

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



DOUGLAS BORINI GALTER

**PLATAFORMA IOT MODULAR PARA AQUISIÇÃO DE
DADOS**

VITÓRIA-ES

OUTUBRO/2021

Douglas Borini Galter

PLATAFORMA IOT MODULAR PARA AQUISIÇÃO DE DADOS

Parte manuscrita do Projeto de Graduação do aluno Douglas Borini Galter, 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

Outubro/2021

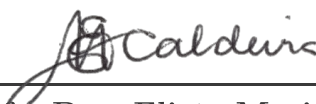
Douglas Borini Galter

PLATAFORMA IOT MODULAR PARA AQUISIÇÃO DE DADOS

Parte manuscrita do Projeto de Graduação do aluno Douglas Borini Galter, 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.

Aprovada em 14 de outubro de 2021.


COMISSÃO EXAMINADORA:



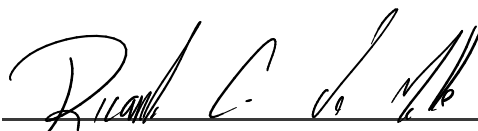
**Profa. Dra. Eliete Maria de Oliveira
Caldeira**
Universidade Federal do Espírito Santo
Orientadora



**Prof. Dr. Hans Jorg Andreas
Schneebeli**
Universidade Federal do Espírito Santo
Coorientador



**Prof. Dr. Camilo Arturo Rodríguez
Díaz**
Universidade Federal do Espírito Santo
Examinador



Prof. Dr. Ricardo Carminati de Mello
Universidade Federal do Espírito Santo
Examinador

Vitória-ES

Outubro/2021

RESUMO

Este projeto mostra uma estrutura de desenvolvimento de sistemas internet das coisas (IoT) para aquisição de dados baseado em uma arquitetura de *software* modular e portátil que seja capaz de fazer interface com diferentes tipos de sensores baseando-se em arquivos de configuração, que podem ser atualizados remotamente. Isto evita a necessidade de alterações no *software* embarcado (reprogramação). Os dados dos sensores são enviados para um banco de dados central usando o protocolo MQTT. A aplicabilidade da solução foi demonstrada a partir de testes com uma placa de desenvolvimento simples e com o uso de um sensor típico (conversor analógico digital) e um sensor que demanda interface e conversão específicas como o DHT22. Foi demonstrado que a alteração de parâmetros como intervalo de amostragem, tipo de interface e o canal usado pode ser feita de maneira simples. O uso de um sistema operacional de tempo real aumenta a modularidade, garante o intervalo de amostragem constante e permite que os dados sejam armazenados temporariamente no caso de interrupção de comunicação e enviados quando do restabelecimento desta.

Palavras-chave: Internet das coisas; Sensoriamento; MQTT; Sistema operacional de tempo real.

ABSTRACT

This project shows a development structure of internet of things systems for data acquisition based on a portable and modular software architecture that can interface diverse sensors types via configurations files, that can be send remotely. This makes changes in embedded software (reprogramming) unnecessary. The sensors data are sent to a database using the MQTT protocol. The applicability of the solution was demonstrated trough tests with a simple development board and the use of a typical sensor (analog-to-digital converter) and a sensor that requires a specific interface and data conversion. It was demonstrated that changing sensors can be done in a simple way. Using a real-time operating system increases modularity, guarantees constant sampling intervals and allows data to be stored when communications fails, to be sent when it returns.

Keywords: Internet of Things; Sensing; MQTT; Real-time operating systems.

LISTA DE FIGURAS

Figura 1 – Exemplo aquisição de dados em um contexto IoT.	10
Figura 2 – Arquitetura de um sistema IoT.	15
Figura 3 – Quantidade de documentos por protocolos de (a) meio de comunicação e de (b) camada de aplicação em diferentes anos.	19
Figura 4 – Alteração de parâmetros no modelo com sensores pré determinados. . .	23
Figura 5 – Arquitetura de capacidades.	24
Figura 6 – Alteração de parâmetros modelo de acesso à periféricos.	25
Figura 7 – Arquitetura completa da solução.	25
Figura 8 – Exemplo de um pacote de dados de um sensor.	29
Figura 9 – Exemplo de um pacote de configuração para um sensor.	29
Figura 10 – Diagrama da solução.	30
Figura 11 – Arquivos de configuração de ADC.	32
Figura 12 – Topologia do protótipo.	39
Figura 13 – Montagem do protótipo.	40
Figura 14 – Arquivos dos sensores ADC.	41
Figura 15 – Arquivos do sensor DHT22.	41
Figura 16 – Execução do interpretador e aquisição de dados.	42

LISTA DE QUADROS

Quadro 1 – Camadas das aplicações em IoT.	16
Quadro 2 – Suporte a tempo real em sistemas operacionais comuns em IoT.	18

LISTA DE ABREVIATURAS E SIGLAS

ADC	<i>Analog-to-Digital Converter</i>
ASCII	<i>American Standard Code for Information Interchange</i>
BLE	<i>Bluetooth Low Energy</i>
CMOS	<i>Complementary Metal Oxide Semiconductor</i>
CoAP	<i>Constrained Application Protocol</i>
CPU	<i>Central Processing Unit</i>
GPIO	<i>General Purpose Input/Output</i>
HTTP	<i>Hypertext Transfer Protocol</i>
I ² C	<i>Inter Integrated Circuit</i>
IBM	<i>International Business Machines</i>
IoT	<i>Internet of Things</i>
IETF	<i>Internet Engineering Task Force</i>
LoRa	<i>Long Range</i>
MCU	<i>Microcontroller Unit</i>
MQTT	<i>Message Queuing Telemetry Transport</i>
RFID	<i>Radio-frequency identification</i>
RMS	<i>Rate-Monotonic Scheduling</i>
RR	<i>Round-Robin</i>
RTOS	<i>Real Time Operating Systems</i>
SPI	<i>Serial Peripheral Interface</i>
TCP	<i>Transmission Control Protocol</i>
UDP	<i>User Datagram Protocol</i>
UART	<i>Universal Asynchronous Receiver-Transmitter</i>
WSN	<i>Wireless Sensors Networks</i>

SUMÁRIO

1	INTRODUÇÃO	9
1.1	IoT	9
1.2	Aquisição de dados usando IoT	9
1.3	Objetivos	12
1.3.1	Objetivo geral	12
1.3.2	Objetivos específicos	12
1.4	Metodologia	13
1.5	Estrutura do trabalho	14
2	SISTEMA DISTRIBUÍDO DE AQUISIÇÃO DE DADOS	15
2.1	Camadas de estudo em IoT	15
2.2	Frameworks para IoT	16
2.3	Sistemas operacionais	17
2.4	Protocolos de comunicação remota	19
2.5	Microcontrolador	20
2.6	Arquitetura da solução	22
3	IMPLEMENTAÇÃO	27
3.1	Definição dos sensores	27
3.2	Transmissão dos dados	28
3.3	Arquitetura da solução	29
3.4	Recepção dos dados	30
3.5	Arquivo de configuração e interpretador	31
3.6	Sensores com protocolos comuns	31
3.7	Sensor com protocolo específico	32
3.8	Arquitetura dos sensores e operação	34
3.9	Arquitetura da transmissão	35
3.10	Arquitetura de recepção	36
3.11	Interpretador	37
4	RESULTADOS	39
5	CONCLUSÕES E TRABALHOS FUTUROS	44
	REFERÊNCIAS	45

1 INTRODUÇÃO

1.1 IoT

IoT é uma tecnologia que vem sendo considerada como a base da próxima revolução industrial. O conceito principal é a ideia de promover a comunicação entre os mais diversos objetos, máquinas e dispositivos (FEKI et al., 2013). Apoiado pelo desenvolvimento das tecnologias em eletrônica, informática, telecomunicações e ciências sociais, IoT é um tópico que continuará relevante no futuro (ATZORI; IERA; MORABITO, 2010).

O custo dos dispositivos IoT é um fator crítico, já que a grande quantidade de objetos a serem integrados implica em muitos equipamentos na solução. Tradicionalmente, os dispositivos IoT possuem capacidade reduzida de armazenamento em memória, o que torna essencial o uso de técnicas que demandem pouca memória (JASKANI et al., 2019). Além disso, o consumo de energia, em geral, é outro fator crítico devido ao fato de que os dispositivos normalmente são alimentados por bateria. Assim, como em tecnologia CMOS o consumo é proporcional à frequência de relógio, devem ser utilizadas técnicas que usem a alteração da frequência de relógio para reduzir o consumo. O desligamento seletivo de periféricos que não estão sendo utilizados no momento pode também diminuir significativamente o consumo.

Arquiteturas modernas permitem isto sem muito impacto na capacidade de processamento ou no custo, tendo em vista o crescente desenvolvimento de circuitos integrados e placas de desenvolvimento com custos cada vez menores. Isso permite a implantação de novas funcionalidades e torna mais simples e acessível o desenvolvimento da solução no lugar de criar *hardware* customizado (TAIVALSAARI; MIKKONEN, 2018).

Microcontroladores do tipo ARM Cortex-M podem ter quantidades de memória *flash*, RAM e alta frequência de relógio que permitem um desenvolvimento mais simples de soluções por ser mais flexível com essas restrições. Além de fornecer ferramentas para atendê-las, possuem também um custo comparável com os microcontroladores utilizados tradicionalmente.

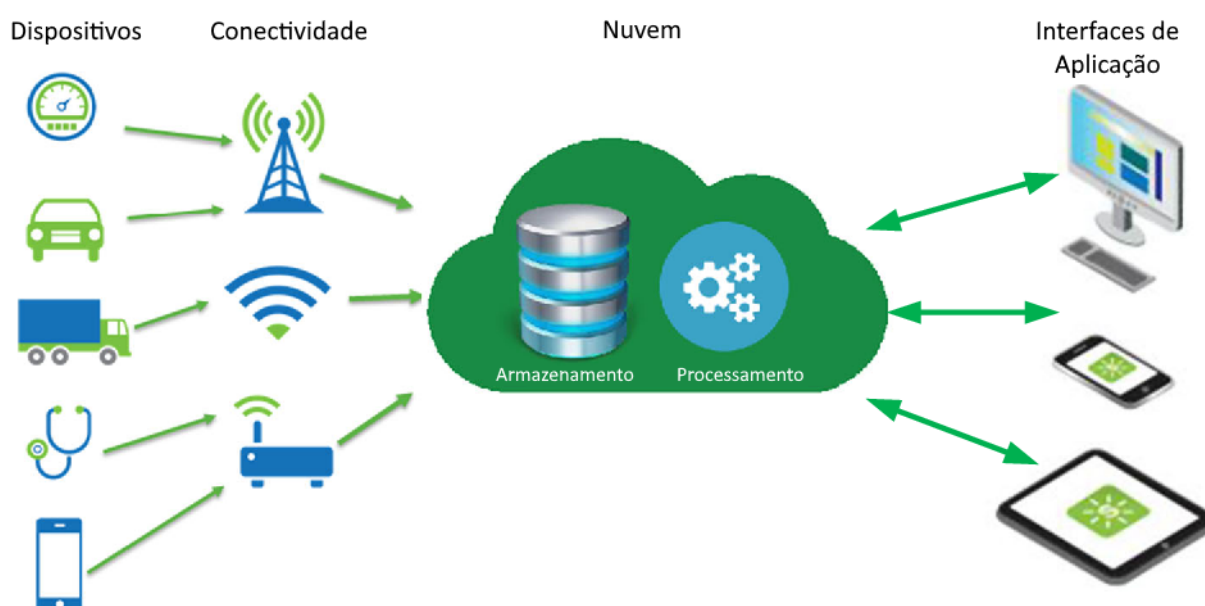
1.2 Aquisição de dados usando IoT

Mishra et al. (2016) dividem as aplicações em quatro áreas principais:

- área da indústria, citando reduções no desperdício e aumento das margens de lucro;
- área da saúde, onde as condições dos pacientes são monitoradas e transmitidas para os profissionais de saúde remotamente;
- área do ambiente, demonstrando o uso de dispositivos e IoT para beneficiar pessoas com exemplos de aplicações em estacionamentos, academia e turismo;
- área do setor pessoal e social, citando o uso em automação residencial e redes sociais.

Nas aplicações é comum a presença de *Wireless Sensors Networks* - WSN (tradução livre, redes de sensores sem fio), que é o nome dado ao conjunto de sistemas microeletrônicos e mecânicos, comunicações sem fio e componentes eletrônicos digitais que são capazes de interagir com o ambiente ao seu redor, interagir com os eventos do mundo real e comunicar-se. Essas características tornam esse conjunto um ponto chave para a idealização do IoT (SRIVASTAVA; SINGH; GUPTA, 2018). A Figura 1 demonstra um sistema de aquisição de dados em um contexto IoT que envolve diversos parâmetros.

Figura 1 – Exemplo aquisição de dados em um contexto IoT.



Fonte: Tomovic et al. (2017).

Nota: Adaptado pelo autor.

Para cumprir os objetivos dessas aplicações há uma grande diversidade de arquiteturas de *software* (TAIVALSAARI; MIKKONEN, 2018). Muitas dessas arquiteturas usam um *kernel* denominado *Real Time Operating System* - RTOS (tradução livre, sistema operacional de tempo real), já que muitas das aplicações que estão sendo estudadas no contexto de IoT exigem restrições rígidas de segurança e temporização para garantir controle e monitoramento adequados. Em alguns casos o não cumprimento dessas restrições pode acarretar prejuízos aos equipamentos e ao ambiente, assim como riscos à saúde humana

nos sistemas em que esses dispositivos são implementados (CHEN; HASAN; MOHAN, 2018). Para o caso de sistemas de sensores, não costuma haver atuação e portanto não podem causar diretamente danos, entretanto, dados errados ou faltantes podem levar a conclusões errôneas e ocasionar problemas.

A execução de análise de dados, controle e tarefas restritas pela temporização nos equipamentos de ponta (sensores, atuadores etc.) da rede IoT é a opção ideal e em alguns casos a única solução viável para cumprir esses quesitos (CHIANG; ZHANG, 2016).

Para fazer as medições, existem vários tipos de sensores disponíveis no mercado, e eles possuem diferentes interfaces para fornecer essas medições. As mais comuns usam interfaces de conversão analógica e também digitais, como RS232, I²C e SPI para transmitir os dados (MATTOLI et al., 2010). Além disso, esses dados podem ser entregues representados de diferentes formas, como sinais de tensão ou valores binários, por exemplo.

Essa grande diversidade de interfaces muitas vezes torna difícil a implementação de sistemas complexos ou distribuídos de monitoramento. Apesar da introdução do padrão IEEE 1451 buscar mitigar essa dificuldade, muitos sensores não são compatíveis com o padrão, e a adaptação pode se tornar complexa (MATTOLI et al., 2010).

A adição de um novo sensor a um nó IoT demanda que o software embarcado seja alterado, acarretando custos de desenvolvimento e testes. Em um sistema de aquisição de dados, há uma considerável similaridade entre a aquisição e o processamento dos dados dos diversos sensores que podem ser utilizados. Essa operação pode ser exemplificada nos seguintes passos:

- Inicializar interface;
- Inicializar sensor;
- Dar partida ao processo de aquisição;
- Esperar que a aquisição esteja completa;
- Ler o dado através da interface;
- Converter o dado para uma representação interna adequada (se necessário);
- Enviar o dado para um repositório central.

Assim, se o nó dispusesse de um conjunto de capacidades de interfaceamento e conversão, seria possível a implementação de um sistema que permitisse adição de sensores e/ou

a alteração de parâmetros sem a necessidade de reprogramação. Nesta abordagem, as alterações poderiam até mesmo serem feitas remotamente.

1.3 Objetivos

1.3.1 Objetivo geral

Desenvolver uma infraestrutura de desenvolvimento de *software* que permita a construção de sistemas de aquisição de dados em um contexto IoT, de forma robusta, flexível e simples evitando a reprogramação do *software* embarcado, permitindo a troca de parâmetros de operação inclusive a adição e remoção de sensores.

1.3.2 Objetivos específicos

Para atingir o objetivo geral, a infraestrutura de desenvolvimento busca cumprir com os seguintes objetivos específicos:

- Utilize de ferramentas de baixo custo;
- Permita a interface com diferentes tipos de sensores de forma simples e modular;
- Permita o armazenamento em um repositório central (banco de dados);
- Permita a transmissão dos dados dos sensores remotamente;
- Permita a alteração de parâmetros do sensor e/ou troca destes sem necessidade de reprogramação;
- Seja capaz de cumprir com restrições de períodos de amostragem;
- Permita a alteração de parâmetros relacionados aos sensores remotamente;
- Seja capaz de armazenar (dentro de limites) os dados no caso de falhas de comunicação e enviá-los quando esta for restabelecida.

Esses pontos contribuem para uma solução que abrange amplamente áreas de atuação da tecnologia IoT. A utilização de ferramentas de baixo custo permitem a integração de mais objetos na mesma aplicação e a capacidade de interface com diversos sensores

permite a aquisição de parâmetros diferentes. Além disso, a capacidade de transmissão remota contribui para a disseminação dos equipamentos e facilidade na implementação e, finalmente, a utilização de um sistema operacional de tempo real permite a capacidade de cumprir com períodos de amostragem restritos e auxilia no armazenamento dos dados no caso de falhas, tornando a aplicação mais confiável e eficaz.

1.4 Metodologia

Uma pesquisa sobre as arquiteturas e ferramentas utilizadas em outros projetos relacionados a IoT deve ser realizada, buscando opções para a implementação deste projeto. Da mesma forma, alguns microcontroladores foram analisados, e as considerações feitas são contempladas no Capítulo 2.

Como sensores são transdutores, convertem um tipo de grandeza para outro para permitir medição. Assim a forma como entregam esse dado é variada, e podem ser classificadas da seguinte maneira:

- Domínio analógico, onde os dados são entregues na forma de sinais de tensão ou corrente, por exemplo;
- Domínio digital, onde os dados são representados por valores binários, trem de pulsos ou frequência, por exemplo.

Além disso, podem ser conectadas a um microcontrolador através de diferentes interfaces. Podem ser destacadas as mais comuns:

- Barramento, como é o caso de um sensor de conversão analógica digital integrado a microcontroladores;
- Serial síncrona, onde há sincronização através de relógio, como é o caso das interfaces SPI, I²C, CAN;
- Serial assíncrona, onde não há sincronização através de relógio, como é o caso da interface RS232.

Entretanto, alguns sensores possuem protocolos de comunicação próprios, que precisam ser implementados para comunicação com o microcontrolador, como é o caso do sensor

DHT22, desenvolvido pela Aosong Electronics Co.,Ltd. Esse sensor entrega seus dados no formato binário através de comunicação serial assíncrona, usando uma codificação de largura de pulsos para representar os bits, o que o torna uma opção ideal para demonstrar a aplicabilidade do projeto.

Uma arquitetura deve ser proposta, que atenda aos requisitos apresentados e seja capaz de fazer a aquisição de dados de diferentes sensores. Essa arquitetura deve usar as ferramentas analisadas em conjunto de forma adequada, alterando os parâmetros, como intervalo de aquisição e formato de entrega dos dados, dos sensores e configurações de interfaces do microcontrolador para cumprir com os objetivos.

Para a validação da proposta foi escolhida uma área de atuação. Muito comum nas aplicações IoT é o monitoramento de dados de processos e ambientes. Parâmetros como pressão, temperatura, umidade, vazão e consumo de energia são medidos e analisados na busca de, por exemplo, garantir um funcionamento adequado de um equipamento, conforto de um ambiente em uma residência ou escritório e buscar formas de economizar gastos.

Finalmente, um teste das funcionalidades deve ser realizado e descrito para validação, demonstrando a flexibilidade da solução. O teste é realizado implementando a arquitetura em um dispositivo de fácil acesso, através do uso de uma placa de desenvolvimento e um dispositivo de comunicação, representando o *hardware* utilizado em nós IoT.

A escolha de um sensor que possui um protocolo próprio, exigindo a adaptação da sua interface, e um sensor comum conversor analógico digital, aquisição dos dados desses sensores, transmissão remota e alteração dos parâmetros dos sensores compõem o teste para essa finalidade.

1.5 Estrutura do trabalho

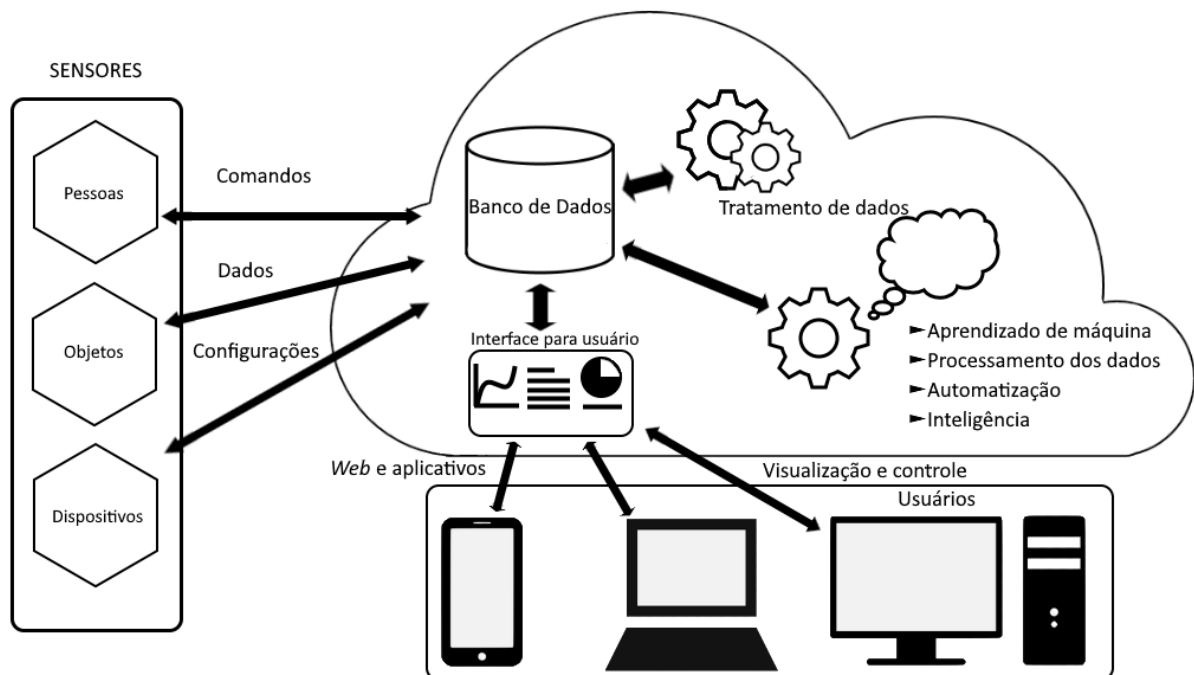
Este capítulo apresenta o tema e alguns desafios relacionados à proposta. O Capítulo 2 apresenta as especificações das ferramentas escolhidas, suas características, algumas limitações e alternativas. Além disso algumas das considerações das escolhas dessas ferramentas são apresentadas. No Capítulo 3 é descrita como é feita a união dessas ferramentas, a implementação do *software*, assim como o funcionamento do *software*. É descrita também a utilização dos arquivos de configuração e interpretador, para que finalmente no Capítulo 4 sejam descrito os testes realizados e no Capítulo 5 são apresentadas as conclusões finais do projeto.

2 SISTEMA DISTRIBUÍDO DE AQUISIÇÃO DE DADOS

2.1 Camadas de estudo em IoT

Internet das coisas é uma tecnologia que busca promover a integração entre os mais diversos objetos, equipamentos e dispositivos que fazem interface com o mundo real através de sensores e atuadores. Automatizar ou melhorar processos, aumentar eficiência, auxiliar na logística, promover bem-estar, são alguns objetivos da tecnologia. A Figura 2 é um diagrama que exemplifica uma arquitetura comum de um sistema IoT, descrevendo as interações entre as partes do sistema.

Figura 2 – Arquitetura de um sistema IoT.



Fonte: Produção do próprio autor.

Xu, He e Li (2014) sugerem que sistemas como o do diagrama da Figura 2 podem ser divididos em 4 camadas distintas, onde as “coisas” de onde se obtêm os dados pertencem à camada de sensores, o processamento, armazenamento e inteligência dos dados ficam na camada de serviços, a visualização dos resultados e controle na camada de interface e a comunicação entre as partes do sistema, bem como a transmissão dos dados na camada de redes. Suas colocações são resumidas no Quadro 1.

Por ser a camada mais relacionada ao *hardware* a ser implementado em uma aplicação

Quadro 1 – Camadas das aplicações em IoT.

Camada	Descrição
Camada de sensoria- mento	União do <i>hardware</i> existente (RFID, sensores, atuadores, etc.) responsável por controlar e/ou adquirir dados do mundo físico.
Camada de redes	Responsável pela comunicação básica da rede e transferência de dados de forma remota ou com fio.
Camada de serviços	Nessa camada é desenvolvida a integração dos dados, seu armazenamento e fornecimento de acordo com a lógica de negócio da aplicação.
Camada de interface	Responsável por simplificar a gerencia e interação com os serviços, além de facilitar a conexão dos objetos.

Fonte – Produção do próprio autor.

IoT, é principalmente nessa etapa, devido à variedade de objetos, dispositivos e dados dessa camada, que questões como custo, tamanho, recurso e consumo de energia devem ser levados em consideração, devendo serem projetados para minimizar custos e recursos utilizados (LI; XU; ZHAO, 2015).

A camada de redes é a responsável pela transmissão dos dados para os serviços mais complexos, e normalmente permite a integração de dispositivos, tanto com fios quando remotos. A confiabilidade da transmissão, qualidade e, em algumas aplicações, privacidade, são parâmetros que aplicações nessa camada devem levar em consideração (LI; XU; ZHAO, 2015).

As camadas de serviços e de interface englobam o tratamento dos dados e interação do usuário com os serviços, de forma a tornar o uso da aplicação mais fácil (LI; XU; ZHAO, 2015).

A partir desses pontos, este projeto classifica-se então como uma aplicação na camada de sensoriamento, já que as restrições apresentadas se enquadram nessa camada, e, apesar de contemplar alguns pontos de outras camadas como parâmetros da transmissão remota, e o recebimento desses dados em uma aplicação, esses não fazem parte do foco da solução.

2.2 Frameworks para IoT

Os sistemas operacionais voltados para IoT mais utilizados nas pesquisas recentes são: TinyOS, Contiki, RIOT, Zephyr, MbedOS e Brillo, e suas características distintas os tornam ideais para diferentes aplicações, no caso de WSN, o mais utilizado era TinyOS, porém vem perdendo relevância devido ao seu desenvolvimento estar lento (ZIKRIA et al., 2019). Dessa forma, a utilização de outros sistemas operacionais, como uCOS ou FreeRTOS

no tema de redes de sensores sem fio é interessante por contribuir para exploração de sistemas operacionais capazes de tempo real no contexto de IoT.

Além disso, algumas soluções que permitem múltiplos sensores foram analisadas:

Rodriguez-Zurrunero, Tirado-Andrés e Araujo (2018) propõem o desenvolvimento do sistema operacional em tempo real YetiOS que é baseado em FreeRTOS. O sistema permite a programação de aplicativos e algoritmos adaptativos para otimizar aplicações em sensoriamento sem fio. Esse projeto demonstra a viabilidade do uso de FreeRTOS, destacando a sua capacidade de manter as restrições e também permitir economia de energia por meio do modo de baixo consumo do sistema operacional. Entretanto adiciona complexidade e utiliza quantidade de memória relativamente alta, limitando aplicabilidade.

Algumas soluções que envolvem *hardware* são propostas. Mattoli et al. (2010) utilizam um PSoC (*Programmable System on Chip*) e TEDS (*Transducer Electronic Datasheet* ou IEEE 1451). Mikhaylov e Huttunen (2014) desenvolvem a interface *Intelligent Modular Periphery Interface* (IMPI). Os projetos permitem grande flexibilidade de sensores, ao custo da implementação do *hardware*.

2.3 Sistemas operacionais

Sistemas que exigem que o tempo de execução de tarefas seja pré-definido são classificados como sistemas de tempo real. Esses sistemas podem ser diferenciados em sistemas rígidos (do inglês *hard real time*), onde a falha na execução da atividade dentro do intervalo determinado é considerada uma falha catastrófica, e suaves (do inglês *soft real time*), onde uma falha degrada o desempenho do sistema mas não o torna inoperante (LABROSSE, 2010).

Programas mais simples podem ser divididos em aplicações que são executadas em primeiro e segundo plano. Em segundo plano são executadas as *tasks*, ou tarefas, que cumprem as funções desejadas do sistema em um *loop*. Durante a execução do *loop* podem ocorrer eventos assíncronos, que são tratados pelas interrupções. Tarefas críticas devem ser executadas imediatamente, e, portanto, devem ser executadas em primeiro plano, sendo esse o plano das interrupções (LABROSSE, 2010).

O núcleo, ou *kernel* de tempo real é responsável pelo *multitasking*, processo onde várias *tasks* são executadas simultaneamente, o que é possível devido a gerenciamento do tempo e dos recursos do microcontrolador, como CPU, para cada *task*. Esse gerenciamento é

realizado pelo kernel de tempo real através da definição de prioridades, preemptividade e sincronismo (LABROSSE, 2010).

O uso de um *kernel* com essas capacidades permite o desenvolvimento de uma solução eficiente e, mesmo que não orientada para as restrições rígidas de tempo real, capaz de aquisição de dados de forma precisa e consistente. Já que permite o desacoplamento das funções de aquisição de dados e comunicação, e através do processo de execução simultânea, o *kernel* evita que essas funções bloqueiem a execução, interferindo minimamente umas com as outras. Essa modularidade e flexibilidade é obtida ao custo da necessidade de mais memória no sistema, já que para cada tarefa é necessário uma pilha para operação. Entretanto, processadores com pilha endereçável e memória suficiente para diversas pilhas são competitivos em termos de custo com os tradicionais.

Liu et al. (2014) comparam os vários sistemas operacionais que são comumente utilizados no contexto de IoT. Alguns dados da comparação são exibidos no Quadro 2.

Quadro 2 – Suporte a tempo real em sistemas operacionais comuns em IoT.

Sistema Operacional	Suporte ao tempo real
TinyOS	Não ou tem suporte ruim a partir de adaptação
Contiki	Não ou tem suporte ruim a partir de adaptação
SOS	Não
MantiOS	Não por utilizar algoritmo RR (<i>Round-Robin</i>)
openWSN	Não
SimpleRTJ	Não por utilizar algoritmo RR
uCOS	Sim, com algoritmo RMS (<i>Rate-Monotonic Scheduling</i>) preemptivo
FreeRTOS	Sim com suporte a algoritmo preemptivo e cooperativo

Fonte – Produção do próprio autor.

Dentre os sistemas apresentados, os que possuem as capacidades que foram consideradas interessantes para o projeto são o uCOS e FreeRTOS. Além disso, são sistemas bastante utilizados facilitando a implementação (UNGUREAN, 2020).

FreeRTOS é um sistema operacional de tempo real de código aberto. Sua licença permite uso em qualquer situação. Ele ocupa normalmente entre 6k e 12k *bytes* e possui uma comunidade ativa, sendo amplamente utilizado (FREERTOS, 2020).

uCOS por outro lado é um sistema operacional que possuía licença comercial, porém durante o desenvolvimento do projeto, o sistema operacional ficou disponível no Github. Normalmente ocupa entre 6k e 24k *bytes* de memória. Possui uma documentação bem elaborada e é certificado para aplicações onde a segurança é crítica (WESTONEMBEDDED, 2021).

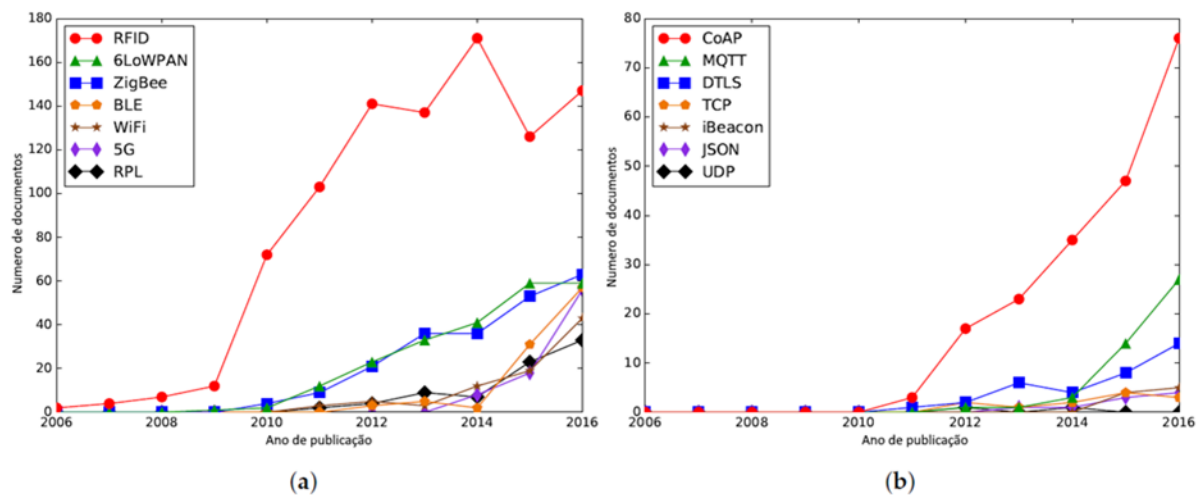
Apesar de ambos os sistemas cumprirem com as demandas do projeto, a decisão da

ferramenta a ser utilizada foi o FreeRTOS, pois sua licença de uso geral permitia mais abrangência do projeto e a comunidade de código aberto é considerada um atrativo.

2.4 Protocolos de comunicação remota

Ruiz-Rosero et al. (2017) fazem um levantamento dos protocolos de comunicação que estão sendo utilizados na literatura recente. De acordo com a Figura 3, nota-se uma tendência de crescimento nos documentos que pesquisam a utilização de BLE (*Bluetooth Low Energy*), 5G e Wi-Fi no contexto de IoT, e o amplo uso de MQTT (*Message Queuing Telemetry Transport*) e CoAP (*Constrained Application Protocol*).

Figura 3 – Quantidade de documentos por protocolos de (a) meio de comunicação e de (b) camada de aplicação em diferentes anos.



Fonte: Ruiz-Rosero et al. (2017).

Nota: Adaptado pelo autor.

Quanto ao meio de comunicação, apesar de outras tecnologias bastante difundidas, como ZigBee, Bluetooth e RFID (*Radio-frequency identification*) foi optado pela utilização do Wi-Fi por alguns motivos:

- A tecnologia já é bastante difundida, presente na maioria dos locais onde o projeto pode ser aplicado, principalmente residências;
- Outras tecnologias envolveriam utilização de mais *hardware* específico;
- Familiaridade com a tecnologia, facilitando o desenvolvimento.

Quanto aos protocolos da camada de aplicação, os protocolos que chamaram atenção, não somente por serem bastante explorados, mas também por serem projetados para redes com restrições, foram CoAP e MQTT;

CoAP é um protocolo baseado em *User Datagram Protocol* (UDP), desenvolvido pela *Internet Engineering Task Force* (IETF), visando substituir o uso de *Hypertext Transfer Protocol* (HTTP). Possui um pequeno *overhead*, porém é baseado em UDP e, portanto, não garante a entrega de pacotes (CHEN; KUNZ, 2016). Isto implica que deve haver um mecanismo de verificação de entrega no uso desse protocolo.

MQTT é baseado em *Transmission Control Protocol* (TCP), desenvolvido pela IBM. O protocolo opera no formato publicador e assinante. Nesse modelo, os clientes podem “publicar” dados em um determinado tópico para um servidor e também “assinar” tópicos para receber dados do servidor. O MQTT combina um *overhead* relativamente alto com a alta robustez do protocolo TCP (CHEN; KUNZ, 2016).

Dentre as opções, foi optado pelo uso do protocolo MQTT, tendo em vista os seguintes pontos:

- Garante a entrega de pacotes por utilizar TCP;
- O modelo publicador e assinante é atrativo no contexto da proposta, onde vários sensores devem operar simultaneamente;
- Permite comunicação bidirecional;
- Bastante difundido com diversas soluções de código aberto disponíveis.

2.5 Microcontrolador

Muitos projetos voltados ao IoT têm sua prototipagem elaborada em placas de desenvolvimento como Arduino e ESP32. O Arduino, apesar do suporte ao FreeRTOS, possui pouca memória RAM, limitando a aplicabilidade já que cada tarefa exige memória para a pilha.

Quanto a ESP32, apesar do suporte às ferramentas de tempo real e já possuir integração com rede Wi-Fi, por ter de gerenciar essa tarefa em conjunto com as funcionalidades que são adicionadas à plataforma, a solução torna-se mais complexa e limitada, já que o recurso computacional fica dividido.

Microcontroladores modernos baseados em ARM Cortex-M e MSP430, por exemplo, possuem características como:

- Frequência de relógio variável (permitindo economia de energia durante operação);
- Configuração flexível de periféricos (desabilitar o periférico para economia);
- Maior capacidade de endereçamento de memória;
- Pilha endereçável e que pode ser remanejada, facilitando troca de contexto;
- Conjunto de instruções otimizados para a linguagem C;
- Mecanismos eficientes de interrupção.

Quando aliados ao desenvolvimento desses sistemas com custo cada vez menor, esses pontos permitem que a solução seja flexível com relação às restrições de sistemas tradicionais de 8 bits por exemplo.

Placas de desenvolvimento que sejam baseadas em processador ARM da família Cortex-M, projetadas para sistemas de baixo custo e possuem um baixo consumo (ARM, 2020) são dispositivos compatíveis com o FreeRTOS, e portanto são opções que atendem os requisitos. São os casos dos modelos STM32, da fabricante ST Microelectronics¹ e EFM32 Gecko, da fabricante Silicon Labs².

Para a transmissão remota, a utilização de outro dispositivo, apenas para gerenciar a comunicação remota atuando como adaptador de uma MCU (*Microcontroller Unit*) para transmissão através de um meio de comunicação como 3G, LoRa (*Long Range*), ou qualquer outro da pesquisa citada anteriormente. Isso evita que o mesmo dispositivo fique sobrecarregado precisando executar as rotinas de transmissão ao mesmo tempo que faz a aquisição dos sensores, simplificando o desenvolvimento e melhorando a performance, com a desvantagem de aumentar o custo, tamanho e consumo.

Dessa forma, pode ser utilizada uma versão mais simples da família ESP citada anteriormente, como a ESP8266, que é um dispositivo adaptado para baixo consumo, com design voltado para o IoT no contexto industrial e que possui toda a configuração necessária para comunicação Wi-Fi (ESPRESSIFSYSTEMS, 2020).

¹ <https://www.st.com/en/microcontrollers-microprocessors/stm32-32-bit-arm-cortex-mcus.html>

² <https://www.silabs.com/mcu/32-bit>

Assim, a partir dos pontos apresentados, a placa utilizada no desenvolvimento do projeto foi a placa EFM32GG-STK3700 em conjunto com uma ESP8266 para adaptação à transmissão Wi-Fi. A placa EFM foi escolhida por cumprir com as demandas discutidas anteriormente, e pela disponibilidade, já que são utilizadas na disciplina de Sistemas Embarcados 2, na Universidade Federal do Espírito Santo, enquanto que a ESP possui a capacidade de comunicação Wi-Fi desejada, além de ser bastante difundida em aplicações disponibilizadas na *internet*.

Para a utilização do protocolo MQTT com a ESP8266, foi utilizada a implementação de código aberto *pubsubclient*, desenvolvida por Nick O’Leary, disponível em sua página do Github³. Essa implementação permite o uso da interface de desenvolvimento da Arduino para compilação e *download* do binário.

As bibliotecas para configuração do *hardware* do microcontrolador são disponibilizadas pelos fabricantes e o manual do FreeRTOS é disponibilizado de forma gratuita. Finalmente, além de exemplos de implementações, um conjunto de ferramentas para compilação e *download* do binário no microcontrolador escolhido é fornecido pelo professor Hans Jorg Andreas Schneebeli em sua página no Github⁴.

2.6 Arquitetura da solução

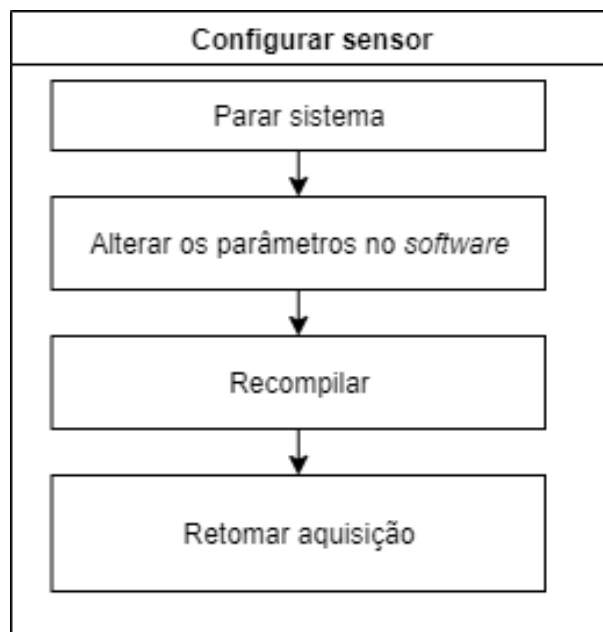
No contexto apresentado, a aquisição dos dados pode ser feita através de diversos sensores, para diversos parâmetros. Muitos desses sensores, apesar de serem especificados para situações e condições diferentes, possuem alterações em questões como taxa de aquisição, formato de entrega dos dados, ou quantidade de dados, enquanto utilizam de uma das interfaces de comunicação comuns, citadas anteriormente.

Muitas vezes, uma arquitetura que precisa fazer interface com esses diversos sensores implementa um modelo onde as características específicas de cada sensor são armazenadas previamente no dispositivo e a sua utilização deve ser também definida previamente e carregada no *software*. Dessa forma alterações de configurações simples, como taxa de aquisição, troca de velocidade de comunicação no caso de uma interface serial, ou endereço de um dispositivo I²C, envolveriam o acesso ao dispositivo e alterações diretamente no *software*, exigindo ferramentas e recompilação. A Figura 4 mostra o passo a passo da alteração de um sensor nesse modelo.

³ <https://github.com/knolleary/pubsubclient>

⁴ <https://github.com/hans-jorg/efm32gg-stk3700-gcc-cmsis>

Figura 4 – Alteração de parâmetros no modelo com sensores pré determinados.



Fonte: Produção do próprio autor.

Uma arquitetura mais genérica pode ser obtida adicionando duas capacidades ao dispositivo: A capacidade de comunicação com os sensores a partir dos canais que eles utilizam, e a capacidade de converter os dados recebidos para um formato padronizado que possa ser utilizado no sistema.

A primeira capacidade envolve a configuração direta dos parâmetros dos periféricos presentes no dispositivo, já que eles são o canal de comunicação com o sensor. Dessa forma a utilização de outros sensores com a mesma interface, que possuam parâmetros diferentes é feita apenas com a alteração dessas configurações.

A segunda capacidade envolve como os dados são fornecidos pelo sensor. A utilização de sensores que retornam dados no formato decimal, hexadecimal ou binário, por exemplo, pode ser feita alterando qual a função de conversão necessária para determinado tipo.

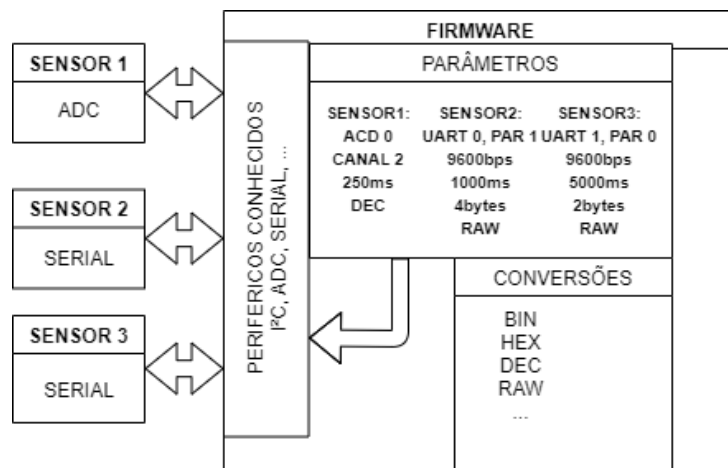
Em contradição ao primeiro modelo, nessa arquitetura um sensor é definido não pelas suas características específicas, e sim por um conjunto de capacidades, ou seja, a configuração do canal de comunicação, com os parâmetros para o periférico, e a função de conversão do formato retornado por esse sensor. Assim, a utilização de um outro sensor é feita a partir da seleção das capacidades já conhecidas pelo dispositivo.

Além disso, as capacidades do dispositivo podem ser ampliadas, adicionando novas funções de conversão conforme a necessidade, e implementando camadas de aplicação para os

periféricos de forma a emular interfaces não convencionais. Dessa forma, um novo sensor pode já ter as ferramentas necessárias para comunicação no dispositivo, e recompilação e alterações no *software* são necessárias apenas quando tipos não convencionais são adicionados.

A Figura 5 mostra um diagrama exemplificando o modelo das capacidades.

Figura 5 – Arquitetura de capacidades.



Fonte: Produção do próprio autor.

A adição de um programa interpretador e um conjunto de arquivos de configuração simplifica a utilização das capacidades. Para cada sensor podem ser registrados nos arquivos o conjunto de capacidades no dispositivo a serem utilizados, com as configurações dos periféricos e a função de conversão, além de ser possível registrar limitações para os tipos.

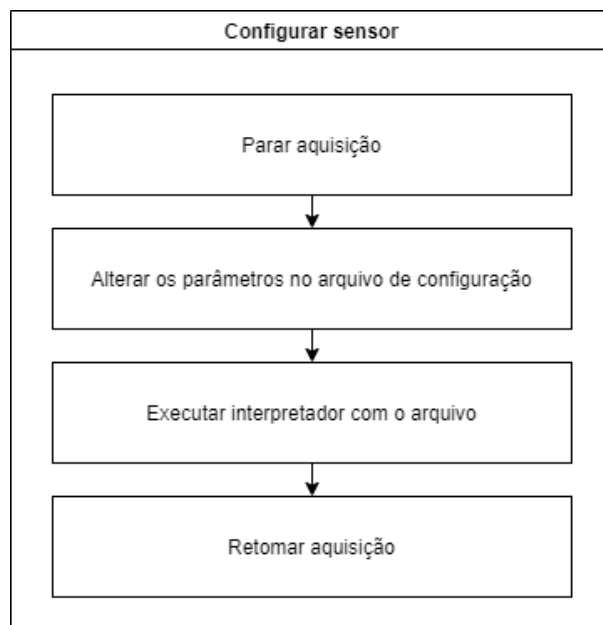
O acesso ao microcontrolador para alterar os parâmetros das interfaces já implementadas torna-se desnecessário ao fazer a adaptação do microcontrolador para comunicação remota através de um dispositivo capaz de receber e enviar dados. A configuração dos periféricos e a ativação de sensores pode ser feita também de forma remota, utilizando arquivos salvos em um repositório central, e controlando os sensores através de comandos.

A utilização da arquitetura do sistema operacional de tempo real, permite que a operação seja feita de forma robusta, através da aquisição de diversos sensores simultaneamente à recepção das alterações.

A Figura 6 mostra o passo a passo para configuração do sensor.

A implementação de um sistema de associação ao dado do sensor com o tempo da sua aquisição, através de um sistema de numeração dos pacotes, ou de contagem de tempo

Figura 6 – Alteração de parâmetros modelo de acesso à periféricos.



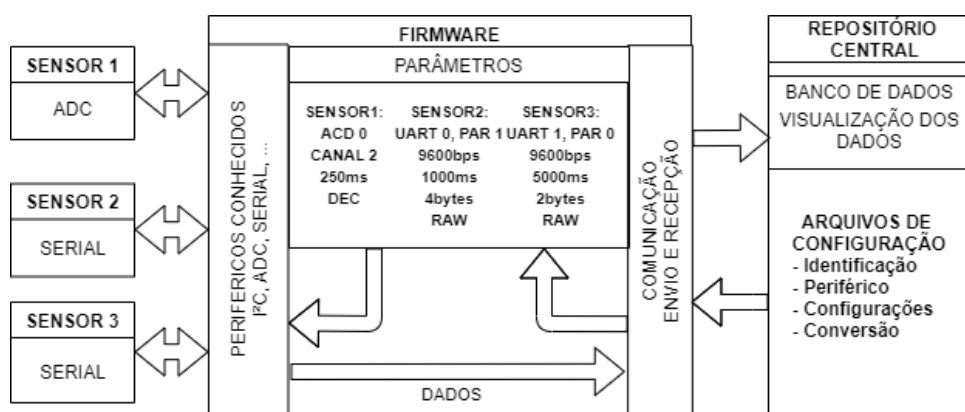
Fonte: Produção do próprio autor.

no próprio microcontrolador e utilização das ferramentas de filas do sistema operacional de tempo real permitem que mesmo que haja uma indisponibilidade momentânea da comunicação, o dado do sensor possa ser identificado posteriormente em um banco de dados, e sua aquisição estará associada a intervalos constantes.

O dado a ser transmitido é encapsulado no protocolo da camada de aplicação e enviado para um repositório central onde deve ser tratado de acordo com a necessidade da aplicação. O repositório deve ser capaz de enviar comandos de volta para o microcontrolador, onde farão a alteração dos parâmetros dos sensores.

A Figura 7 mostra um diagrama da arquitetura completa da solução.

Figura 7 – Arquitetura completa da solução.



Fonte: Produção do próprio autor.

Assim, tem-se uma arquitetura onde a implementação de cada vez mais funções de conversão e aplicações para interfaces não convencionais adicionam compatibilidade à solução. A utilização de arquivos de configuração simplifica a alteração e o uso dos sensores, evitando programação e recompilação de *software*, desde que o sensor utilize de capacidades já conhecidas. A transmissão remota dos arquivos evita ser necessário o acesso ao dispositivo, ao mesmo tempo que permite ao usuário a configuração dos sensores apenas pela alteração de arquivos armazenados em um repositório central. A operação do sistema é robusta por utilizar um sistema operacional de tempo real, de forma que a recepção de comandos e envio dos dados interferem minimamente entre si.

O detalhamento do uso das ferramentas, o funcionamento da rotina de comunicação, o formato do encapsulamento para transmissão e a utilização dos arquivos de configuração são descritos nos capítulos a seguir.

3 IMPLEMENTAÇÃO

3.1 Definição dos sensores

Na área de atuação escolhida, ou seja, o monitoramento de dados de processos e ambientes, é comum a aquisição de dados como pressão, temperatura, umidade, vazão e consumo de energia. São esperados para a plataforma, portanto, sensores que fazem aquisição de dados escalares, permitindo sua manipulação de forma numérica.

A implementação de um sensor pode ser realizada através de um *driver*. Isso implica na pré compilação dos parâmetros do sensor no microcontrolador, relacionados a um nome que será utilizado para identificar o sensor e fazer sua utilização. Para cumprir esse objetivo, os sensores são representados por um conjunto de variáveis, que identificam seu tipo e configurações específicas, e uma função, que executa a sequência de ações necessária para realizar a aquisição dos dados, de forma que os dados são retornados pelo *driver* após serem tratados. Nesse modelo, a implementação é realizada facilmente, precisando apenas identificar o sensor. Entretanto, as características de cada sensor devem estar salvas em memória, o que torna a solução bastante limitada.

O projeto então buscou a implementação dos sensores de forma mais dinâmica, dando ao arquivo de configuração a capacidade de acessar e alterar parâmetros do periférico diretamente. Nesse modelo, parte das variáveis que definem o sensor são os parâmetros do periférico. Em um sensor analógico, por exemplo, alterar qual o número do periférico, caso haja mais de um, o canal de aquisição em que o sensor está conectado, a tensão de referência e a resolução desejada são exemplos de parâmetros que podem ser modificados.

Dessa forma, durante a execução do programa as variáveis que representam as características do sensor podem ser alteradas, mudando o comportamento do periférico para cada sensor, evitando que os sensores precisem estar pré compilados no dispositivo. Funções de conversão podem ser utilizadas para alterar a forma como os dados são recebidos, como por exemplo, formato binário, decimal, hexadecimal ou sem tratamento, denominado *raw*. Essas funções devem ser armazenadas no microcontrolador, entretanto não precisam ser relacionadas a um sensor especificamente e quando em conjunto com as configurações do periférico, definem o sensor.

Assim, a inclusão de um novo tipo de sensor implica na configuração do periférico com os

parâmetros necessários para aquisição dos dados. A quantidade de sensores fica limitada ao *hardware* em que será implementado, devido a fatores como capacidade de processamento e quantidade de periféricos.

3.2 Transmissão dos dados

No ponto de medição, a comunicação entre os dispositivos embarcados é feita a partir da transmissão no formato serial utilizando o protocolo *Universal Asynchronous Receiver-Transmitter* (UART). Esse protocolo é muito comum nessas plataformas e sua utilização é bem direta, permitindo simplicidade e facilidade na depuração através da transmissão de 8 *bits* (um *byte*) no formato de codificação *American Standard Code for Information Interchange* (ASCII) que transforma os *bytes* em caracteres alfanuméricos para que possam ser lidos. Para manter a consistência, os dados enviados remotamente também seguem essa codificação.

As mensagens entre os dispositivos embarcados são enviadas no formato de pacotes, definidos por marcadores que indicam o início e o fim de um pacote contendo os dados que podem ser a aquisição dos sensores ou os comandos de configuração.

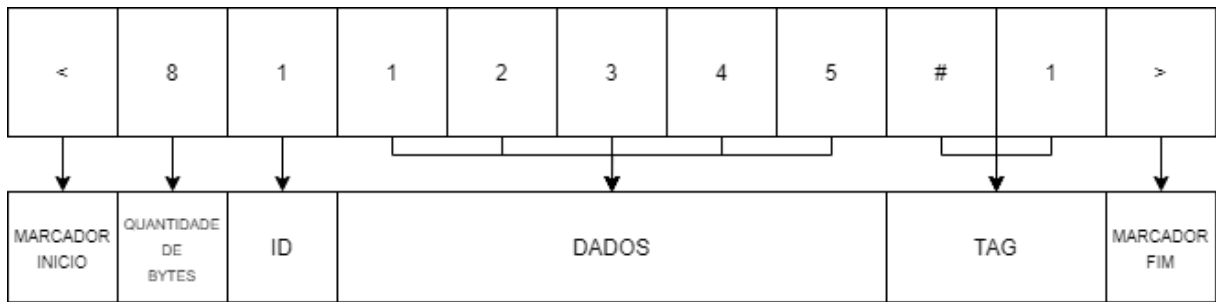
Para o envio dos dados da aquisição, o formato a ser seguido é: o primeiro *byte* após o marcador identifica a quantidade de dados que foram enviados pelo sensor (contando o *byte* de identificação), o segundo *byte* representa o sensor, a partir do seu parâmetro de identificação e os demais o valor retornado pelo sensor, seguido pelo marcador de *tag*, com o número do pacote, que representa o momento da sua aquisição e finalmente, o marcador de fim do pacote. Dessa forma a extração dos dados é feita de forma simples, já que se obtém a quantidade de dados e a posição em que eles começam é fixa, e é possível relacionar o dado ao momento de sua aquisição.

A função de conversão deve, portanto, adequar seus dados seguindo essa formatação, convertendo os valores após a aquisição para uma sequência de caracteres, calculando a quantidade de caracteres que serão enviados, e inserir esses dados nas posições corretas.

A Figura 8 mostra o exemplo de um dado enviado pelo sensor 1, cujo valor medido é um número de 5 dígitos, e é o segundo pacote enviado.

Os pacotes de configuração seguem o mesmo formato de pacotes encapsulados por marcadores de início e fim de dados, porém seu conteúdo possui um formato diferente: Após o primeiro marcador o primeiro *byte* é o identificador de qual sensor está sendo configurado,

Figura 8 – Exemplo de um pacote de dados de um sensor.

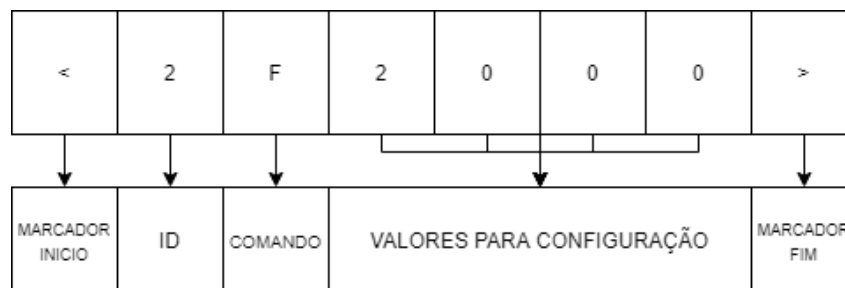


Fonte: Produção do próprio autor.

o segundo *byte* é qual parâmetro do sensor está sendo alterado e os demais são o valor. Dessa forma, a recepção e o tratamento da configuração são mais simples, permitindo maior granularidade na configuração e quantidade de dados é bem definida para cada tipo de configuração a ser realizada.

A Figura 9 mostra um exemplo de comando alteração de intervalo de aquisição do sensor 2, onde “F” é o identificador de comando de frequência de aquisição e o valor está em milissegundos.

Figura 9 – Exemplo de um pacote de configuração para um sensor.



Fonte: Produção do próprio autor.

3.3 Arquitetura da solução

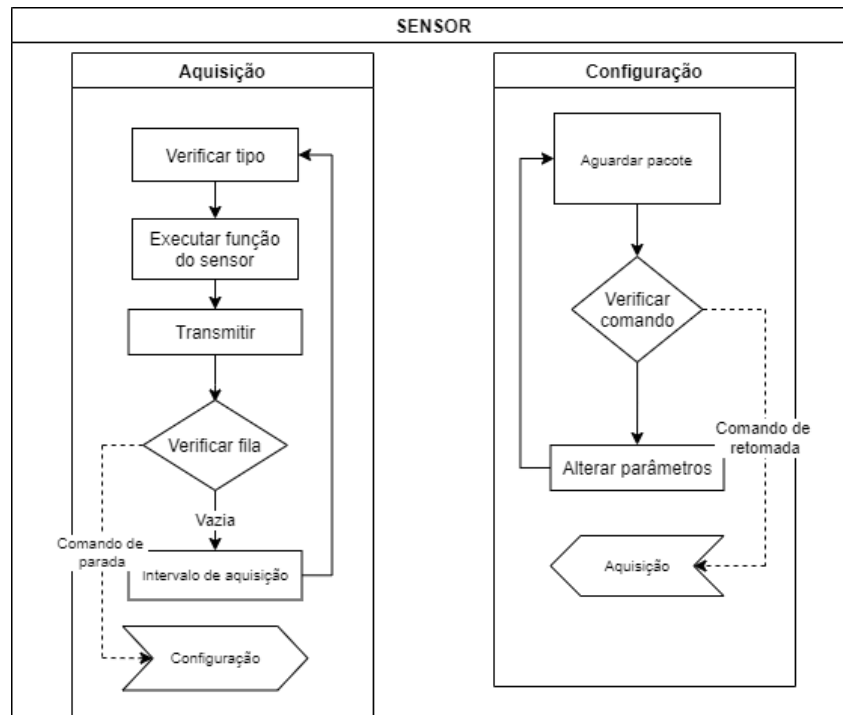
Para permitir as alterações e o controle dos sensores, o programa opera de forma similar a uma máquina de estados: Em um estado o processamento é utilizado para obter os dados do sensor e transmitir enquanto que no outro o processamento é utilizado para fazer as alterações dos parâmetros. Dessa forma, a implementação é mais simples, evitando interferência no processo de aquisição de dados, e facilitando o tratamento dos pacotes de configuração já que durante esse período o programa executa exclusivamente essa função.

Como é esperado mais de um sensor operando simultaneamente e sendo gerenciado pelo

sistema operacional, é preciso um sistema de filas para transmissão de mensagens entre as funções responsáveis pela aquisição de dados e pela recepção de comandos entre os sensores.

A Figura 10 mostra um diagrama que representa esse processo.

Figura 10 – Diagrama da solução.



Fonte: Produção do próprio autor.

3.4 Recepção dos dados

Como a recepção e o tratamento dos dados não fazem parte do escopo principal do projeto, uma solução simples apenas para visualização foi implementada utilizando um agente de serviços para MQTT. A função desse agente é permitir comunicação entre dispositivos compatíveis com esse protocolo em uma rede, além de fornecer uma interface que permita inscrever ou publicar em tópicos.

A interface do agente de serviços para publicação em um tópico foi utilizada para o envio dos comandos de configuração, de forma que esse comando seja recebido por outro dispositivo que esteja inscrito nesse tópico via Wi-Fi e então seja feito o tratamento. Da mesma forma, o agente pode se inscrever no tópico de cada sensor presente na solução para receber e apresentar seus dados de forma remota.

3.5 Arquivo de configuração e interpretador

Para enviar os comandos ao microcontrolador, foi elaborado um interpretador que lê os arquivos com os parâmetros desejados. O arquivo deve identificar o tipo do sensor (ADC (*Analog-to-Digital Converter*), SPI, I²C, etc), a frequência desejada e as configurações do periférico. Esses arquivos seguem o formato JSON, que é uma linguagem de representação de dados que facilita a leitura dos dados por humanos. Nesse formato, conjuntos de variáveis formam pares de nomes e valores, que são organizados em objetos.

A configuração de um sensor envolve dois arquivos de configuração: um deles para as configurações desejadas para esse sensor, e outro denominado de arquivo de especificações. Esse arquivo não deve ser alterado pelo usuário, e define para cada tipo de sensor as variáveis compatíveis com a placa, traduzindo o tipo do sensor para o periférico desejado e os valores permitidos, por exemplo. Dessa forma é possível impedir configurações erradas, como por exemplo a escolha de pinos que não são do periférico.

O interpretador lê o arquivo de especificações, e também o do sensor que será configurado, que é indicado pelo usuário. Do primeiro arquivo, é verificado se aquele sensor possui compatibilidade com o programa, verificando sua existência na lista de especificações, e registrando suas limitações e o seu identificador de tipo. Do segundo arquivo, o interpretador verifica qual o sensor que será alterado no programa, através do identificador fornecido pelo usuário, e lê as variáveis que serão alteradas e compara com os limites.

Se não há inconsistências, o interpretador gera os comandos de configuração e envia para o microcontrolador. A Figura 11 mostra um exemplo do formato dos arquivos para um sensor ADC.

Além disso, o interpretador é capaz de enviar os comandos de parada e de retomada para o sensor.

3.6 Sensores com protocolos comuns

Para demonstrar a aplicabilidade do protótipo utilizando a transmissão dos parâmetros através do arquivo de configuração, foi realizada a aquisição de dados de um sensor analógico, através da configuração da interface ADC do microcontrolador, alterando a resolução e o canal utilizado pelo periférico para aquisição.

Figura 11 – Arquivos de configuração de ADC.

SPEC.json	ADCx.json
<pre>{ "ADC": { "TIPO": 1, "FREQUENCIA": { "MIN": 250, "MAX": 0 }, "NUMERO": [0], "CANAL": [0,1,2,3,4,5,6,7], "REFERENCIA": ["VDD"], "RESOLUCAO": [6,8,12], "CONVERSAO": ["RAW","BIN","DEC"] } }</pre>	<pre>{ "ADC": { "NUMERO": 0, "FREQUENCIA": 250, "CANAL": 6, "REFERENCIA": "VDD", "RESOLUCAO": 12, "CONVERSAO": "DEC" } }</pre>

Fonte: Produção do próprio autor.

Os parâmetros para o periférico são salvos em uma estrutura na tarefa, e alterados no recebimento das configurações. Dessa forma, o canal e as demais variáveis são extraídos do arquivo de configuração especificado pelo usuário. A configuração dos registradores e a execução da aquisição utilizam esses parâmetros salvos e os dados retornados pelo ADC são então utilizados na função de conversão e transmitidos.

Como são sete canais disponíveis na placa, e ela só possui um conversor, é definido no arquivo *SPEC.json* os valores máximos e mínimo que essas configurações podem assumir, conforme a Figura 11.

O mesmo procedimento pode ser utilizado para os dispositivos digitais, criando as estruturas para salvar as configurações como *baud-rate*, quantidade de bits, bits de paridade e de parada, e quantidade de *bytes* para o dispositivo serial, enquanto que para o I²C, o endereço do dispositivo e os comando, por exemplo.

3.7 Sensor com protocolo específico

A extensibilidade para sensores com protocolos não convencionais foi demonstrada a partir da adaptação para utilização do sensor de temperatura e umidade DHT22, desenvolvido pela Aosong Electronics Co.,Ltd, utilizando as ferramentas anteriormente discutidas. O sensor é apresentado em seu *datasheet* como uma solução de medição de temperatura e umidade com calibração, duradouro e de baixo consumo (AOSONG, 2021).

Sua comunicação ocorre através de uma interface de um fio, da seguinte forma:

- Primeiro o microcontrolador leva a linha de transmissão ao nível baixo, por ao menos 1ms;
- O sensor responde após um intervalo de 20-40 μ s, levando a linha para o nível baixo por 80 μ s e então alto por 80 μ s;
- Para cada *bit* de dado, precede um intervalo em nível baixo pelo sensor de 50 μ s e são transmitidos 40 *bits*, onde após o intervalo baixo, um intervalo alto de 70 μ s representa um “1” e um intervalo de 26-28 μ s representa um “0”;
- A transmissão de uma aquisição leva 5 ms.

Os dados finais são 8 *bits* representando a parte decimal da umidade, seguidos de 8 *bits* representando a parte fracionária da umidade, mais 8 *bits* representando a parte decimal da temperatura seguidos de 8 *bits* representando a parte fracionária da temperatura e por fim 8 *bits* com o *checksum* dos valores para validar. As partes decimais e fracionárias devem ser somadas em um número de 16 *bits*, formando o resultado final para a umidade e para a temperatura (AOSONG, 2021).

A interface para o sensor foi implementada utilizando o *hardware* de *timer* do microcontrolador para a medição dos tamanhos dos pulsos gerados pelo sensor e determinação dos *bits*. Para isso é necessário configurar os registradores que controlam o periférico. Toda a descrição de funcionalidades e registradores é encontrada no manual de referência fornecido pelo fabricante.

A configuração consistiu na ativação do relógio para os periféricos, e para esse caso de aplicação, além de habilitar para o próprio *timer*, é preciso habilitar para a interface *General Purpose Input/Output* (GPIO), já que o *timer* utiliza essa interface para detectar a variação do sinal pino de transmissão do sensor.

O próximo passo é a configuração do pino escolhido para conectar no sensor, que precisa ser em modo de entrada. O *datasheet* explicita quais pinos são roteados para o periférico escolhido, além de indicar a possibilidade de fazer o roteamento para outros pinos, conforme o modo de função alternada. Para isso é necessário configurar a localização de acordo com o *datasheet* e habilitar o roteamento nos registradores.

Com o pino configurado, resta definir o comportamento do *timer*, como interrupções, divisão de relógio na contagem, limite de contagem dentre outros. Para a interface com

o DHT22, ele foi configurado de modo a gerar uma interrupção na borda de descida do sinal no pino, e redefinir o contador na borda de subida operando no modo de captura de entrada, como sugere o manual de referência para medição de pulsos.

Com a configuração do *timer* finalizada, desenvolveram-se as funções para comunicar com o DHT22: Uma função para colocar a linha em nível baixo, exigindo que seja refeita a configuração do pino GPIO para o modo de saída, e desabilitada a função alternada se isso houver sido realizado, outra para colocar a linha novamente em nível alto, e finalmente a função que é executada quando é gerada uma interrupção pelo *timer*. Essa interrupção lê o registrador que armazena a contagem do *timer*, converte para microssegundos e verifica o comprimento do pulso, armazenando o valor do *bit* em um vetor e retorna da interrupção. Todo esse procedimento é definido no arquivo *timer.c*.

Essas funções e valores são utilizados pelo sistema operacional posteriormente no programa desenvolvido, através da função de aquisição desenvolvida para o DHT22, chamada *DHT22_Rec*, que segue os passos citados anteriormente. Dessa forma, a criação de uma camada de aplicação para o *timer* pode permitir a utilização de outros sensores que se comunicam através da largura de pulsos seguindo o mesmo modelo.

3.8 Arquitetura dos sensores e operação

O sistema operacional foi desenvolvido utilizando da arquitetura de *tasks*, ou tarefas, do FreeRTOS. Nessa arquitetura, cada tarefa possui seu próprio espaço em memória, isolando assim suas variáveis, e executa em um ciclo infinito independente, e é papel do RTOS gerenciar esses ciclos. Esse formato permite então que cada *task* represente um sensor, com seus parâmetros de configuração definidos como variáveis criadas em cada *task*, e sua função de aquisição que é executada repetidamente nesse ciclo.

Essa arquitetura também permite a execução das tarefas de forma periódica, através da utilização conjunta das funções *vTaskDelayUntil* com *xTaskGetTickCount*, que são inseridas dentro do ciclo da *task* (FREERTOS, 2020). É definida uma frequência em que deve ser executada a *task* e essa frequência é passada como parâmetro para a função. A função *xTaskGetTickCount* salva o tempo atual em outra variável que também é passada como parâmetro para *vTaskDelayUntil*, e após isso, a função faz o cálculo do tempo restante para que a execução seja periódica. Assim, tem-se que essa frequência representa o intervalo de aquisição do sensor.

Foram implementadas também variáveis que representam a identificação da *task*, o tipo de

sensor, que identifica o periférico utilizado para obter os dados, e uma variável de controle que altera o fluxo de execução para o estado de aquisição ou de configuração.

O FreeRTOS também permite que tarefas sejam suspensas, a partir da habilitação do parâmetro `INCLUDE_vTaskSuspend` no arquivo de configuração de biblioteca do sistema operacional de tempo real. A suspensão de uma tarefa significa que ela irá esperar indefinitivamente, sendo que nesse período ela não consome recursos de processamento até que seja ativada pelo sistema operacional novamente. Dessa forma, a adição ou a remoção de um novo sensor apenas suspende a *task* que o representa, ou retoma a execução, o que também simplifica o desenvolvimento.

A ferramenta do FreeRTOS que permite esse funcionamento é a *Queue*, que opera como uma FIFO (do inglês, *First In First Out*), onde mensagens são adicionadas e removidas por *tasks*, permitindo comunicação entre elas. O tamanho do elemento que representa uma posição na fila e a quantidade de elementos são definidos na criação da *Queue*. Quando é feita uma leitura de mensagem, se a fila estiver vazia, é configurado um período em que ela deve aguardar a chegada de uma nova mensagem. A utilização do parâmetro `portMAX_DELAY` faz com que a *task* espere indefinitivamente, até que chegue uma nova mensagem. Nesse período ela entra no modo de suspensão. Alternativamente, no lugar do parâmetro que suspende a tarefa, pode ser configurado o valor 0 para que ela volte a executar imediatamente se a fila estiver vazia (FREERTOS, 2020).

3.9 Arquitetura da transmissão

Para a troca das mensagens, dois vetores de caracteres são adicionados também às *tasks*, um para tratar o pacote de configuração, e outro para os pacotes de dados. Ambos são transmitidos entre as *tasks* utilizando o sistema de filas. O pacote de dados é passado por referência para as funções de conversão, onde precisam ser preenchidos pela implementação específica do sensor, nas posições indicadas, conforme citado no Capítulo 3. A posição de identificação do sensor é preenchida pela própria tarefa, utilizando sua variável de identificação.

Cada sensor tem duas filas implementadas. Uma delas é uma fila para transmitir seus dados, denominada *xDataQueue*, que é compartilhada entre todos os sensores e uma *task* (*Task_DATATX*) que tem como único objetivo aguardar a chegada de dados nessa fila e imediatamente enviar esses dados para a interface serial. Isso é feito para evitar que duas tarefas tentem utilizar a interface serial no mesmo momento, o que causaria a perda dos dados de um dos sensores. A outra fila é para recebimento dos comandos de configuração,

que opera em modo de retornar à execução imediatamente durante o estado de aquisição caso a fila esteja vazia, para verificar se o comando de parada foi recebido, e configurada para esperar indefinidamente no estado de configuração, aguardando os comandos de configuração ou de retomada.

A tarefa responsável em verificar a chegada de pacotes de configuração é a *Task_CONFIGRX*, que roda periodicamente verificando se foi recebido um novo pacote. Se uma nova configuração foi recebida na interface serial, o pacote de configuração é encaminhado para a *task* especificada, através da *Queue* que é relacionada à identificação no pacote.

Os parâmetros que podem ser alterados são: Retomada/Parada, identificado pelo caractere "S", Tipo, pelo caractere "T" e Frequência de aquisição, pelo caractere "F". Essa separação foi feita para simplicidade no tratamento. Os comandos são seguidos por valores em ASCII, que serão convertidos. O caractere "S" espera o dígito zero ou um, o "F" espera quatro dígitos representando o intervalo de aquisição e o "T" é o caso especial para configurações. Os próximos dígitos identificam o periférico e o valor a ser salvo, por exemplo no caso do ADC, o comando "1T226" indica configuração do sensor um, como tipo dois (ADC), parâmetro "canal" (quarto dígito com valor dois) e valor para esse parâmetro (seis).

A atualização das variáveis precisa ser realizada com a aquisição suspensa. Antes de retomar a execução é esperada a alteração das variáveis e a configuração dos periféricos, até que o comando de retomada seja recebido. Esse conjunto de filas, *tasks* e pacotes definem a transmissão e a arquitetura da solução.

3.10 Arquitetura de recepção

Os dados são transmitidos pelo microcontrolador para a ESP8266, através de uma de suas interfaces seriais que permitem comunicação nos dois sentidos. Não é esperada perda de dados, já que tanto a ESP8266 quanto o microcontrolador estão configurados para realizar interrupções quando ocorre a chegada de dados na interface serial e armazenar esse dado na memória, em um *buffer* que permite que esse dado seja analisado quando possível ou necessário.

A ESP8266 pode ser configurada com um IP estático utilizando da biblioteca *ESP8266WiFi* disponível na plataforma de desenvolvimento Arduino. A biblioteca permite também a utilização do parâmetro *WIFI_STA*, para operação no modo estação, desabilitando a capacidade do dispositivo de operar como ponto de acesso Wi-Fi para outros dispositivos, atuando apenas como cliente e conseqüentemente reduzindo o esforço computacional, já

que sua função é apenas transmissão dos dados para a rede.

Além da configuração da rede Wi-Fi, é preciso especificar o IP do servidor MQTT, através da biblioteca *pubsubclient*, disponível na Arduino IDE. Esse IP é o da máquina onde foi instalado e habilitado o agente MQTT *mosquitto*. O agente é uma solução de código aberto desenvolvida pela fundação Eclipse, e possibilita a instalação de um serviço MQTT em máquinas Linux e Windows tornando-as servidores MQTT, capazes de permitir a comunicação desse protocolo em uma rede, além de fornecer uma interface simples de linhas de comandos para se inscrever e publicar em tópicos (LIGHT, 2017). No caso de teste o servidor foi implementado na máquina pessoal do autor e tendo em vista apenas a demonstração da recepção, o serviço foi configurado da maneira mais simples, sem autenticação dos dispositivos conectados ou tratamento dos dados.

A configuração da ESP8266 consiste em uma rotina que aguarda a chegada de dados na interface serial, através da função *Serial.Available*, dados estes que são enviados pela *xDataQueue* do sistema operacional. A partir do ID do sensor, a ESP8266 define qual o tópico MQTT que pertence ao sensor responsável por aqueles dados e publica apenas a parte do pacote que representa o que foi medido pelo sensor através do protocolo MQTT via Wi-Fi. A função responsável pela transmissão é a *client.publish* que recebe como parâmetros o tópico em que será publicado, a mensagem e a quantidade de *bytes* a serem enviados.

Para o envio dos pacotes de configuração para o microcontrolador, a ESP8266 está configurada para estar inscrita no tópico “config” através da função *client.subscribe*. Quando uma mensagem é publicada nesse tópico pelo agente *mosquitto*, uma função chamada *callback* é executada na ESP8266, repassando os dados que foram recebidos por Wi-Fi para sua interface serial, transmitindo para o microcontrolador, permitindo então a configuração remotamente. O uso do agente *mosquitto*, da biblioteca *pubsubclient* e Wi-Fi definem a recepção e a transmissão de dados remotamente da solução.

3.11 Interpretador

O interpretador fica localizado no repositório central, de onde podem ser enviados os comandos para os sensores através da arquitetura do serviço MQTT. O código do interpretador foi desenvolvido em Python, por ser uma linguagem amplamente utilizada, e possuir um leitor e escritor de arquivos JSON nativamente. O código recebe como entrada o arquivo que será lido para configurar o sensor, com o nome no formato NOME_SENSOR.json, e lê o arquivo de configurações SPEC.json, formatados de acordo com a Seção 3.5.

Após a execução, o interpretador envia para o usuário qual o tipo do sensor que está sendo configurado, se ele foi encontrado no arquivo de especificações e se foram definidos limites para aquele sensor. Além disso encerra a configuração sem alterações e avisa ao usuário caso algum parâmetro esteja fora dos limites.

Se não são identificadas inconsistências, os comandos de configuração são gerados e enviados para o agente MQTT fazer a publicação. O exemplo de comando a seguir foi utilizado para configurar o sensor usado no protótipo: *python3 config.py ADC1.json 2*.

Para fazer a parada do sensor ou sua retomada, o código recebe como entrada o comando PARAR ou RETOMAR e o ID do sensor: *python3 config.py PARAR 2*. Dessa forma, o procedimento para configurar um sensor consiste em enviar o comando de parada, aguardar o fim do ciclo de aquisição, e fazer a tradução do arquivo de configuração. Quando estes passos estiverem completos, envia-se o comando de retomada.

4 RESULTADOS

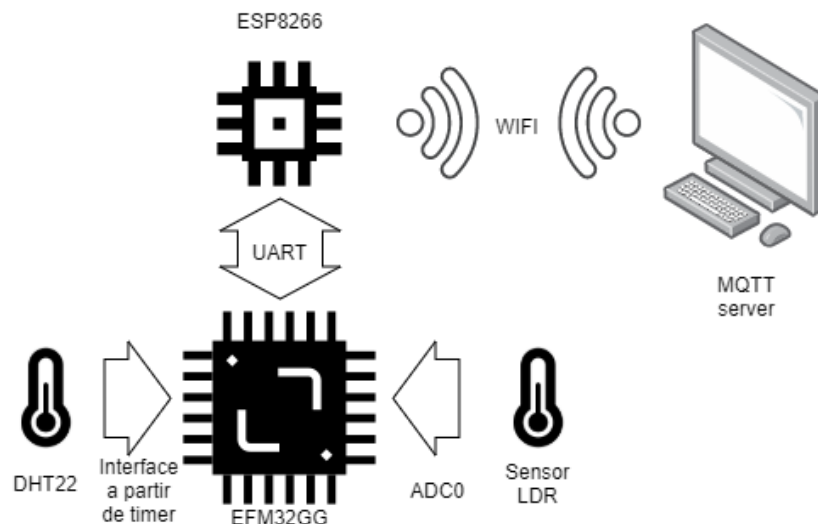
Para demonstrar as funcionalidades da arquitetura idealizada, foi desenvolvido um protótipo com características próximas aos de um nó IoT.

Foram realizados testes para verificar o *software* resultante do processo apresentado no Capítulo 3 com as seguintes características:

- Ser o mais próximo da realidade possível, demonstrando uma situação de medição de dados real;
- Tornar mais simples a alteração de sensores e seus parâmetros;
- Demonstrar a aquisição de dados simultânea à configuração dos sensores;
- Demonstrar a aquisição remota dos dados dos sensores associados a intervalos constantes.

A Figura 12 mostra a topologia do protótipo.

Figura 12 – Topologia do protótipo.

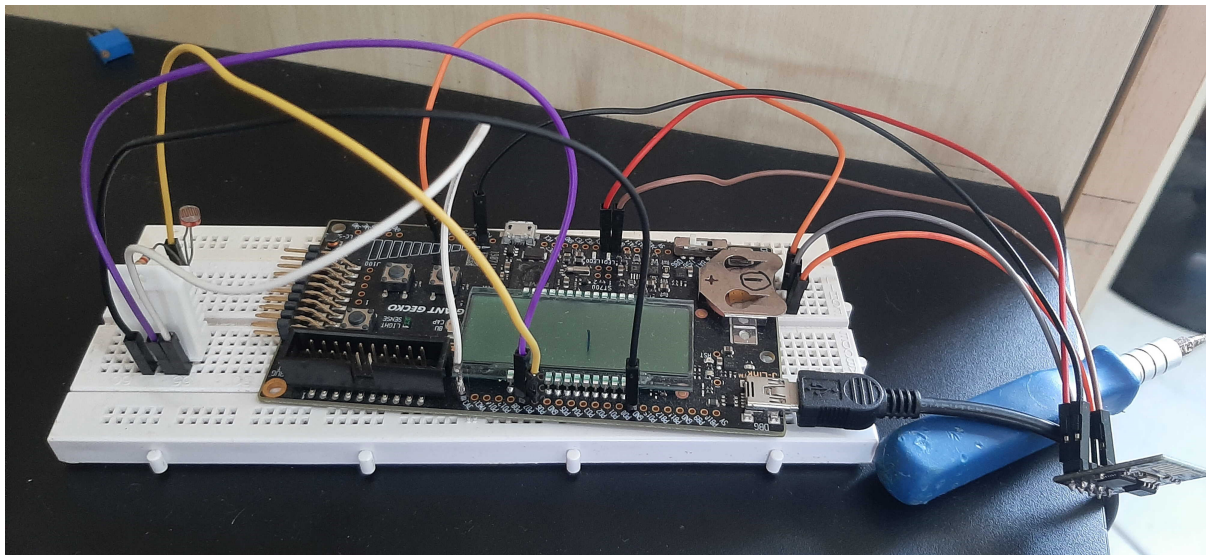


Fonte: Produção do próprio autor.

A Figura 13 mostra a montagem do protótipo em uma *protoboard*, com os sensores DHT22 e de luminosidade conectados. A plataforma utilizada para os testes foi a placa EFM32GG, adaptada para transmissão remota através de outro dispositivo, no caso a ESP8266. A plataforma representa sistemas modernos típicos, com um processador de

32 bits (Cortex-M3) e um componente para transmissão remota, permitindo todas as vantagens apresentadas nas discussões anteriores.

Figura 13 – Montagem do protótipo.



Fonte: Produção do próprio autor.

O teste consistiu na elaboração dos arquivos de configuração e de especificações para sensores ADC e para o sensor DHT22. Dois arquivos de configuração chamados de LDR.json e POT.json foram salvos no repositório, para representar os sensores ADC. Destaca-se que esses arquivos JSON estão integrados ao *software* no repositório, entretanto, poderiam ser enviados para o microcontrolador em formato semelhante para serem lidos. A reprogramação é evitada da mesma forma, já que haverá alterações apenas nos arquivos, e assim outras formas de transmitir o arquivo podem ser implementadas no repositório.

O primeiro arquivo foi usado para recepção de dados de iluminação do ambiente através de um resistor fotossensível e o segundo para leitura de um resistor variável simulando outro sensor ADC. A Figura 14 mostra esses arquivos e suas configurações.

Para o sensor de interface não convencional DHT22, foram criados os arquivos mostrados na Figura 15. Esses arquivos foram utilizados para demonstrar a aplicabilidade das funções de conversão, e apenas permitiam a troca da conversão escolhida. A primeira função possuía *offset* de 16 bits, adquirindo a temperatura medida pelo sensor e a segunda retornava os primeiros dados, que representavam a umidade.

A primeira etapa do teste foi a ativação do servidor MQTT, fazendo as inscrições nos tópicos "sensor1" e "sensor2", e a verificação da conexão da ESP ao servidor. Após verificar a conexão, foi executado o interpretador com o arquivo e identificação do sensor desejado.

Figura 14 – Arquivos dos sensores ADC.

LDR.json	POT.json
<pre>{ "ADC": { "FREQUENCIA": 250, "NUMERO": 0, "CANAL": 6, "REFERENCIA": "VDD", "RESOLUCAO": 8, "CONVERSAO": "DEC" } }</pre>	<pre>{ "ADC": { "FREQUENCIA": 250, "NUMERO": 0, "CANAL": 7, "REFERENCIA": "VDD", "RESOLUCAO": 12, "CONVERSAO": "DEC" } }</pre>

Fonte: Produção do próprio autor.

Figura 15 – Arquivos do sensor DHT22.

DHT_TEMPERATURA.json	DHT_UMIDADE.json
<pre>{ "DHT22": { "FREQUENCIA": 2500, "CONVERSAO": "BIN2X8X2_1" } }</pre>	<pre>{ "DHT22": { "FREQUENCIA": 2500, "CONVERSAO": "BIN2X8X2_0" } }</pre>

Fonte: Produção do próprio autor.

O interpretador gera os comandos que serão utilizados no microcontrolador e envia os comandos através do MQTT. Além disso mostra ao usuário as informações e configurações para o sensor.

Após a configuração dos sensores ADC, o interpretador foi executado com o comando de retomar a aquisição, e os dados são enviados para o repositório. A alteração da luminosidade e valor do resistor variável alteram os valores recebidos, verificando o funcionamento. A Figura 16 mostra o exemplo das informações retornadas pelo interpretador além da aquisição dos dados.

Para a alteração do sensor que está sendo utilizado, é executado o interpretador com o comando de parada, depois com a configuração do DHT22 para aquisição de temperatura e finalmente, após trocar o sensor na *protoboard*, com o comando de retomada. O outro sensor não tem sua aquisição interrompida, e imediatamente o repositório já começa a receber os dados de temperatura. O mesmo é feito para umidade.

Figura 16 – Execução do interpretador e aquisição de dados.

```

douglass@douglas-linux: ~/pg2
File Edit View Search Terminal Help
douglass@douglas-linux:~/pg2$ python3 config.py POT.json 1
Configurando tipo 1 (ADC) no sensor 1
Definido valor min para frequencia: 0
Frequencia escolhida: 250ms
Numeros dos ADC na placa: [0]
Numeros dos canais ADC: [0, 1, 2, 3, 4, 5, 6, 7]
Referencias de tensao permitidas: ['VDD']
Resolucoes permitidas: [6, 8, 12]
Formatos de conversao permitidos: ['DEC']
-----Verificacao-----
Numero do ADC escolhido : 0
Numero do canal escolhido: 7
Referencia escolhida: VDD
Valor de resolucao escolhido: 12
Conversao escolhida: DEC
-----Gerando comandos-----
1T100
1T117
1T120
1T132
1T140
1F250
douglass@douglas-linux:~/pg2$ python3 config.py LDR.json 2
536#115
542#116
540#117
538#118
539#119
292#0
292#1
292#2
291#3
292#4
291#5
844#0
838#1
835#2
857#3
897#4
909#5
916#6
914#7
904#8
887#9
871#10
859#11
21#399
21#400
21#401
22#402
21#403
21#404
21#405
21#406
21#407
21#408
21#409
21#410
21#411
22#412
22#413
21#414
21#415
21#416
21#417
21#418
21#419
21#420
21#421
1634411694: New client connected from 127.0.0.1 as mosqpub|10931-douglas-l (c1, k60).
1634411694: Client mosqpub|10931-douglas-l disconnected.
1634411702: New connection from 127.0.0.1 on port 1883.
1634411702: New client connected from 127.0.0.1 as mosqpub|10940-douglas-l (c1, k60).
1634411702: Client mosqpub|10940-douglas-l disconnected.
1634411702: New connection from 127.0.0.1 on port 1883.
1634411702: New client connected from 127.0.0.1 as mosqpub|10941-douglas-l (c1, k60).
1634411702: Client mosqpub|10941-douglas-l disconnected.
1634411704: New connection from 127.0.0.1 on port 1883.
1634411704: New client connected from 127.0.0.1 as mosqpub|10943-douglas-l (c1, k60).
1634411704: Client mosqpub|10943-douglas-l disconnected.
1634411729: New connection from 127.0.0.1 on port 1883.
1634411729: New client connected from 127.0.0.1 as mosqpub|10946-douglas-l (c1, k60).
1634411729: Client mosqpub|10946-douglas-l disconnected.
1634411731: New connection from 127.0.0.1 on port 1883.
1634411731: New client connected from 127.0.0.1 as mosqpub|10948-douglas-l (c1, k60).
1634411731: Client mosqpub|10948-douglas-l disconnected.

```

Fonte: Produção do próprio autor.

Um vídeo foi gravado com o protótipo em funcionamento e disponibilizado no YouTube.¹

Como demonstrado na implementação do sensor ADC, que somente envolve interfaces comuns e demanda uma rotina de conversão simples, basicamente a única necessidade é alterar o arquivo de configuração.

Como demonstrado na implementação do sensor DHT22, a adaptação de com características não convencionais envolve muitos passos: configuração dos periféricos, elaboração da rotina de conversão (se necessário), adaptação para o formato dos pacotes e elaboração do arquivo de configuração. Entretanto, os passos são relativamente simples e o teste demonstra que após a implementação o uso não é complexo. As características de interfaceamento e de conversão passam a fazer parte da capacidade do nó, e podem ser usadas em novos sensores e a utilização deste depende apenas das alterações nos arquivos de configuração.

Uma limitação bastante impactante na utilização do programa é o fato de ser necessário aguardar o fim de um ciclo de aquisição para alteração das configurações dos sensores, o que degrada a experiência do usuário, já que alguns sensores, como é o caso do próprio

¹ <https://youtu.be/5BVYbPzWRw8>

DHT22 usado no teste, podem ter intervalos de aquisição relativamente longos.

Apesar de não ser esperado perda dos dados por falta de conectividade, já que há a utilização de *buffers* e protocolo TCP, o sistema de filas não foi configurado para armazenar os dados mais recentes. A partir do momento que os *buffers* ficarem cheios, os próximos dados serão descartados. Entretanto, essa limitação pode ser contornada utilizando as ferramentas de manipulação de filas do FreeRTOS, retirando o dado antigo da fila e adicionando o novo conforme o necessário, por exemplo.

5 CONCLUSÕES E TRABALHOS FUTUROS

Foi apresentada uma prova de conceito para uma arquitetura que é capaz de simplificar a alteração de parâmetros e troca de sensores com interfaces comuns, e permitir o uso de interfaces não convencionais em aplicações no contexto IoT. No caso de testes, foi realizado o monitoramento de temperatura, umidade e luminosidade em um ambiente.

Foi desenvolvida uma arquitetura que permite a recepção de comandos de configuração de forma remota para acesso direto ao periférico relacionado à interface do sensor. Assim, a alteração de sensores que utilizam de interfaces já implementadas no sistema pode ser realizada sem recompilação de *software*, evitando a necessidade de ferramentas específicas e parada do sistema.

A partir de um *software* baseado em sistema operacional de tempo real, o protótipo é capaz de fazer aquisição de dados remotamente em uma aplicação IoT de forma robusta, com várias aquisições simultâneas e reconfigurações interferindo minimamente na operação.

O aprimoramento do protótipo, buscando otimizar a operação e remover as limitações são sugestões de trabalhos futuros. Por exemplo, avaliar as ferramentas de criação dinâmica de tarefas do FreeRTOS, para permitir mais sensores atuando simultaneamente. As ferramentas de suspensão de tarefas e filas do FreeRTOS também podem ser avaliadas para evitar aguardar o fim da aquisição e a perda dos dados recentes quando as filas estiverem cheias.

Testar o programa em outras plataformas modernas compatíveis com as ferramentas e a inclusão de sensores que utilizam outros periféricos da placa, como I²C ou uma das interfaces UART, e adicionar funções de conversão irão ampliar a abrangência do projeto.

A prova de conceito mostra que transmitir os arquivos de configuração para a memória do microcontrolador em um formato compatível com o que é lido do arquivo no repositório não é complexo. Isso permitiria simplicidade no protocolo de transmissão remota das configurações. As capacidades de cada nó podem ser registradas no banco de dados e verificada a validade antes do envio.

Além disso, implementar uma sinalização de desativação do sensor e adicionar ao repositório central um banco de dados com interfaces gráficas, permitindo uma visualização mais clara dos dados, são trabalhos que poderão aprimorar a experiência do usuário.

REFERÊNCIAS

- AOSONG. Digital-output relative humidity and temperature sensor/module DHT22. 2021. Disponível em: <<https://pdf1.alldatasheet.com/datasheet-pdf/view/1132459/ETC2/DHT22.html>>. Acesso em: 9 set. 2021. Citado 2 vezes nas páginas 32 e 33.
- ARM. The FreeRTOS™ Kernel. 2020. Disponível em: <<https://www.arm.com/products/silicon-ip-cpu/cortex-m/cortex-m3>>. Acesso em: 26 out. 2020. Citado na página 21.
- ATZORI, L.; IERA, A.; MORABITO, G. The internet of things: A survey. Computer networks, Elsevier, v. 54, n. 15, p. 2787–2805, 2010. Citado na página 9.
- CHEN, C.-Y.; HASAN, M.; MOHAN, S. Securing real-time internet-of-things. Sensors, Multidisciplinary Digital Publishing Institute, v. 18, n. 12, p. 4356, 2018. Citado na página 11.
- CHEN, Y.; KUNZ, T. Performance evaluation of iot protocols under a constrained wireless access network. In: IEEE. 2016 International Conference on Selected Topics in Mobile & Wireless Networking (MoWNeT). [S.l.], 2016. p. 1–7. Citado na página 20.
- CHIANG, M.; ZHANG, T. Fog and iot: An overview of research opportunities. IEEE Internet of things journal, IEEE, v. 3, n. 6, p. 854–864, 2016. Citado na página 11.
- ESPRESSIFSYSTEMS. ESP8266X Datasheet. 2020. Disponível em: <https://www.espressif.com/sites/default/files/documentation/0a-esp8266ex_datasheet_en.pdf>. Acesso em: 22 out. 2020. Citado na página 21.
- FEKI, M. A.; KAWSAR, F.; BOUSSARD, M.; TRAPPENIERS, L. The internet of things: the next technological revolution. Computer, IEEE, v. 46, n. 2, p. 24–25, 2013. Citado na página 9.
- FREERTOS. The FreeRTOS™ Kernel. 2020. Disponível em: <<https://www.freertos.org/RTOS.html>>. Acesso em: 26 out. 2020. Citado 3 vezes nas páginas 18, 34 e 35.
- JASKANI, F. H.; MANZOOR, S.; AMIN, M. T.; ASIF, M.; IRFAN, M. An investigation on several operating systems for internet of things. EAI Endorsed Transactions on Creative Technologies, European Alliance for Innovation, v. 6, n. 18, 2019. Citado na página 9.
- LABROSSE, J. J. uC/OS-III – The Real-Time Kernel. Weston, FL 33326, USA: Micrium Press, 2010. Citado 2 vezes nas páginas 17 e 18.
- LI, S.; XU, L. D.; ZHAO, S. The internet of things: a survey. Information Systems Frontiers, Springer, v. 17, n. 2, p. 243–259, 2015. Citado na página 16.
- LIGHT, R. A. Mosquitto: server and client implementation of the mqtt protocol. Journal of Open Source Software, The Open Journal, v. 2, n. 13, p. 265, 2017. Disponível em: <<https://doi.org/10.21105/joss.00265>>. Citado na página 37.

- LIU, X.; HOU, K. M.; VAULX, C. D.; SHI, H.; GHOLAMI, K. E. Miros: A hybrid real-time energy-efficient operating system for the resource-constrained wireless sensor nodes. Sensors, Multidisciplinary Digital Publishing Institute, v. 14, n. 9, p. 17621–17654, 2014. Citado na página 18.
- MATTOLI, V.; MONDINI, A.; MAZZOLAI, B.; FERRI, G.; DARIO, P. A universal intelligent system-on-chip based sensor interface. Sensors, Molecular Diversity Preservation International, v. 10, n. 8, p. 7716–7747, 2010. Citado 2 vezes nas páginas 11 e 17.
- MIKHAYLOV, K.; HUTTUNEN, M. Modular wireless sensor and actuator network nodes with plug-and-play module connection. In: IEEE. SENSORS, 2014 IEEE. [S.l.], 2014. p. 470–473. Citado na página 17.
- MISHRA, D.; GUNASEKARAN, A.; CHILDE, S. J.; PAPADOPOULOS, T.; DUBEY, R.; WAMBA, S. Vision, applications and future challenges of internet of things: A bibliometric study of the recent literature. Industrial Management & Data Systems, Emerald Group Publishing Limited, 2016. Citado na página 9.
- RODRIGUEZ-ZURRUNERO, R.; TIRADO-ANDRÉS, F.; ARAUJO, A. Yetios: An adaptive operating system for wireless sensor networks. In: IEEE. 2018 IEEE 43rd Conference on Local Computer Networks Workshops (LCN Workshops). [S.l.], 2018. p. 16–22. Citado na página 17.
- RUIZ-ROSETO, J.; RAMIREZ-GONZALEZ, G.; WILLIAMS, J. M.; LIU, H.; KHANNA, R.; PISHARODY, G. Internet of things: A scientometric review. Symmetry, Multidisciplinary Digital Publishing Institute, v. 9, n. 12, p. 301, 2017. Citado na página 19.
- SRIVASTAVA, S.; SINGH, M.; GUPTA, S. Wireless sensor network: a survey. In: IEEE. 2018 International Conference on Automation and Computational Engineering (ICACE). [S.l.], 2018. p. 159–163. Citado na página 10.
- TAIVALSAARI, A.; MIKKONEN, T. A taxonomy of iot client architectures. IEEE software, IEEE, v. 35, n. 3, p. 83–88, 2018. Citado 2 vezes nas páginas 9 e 10.
- TOMOVIC, S.; YOSHIGOE, K.; MALJEVIC, I.; RADUSINOVIC, I. Software-defined fog network architecture for iot. Wireless Personal Communications, Springer, v. 92, n. 1, p. 181–196, 2017. Citado na página 10.
- UNGUREAN, I. Timing comparison of the real-time operating systems for small microcontrollers. Symmetry, Multidisciplinary Digital Publishing Institute, v. 12, n. 4, p. 592, 2020. Citado na página 18.
- WESTONEMBEDDED. Real-Time Kernels: uC/OS-II and uC/OS-III. 2021. Disponível em: <<https://weston-embedded.com/micrium-kernels>>. Acesso em: 19 set. 2021. Citado na página 18.
- XU, L. D.; HE, W.; LI, S. Internet of things in industries: A survey. IEEE Transactions on industrial informatics, IEEE, v. 10, n. 4, p. 2233–2243, 2014. Citado na página 15.
- ZIKRIA, Y. B.; KIM, S. W.; HAHM, O.; AFZAL, M. K.; AALSALEM, M. Y. Internet of things (iot) operating systems management: Opportunities, challenges, and solution. Sensors, v. 19, n. 8, 2019. Citado na página 16.