 kΩ
C1=C2	10 μF

O circuito do TC, apresentado na figura 7, possui uma resistência R1 para evitar com que a saída do TC fique em aberto. O circuito apresenta uma elevação na tensão na saída

V_{OUT} em relação a saída do TC no valor de 2,5 V. Isso ocorre pelo divisor de tensão nos resistores R2 e R3, que possuem valores iguais, assim como os resistores R6 e R7 no caso do TP. A adição desse nível DC é realizada afim de ter somente valores positivos na entrada do ADC. No referencial teórico vimos que esse nível DC é tratado posteriormente pela biblioteca EmonLib. Em paralelo com a entrada do TP foi adicionado um varistor de 200 V afim de proteger o circuito, que para tensão da rede de 127 V possui tensão de pico de 179,61 V. As resistências R4 e R5 foram selecionadas afim de limitar a tensão V_{OUT}.

$$V_{OUT}[MAX] = 2,5 + 9\sqrt{2}\frac{R5}{(R4 + R5)} = 3.6571V \tag{3.2}$$

Assim, os circuitos do TC e TP adequam a faixa de tensão para a entrada do ADC do Arduino. No referencial teórico foi citado que a biblioteca EmonLib realiza a leitura de uma única fase por vez. Com o número de *crossings* 30 escolhido e sendo o número de períodos da tensão da rede elétrica ocorridos no decorrer da função a metade do número de *crossings*, conclui-se que a leitura leva 15 períodos, equivalente a 250 ms para a rede elétrica de 60 Hz. Como o Arduino possui somente um core, a realização da leitura de cada fase sucessivamente levaria um total de 750 ms, com um tempo de 500 ms para a leitura de uma mesma fase inciar novamente, não incluindo o tempo dos demais processamentos realizados no microcontrolador. Assim, a função *calcVI* foi modificada para que em cada iteração durante a contagem dos *crossings* seja realizada a leitura de amostras das três fases. Para isso, todas as variáveis da classe EnergyMonitor foram transformadas em vetores de três elementos, um para cada fase, e as leituras e operações para as três fases são realizadas dentro de um mesmo período de 250 ms utilizando um *for* interno a contagem dos *crossings*. Como as três fases possuem a mesma frequência, os *crossings* são contabilizados usando a primeira fase como referência. Assim é possível medir as três fases em um só *loop*, levando os mesmos 250 ms.

Para não perder dados de energia, o desejado é manter o esforço computacional sempre na aquisição de amostras dos ADCs. Com a intenção de manter o Arduino realizando somente esse processamento, optou-se por utilizar um ESP8266 que recebe os dados calculados pela EmonLib no Arduino través de uma conexão serial, processo que demanda menos processamento que o funcionamento do Arduino como cliente Modbus e servidor HTTP simultaneamente. Assim, o ESP8266 fica responsável por toda a conexão externa. Outra solução para esse problema testada foi o uso do ESP32, que possui dois cores, onde um deles poderia ser dedicado exclusivamente para a leitura de energia. Os ADCs do ESP32 se mostraram bastante suceptíveis a ruído, possivelmente pelo seu nível de tensão 3,3V, além do MCU não possuir referência interna de tensão (como o 1,1V do Arduino), utilizada para realizar a filtragem do *offset* com maior precisão. Assim, o uso de dois microcontroladores,

Arduino e ESP8266 foi escolhido por apresentar-se como uma solução mais robusta. O esquema do medidor de energia está apresentado na figura 9.

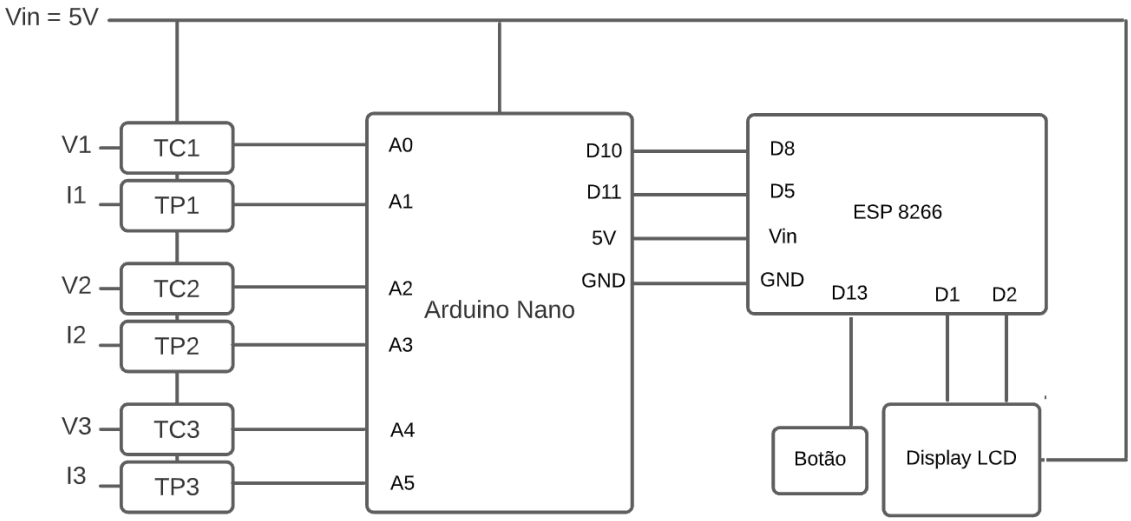


Figura 9 – Esquemático do medidor de energia.

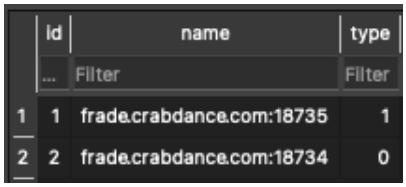
Nota-se que para a comunicação serial ocorrer com sucesso o ESP8266 é alimentado pelo Arduino. O ESP8266 mostra o IP do dispositivo no display LCD conectado a ele, e ao segurar um botão ligado no pino D13, ele inicia o modo de visualização dos valores de corrente, tensão, potência e fator de potência no display, alternando entre as informações de cada fase em intervalos de 3 segundos enquanto o botão continua pressionado.

Por fim, ambos os medidores, além de fornecer os dados nos registradores Modbus e via JSON na rota /data, também oferece uma forma de visualizar as informações por uma pagina *web*. Para isso basta acessar o endereço no navegador do dispositivo no navegador.

3.3.3 Gerenciamento de dados

O banco de dados é gerenciado utilizando SQLite 3. Quando inicializado através do script *setup.py*, três tabelas são criadas e inicializadas vazias.

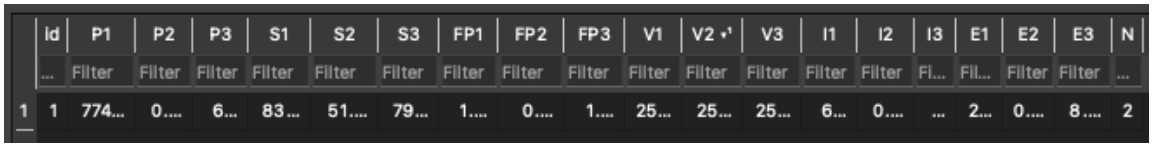
- **devices**, possui colunas id (inteiro), name (texto) e type (inteiro). Quando um dispositivo é adicionado através do script *addDevice.py* ele é registrado nessa tabela, contendo id definido pelo banco, o campo name no modelo "endereço:porta"e type 1 para medidor de energia ou 0 para medidor de água.



	id	name	type
...		Filter	Filter
1	1	frade.crabdance.com:18735	1
2	2	frade.crabdance.com:18734	0

Figura 10 – Exemplo de tabela device.

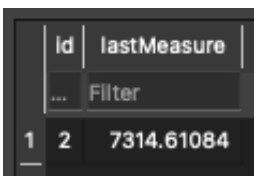
- **avgEnergy**, possui colunas id, correspondente ao id do dispositivo registrado na tabela "*devices*" e campos do tipo "*real*" para potência ativa (PX), potência aparente (SX), fator de potência(FPX), tensão (VX), corrente (IX) e energia (EX), sendo X=1,2 ou 3, para cada fase do dispositivo, e também um campo inteiro N. Em cada minuto uma leitura é realizada e acumulada nas variáveis reais, com exceção dos campos de energia que guardam o valor lido desde a última escrita na tabela do dispositivo de id correspondente (será mostrada posteriormente), enquanto o valor de N é acrescido em 1 em cada leitura. Quando alcançado o horário programado para registro no banco de dados, calcula-se a média a partir da divisão dos valores acumulados por N e o calculo da energia pelo valor lido atual subtraído do valor registrado nesta tabela. Os valores são registrados na tabela do dispositivo e os acumuladores são reiniciados, enquanto os valores de energia são atualizados para essa última leitura.



	id	P1	P2	P3	S1	S2	S3	FP1	FP2	FP3	V1	V2	V3	I1	I2	I3	E1	E2	E3	N
...		Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Filter	Fi...	Fil...	Filter	Filter	...
1	1	774...	0....	6...	83...	51....	79...	1....	0....	1....	25...	25...	25...	6...	0....	...	2...	0....	8....	2

Figura 11 – Exemplo de tabela avgEnergy.

- **lastWater**, com funcionamento similar ao cálculo da energia para a tabela anterior, essa tabela possui colunas id (inteiro) e lastMeasure (real). Essa segunda coluna registra o valor lido de consumo de água na última escrita na tabela do dispostivo de id correspondente, que é usado para calcular o consumo do período com o valor lido no momento da próxima escrita. Após esse procedimento, lastMeasure é atualizado com essa última leitura.



	id	lastMeasure
...		Filter
1	2	7314.61084

Figura 12 – Exemplo de tabela lastWater.

Além dessas três tabelas, uma tabela é criada para cada dispositivo adicionado, onde os dados da tabela avgEnergy e lastWater após processados (como descrito anteriormente) são registrados com a data. O tempo decorrido em minutos entre cada registro está configurado no arquivo *settings.json*, na variável *writeOnDbInterval*. Há também um script *deleteDevices.py* que lista os dispositivos registrados, permitindo selecionar algum deles para remoção da rotina de leitura periódica. A leitura é realizada de minuto em minuto agendando o script *readDevices.py* por um *cron job*.

Para visualização dos dados, usa-se o script *display.py*, selecionando o dispositivo e informando os parâmetros e período que deseja-se visualizar.

4 RESULTADOS

O medidor de energia, após confecção de placa de circuito e acoplamento dos componentes em uma caixa apropriada para uso externo pode ser visualizado internamente e externamente pelas figuras 13 e 14.

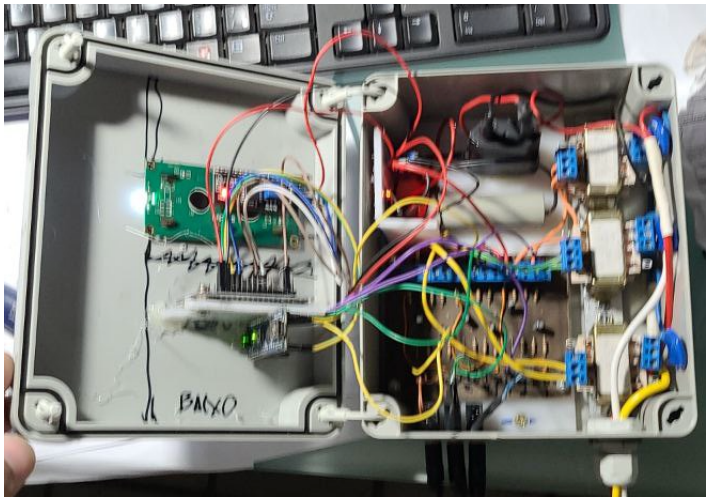


Figura 13 – Medidor de energia aberto.

Figura 14 – Medidor de energia fechado.

O botão para alteração das informações no display está localizado na parte inferior do medidor. A figura 15 apresenta a alocação interna da caixa do medidor de água.

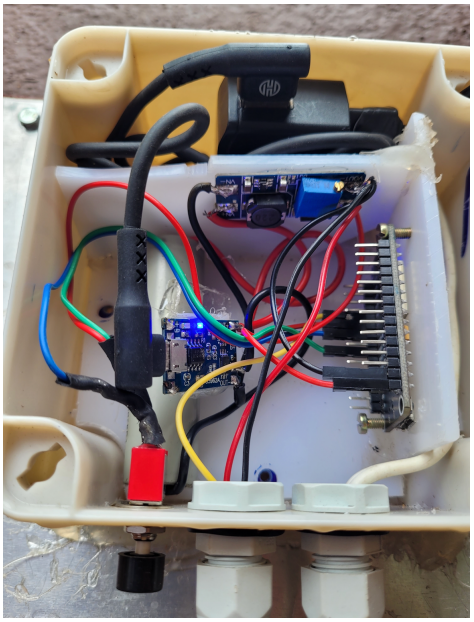


Figura 15 – Medidor de água internamente.

Como os medidores foram instalados na casa do meu orientador, foi necessário configurar um DNS (*domain name system*) para acesso via internet. Essa configuração foi realizada pelo domínio https://freedns.afraid.org/, que possibilita o registro gratuito de DNS. Assim, foi realizada a coleta de dados remotamente. Vale ressaltar que esse esquema foi realizado para fins de pesquisa. Em uma instalação ideal a aquisição de dados deve ser feita no mesmo ambiente pela rede local afim de minimizar problemas relacionados a conexão de internet. Durante um período de 3 dias de monitoramento, o medidor de água funcionou continuamente, enquanto o medidor de energia só apresentou problema de conexão entre o ESP8266 e a rede LAN uma única vez. As figuras 16 e 17 foram geradas utilizando o script *display.py*, exibindo os dados de consumo de água, energia e potência (total e por fase).

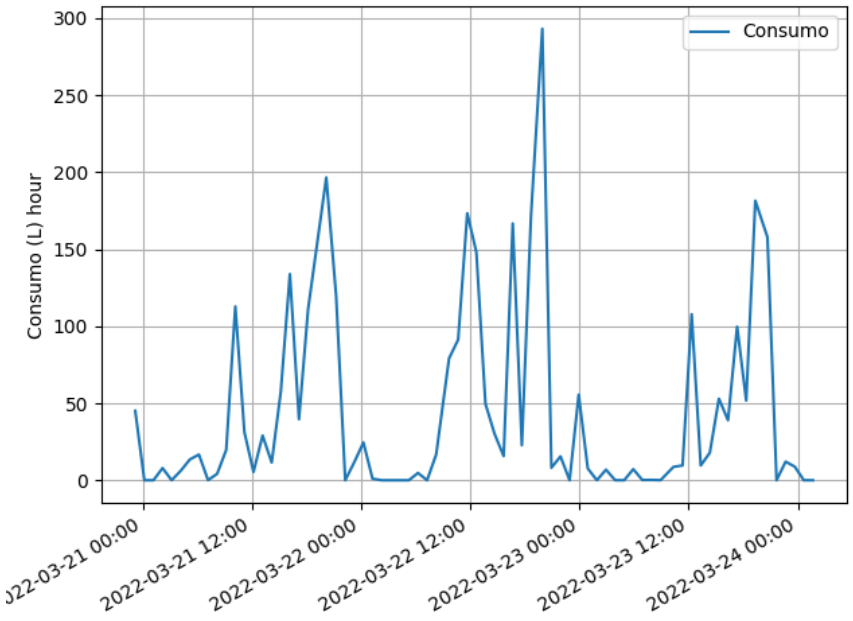


Figura 16 – Consumo de água no período de teste.

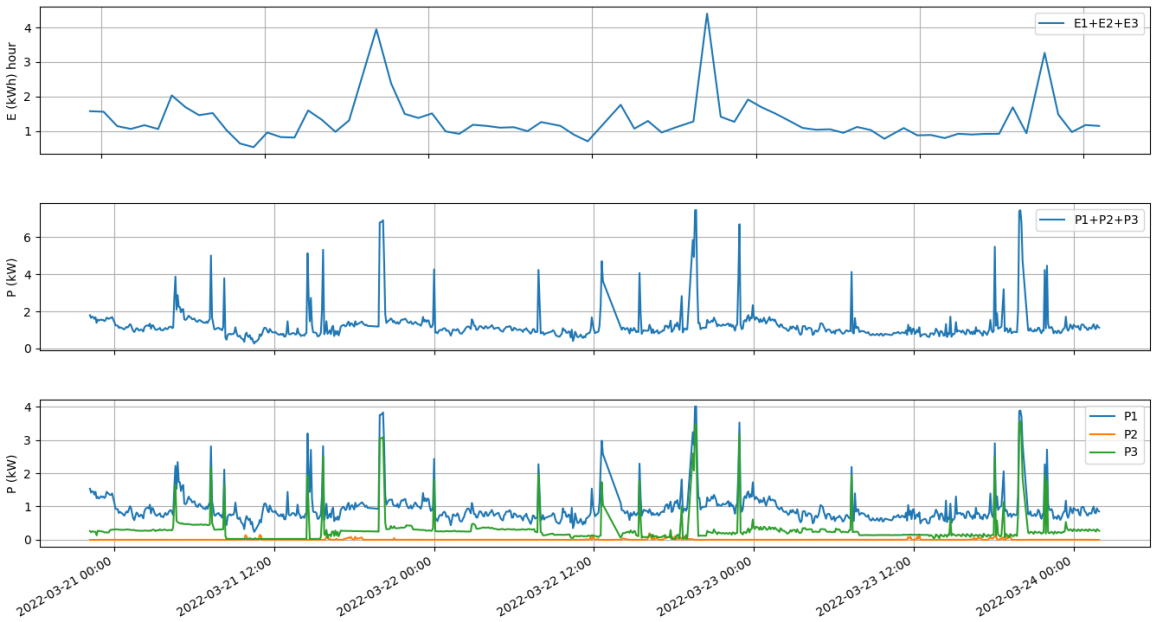


Figura 17 – Consumo de energia no período de teste.

5 CONCLUSÃO E TRABALHOS FUTUROS

O projeto foi concluído com ambos os protótipos de medidores funcionais, prontos para operação. Além de ser possível realizar a coleta de dados pelo *script* desenvolvido neste trabalho, os medidores podem ser utilizados em qualquer implementação que seja compatível com os protocolos Modbus TCP ou requisições HTTP, atentando-se a forma que os dados são entregues (essa informação está disponível no Apêndice C).

O *script* de aquisição de dados depende da formatação dos dados recebidos na forma esperada. É possível a sua comunicação com outros medidores, desde que ele apresente uma customização na formatação dos seus dados para adequar ao modelo esperado. Caso contrário, é necessário um dispositivo intermediário que realiza a leitura desse medidor e converte para o formato esperado. Por exemplo, para um medidor compatível com Modbus, é possível coletar os dados dos registradores especificados pelo fabricante com um MCU da família ESP e reenviá-los em um JSON via API REST.

Há melhorias a serem realizadas no sistema em relação a confiabilidade, conectividade e acesso de dados, podendo ser desenvolvidos por uma série de trabalhos futuros,

- **Implementação de um *datalogger***. Para casos de problemas de conexão entre os medidores e a rede LAN, uma solução para evitar a perda dos dados desse período é registrá-los em alguma forma de armazenamento, como um cartão SD. Mantendo-se dentro da solução proposta nesse trabalho, existem módulos de cartão SD no mercado compatíveis com o ESP8266.
- **Estudo de microcontroladores**. O medidor de energia acabou utilizando dois microcontroladores em sua versão final. Uma possibilidade para barateamento do sistema seria o levantamento de opções que possuem mais de um core para dedicar um deles somente para leitura de dados, enquanto as outras funcionalidades sejam executadas por outro core. Pensando no *datalogger* sugerido, existem microcontroladores no mercado que já oferecem essa funcionalidade nativa.
- **Uso de *watchdog***. Para dispositivos que funcionam constantemente, é comum o uso de *watchdogs*, sistemas via *software* ou *hardware* que verificam se houve o travamento do microcontrolador. O uso dessa ferramenta aumentaria a confiabilidade do sistema.
- **Estudo da precisão dos medidores**. Cabe um estudo númerico a partir dos dados

levantados. Ao comparar uma quantidade satisfatória de dados com medidores de precisão garantida, seria possível quantizar os erros dos medidores.

- **Expandir opções de visualização de dados**. A execução de *scripts* via terminal pode ser aprimorada para o usuário através de uma interface gráfica. Para aproveitar o processamento já realizado, é possível embutir gráficos do Matplotlib em interface gerada pela biblioteca Tkinter, inclusive tornando-os interativos. Outra opção é o desenvolvimento de uma aplicação *web*, que também pode ser transformada em aplicações *desktop* e *mobile*.

Todos os códigos utilizados estão disponíveis no GitHub.

https://github.com/felixgaleano/TCC2022

REFERÊNCIAS

ARDUINO. Arduino® UNO R3 Product Reference Manual. https://docs.arduino.cc/resources/datasheets/A000066-datasheet.pdf, 2022. Citado na página 17.

BRITO, A. A. F. D. A quarta revolução industrial e as perspectivas para o brasil. 2017. Disponível em: <https://www.nucleodoconhecimento.com.br/administracao/quarta-revolucao-industrial>. Acesso em: 20 set. 2021. Citado na página 11.

CHELIKOWSKY, J. Silicon: Evolution and future of a technology. 2004. Disponível em: <https://books.google.com.br/books?hl=en&lr=&id=Qxj_CAAAQBAJ>. Acesso em: 23 set. 2021. Citado na página 11.

ESPRESSIF. ESP8266EX Datasheet. https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex-datasheet-en.pdf, 2020. Citado na página 18.

FILHO, M. F. Internet das coisas. 2016. Disponível em: <https://www.researchgate.net/profile/Mauro-Fazion-Filho/publication/319881659_Internet_das_Coisas_Internet_of_Things/links/59c038d5458515e9cfd54ff9/Internet-das-Coisas-Internet-of-Things.pdf>. Acesso em: 23 set. 2021. Citado na página 11.

GRIDLING, G.; WEISS, B. Introduction to microcontrollers. 2007. Disponível em: <https://www.skylineuniversity.ac.ae/pdf/software-engineering/Microcontroller.pdf>. Acesso em: 27 set. 2021. Citado na página 12.

HUNTER, J. D. Matplotlib: A 2d graphics environment. Computing in Science & Engineering, IEEE COMPUTER SOC, v. 9, n. 3, p. 90–95, 2007. Citado na página 24.

MALAR, J. P. Brasil, europa e china têm crises energéticas com causas diferentes. CNN Brasil, 2021. Disponível em: <https://www.cnnbrasil.com.br/business/brasil-europa-e-china-tem-crises-energeticas-com-causas-diferentes-entenda/>. Acesso em: 2 out. 2021. Citado na página 12.

MAZZAFERRO, J. A. E. Indústria 4.0 e a qualidade da informação. 2018. Disponível em: <https://www.scielo.br/j/si/a/z9VfBmny3hhvcWvTsxW6YzN/>. Acesso em: 23 set. 2021. Citado na página 11.

MICROCHIP. megaAVR® Data Sheet. https://ww1.microchip.com/downloads/en/DeviceDoc/ATmega48A-PA-88A-PA-168A-PA-328-P-DS-DS40002061B.pdf, 2020. Citado na página 21.

MODBUS ORGANIZATION. MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE V1.0b. https://www.modbus.org/docs/Modbus-Messaging-Implementation-Guide-V1-0b.pdf, 2006. Citado 2 vezes nas páginas 22 e 23.

MORANDINI, M. Subsidios para o teste de software orientado a objetos: Definição e mapeamento de programas c++ para a li++. 1996. Disponível em: <https://www.teses.usp.br/teses/disponiveis/55/55134/tde-29082017-142725/en.php>. Acesso em: 22 mar. 2022. Citado na página 17.

MOZILLA FOUNDATION. Structuring the web with HTML. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods, 2022. Citado na página 23.

RAMOS, M. C.; ANDRADE, V. S. Desenvolvimento, construção e calibração de uma central de monitoramento de consumo de energia elétrica e de água utilizando o microcontrolador arduino. 2016. Disponível em: <https://revistas.cefet-rj.br/index.php/producaoedesenvolvimento/article/view/e98/119>. Acesso em: 30 set. 2021. Citado na página 14.

SCHWAB, K. The fourth industrial revolution: what it means, how to respond. 2016. Disponível em: <https://www.weforum.org/agenda/2016/01/the-fourth-industrial-revolution-what-it-means-and-how-to-respond/>. Acesso em: 20 set. 2021. Citado na página 11.

SOUZA, C. da Silva e; AIROLDI, D. Sistema para leitura de vazão de água. 2018. Disponível em: <http://revistabrmecatronica.com.br/ojs/index.php/revistabrmecatronica/article/view/23/28>. Acesso em: 30 set. 2021. Citado na página 14.

URZEDA, C. C. de. Software scada como plataforma para a racionalização inteligente de energia elétrica em automação predial. 2006. Disponível em: <https://repositorio.unb.br/handle/10482/3285>. Acesso em: 25 set. 2021. Citado na página 14.

Apêndices

APÊNDICE A – DEMONSTRAÇÃO DE FILTRO PASSA-BAIXAS

Considere o circuito RC da figura 18, com entrada x(t) sobre o resistor e capacitância em série e saída y(t) correspondente a tensão sobre o capacitor. Essa ligação funciona como um filtro passa-baixa.

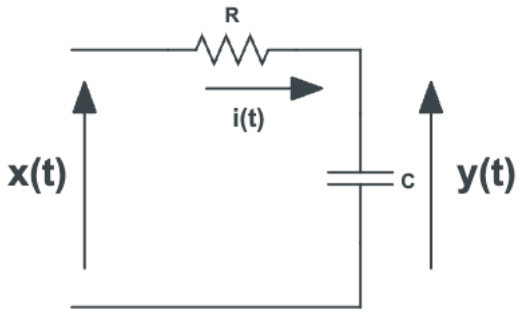


Figura 18 – Circuito RC.

A corrente i(t) descrita de acordo com os parâmetros da figura é dada pela equação A.1.

$$i(t) = C\frac{dy}{dt} = \frac{x(t) - y(t)}{R} \tag{A.1}$$

A constante de tempo é dada por $\tau = RC$ e frequência de corte do filtro $f_c = 1/2\pi\tau$. Na discretização do tempo, considera-se $y^n = y[n]$. Assim, é possível reescrever a equação A.1.

$$RC\frac{dy}{dt} = x(t) - y(t) \implies \frac{\tau}{\delta t}(y^n - y^{n-1}) \approx x^n - y^n$$

$$y^n \approx \frac{\delta t}{(\tau + \delta t)}x^n + \frac{\tau}{\delta t}\frac{\delta t}{(\tau + \delta t)}y^{n-1} \tag{A.2}$$

Define-se uma constante α para simplificação da equação A.2.

$$\alpha = \frac{\delta t}{(\tau + \delta t)} \implies 1 - \alpha = 1 - \frac{\delta t}{(\tau + \delta t)} = \frac{(\tau + \delta t) - \delta t}{(\tau + \delta t)} = \frac{\tau}{(\tau + \delta t)}$$

$$y^n \approx \alpha x^n + (1 - \alpha)y^{n-1}$$

Reescrevendo o desenvolvimento anterior chega-se na equação A.3, que descreve um filtro passa-baixas.

$$y^n \approx y^{n-1} + \alpha(x^n - y^{n-1}) \tag{A.3}$$

A equação 2.2 pode ser reescrita da forma da A.4.

$$V^n_{offset} = V^{n-1}_{offset} + \frac{1}{1024}(V_{amostra} - V^{n-1}_{offset}) \tag{A.4}$$

Comparando a equação A.4 com a equação A.3, nota-se que ela se comporta como um filtro passa-baixas, onde a entrada é $y = V_{offset}$, e conclui-se que a biblioteca EmonLib utiliza um filtro com $\alpha = 1/1024$. Assim, o *offset* que corresponde a um nível DC é filtrado. A frequência de amostragem f_a é dada pelo tempo decorrido entre cada iteração da segunda etapa do algoritmo de medição de energia. Utilizando as função *millis* do Arduino conclui-se empiricamente que $f_a = 834.93$ Hz. Assim, através da equação do *alpha* deduz-se a frequência de corte f_c.

$$\alpha = \frac{\delta t}{(\tau + \delta t)} \implies \frac{1}{\alpha} = \frac{\tau}{\delta t} + 1 \implies \frac{1}{\alpha} = \tau f_a + 1 \implies \tau = \frac{1/\alpha - 1}{f_a} = \frac{1024 - 1}{834,93} \approx 1,225$$

$$f_c = \frac{1}{2\pi\tau} = 0,1299Hz$$

Com os parâmetros do sistema encontrados, considerando uma entrada $V_{amostra}[n] = 2,5 + 9\sqrt{2}/11\cos(2\pi 60 n\delta t)V$, e considerando que a biblioteca EmonLib inicializa a váriavel V_{offset} na metade do número de níveis (512, correspondente a 2,5V), a saída do filtro é estabilizada, oscilando entre 2,4969 V e 2,5020 V, como mostra a figura 20.

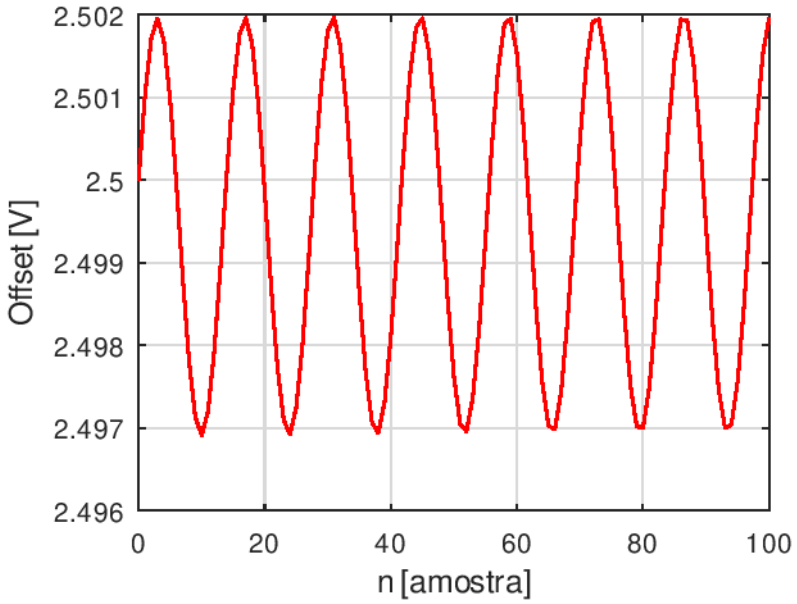


Figura 19 – Saída do filtro.

APÊNDICE B – AJUSTE DE FASE DA EMONLIB

A biblioteca EmonLib descreve o ajuste de fase da forma demonstrada na equação B.1.

$$phaseShiftedV = lastFilteredV + PHASECAL \times (filteredV - lastFilteredV) \quad \text{(B.1)}$$

Nomeando as variáveis, obtêm-se a representação da equação B.2.

$$V_{fase} = V_{filtrado}^{n-1} + \phi_{cal}(V_{filtrado}^{n} - V_{filtrado}^{n-1}) \quad \text{(B.2)}$$

Utilizando a identidade trigonométrica cos(a-b) = cos(a)cos(b) + sin(a)sin(b), pode-se realizar o seguinte desenvolvimento.

$$V_{filtrado}^{n-1} = V_{pico}cos(2\pi f(n-1)\delta t) = V_{pico}[cos(2\pi f\delta t)cos(2\pi n\delta t) + sin(2\pi f\delta t)sin(2\pi n\delta t)]$$

Reescrevendo a equação B.2 e explicitando a tensão filtrada no instante n na forma senoidal.

$$V_{fase} = (1 - \phi_{cal})V_{filtrado}^{n-1} + \phi_{cal}V_{filtrado}$$

$$V_{filtrado}^{n} = V_{pico}cos(2\pi fn\delta t)$$

Substituindo os valores de tensão filtrada em n e n+1 na equação do ajuste de fase.

$$V_{fase} = (1 - \phi_{cal})V_{pico}[cos(2\pi f\delta t)cos(2\pi n\delta t) + sen(2\pi f\delta t)sen(2\pi n\delta t)] + \phi_{cal}V_{pico}cos(2\pi fn\delta t)$$

$$V_{fase} = [(1 - \phi_{cal})cos(2\pi f\delta t) + \phi_{cal}]V_{pico}cos(2\pi n\delta t) + (1 - \phi_{cal})sen(2\pi f\delta t)sen(2\pi n\delta t)$$

Considerando a seguinte propriedade,

$$Mcos(\omega t - \phi) = Mcos(\phi)cos(\omega t) + Msen(\phi)sen(\omega t) = Acos(\phi) + Bsin(\omega t)$$

$$com \;\; M = \sqrt{A^2 + B^2} \quad e \quad \phi = tan^{-1}(\frac{B}{A})$$

Define-se $A = (1 - \phi_{cal})cos(2\pi f \delta t) + \phi_{cal}$ e $B = (1 - \phi_{cal})sen(2\pi f \delta t)$, para escrever a tensão defasada V_{fase} na forma da equação B.3.

$$V_{fase} = V_{pico} M cos(2\pi f n \delta t - \phi) \tag{B.3}$$

Nota-se que o termo M implica em que além da correção na fase ϕ há alteração na amplitude do sinal no processo. A partir da frequência de amostragem adquirida empiricamente obtêm-se $\delta t = 1/f_a = 1/834,93$ [s]. A frequência f é a frequência de rede elétrica, 60 Hz. Com esses valores é possível observar a alteração na amplitude M e o desvio de fase ϕ de acordo com a constante de calibração ϕ_{cal}.

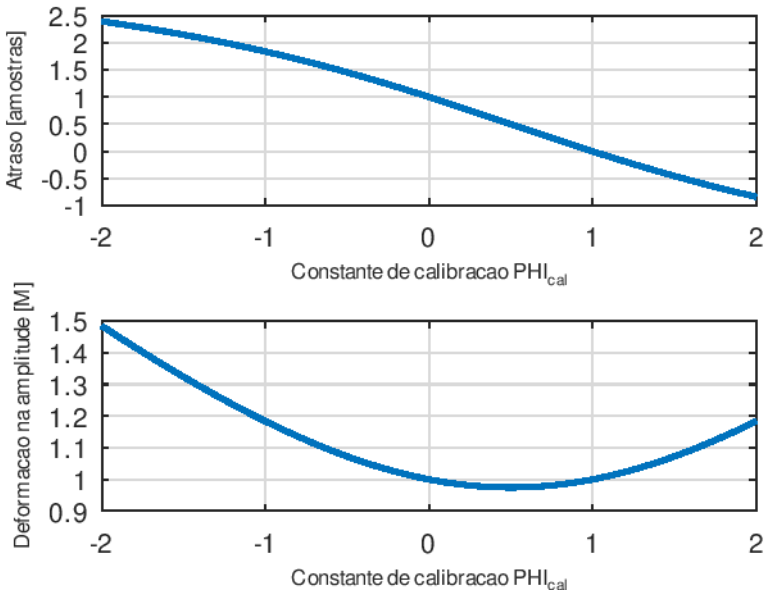


Figura 20 – Alteração em fase e amplitude pela constante de calibração.

Para testar expressão em simulação, utiliza-se uma tensão amostrada $V_{amostrado} = sen(2f\pi t)$ e um $\phi_{cal} = 2$, e também foi utilizada a mesma frequência de amostragem e frequência da rede expostas anteriormente, calcula-se o atraso teórico dessa amostragem em relação ao sinal de entrada (supondo um ϕ_{cal} bem selecionado) pela fórmula de ϕ deduzida, chegando ao sinal de entrada teórico por $V_{teórico} = sen(2\pi f(t - t_{atraso}))$, onde $t_{atraso} = \phi/2\pi f$.

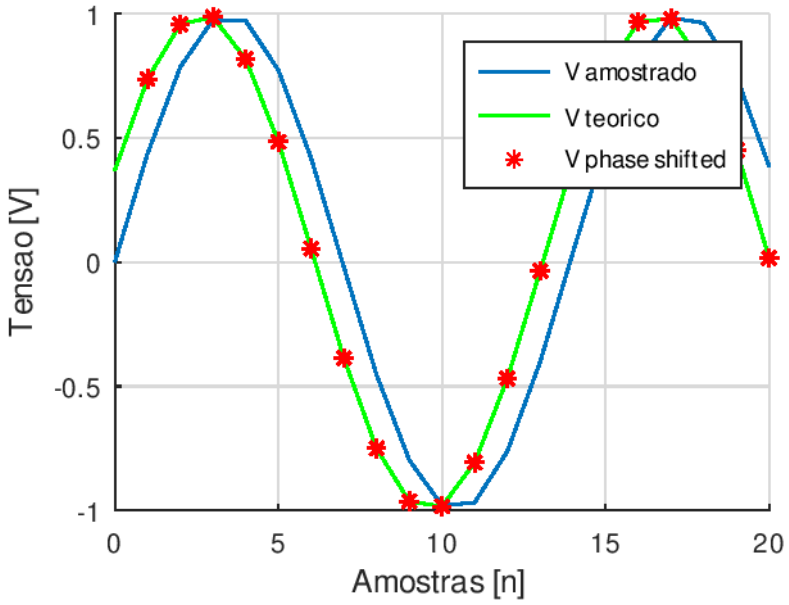


Figura 21 – Comparação da EmonLib com a teoria desenvolvida.

Os valores marcados com um asterisco na figura 21 foram calculados pela equação B.1 da EmonLib. Nota-se que a correção realizada pela biblioteca está sobre o gráfico da tensão de entrada teórica.

APÊNDICE C – MANUAL DO SISTEMA

C.1 Formato dos dados

Os medidores de energia e água fornecem os dados nos registros Modbus TCP e em JSON via HTTP a partir de um GET em /data. O endereçamento Modbus para medidor de energia está na tabela 5. Todos os registros são *input registers* e as variáveis devem ser interpretadas como floats de 32 bits.

Tabela 5 – Endereçamento do medidor de energia.

Endereço	Variável	Descrição
#0000	V1 LSB	Tensão RMS Fase 1 (LSB)
#0002	V1 MSB	Tensão RMS Fase 1 (MSB)
#0004	I1 LSB	Corrente RMS Fase 1 (LSB)
#0006	I1 MSB	Corrente RMS Fase 1 (MSB)
#0008	P1 LSB	Potência Ativa Fase 1 (LSB)
#0010	P1 MSB	Potência Ativa Fase 1 (MSB)
#0012	S1 LSB	Potência Aparente Fase 1 (LSB)
#0014	S1 MSB	Potência Aparente Fase 1 (MSB)
#0016	FP1 LSB	Fator de Potência Fase 1 (LSB)
#0018	FP1 MSB	Fator de Potência Fase 1 (MSB)
#0020	E1 LSB	Energia Fase 1 (LSB)
#0022	E1 MSB	Energia Fase 1 (MSB)
#0024	V2 LSB	Tensão RMS Fase 2 (LSB)
#0026	V2 MSB	Tensão RMS Fase 2 (MSB)
#0028	I2 LSB	Corrente RMS Fase 2 (LSB)
#0030	I2 MSB	Corrente RMS Fase 2 (MSB)
#0032	P2 LSB	Potência Ativa Fase 2 (LSB)
#0034	P2 MSB	Potência Ativa Fase 2 (MSB)
#0036	S2 LSB	Potência Aparente Fase 2 (LSB)
#0038	S2 MSB	Potência Aparente Fase 2 (MSB)
#0040	FP2 LSB	Fator de Potência Fase 2 (LSB)
#0042	FP2 MSB	Fator de Potência Fase 2 (MSB)
#0044	E2 LSB	Energia Fase 2 (LSB)
#0046	E2 MSB	Energia Fase 2 (MSB)
#0048	V3 LSB	Tensão RMS Fase 3 (LSB)
#0050	V3 MSB	Tensão RMS Fase 3 (MSB)
#0052	I3 LSB	Corrente RMS Fase 3 (LSB)
#0054	I3 MSB	Corrente RMS Fase 3 (MSB)
#0056	P3 LSB	Potência Ativa Fase 3 (LSB)
#0058	P3 MSB	Potência Ativa Fase 3 (MSB)
#0060	S3 LSB	Potência Aparente Fase 3 (LSB)
#0062	S3 MSB	Potência Aparente Fase 3 (MSB)
#0064	FP3 LSB	Fator de Potência Fase 3 (LSB)
#0066	FP3 MSB	Fator de Potência Fase 3 (MSB)
#0068	E3 LSB	Energia Fase 3 (LSB)
#0070	E3 MSB	Energia Fase 3 (MSB)

O endereçamento Modbus para medidor de água está na tabela 6. Todos os registros são

input registers e a variável deve ser interpretada como double de 64 bits.

Tabela 6 – Endereçamento do medidor de água.

Endereço	Variável	Descrição
#0000	L LSB0	Consumo (LSB0)
#0002	L LSB1	Consumo (LSB1)
#0002	L LSB2	Consumo (LSB2)
#0002	L MSB	Consumo (MSB)

A formatação do arquivo JSON recebido em /data está ilustrada nas figuras 22 e 23 para energia e água, respectivamente.

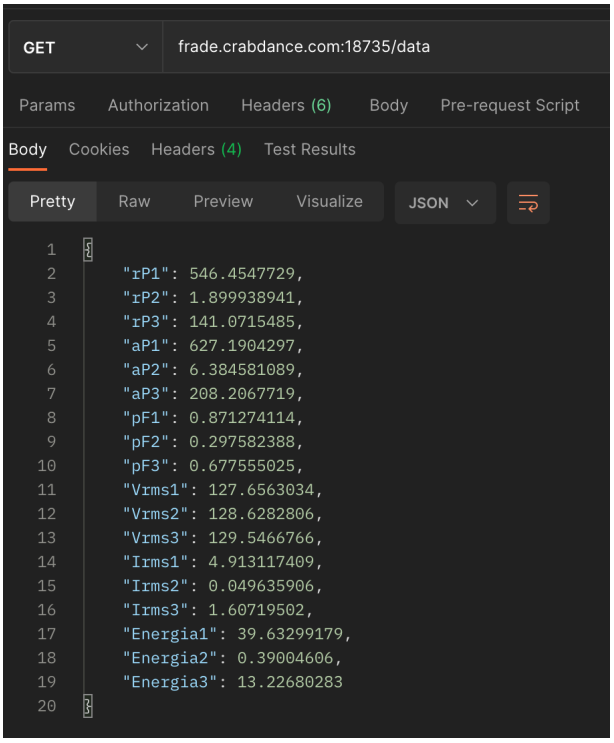


Figura 22 – Documento JSON do medidor de energia.

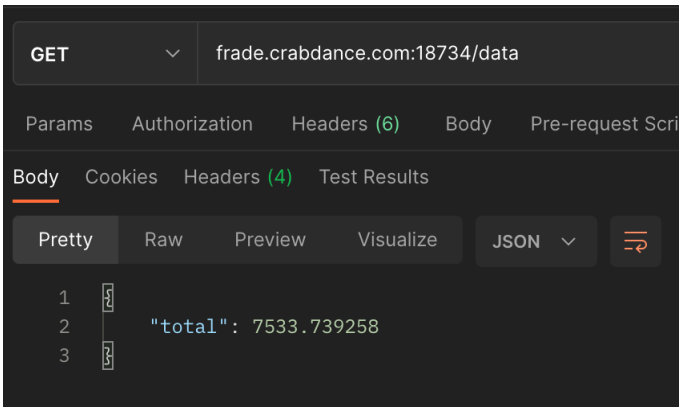
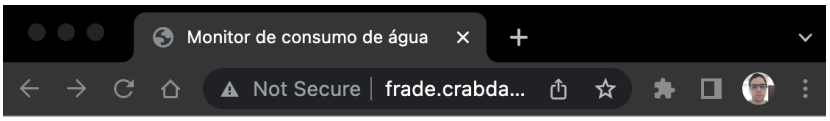


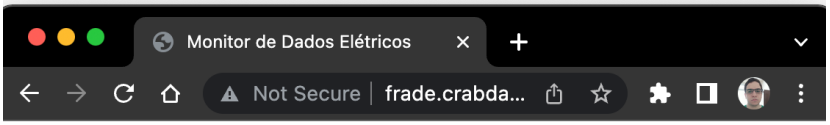
Figura 23 – Documento JSON do medidor de água.

Também é possível visualizar os dados pelo navegador, acessando o endereço do medidor, da forma demonstrada na figura 24.



Monitor de consumo de água

Consumo
Consumo Total = 7538.21 [litros]



Monitor de Dados Elétricos na Entrada da casa

Consumo	
Consumo Total = 53.44 [kWh]	
Fase 1	
$Vrms_1$ = 125.42 [V]	$Irms_1$ = 7.59 [A]
PR_1 = 730.62 [W]	PA_1 = 951.83 [VA]
FP_1 = 0.77	$Energia_1$ = 39.78 [kWh]
Fase 2	
$Vrms_2$ = 127.22 [V]	$Irms_2$ = 0.05 [A]
PR_2 = 1.28 [W]	PA_2 = 6.39 [VA]
FP_2 = 0.20	$Energia_2$ = 0.39 [kWh]
Fase 3	
$Vrms_3$ = 127.27 [V]	$Irms_3$ = 1.55 [A]
PR_3 = 144.07 [W]	PA_3 = 197.03 [VA]
FP_3 = 0.73	$Energia_3$ = 13.26 [kWh]

Figura 24 – Página web para visualização.

C.2 Banco de dados

Os *scripts* para gerenciamento do banco estão disponíveis no GitHub, com link na conclusão.

C.2.1 Inicialização

Para inicializar o banco de dados abra um terminal e vá até a pasta contendo os *scripts*. Digite 'python3 -m pip install -r requirements.txt' para fazer o *download* das bibliotecas necessárias. Em seguida, execute 'python3 setup.py'. Os arquivos necessários serão criados. Siga as instruções que aparecerão para configurar o *cron job*.

C.2.2 Gerenciar dispositivos

Para adicionar um medidor, execute 'python3 addDevice.py' e informe o endereço do dispositivo que será adicionado. Para a remoção de um dispositivo utilize o comando 'python3 deleteDevice.py'.

C.2.3 Visualizar dados

Para visualização dos dados execute 'python3 display.py'. Quando solicitado, informe o dispositivo a ser visualizado, as variáveis de interesse e a data que deseja plotar.

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

PROTOCOLO DE ASSINATURA

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
ANTONIO MANOEL FERREIRA FRASSON - SIAPE 298130
Departamento de Engenharia Elétrica - DEE/CT
Em 27/03/2022 às 16:57

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/388076?tipoArquivo=O

UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

PROTOCOLO DE ASSINATURA

O documento acima foi assinado digitalmente com senha eletrônica através do Protocolo Web, conforme Portaria UFES nº 1.269 de 30/08/2018, por
ANDRE FERREIRA - SIAPE 1713400
Departamento de Engenharia Elétrica - DEE/CT
Em 28/03/2022 às 17:44

Para verificar as assinaturas e visualizar o documento original acesse o link:
https://api.lepisma.ufes.br/arquivos-assinados/389268?tipoArquivo=O