Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Monitorização Inteligente de Zonas Húmidas Construídas para Tratamento e Valorização de Águas Residuais

Relatório de projeto

Simão Pedro Barcelos Lopes

Mestrado em Engenharia Eletrotécnica

Tomar, setembro de 2024

Instituto Politécnico de Tomar

Escola Superior de Tecnologia de Tomar

Simão Pedro Barcelos Lopes

Monitorização Inteligente de Zonas Húmidas Construídas para Tratamento e Valorização de Águas Residuais

Relatório de projeto

Orientado por:

Professor Doutor Henrique Pinho, IPT

Professor Doutor Manuel Barros, IPT

Projeto apresentado ao Instituto Politécnico de Tomar para cumprimento dos requisitos necessários à obtenção do grau de Mestre em Engenharia Eletrotécnica

Dedico este trabalho aos meus pais e avós.

Sic Parvis Magna.

RESUMO

Os sistemas embebidos, *Internet of Things* e sistemas distribuídos de controlo constituem áreas da engenharia eletrotécnica cuja evolução recente tem contribuído para uma enorme melhoria do desempenho e do consumo racional de recursos em numerosos equipamentos e processos. A melhoria de desempenho traduz-se em benefícios de produtividade e de qualidade dos serviços e dos produtos.

No contexto da economia circular, onde a racionalidade no uso de recursos é fundamental, a eletrotecnia e a química surgem como áreas essenciais na análise e acompanhamento eficiente dos processos de reutilização de recursos. Este trabalho inseriu-se e visou dar continuidade ao projeto cofinanciado de investigação EcoModZHC desenvolvido no Centro de Investigação em Cidades Inteligentes, que consistiu na implementação de um demonstrador baseado em Zonas Húmidas Construídas.

As Zonas Húmidas Construídas são sistemas de tratamento de águas residuais que replicam processos naturais em ambientes controlados. Para aperfeiçoar a monitorização da qualidade da água tratada, dimensionou-se um sistema autónomo e automatizado, impulsionado principalmente por avanços nos sistemas embebidos e otimização de protocolos de comunicação. Este sistema automatizado permite recolher dados relevantes, acesso em tempo real às informações, gestão otimizada das Zonas Húmidas Construídas e análise eficaz dos dados. Com este sistema pretende-se eliminar a necessidade de efetuar medições e análises manuais. Esta abordagem oferece uma visão mais abrangente sobre o impacto de fatores externos, como luz solar e temperatura, no rendimento geral do sistema de tratamento das águas residuais.

Palavras-chave: *Internet of Things*, zonas húmidas construídas, sistemas distribuídos de controlo, sistemas embebidos.

ABSTRACT

Embedded systems, the Internet of Things, and distributed control systems are areas of electrical engineering whose recent evolution has contributed to a significant improvement in performance and the rational consumption of resources in numerous devices and processes. The performance improvement translates into benefits in productivity and quality of services and products.

In the context of the circular economy, where rational resource use is essential, electrical engineering and chemistry emerge as crucial areas for the efficient analysis and monitoring of resource reuse processes. This work aims to continue the co-financed research project EcoModZHC developed at the Smart Cities Research Center, which involved implementing a demonstrator based on Constructed Wetlands.

Constructed Wetlands are wastewater treatment systems that replicate natural processes in controlled environments. To improve the monitoring of treated water quality, an autonomous and automated system was designed, driven primarily by advances in embedded systems and communication protocol optimization. This automated system allows for the collection of relevant data, real-time access to information, optimized management of Constructed Wetlands, and effective data analysis. This system aims to eliminate the need for manual measurements and analyses. This approach offers a more comprehensive view of the impact of external factors, such as sunlight and temperature, on the overall performance of the wastewater treatment system.

Keywords: *Internet of Things*, constructed wetlands, distributed control systems, embedded systems.

AGRADECIMENTOS

A realização do trabalho apresentado neste relatório não teria sido possível sem o apoio e a contribuição de várias pessoas, às quais estou muito grato.

O meu primeiro agradecimento é para os professores Henrique Pinho e Manuel Barros que me acompanharam neste projeto desde a primeira hora, não só como orientadores, mas também, como amigos sempre prontos a ajudar e a colaborar nas várias fases deste projeto. Também, agradeço pelos comentários assertivos e pertinentes aquando da revisão deste relatório e pela permanente disponibilidade em esclarecer as dúvidas que lhes foram colocadas.

Ao professor Carlos Ferreira que sempre me apoiou em todas as fases do projeto e principalmente em todas as fases da minha caminhada no ensino superior. A sua dedicação ao trabalho e a sua abordagem ao ensino inspiram-me verdadeiramente.

Ao engenheiro Pedro Neves, sempre disponível para me ajudar e colaborar no âmbito das suas novas responsabilidades nos laboratórios.

Um agradecimento especial ao técnico da manutenção Carlos Ferreira, do Instituto Politécnico de Tomar, que se disponibilizou para me ajudar em todos os momentos na construção de estruturas e outros tipos de trabalhos.

Ao Instituto Politécnico de Tomar, a todos os professores e funcionários com os quais convivi e aprendi ao longo destes anos e que contribuíram para a minha formação superior.

À minha namorada Inês, sendo ela uma das pessoas que mais me apoiou e se sacrificou para que eu conseguisse completar esta fase da minha vida.

Um agradecimento especial à minha família e amigos que me apoiaram incondicionalmente e sempre me incentivaram a fazer mais e melhor. Sem eles teria sido tudo muito mais difícil.

Agradece-se, também, o suporte dos projetos CENTRO-01-0145-FEDER-179932 (EcoModZHC), UIDB/05567/2020 e UIDP/05567/2020 (Ci2).

Índice

Índice de Figuras

Índice de Tabelas

Lista de siglas e abreviações

Sigla/ Abreviação	Descrição
ADC	*Analog to Digital Converter*
AP	*Access Point*
APA	Agência Portuguesa do Ambiente
API	*Application Programming Interface*
Ci2	Centro de Investigação em Cidades Inteligentes
COD	*Chemical Oxygen Demand*
CPU	*Central Processing Unit*
CQO	Carência Química de Oxigénio
DAC	*Digital to Analog Converter*
EC	*Eletrical Conductivity*
EM	Estação Meteorológica
ESP-IDF	*Espressif IoT Development Framework*
GPIO	*General Porpose Input/Output*
GUI	*Graphical User Interface*
HMI	*Human-Machine Interface*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
I&D	Investigação e Desenvolvimento

I2C	*Inter-Integrated Circuit*
IC	*Integrated Circuit*
IDE	*Integrated Devlopment Enviroment*
IEC	*International Electrotechnical Commission*
IoT	*Internet of Things*
IPT	Instituto Politécnico de Tomar
ISO	*International Organization for Standardization*
JSON	*JavaScript Object Notation*
LED	*Ligh Emitting Diode*
MCP(s)	Microcomputador(es)
MCU(s)	Microcontrolador(es)
MOSFET	*Metal Oxide Semiconductor Field Effect Transistor*
MQTT	*Message Queuing Telemetry Transport*
MRIPTA	Monitorização Remota de Infraestrutura Piloto de Tratamento de Águas Residuais por via Biológica
MT	Módulo de Telemetria
NTU	*Nephelometric Turbidity Units*
OS	*Operating System*
OSI	*Open Systems Interconnection*
OTA	*Over The Air*

PCB	*Printed Circuit Board*
PWM	*Pulse Width Modulation*
RAM	*Random Access Memory*
RasPi	RaspBerry Pi
ROM	*Read Only Memory*
RTOS	*Real Time Operating Systems*
RX	Recetor
SBC	*Single-Board Computers*
SCADA	*Supervisory Control and Data Acquisition*
SDC	Sistemas Distribuídos de Controlo
SoC	*System On a Chip*
SPI	*Serial Peripheral Interface*
SSL	*Secure Shell*
STA	*Station*
TX	Transmissor
UART	*Universal Asynchronous Receiver/Transmitter*
UCAD	Unidades de Controlo e Aquisição de Dados
VSC	*Visual Studio Code*
WWW	*World Wide Web*
ZHC	Zonas Húmidas Construídas

ZHN	Zonas Húmidas Naturais

1 Introdução

1.1 Enquadramento

A Economia Circular é um modelo de desenvolvimento apontado como um dos pilares fundamentais para assegurar a sustentabilidade do nosso planeta.

Perante a crescente integração de processos baseados na economia circular todos os recursos devem ser utilizados de forma racional e servir o maior número de propósitos. Este modelo de desenvolvimento tem sido integrado em diversas áreas tecnológicas, em que se incluem a Engenharia Eletrotécnica e a Engenharia Química, com vista a analisar, monitorizar e otimizar os processos de reutilização de recursos da forma mais eficiente, rentável e autónoma [1].

O presente projeto insere-se nas atividades de investigação e desenvolvimento (I&D), a desenvolver no seio do Centro de Investigação em Cidades Inteligentes (Ci2), tendo como principal objetivo a adição de novas funcionalidades ao protótipo do projeto cofinanciado EcoModZHC [2].

As Zonas Húmidas Construídas (ZHC) consistem em sistemas de tratamento de águas residuais baseados em plantas, e que funcionam de modo semelhante às zonas húmidas naturais, mas de forma controlada [3, 4]. As ZHC são constituídas por três componentes principais: (i) uma bacia, lagoa, ou uma estrutura tipo tanque, impermeável ou impermeabilizada, dotada de um sistema de tubagens para entrada da água residual e saída da água tratada; (ii) material granular que serve de suporte às plantas, usualmente designado por enchimento, que pode ser um material simples como areia, gravilha ou cascalho, ou materiais mais complexos incluindo resíduos, podendo ser uma mistura de materiais; (iii) plantas adaptadas a ambientes aquáticos, usualmente designadas por macrófitas. Porém, existem tipos de ZHC que podem não ter material de suporte, em que as plantas são flutuantes, como é o caso dos jacintos de água. Também existem variantes de ZHC com outros componentes, como por exemplo sistemas de arejamento [3, 4].

As ZHC procedem ao tratamento de águas residuais através de diversos mecanismos, complexos e interligados, como por exemplo: (i) assimilação de nutrientes e outros compostos, pelas plantas; (ii) assimilação e transformação de poluentes por microrganismos que se desenvolvem naturalmente; (iii) intervenção dos materiais de

enchimento por vários fenómenos físico-químicos, como filtração, sedimentação, precipitação e adsorção [3, 4].

Para analisar a qualidade da água tratada, e assim otimizar o desempenho das ZHC, são monitorizados uma série de parâmetros físicos, como o pH, a condutividade elétrica e a turbidez, e diversos parâmetros químicos, como a avaliação da carga orgânica, nutrientes, designadamente os compostos de nitrogénio e de fósforo, e a concentração de outros compostos e elementos, tais como cálcio, potássio, cloro e magnésio. Para o efeito, têm de ser realizadas análises à água residual à entrada e análises à água tratada de modo a se quantificar o rendimento do sistema de tratamento e aferir se a água tratada se encontra dentro dos parâmetros desejados, segundo as diretivas da Agência Portuguesa do Ambiente (APA) [3, 4].

As medições dos parâmetros de qualidade da água à entrada e à saída das ZHC são analisadas entre largos períodos e com intervenção humana, impossibilitando retirar algumas conclusões importantes sobre o efeito da luz solar, temperatura do ar e pluviosidade no sistema de filtragem.

Os avanços dos sistemas embebidos e otimização de protocolos de comunicação, têm melhorado a capacidade e desempenho dos sistemas de controlo e telemetria, no que diz respeito à capacidade de processamento e velocidades de comunicação.

A utilização de um sistema de monitorização autónomo e automatizado baseado em sistemas embebidos e nas mais recentes tecnologias de comunicação e informação aplicado a um sistemas ZHC, permite: (i) a recolha de amostras de parâmetros relevantes da qualidade da água com uma cadência elevada; (ii) o acesso rápido e em tempo real à informação; (iii) fazer o controlo e a gestão otimizada do sistema ZHC; (iv) a visualização, processamento e análise dos dados, assim como, o desenvolvimento ou a utilização de ferramentas computacionais para apoio à tomada de decisão.

1.2 Objetivos

Este projeto visa o estudo e desenvolvimento de uma rede de sensores e atuadores capazes de autonomamente recolher, atuar e remeter para domínios na *World Wide Web* (WWW) informações relevantes sobre o funcionamento de uma ZHC.

Pretende-se desenvolver um conjunto de Unidades de Controlo e Aquisição de Dados (UCAD) para a monitorização de parâmetros aplicados a uma ZHC que sejam capazes de adquirir dados oriundos de vários tipos de sensores. Paralelamente, pretende-se que as UCAD sejam capazes de controlar alguns fatores físicos relacionados com as ZHC como o caudal de entrada e saída. A UCAD deve ser projetada de forma a possibilitar a interligação de vários sensores e atuadores de fabricantes diferentes.

Neste trabalho deseja-se, também, projetar um Módulo de Telemetria (MT) capaz de estabelecer a comunicação entre vários dispositivos. Pretende-se que o MT interligue todos os dispositivos, desde o processo até à *interface* com o utilizador. Objetivamente, o MT estabelecerá a comunicação entre dispositivos com o auxílio de vários protocolos de comunicação.

O objetivo principal deste trabalho consiste em dar continuidade às atividades relacionadas com a vertente de dimensionamento e desenvolvimento de soluções de sistemas de controlo, aquisição de dados e telemetria de um sistema ZHC inserido no projeto de I&D do Ci2 designado EcoModZHC [5, 6]. Neste projeto pretende-se desenvolver uma unidade de controlo, aquisição e transmissão de dados, para ligar e remeter informação de um conjunto de sensores ambientais e de qualidade de água baseados em standards de comunicação.

A Figura 1.1 mostra o esquema principal do projeto. A água residual é armazenada num depósito ligado a um recipiente a jusante de uma bomba peristáltica onde são feitas as análises à água. A bomba peristáltica dispõe o caudal à entrada do sistema e as análises são efetuadas através de sensores elétricos e químicos submersos na água. A água cai por meio da gravidade para o tanque com flora e vários substratos que filtram a água residual (uma ZHC). Novamente com a ajuda da gravidade e do efeito sifão, a água filtrada cai para um recipiente onde são realizadas novas análises. Quando o nível da água atinge um determinado nível é atuada uma bomba de caudal reduzido, para transportar a água para um tanque final, contabilizando assim o caudal à saída. O sistema de monitorização e controlo é responsável por controlar ambas as bombas, pedir para que os sensores efetuem as respetivas medições, receber e remeter as informações relevantes de todo o processo. Assim sendo, todos parâmetros recolhidos são remetidos para sistemas responsáveis por armazenar e facilitar a monitorização e análise de todos os dados.

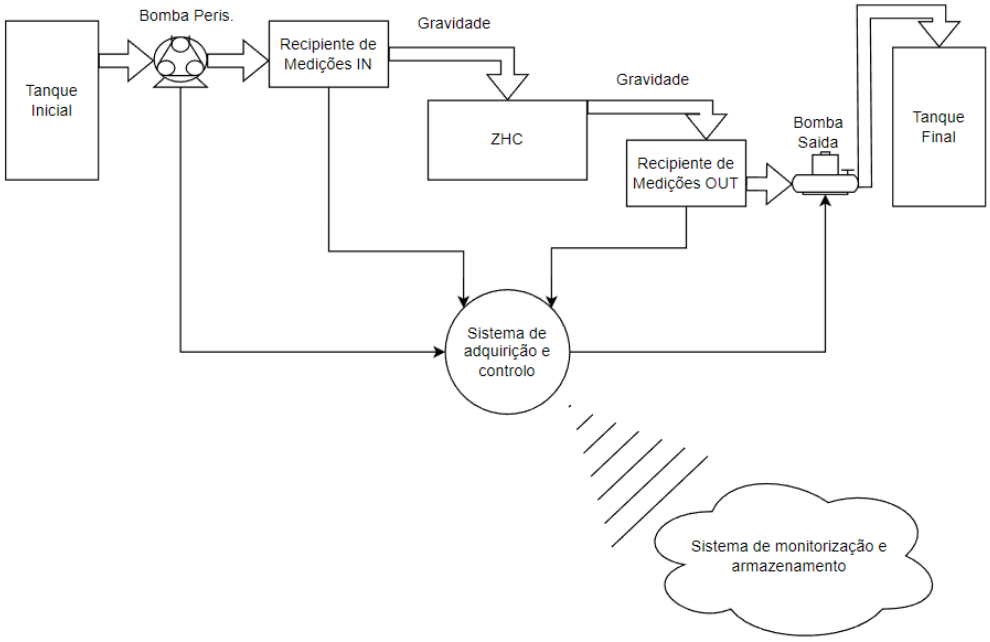


Figura 1.1 - Diagrama geral do projeto.

1.3 Estrutura do documento

O capítulo um descreve de forma sucinta o que se pretende alcançar com este projeto e a motivação subjacente à realização do mesmo.

No capítulo dois, é feito uma abordagem ao estado da arte dos sistemas de monitorização de ZHC e no final apresenta vários conceitos teóricos nos ramos dos sistemas embebidos e protocolos de comunicação. Dá-se primazia ao estudo e exposição das tecnologias empregues neste projeto sem descurar, contudo, tecnologias, normalmente, utilizadas em sistemas semelhantes.

No capítulo três, são apresentadas as diferentes etapas até à implementação no terreno do sistema de aquisição de dados. Neste capítulo encontram-se justificadas as escolhas de procedimentos e dispositivos, utilizados para a implementação dos diversos módulos.

O capítulo quatro expõe os testes e ensaios efetuados aos diferentes módulos projetados. Neste capítulo encontra-se, também, algumas conclusões sobre o comportamento dos módulos dimensionados.

O capítulo cinco expõe os procedimentos efetuados para implementar no terreno os sistemas dimensionados. Este capítulo contém fotografias tiradas para comprovar a implementação dos módulos projetados.

Por fim, o capítulo seis apresenta as principais conclusões que resultam da realização do presente projeto.

2 Estado da arte de sistemas aplicáveis à monitorização de ZHC

Neste capítulo será feito uma abordagem ao estado da arte em sistemas de monitorização de ZHC. Serão abordados alguns conceitos subjacentes ao projeto focados em questões dos domínios da eletrotécnico e da química.

2.1 Breve revisão dos métodos de monitorização de ZHC

As ZHC são sistemas que se assemelham às Zonas Húmidas Naturais (ZHN). As ZHC são projetadas para empregar processos naturais no tratamento de águas residuais, ao mesmo tempo estes sistemas desempenham um papel crucial na biorremediação. As ZHC são constituídas por um sistema composto por plantas, substratos e microrganismos, aproveitando os processos inerentes á natureza para melhorar a qualidade da água [7, 8]. A Figura 2.1 mostra um modelo horizontal de um típica ZHC.

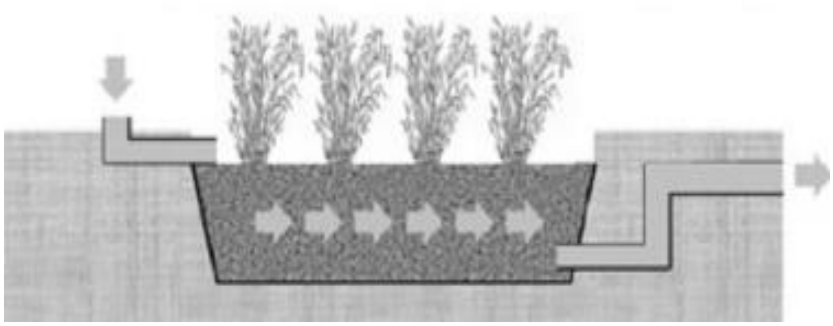


Figura 2.1 – Modelo horizontal de uma ZHC [8].

A monitorização tradicional das ZHC consiste, de uma forma geral, na determinação de parâmetros de qualidade da água com o auxílio de métodos laboratoriais após a recolha e transporte das amostras de água com uma frequência variável e por vezes baixa [7, 9].

A recolha e monitorização remota de dados já se encontra a ser explorada por entidades governamentais para a monitorização e aperfeiçoamento da precisão dos dados de ZHN [10]. Por exemplo, através de imagens capturadas por satélites, como o sistema Landsat. Estas imagens são processadas em programas como o ArcGIS, integrando os dados capturados em estudos científicos. Este processo de captura de imagens recebidas via satélite, pode, no entanto não ser muito viável uma vez os custos para implementar esta

solução são elevados, estão dependentes de fatores meteorológicos e o tempo de transmissão e processamento das imagens em dados concretos é considerável.

A utilização de uma rede de sensores e dispositivos relacionados com os sistemas *Internet of Things* (IoT), especialmente aqueles que possibilitam a determinação ou estimativa da composição da água, consiste no modo mais eficaz no que diz respeito à monitorização de águas residuais [11]. As vantagens inerentes a este processo de recolha de dados passam pela sua rapidez na recolha e processamento dos dados, exatidão, baixos custos relativamente a outras soluções e robustez. Os avanços tecnológicos têm permitido desenvolver sensores e dispositivos associados à recolha e monitorização com custos reduzidos. A proliferação de sistemas IoT e redes de dados pode impulsionar a disseminação da monitorização remota das ZHC.

Existem claras vantagens ao empregar uma monitorização remota, quando comparando com o método tradicional. As vantagens englobam a diminuição dos custos associados às análises laboratoriais, a rapidez na obtenção dos dados e a capacidade de incrementar a cadência de amostragem [8].

O autor Koskiaho [9], relata o uso de monitorização remota e automatizada com o auxílio de dispositivos eletrónicos, para aferir parâmetros de águas residuais na entrada e nas saídas de uma ZHC. Segundo outros testes realizados pelos autores em [12], a monitorização remota e autónoma foi empregue para avaliar a temperatura e o conteúdo volumétrico (entre outros) em intervalos de um a dez minutos.

2.2 Tecnologias aplicadas na monitorização e controlo de processos

Alguns autores [13, 14, 6], afirmam que os sistemas de monitorização e controlo caem sobre os conceitos e topologias dos Sistemas Distribuídos de Controlo (SDC) e/ou IoT.

Num contexto industrial, um SDC é um sistema dimensionado e configurado para controlar e monitorizar vários processos simultaneamente. A descentralização inerente a estes sistemas possibilita a supervisão e controlo dos processos, com uma grande eficiência e redundância [15]. Nos SDC, os controladores desempenham várias funções em distintas fases dos processos, transmitindo informações aos utilizadores por meio de uma variedade de protocolos de comunicação.

A arquitetura de um SDC é normalmente organizada por graus hierárquicos. A representação típica das várias camadas de um SDC pode ser observada com recurso à Figura 2.2 [15]. Nos SDC, as operações são estruturadas em múltiplos níveis. Na base da hierarquia, no nível um, os processos são acompanhados por sensores para recolher dados cruciais e atuadores para controlar as suas características físicas. No nível dois, os controladores interagem diretamente com sensores, atuadores e sistemas de supervisão de nível superior, recolhendo e transmitindo informações. No nível três, no topo, a central de monitorização exibe e armazena informações em tempo real sobre todos os processos. Para a interligar todos os níveis, são empregues protocolos de comunicação que garantem a transferência segura e rápida de dados relevantes.

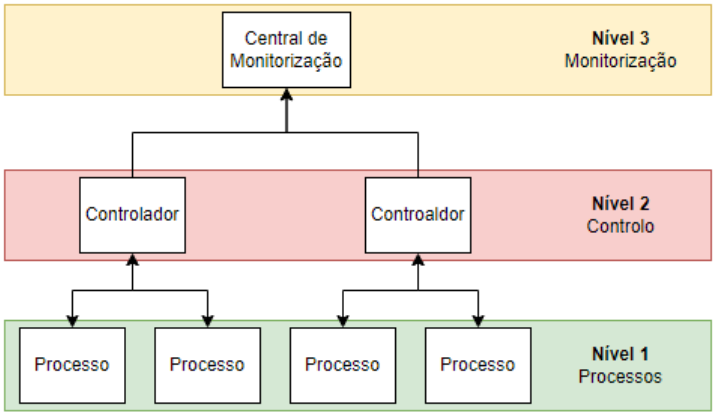


Figura 2.2 – Arquitetura típica de um SDC.

Os sistemas IoT representam um paradigma inovador de interconexão e colaboração de um vasto leque de dispositivos e objetos inteligentes através da Internet. Na sua essência, a IoT permite que certos dispositivos recolham, troquem e atuem sobre diferentes dados. O conceito principal de IoT consiste na capacidade de os objetos físicos incorporarem sensores, atuadores e conectividade, permitindo a recolha, transmissão e análise de dados em tempo real. Esta abordagem veio revolucionar a forma como os dispositivos interagem entre si e com os utilizadores, possibilitando uma integração mais eficiente e uma tomada de decisões mais informada [13].

A arquitetura típica de um sistema IoT consiste na comunicação bilateral entre dispositivos embebidos que acompanham diversos processos e um dispositivo *edge* (ou *Gateway*). O dispositivo *edge* encarrega-se de monitorizar e por vezes, controlar as ações dos dispositivos embebidos. Paralelamente, o dispositivo *edge* remete informações sobre

todos os processos para um repositório de dados conectado à Internet, atuando assim como um ponto de conexão entre os dispositivos embebidos e as infraestruturas alocadas na WWW. A Figura 2.3 mostra um exemplo da arquitetura dos sistemas IoT.

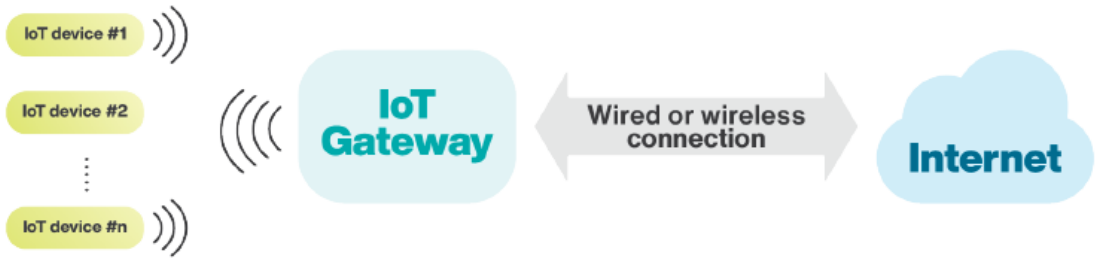


Figura 2.3 – Arquitetura típica de sistemas IoT [13].

Efetuar-se-á, nos subcapítulos seguintes, uma exposição de tipos de dispositivos e temas relevantes para o projeto e ao mesmo tempo que se englobam nos SDC e sistemas IoT.

2.2.1 Microcontroladores

Microcontroladores (MCU) são dispositivos compactos dimensionados para executar tarefas dentro do ramo dos sistemas embebidos. Os MCU são constituídos por uma unidade de processamento central (CPU), memórias (do tipo *Random Access Memory* (RAM), *Read Only Memory* (ROM), Flash entre outros), osciladores, portos porgramáveis de interação externa e outros componentes contidos num único *Integrated Circuit* (IC) ou *System On a Chip* (SoC). Devido ao seu pequeno tamanho, baixos consumos energéticos, relação custo-benefício, os MCU são empregues nas mais variadas aplicações desde brinquedos a equipamentos médicos [16].

Uma das características chave dos MCU é a sua capacidade de incorporar aplicações em tempo real. Estes dispositivos são programados para levar a cabo tarefas especificas, de uma forma periódica e fiável, tornando-os um componente ideal para controlar e monitorizar processos dentro dos ramos dos sistemas embebidos. A natureza dos MCU permite que estes interajam com um vasto leque de sensores, atuadores e outros dispositivos, tendo em conta restrições de *hardware* e dos sistemas em tempo real. Além do mais, os MCU são extremamente versáteis e permitem estabelecer comunicações com outros dispositivos através de diversos protocolos de comunicação.

Os MCU podem ser dimensionados com diferentes arquiteturas e variantes, consoante diversos requisitos. As arquiteturas mais comuns incluem os MCU de 8, 16 e 32 bits, cada uma com diferentes capacidades de poder computacional, tamanho de memória, pinos *General Porpose Input/Output* (GPIO), entre outos [16].

Estes dispositivos são, tipicamente, programados em Ambientes Integrados de Desenvolvimento (em inglês, *Integrated Devlopment Enviroment* (IDE)), com o auxílio de linguagens de programação como C, C++, Python ou em casos específicos com uma linguagem de baixo nível como Assembly. Este conjunto de instruções (programas) são, geralmente, denominadas como *firmware*. Os programas são responsáveis por ditar o comportamento e em que ordem devem ser executadas funções pelo MCU, consoante várias condições.

A Figura 2.4 mostra um exemplo dos principais blocos que compõem um MCU. É importante realçar que os MCU diferem consideravelmente entre si, sendo que os diagramas e circuitos podem diferir bastante de MCU para MCU.

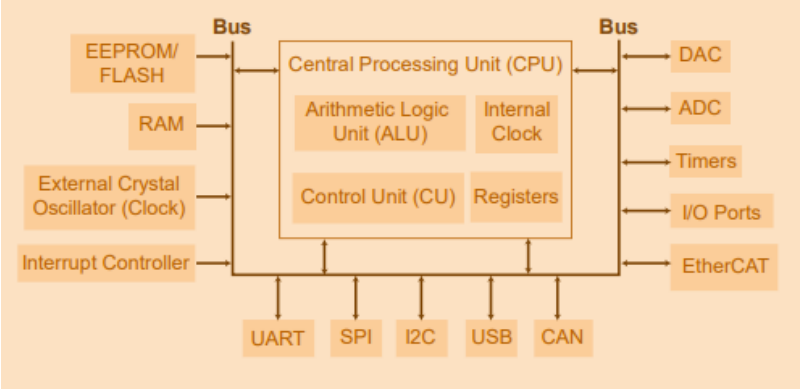


Figura 2.4 – Diagrama de blocos típicos de um MCU [17].

Com o avanço da tecnologia os MCU têm vindo a ficar economicamente mais acessíveis. Ao mesmo tempo estes IC têm visto um aumento significativo na velocidade de processamento, capacidade de armazenamento e uma diminuição nos seus consumos de corrente. Além disso, atualmente vários fabricantes têm, cada vez mais, começado a adicionar novos canais de transmissão de dados, como o Wi-Fi e *Bluetooth* [18].

A inclusão de MCU em projetos desta natureza permite a integração e interação entre vários dispositivos, como sensores, atuadores entre outros. O seu baixo consumo energético e versatilidade possibilitam o desenvolvimento de sistemas eficientes e modulares.

A Tabela 2.1 mostra alguns dos MCU disponíveis no mercado.

Tabela 2.1 - Exemplos de MCU disponíveis no mercado [19, 20, 21, 22].

	Arduino UNO	ESP32	STM32	TI AM23
Processador	ATmega328P	XTensa LX6	ARM Cortex-M33	ARM Cortex-R5F
Frequência de relógio	16 MHz	2.4 GHz	100 MHz	800 MHz
Conectividade	GPIO, UART, I2C e SPI	GPIO, UART, I2C, SPI, Bluetooth e Wi-Fi	GPIO, UART, I2C, SPI, Bluetooth	GPIO, UART, I2C, SPI, Ethernet

2.2.2 Microcomputadores

Microcomputadores (MCP), também denominados como *single-board computers* (SBC), representam uma classe de dispositivos compactos que encapsulam um sistema computacional inteiro numa única *Printed Circuit Board* (PCB). Estes dispositivos são compostos por todos os componentes necessários para realizar operações computacionais, como um CPU, memórias, placas de rede, entre outros.

Os MCP tais como, Raspberry Pi e Jetson Nano, em contraste com os MCU, funcionam, tipicamente, com um sistema operativo (OS) que lhes permitem realizar tarefas e rotinas mais complexas do ponto de vista computacional. O OS concede aos MCP funcionalidades semelhantes a um computador tradicional, como capacidade de realizar multitarefas, organização de documentos, interações com redes externas, um *Graphical User Interface* (GUI), entre outras.

A presença de MCP em projetos desta natureza permite a integração e interação de vários nós de informação. Dentro do contexto deste projeto, os MCP servem como dispositivos *edge* dentro de uma rede distribuída, melhorando o processamento e a transmissão de dados fundamentais para os processos.

2.2.3 Sistemas operativos em tempo real

Sistemas operativos em tempo real, em inglês *Real Time Operating Systems* (RTOS) são um tipo de sistemas computacionais que processam dados e eventos, tal como o nome indica, em tempo real e nos quais, a correta execução das tarefas está intrinsecamente ligada a prazos específicos e restritos. A característica essencial desses sistemas é a capacidade de realizar operações e fornecer resultados dentro de limites temporais predeterminados e previsíveis, tornando-os adequados para aplicações críticas. Além de produzir os resultados corretos, a precisão temporal é fundamental para a eficácia desses sistemas [23].

Existe uma grande série de algoritmos utilizados no contexto do RTOS, como o *Cooperative scheduling*, *Round-Robin scheduling* e *Fixed priority pre-emptive scheduling* [24]. Sendo o último exemplo empregue no RTOS utilizado neste projeto. Os algoritmos definem como devem ser executadas as tarefas, prioridades, tempos mortos, entre outros.

Um componente crucial num RTOS é o escalonador, mais conhecido pelo termo em inglês *scheduler*. O *scheduler* é o componente responsável pela gestão e atribuição eficiente dos recursos do sistema, como o processador, memória e periféricos, para as diferentes tarefas ou processos em execução e pela seleção da próxima tarefa a ser executada. Este componente escolha de execução das diversas tarefas consoante os diversos níveis de prioridade [23].

Os RTOS são fundamentais na implementação de *firmware* robusto em MCU. Implementando rotinas sobre um RTOS, o conceito de multitarefa e segurança entre tarefas (em inglês *thread safety*), tornam-se realidade o que proporciona uma alta eficiência e fiabilidade no *firmware* embutido nos MCU [23].

Alguns exemplos de RTOS:

- FreeRTOS

  Sistema desenvolvido pela Real Time Engineers Ltd e suportado pela Amazon. O código fonte é *open-source* (código fonte e licenças disponíveis gratuitamente), ideal para sistemas com baixos recursos computacionais. O FreeRTOS pode ser

implementado sobre diferentes arquiteturas de CPU o que o torna num RTOS bastante abrangente [25];

- VxWorks

  Sistema desenvolvido pela Wind River. A sua utilização requer uma licença avaliada em alguns milhares de euros. É um RTOS utilizado principalmente para aplicações *safety critical* na indústria do espaço e aviação. O VxWorks tem a particularidade de ser um RTOS que atinge certos requisitos que o permitem ser certificado em algumas diretivas da *International Electrotechnical Commission* (IEC) e *International Organization for Standardization* (ISO), como a DO-178C [26].

Dar-se-á enfase ao FreeRTOS uma vez que foi o RTOS utilizado neste projeto. Sendo as principais razões desta escolha o facto de ser o RTOS nativo do MCU escolhido para realizar este projeto e a vasta e facilmente acessível documentação [27].

2.2.4 FreeRTOS

Como já foi referido anteriormente (secção 2.2.3), o FreeRTOS é um sistema de tempo real criado e desenvolvido pela Real Time Engineers Ltd. Esta empresa trabalha em paralelo com os principais fabricantes de microcontroladores e microprocessadores de modo a acompanhar tendências e fornecer as melhores soluções de forma gratuita [27]. Este RTOS oferece a possibilidade de organizar multitarefas (*multitasking*), vários níveis de priorização das tarefas, filas, semáforos, entre outras, funções essenciais para construir um sistema em tempo real.

O FreeRTOS é essencialmente um *scheduler* que funciona em tempo real. Este *scheduler* tem a capacidade de funcionar sobre as aplicações de baixo nível dos MCU (como por exemplo a contagem de pulsos de relógio e interrupções) de modo a atingir certos requisitos impostos pelos programas.

As próximas seções têm como objetivo contextualizar alguns conceitos importantes relacionados com os sistemas RTOS, particularmente com o FreeRTOS.

2.2.4.1 Tarefas

As tarefas são fundamentais nos sistemas multitarefa e nos RTOS em geral. No FreeRTOS cada tarefa é essencialmente uma função com um ciclo infinito, neste ciclo são chamadas as respetivas instruções. É importante notar que no FreeRTOS as tarefas nunca podem ser finitas, ou seja a sua respetiva função deve conter, salvo algumas raras exceções, um ciclo infinito.

As tarefas podem ser declaradas com diversos níveis de prioridade e indexadas a um núcleo específico. Para além, disto o programador tem um controlo absoluto sobre a alocação de memória para cada tarefa [27].

2.2.4.2 Contabilização de tempo e execução de tarefas

Nos RTOS a contabilização do tempo é uma função extremamente importante, uma vez que o *scheduler* necessita ter uma base temporal para aplicar decisões importante, como por exemplo qual tarefa executar.

O FreeRTOS contabiliza o tempo com a ajuda de uma interrupção periódica, denominada de "tick". Esta interrupção é executada com uma frequência configurável. A interrupção tick tem como função dar a base de tempo necessária para a maioria dos processos e funções do FreeRTOS e RTOS em geral [27].

2.2.4.3 Filas

Como previamente explicado, em sistemas multitarefa podem existir paralelismos entre tarefas, ou seja, mais do que uma tarefa pode encontrar-se a ser executada ao mesmo tempo pelos CPUs. Nestes sistemas as variáveis e recursos de *hardware* devem ser tratadas com muito cuidado de modo a não afetar o funcionamento do *firmware*.

As filas ou *queues*, em inglês, são ferramentas do FreeRTOS que permitem, de uma forma segura, ler e escrever dados entre tarefas. As filas funcionam com o conceito *first in, first out* (FIFO), ou seja, os primeiros dados enviados para a fila, são os primeiros a ser recebidos. Outra grande vantagem das filas, no FreeRTOS, é o facto das variáveis serem passadas por cópia e não por endereço, ou seja, mesmo que as variáveis copiadas para a

fila deixem de existir num contexto global o seu conteúdo mantém-se guardado na RAM até que este seja recolhido [27].

2.2.4.4 Semáforos

Semáforos são ferramentas de sincronização fundamentais nos RTOS para gerir recursos partilhados (como variáveis ou acessos a protocolos de comunicação) e coordenar a execução de tarefas. Os semáforos podem ser vistos como controladores do número de acessos a um determinado recurso ou um controlador de rotinas críticas [27]. Estes sincronizadores podem ser divididos em dois tipos:

- Semáforos Binários

  Como o próprio nome sugere, estes tipos de semáforos podem assumir apenas dois valores, sendo estes 0 ou 1. São, geralmente, utilizados para sinalizar eventos ou proteger secções críticas na execução de tarefas;

- Semáforos contadores

  Os semáforos contadores permitem sincronizar e gerir o acesso de a recursos críticos. A principal vantagem deste tipo de semáforos é a capacidade de conter valores superiores a 1, o que significa que podem controlar o acesso a recursos com base no valor atual do semáforo contador.

No contexto dos semáforos, é importante delimitar o conjunto de instruções que acedem a recursos partilhados. Este conjunto de instruções devem executar o mais rapidamente possível e somente instruções que acedam a recursos partilhados por várias tarefas. A este conjunto de instruções dá-se o nome de secção critica, em inglês *critical section* [27].

2.2.4.5 Notificações intertarefas

As notificações intertarefas permitem a comunicação entre tarefas de uma forma flexível e eficaz. As notificações intertarefas não necessitam de uma variável do tipo *handler* dedicada para estabelecer o sincronismo ou aceder a um recurso partilhado. Estas notificações podem, em muitos casos, ser mais eficientes, em termos de velocidade de

processamento e de memória, do que semáforos binários, semáforos contadores ou até mesmo filas [27].

2.2.5 Protocolos de comunicação

Protocolos de comunicação são a linguagem que permite vários dispositivos e sistemas de diversas complexidades, partilhar informação e executar ações coordenadas. No contexto deste projeto o papel dos protocolos de comunicação não deve ser subestimado. Estes protocolos são a espinha dorsal deste sistema de monitorização, isto porque são responsáveis por interligar sensores, atuadores, controladores entre outros sistemas.

Os próximos subcapítulos têm como objetivo contextualizar os protocolos de comunicação dentro dos vários ramos em que estes se inserem. Explicar-se-ão as suas propriedades gerais, esquemas de ligação e como se definem dentro do modelo *Open Systems Interconnection* (OSI).

2.2.5.1 Modelo OSI

O modelo OSI é uma *framework* fundamental na área de redes e comunicações. Desenvolvido pela *International Organization for Standardization* (ISO), o modelo OSI fornece uma estrutura clara que facilita a análise de como os dados são transmitidos entre dispositivos. Este modelo dividido em sete camadas permite separar a análise de processos complexos que surgem na comunicação de dados. Ao dividir o processo de comunicação em diversas camadas, o modelo OSI facilita a compreensão de vários protocolos de comunicação. Sem o modelo OSI as redes e protocolos de comunicação seriam muito difíceis de entender e implementar [28].

A Figura 2.5 mostra as sete camadas do modelo OSI. Cada camada é responsável por definir uma série de parâmetros:

1. Camada Física

   A primeira camada define os parâmetros relacionados com o *hardware* e transmissão de dados binários. A camada física foca-se em definir características como níveis de tensão nos condutores, os tipos de cabos e condutores, conectores e taxas de transmissão;

2. Camada de Ligação de Dados

A segunda camada do modelo OSI responsabiliza-se por definir os pacotes de dados, deteção de erros e colisões. A camada de ligação de dados organiza os dados em pacotes e define formas de evitar erros no envio de dados;

3. Camada de Rede

Esta camada é responsável por definir o encaminhamento e endereçamento. A camada de rede encaminha os dados pelo melhor caminho, tendo em conta diversas redes de dados;

4. Camada de Transporte

A quarta camada do modelo OSI responsabiliza-se pela comunicação ponto a ponto entre dispositivos. Dependendo da aplicação e do protocolo, a camada de transporte define a verificação de erros, controlo de fluxo e segmentação dos dados;

5. Camada de Sessão

Esta camada é responsável por definir métodos de controlar e manter a sessão de comunicação entre dispositivos. A camada de sessão estabelece, mantém e termina as conexões entre dispositivos;

6. Camada de Apresentação

A sexta camada do modelo OSI responsabiliza-se pela tradução e encriptação dos dados. Esta camada é responsável por definir métodos de formatação de dados de modo a serem corretamente interpretados pela sétima camada;

7. Camada de Aplicação

A última camada do modelo OSI é a mais próxima do utilizador. Esta camada organiza interfaces e aplicações para o acesso dos dados pelo utilizador.

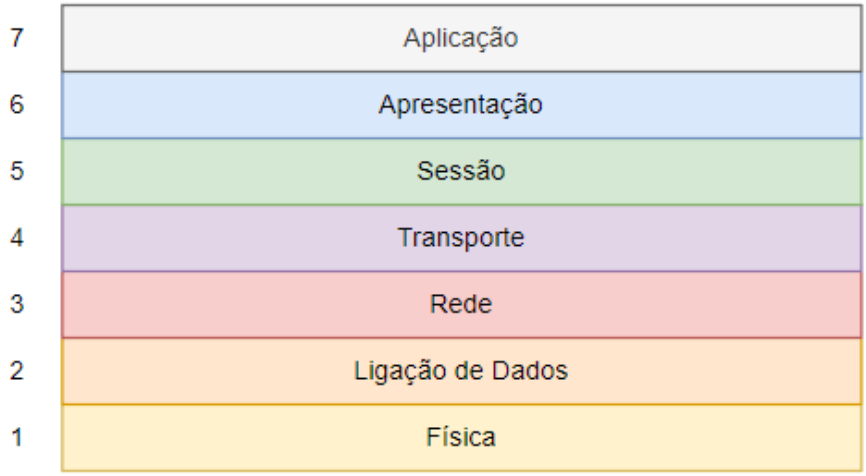


Figura 2.5 – Representação das camadas do Modelo OSI.

O modelo OSI será utilizado para enquadrar os protocolos de comunicação enunciados de seguida.

2.2.5.2 Universal Asynchronous Receiver/Transmitter (UART)

O protocolo de comunicação UART é utilizado para estabelecer a comunicação série entre dois dispositivos eletrónicos entre curtas distâncias. Aperfeiçoado pela IBM nos meados do século XX, o protocolo UART é conhecido pela sua simplicidade e versatilidade [29].

UART é um protocolo de comunicação, assíncrono e *full-duplex*, que permite interligar dois dispositivos entre si. O tamanho dos dados efetivos transmitidos pode variar entre cinco e nove bits. Para que a comunicação seja feita corretamente entre os dois dispositivos, ambos devem estar configurados com a mesma taxa de transmissão, em inglês *baud rate*. No contexto do protocolo UART, o *baud rate* define o número máximo de bits por segundo que podem ser transferidos [29].

Do ponto de vista do modelo OSI, o protocolo de comunicação UART é de baixo nível. Este protocolo é definido apenas pelas primeiras camadas do modelo OSI, nomeadamente a primeira e segunda camada. Na primeira camada e associado ao protocolo UART, surgem protocolos como o RS-232 ou RS-485 que definem aspetos físicos como níveis

de tensão, comprimentos de condutores entre outros. Na segunda camada são definidos os *frames* de dados típicos do protocolo UART e interpretação de erros.

A comunicação entre os dois dispositivos é feita através dos canais transmissores (TX) e recetores (RX). A conexão entre os dois dispositivos é cruzada, ou seja, o RX de um dispositivo é conectado ao TX do outro dispositivo e vice-versa. A Figura 2.6 mostra a ligação entre dois dispositivos que pretendem comunicar via UART.

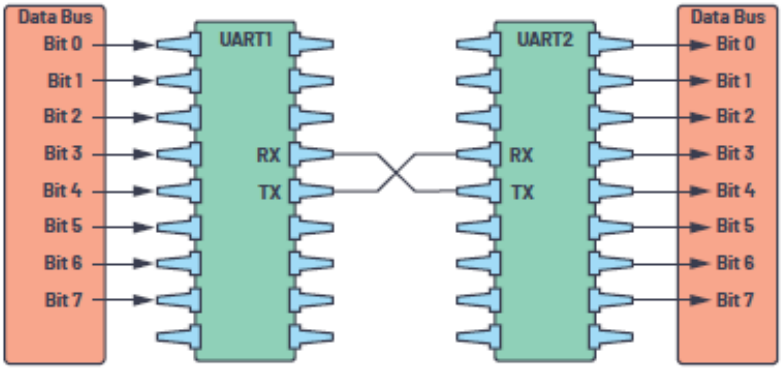


Figura 2.6 - Esquema de ligação UART [29].

A Tabela 2.2 sumariza alguns aspetos fundamentais do protocolo de comunicação UART.

Tabela 2.2 – Sumário do protocolo UART [29].

Número de fios	2
Método de transmissão	Assíncrono
Número máximo de mestres	1
Número máximo de escravos	1

2.2.5.3 Inter-Integrated Circuit (I2C)

O protocolo de comunicação *Inter-Integrated Circuit* (I2C) é muito utilizado no mundo da eletrónica e dos sistemas embebidos. Desenvolvido pela Philips (agora NXP Semiconductors) nos anos oitenta, o I2C é um protocolo de comunicação *half-duplex*, síncrono do tipo mestre-escravo e é, normalmente, utilizado para estabelecer a comunicação entre dispositivos a curtas distâncias [30].

O I2C é um protocolo de baixo nível. Este protocolo fixa-se na primeira e segunda camada do modelo OSI. Na primeira camada e tal como já foi referido anteriormente (secção 2.2.5.1), são definidos os aspetos físicos como níveis de tensão nos barramentos, distâncias máximas entre dispositivos, entre outros. Na camada de ligação de dados é definido como são construídos os pacotes I2C, endereçamento e deteção de erros.

No protocolo I2C a comunicação é efetuada através de dois barramentos: *Serial Data* (SDA) e *Serial ClockI* (SCL). O barramento SDA é responsável por transmitir os dados entre os dispositivos, enquanto o barramento SCL transmite o sinal temporal necessário para estabelecer a sincronização.

Os dispositivos que comunicam via I2C têm um endereço distinto de 7 bits que os identifica. Os endereços podem ser, muitas vezes, alterados através de comandos I2C sempre que seja necessário. O número máximo de dispositivos em cada barramento é de cento e vinte e oito. O dispositivo mestre está responsável por gerar a frequência (*clock*) e iniciar a transmissão de dados com os dispositivos escravos [30].

Os barramentos SDA e SCL devem estar sempre conectados a uma tensão positiva com a ajuda de uma resistência de *pull-up*. As resistências *pull-up* são utilizadas para assegurar um nível lógico *HIGH* sempre que o sinal digital se encontre aberto ou o barramento inativo. A Figura 2.7 exemplifica o circuito típico do protocolo I2C.

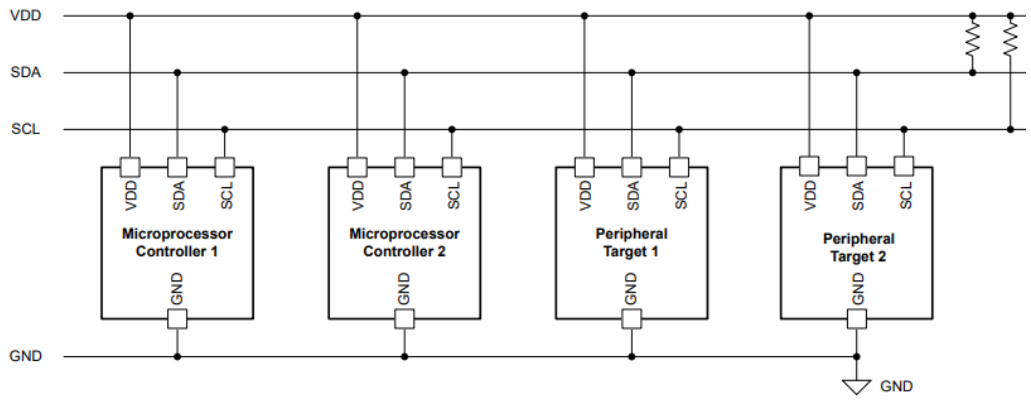


Figura 2.7 – Esquema de ligação I2C [30].

A Tabela 2.3 sintetiza alguns dos aspetos mais importantes do protocolo I2C.

Tabela 2.3 – Sumário do protocolo I2C [30].

Número de fios	2
Método de transmissão	Síncrono
Número máximo de mestres	Não especificado (*)
Número máximo de escravos	128
(*) – O número máximo de escravos não é especificado. Porém o número máximo de dispositivos acoplados aos barramentos I2C pode ser limitado por aspetos físicos, como capacidades, indutâncias e impedâncias parasitas.	

2.2.5.4 Serial Peripheral Interface (SPI)

O protocolo série, síncrono e *full-duplex, Serial Peripheral Interface* (SPI) é comumente utilizado no mundo dos sistemas embebidos. Criado e popularizado pela Motorola, o SPI é um protocolo de curta distância, do tipo mestre-escravo, que permite interligar um dispositivo mestre a vários dispositivos escravos [31].

Do ponto de vista do modelo OSI, o protocolo de comunicação SPI é de baixo nível. Este protocolo é definido apenas pelas primeiras camadas do modelo OSI, nomeadamente a primeira e segunda camada. Na primeira camada e associado ao protocolo SPI, são definidos os aspetos físicos como níveis de tensão, comprimentos de condutores entre outros. Na segunda camada são construídos os *frames* de dados típicos do protocolo SPI.

Para estabelecer a comunicação SPI são necessários três barramentos e um condutor distinto para cada dispositivo. O barramento responsável por transportar informação de um dispositivo escravo para o mestre é denominado como *Master In Slave* Out (MISO). Por outro lado, o barramento responsável por transportar informação do dispositivo mestre para um dispositivo escravo é denominado como *Master Out Slave In* (MOSI). Como já foi referido anteriormente o protocolo SPI é síncrono, logo e para que a comunicação seja assertiva um barramento (SCLK ou CLK) é dedicado a partilhar os pulsos de relógio estabelecido pelo mestre. A forma de endereçamento do protocolo SPI é feita aplicando um nível lógico num condutor (*Chip Select* (CS)) dedicado a cada

escravo. A Figura 2.8 ilustra a ligação entre um dispositivo mestre e quatro dispositivos escravos.

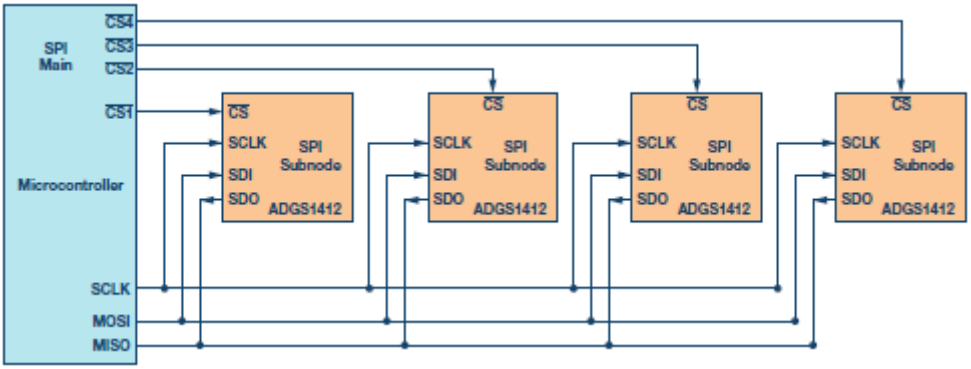


Figura 2.8 - Esquema de ligação usual do protocolo SPI [31].

A Tabela 2.4 sumariza alguns dos aspetos mais importantes do protocolo de comunicação SPI.

Tabela 2.4 – Sumário do protocolo SPI [31].

Número de fios	3 + número de escravos
Método de transmissão	Síncrono
Número máximo de mestres	1 (*)
Número máximo de escravos	Não especificado (#)
(*) - Por norma, no protocolo SPI apenas um mestre comunica com vários escravos, porém e com ajuda de lógica adicional é possível acoplar aos barramentos outros dispositivos mestres; (#) – O número máximo de escravos não é especificado, sendo este número apenas limitado por questões físicas.	

2.2.5.5 ModBus

O protocolo série, assíncrono, *half* ou *full-duplex* ModBus é empregue, maioritariamente na indústria. Desenvolvido pela Modicon em 1979, o protocolo ModBus é do tipo mestre-

escravo, utilizado normalmente para interligar *Porgrammable Logic Controllers* (PLCs) e dispositivos industriais [32, 33].

No modelo OSI, o protocolo ModBus encontra-se definido na camada física e na camada de ligação de dados. A camada física define os aspetos físicos do próprio protocolo, como o comprimento dos condutores, métodos de ligação, conectores, níveis de tensão, entre outros. A segunda camada define os *frames* típicos do ModBus, deteção de erros, endereçamento entre outros.

O ModBus opera sobre diversos protocolos presentes na camada física, como o RS-232, o RS-485 e a Ethernet. Uma vez que a maioria dos sensores e equipamentos industriais, que adotam o protocolo ModBus, comunicam através do padrão RS-485, as seguintes explicações e estudos do ModBus seguirão como princípio este método de ligação [32, 33].

De forma a estabelecer a comunicação via ModBus, com o padrão de ligação RS-485, são necessários dois condutores (um par). O modelo RS-485 utiliza sinais diferencias entre dois condutores, denominados por A e B. A tensão aplicada no condutor A é simétrica da tensão do condutor B. O nível lógico dos barramentos é assumido através de diferenças de tensão entre os condutores A e B [33].

É importante notar que para garantir um bom funcionamento, nas terminações dos pares diferenciais A e B devem ser colocadas resistências com um valor determinado a partir do comprimento dos próprios barramentos. Os pares A e B são também, por norma, interlaçados de modo a reduzir indutâncias e capacidades parasitas responsáveis pelo efeito *crosstalk*. Podem ser acoplados aos pares ModBus duzentos e quarenta e sete escravos, com o princípio de que cada um tenha um endereço específico. A Figura 2.9 exemplifica o esquema de ligação ModBus segundo o padrão RS-485 [33].

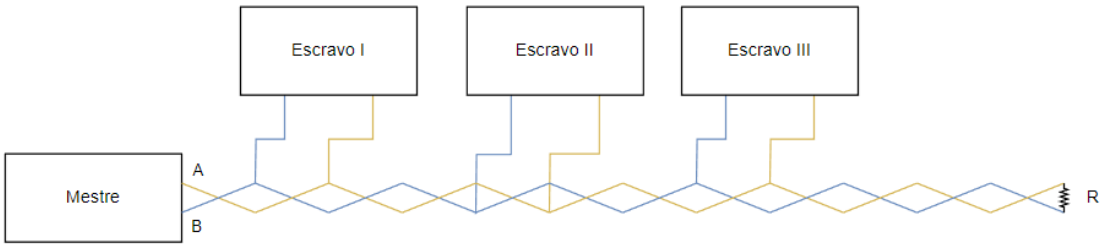


Figura 2.9 - Esquema de ligação do protocolo ModBus.

A Tabela 2.5 sumariza alguns dos aspetos mais importantes do protocolo de comunicação ModBus.

Tabela 2.5 – Sumário do protocolo ModBus [33].

Número de fios	2
Método de transmissão	Assíncrono
Número máximo de mestres	1
Número máximo de escravos	254 (*)
(*) – O número máximo de escravos acoplados aos barramentos A e B é limitado pelo tamanho do bloco de oito bits de endereçamento presente no pacote ModBus. Porém, o modelo RS485 define um número máximo de dispositivos acoplados aos seus barramentos de trinta e dois [33].	

2.2.5.6 Message Queuing Telemetry Transport (MQTT)

MQTT é um protocolo bidirecional, eficiente e flexível, do ponto de vista computacional, dimensionado para aplicações *machine-to-machine* (M2M) e IoT. Criado no final do século XX pela *International Business Machines Corporation* (IBM), o protocolo MQTT é, atualmente, *opne-source* e adotado por várias indústrias para desempenhar vários propósitos [34].

O protocolo MQTT é considerado um protocolo de alto nível. Do ponto de vista do modelo OSI, o MQTT opera e é definido principalmente na camada de aplicação. Este protocolo de comunicação fornece métodos para que dispositivos e aplicações troquem mensagens bilateralmente. É importante notar que o protocolo MQTT opera sobre vários protocolos de mais baixo nível como o *Transmission Control Protocol/ Internet Protocol* (TCP/IP) [34].

Na transmissão de informação, o protocolo MQTT segue o padrão publicação/subscrição, onde os clientes podem agir como publicadores e subscritores. Neste modelo, clientes publicam mensagens categorizadas ou endereçadas por tópicos e outros clientes subscrevem aos tópicos de interesse. É importante notar que os clientes MQTT podem

atuar como publicadores e subscritores simultaneamente. Este método de comunicação assíncrono é bastante viável para cenários onde as aplicações ou clientes necessitem de atualizar parâmetros regularmente sem ter de realizar pedidos regulares. Em termos de segurança, o protocolo MQTT pode operar sobre a camada de encriptação *Transport Layer Security* TLS e ainda adicionar credenciais associadas a tópicos [34].

O *broker* é um *software* de apoio importante na comunicação MQTT. É um intermediário entre os clientes MQTT que facilita a troca de mensagens entre os dispositivos. O objetivo principal do *broker* MQTT é delegar mensagens de clientes publicadores para os clientes subscritores. Assim que um cliente publica uma mensagem com um tópico específico, o *broker* encarrega-se de entregar a mensagem a todos os dispositivos que subscreveram ao determinado tópico. Certos tipos de *brokers* podem também guardar temporariamente certas mensagens. Esta característica pode ser útil em casos onde certos clientes se encontrem offline e não consigam receber certos tópicos subscritos. Assim que os dispositivos clientes fiquem online o *broker* encarrega-se de enviar as mensagens perdidas. É importante notar que o protocolo MQTT não exige o uso de *brokers,* porém são aconselhados, uma vez que tornam o processo de troca de mensagens menos exigente entre os clientes e aumentam a robustez, modularidade e eficiência do processo [34, 35].

A Figura 2.10 mostra um exemplo da partilha de informação através do protocolo MQTT. Neste exemplo cinco dispositivos clientes publicam e subscrevem diversos tópicos. Os clientes três e quarto publicam e subscrevem a tópicos distintos simultaneamente. O cliente cinco subscreve dois tópicos e os clientes um e dois publicam apenas um tópico cada. O dispositivo *broker* encontra-se no meio de todos os clientes e tem como função principal delegar as mensagens MQTT para os clientes corretos.

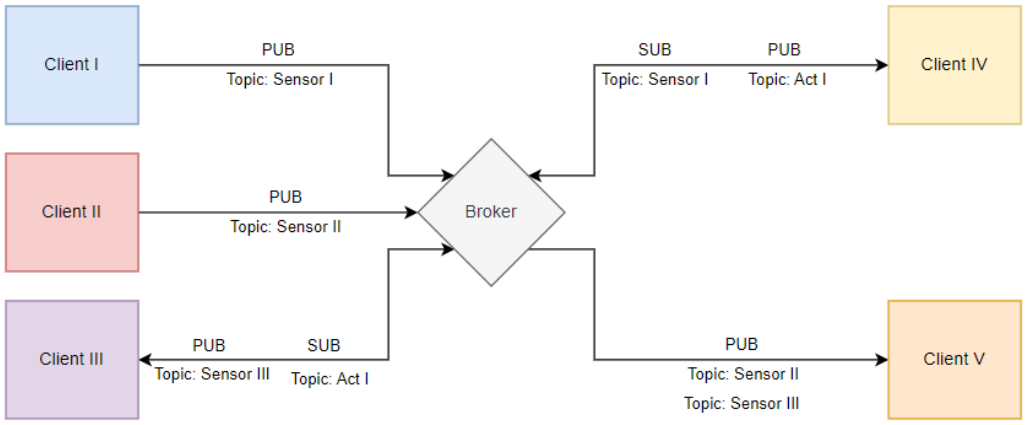


Figura 2.10 - Esquema de transmissão sobre o protocolo MQTT.

2.2.5.7 Hypertext Transfer Protocol (HTTP)

O protocolo HTTP é fundamental na comunicação e interligação de dispositivos e aplicação via web. Este protocolo estabelece a base da comunicação de dispositivos sobre a WWW. Criado no final do século XX em conjunto com a própria WWW, este protocolo permite, essencialmente que páginas web interajam com informações provenientes de servidores [36].

No contexto do modelo OSI, o protocolo de comunicação HTTP é considerado como um protocolo de alto nível. Este protocolo é definido na camada de aplicação e a sua função é criar uma ponte entre as aplicações web, *software* e outros protocolos nas camadas inferiores. É importante notar que o protocolo HTTP funciona sobre alguns protocolos de mais baixo nível, como os TCP/IP, *Domain Name System* (DNS), *Uniform Resource Locator* (URL) entre outros. Por uma questão de simplicidade estes protocolos não serão abordados, uma vez que não se enquadram no contexto geral do projeto [36].

A comunicação entre dispositivos via HTTP é feita através de um modelo cliente-servidor. O início da comunicação parte do dispositivo cliente, sendo que o pedido desencadeado pode ser de várias naturezas. Assim que o pedido chega ao servidor, este processa-o de acordo com o próprio pedido. Esta ação pode envolver recolher dados do servidor e/ou gerar uma resposta. A resposta e/ou dados pedidos são enviados de volta pelo servidor para o dispositivo cliente. Por fim, o dispositivo cliente recebe e processa

os dados recebidos. A Figura 2.11 exemplifica o processo, ao longo do tempo, descrito anteriormente [36].

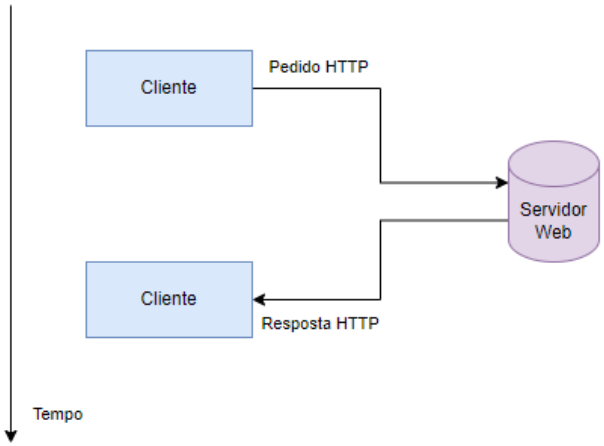


Figura 2.11 – Diagrama temporal de um pedido HTTP.

Tal como já foi referido anteriormente, existem vários pedidos HTTP. Cada pedido desencadeia por parte do servidor um processo distinto. Estes métodos HTTP possibilitam a interação entre clientes e servidores [36].

2.2.6 Sistemas de monitorização e armazenamento de dados

Sistemas de monitorização de dados no contexto dos sistemas distribuídos de controlo envolve a observação, recolha e análise continua de diversos parâmetros e sinais críticos para as operações dos sistemas. Múltiplos sensores recolhem dados em tempo real que são posteriormente processados e monitorizados. A monitorização em tempo real permite que os processos sejam facilmente acompanhados, aumentando a sua eficiência.

Sempre que é referido a monitorização ou interação com os dados dos processos, é importante contextualizar os *Human-Machine Interface* (HMI). Estes *softwares* têm como objetivo criar uma ponte entre os humanos e os processos. Os HMI são plataformas, por norma, interativas que permitem monitorizar, gerir e interagir processos complexos em tempo real. Estas plataformas oferecem representações gráficas e métodos de controlo intuitivos, como por exemplo *Supervisory Control and Data Acquisition* (SCADA) ou o Simatic WinCC [37].

A Figura 2.12 mostra um exemplo de um HMI baseado no sistema da Siemens WinCC. Neste HMI encontram-se representados vários atuadores e informações provenientes de sensores que permitem acompanhar um certo processo [15].

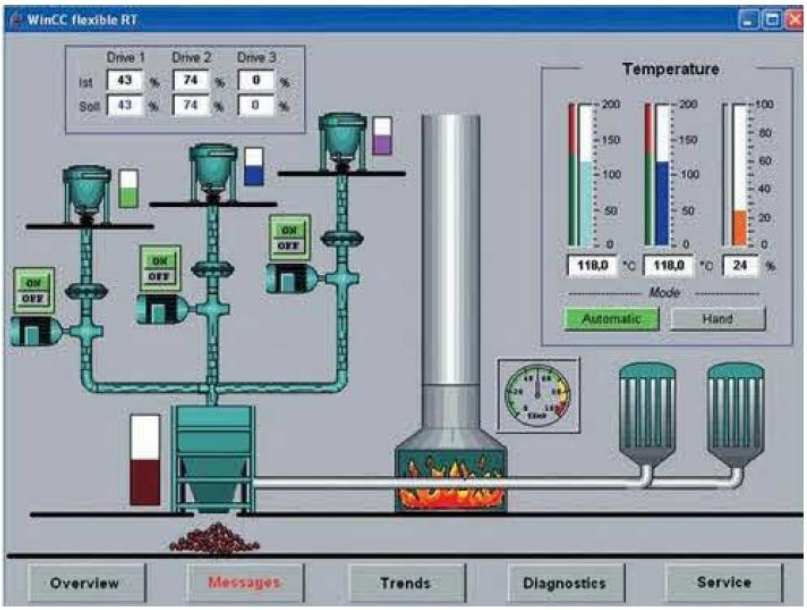


Figura 2.12 – Exemplo de um HMI [15].

Para além dos HMI, existem *softwares* que permitem visualizar os dados em painéis interativos. Estes *soft*wares são menos interativos que os HMI, porém são os mais indicados para aplicações que apenas seja necessário visualizar dados em forma de gráficos ou manómetros. Alguns exemplos destes *softwares* incluem o Grafana e o *software* desenvolvido por colegas do curso de Engenharia Informática do Instituto Politécnico de Tomar (IPT), conhecido por Monitorização Remota de Infraestrutura Piloto de Tratamento de Águas Residuais por via Biológica (MRIPTA) [38].

A Figura 2.13 mostra um exemplo de um *dashboard* criado com o *software* Grafana. Neste exemplo estão a ser monitorizados dados relevantes de um sistema de monitorização de energia elétrica [38].

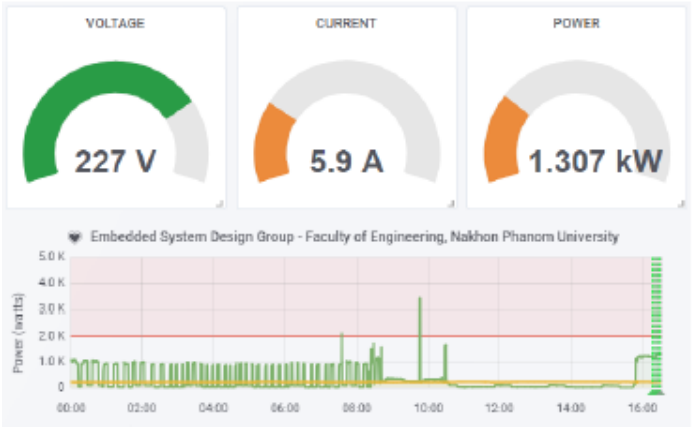


Figura 2.13 – Exemplo de um dashboard [38].

Aliado aos sistemas de monitorização, encontram-se os sistemas de armazenamento de dados. Os sistemas de armazenamento são responsáveis por reter e por vezes organizar um vasto volume de dados gerados por sensores, atuadores e outros sistemas [38]. A retenção de dados é fundamental na monitorização de processos, uma vez que permite efetuar análises históricas e identificar tendências que podem ocorrer nos processos. Estes sistemas são por norma empregues em servidores ou computadores dedicados que permitem guardar e organizar grandes fluxos de dados.

2.3 Conclusões do capítulo

Neste capítulo foram abordados e introduzidos alguns conceitos subjacentes às ZHC e monitorização e controlo de processos de forma autónoma e remotamente.

Foram enfatizados os sistemas IoT e SDC e abordados os dispositivos, RTOS, protocolos de comunicação e sistemas de monitorização considerados importantes no estudo e análise dos sistemas de monitorização remota.

Concluindo, a integração de MCU e SBC constitui a espinha dorsal da aquisição e processamento de dados, garantindo a aquisição e análise de informações em tempo real de diversos sensores e atuadores. A implementação dos RTOS garante precisão e exatidão na execução de tarefas críticas, contribuindo para a eficiência do sistema de monitorização. Os protocolos de comunicação possibilitam a troca periódica de dados, promovendo a comunicação entre dispositivos e utilizadores, criando assim uma rede de dados coesa. Além do mais, os sistemas de monitorização e armazenamento permitem

que os utilizadores supervisionem, estudem e controlem os processos monitorizados. Esta sinergia coletiva não só melhora a eficiência operacional, mas também fornece uma base sólida para garantir a integridade, segurança e otimização do processo.

3 Implementação do sistema de monitorização do projeto EcoModZHC

Para a implementação do sistema de monitorização de águas residuais foram selecionados, adquiridos e utilizados componentes disponíveis no mercado, com as características adequadas aos requisitos definidos para este projeto.

Neste capítulo são apresentadas as diferentes etapas até à implementação no terreno do sistema de aquisição de dados.

3.1 Projeto e desenvolvimento da UCAD

No contexto deste projeto, a UCAD tem como função a aquisição, armazenamento e a gestão dos parâmetros da ZHC. Trata-se de um sistema composto por vários componentes eletrónicos capazes de adquirir e gerir dados fundamentais para o projeto.

3.1.1 Componentes

Tal como mencionado anteriormente (secção 1.2), este projeto visa a recolha automática de informações críticas sobre o processo de filtragem de água de uma ZHC. De modo a realizar a análise de certos parâmetros físico-químicos foram selecionados uma série de sensores e atuadores com base em diferentes fatores.

Neste subcapítulo serão explicadas e justificadas as escolhas dos sensores, atuadores e microcontroladores que compõem a UCAD.

3.1.1.1 Sensores

O tratamento das águas residuais deve atender a diversos parâmetros exigidos pela APA no Decreto-Lei nº.152/97 [39]. Os sensores selecionados permitem analisar e conferir se os parâmetros exigidos estão dentro dos valores aceitáveis. É importante notar que alguns dos parâmetros exigidos, como a Carência Química de Oxigénio (CQO), têm de ser determinados por métodos convencionais de análise química, uma vez que não existem sensores deste tipo no mercado.

Escolheram-se os sensores que se descrevem no seguimento, pois são relativamente baratos, permitem estabelecer a comunicação entre dispositivos através de protocolos conhecidos e permitem analisar a maioria dos parâmetros químicos exigidos pela APA.

Neste projeto existem dois grupos de sensores: O primeiro grupo é constituído pelos sensores que se encontram sempre submersos em água; o segundo grupo é referente aos sensores auxiliares e que se encontram fora da água.

Os seguintes sensores encontram-se sempre submersos em água e têm como principal função aferir alguns parâmetros de qualidade da água.

- EZO-pH

  Sensor fabricado pela Atlas Scientific, tem como propósito recolher informações sobre o pH da água. O pH pode variar numa gama de zero a catorze. A sonda mede a atividade dos iões de hidrogénio na água através de uma membrana de vidro. Os dados deste sensor podem ser recolhidos através dos protocolos I2C ou UART. É importante notar que este sensor requer um IC auxiliar que traduz todos os dados recebido em pacotes UART ou I2C [40];

- EZO-EC

  Sensor desenvolvido pela Atlas Scientific, tem como objetivo analisar o parâmetro de condutividade elétrica (*Eletrical Conductivity* (EC)). A unidade de medida da EC é Siemens/m. A sonda aplica uma tensão entre dois elétrodos, o que causa com que os catiões se aproximem do elétrodo com cargas negativas, enquanto os aniões se aproximam do elétrodo carregado com cargas positivas. Os dados deste sensor podem ser recolhidos através dos protocolos I2C ou UART. É importante notar que este sensor requer um IC auxiliar que traduz todos os dados recebido em pacotes UART ou I2C [41];

- NTU SN-PNEP

  Este sensor tem como objetivo analisar a redução da transparência da água, que ocorre devido à presença de materiais em suspensão, também conhecido como turbidez. A *Nephelometric Turbidity Units* (NTU) em suspensão, propriedade também designada por turbidez. Fabricado pela Aqualabo Ponsel, este sensor analisa a turbidez emitindo e recebendo feixes de pulsos luminosos. Os dados deste

sensor podem ser analisados com a ajuda do protocolo de comunicação ModBus (RS-485) [42];

- ZWQ-COD

  Sensor fabricado pela Zata, tem como principal função aferir a estimativa da quantidade de carga orgânica presente na água. Este sensor usa a absorção de luz ultravioleta para obter uma estimativa do valor de Carência Química de Oxigénio (CQO) (em inglês *Chemical Oxygen Demand* (COD)), em miligramas de oxigénio por litro. Este sensor utiliza o protocolo ModBus (RS-485), como forma de dispositivos acederem às suas leituras [43].

É importante notar que como os parâmetros químicos a analisar são muito dependentes da temperatura da água, todos os sensores anteriormente mencionados permitem, adicionalmente, obter o valor da temperatura da mesma.

Os seguintes sensores não se encontram submersos, porém oferecem alguns parâmetros complementares para este projeto e auxiliam o processo em fases críticas.

- Estação Meteorológica ZWS

  Para analisar o comportamento da ZHC durante diferentes condições meteorológicas utilizou-se a estação meteorológica (EM) Zata ZWS que fornece uma série de dados, como a velocidade do vento, temperatura do ar, humidade, pluviosidade, radiação solar, entre outros. Fabricado pela Zata, a EM comunica os seus dados através do protocolo ModBus (RS-485) [46];
- Sensores de nível SEN0204

  Para aferir o nível de água num determinado recipiente, utilizaram-se sensores de nível capacitivos da DFRobot que não necessitam de estar em contacto direto com o líquido. Estes sensores aplicam um sinal digital *HIGH* no pino de sinal assim que algum líquido se encontre próximo [47].

3.1.1.2 Atuadores

De forma a controlar o caudal de água na entrada e na saída utilizaram-se duas bombas diferentes:

- Bomba Peristáltica BT100S

De modo a introduzir as águas residuais na ZHC, utilizou-se uma bomba peristáltica na entrada do sistema, para que os resíduos presentes na água residual não afetem o funcionamento da bomba e para que o controlo de caudal seja exato e preciso. A bomba peristáltica utilizada para garantir o caudal na entrada foi a BT100S da LeadFluid. Escolheu-se esta bomba peristáltica, uma vez que disponibiliza permite a sua monitorização através do protocolo ModBus [44];

- Bomba Submergível JT-180

  Como o caudal na saída do sistema não é significativo e a água já se encontra parcialmente filtrada, escolheu-se a bomba submergível JT-180 para pequenos caudais. Esta bomba consome uma corrente reduzida, e funciona com uma tensão de cinco VDC. Ao mesmo tempo consegue transportar líquido até 2,2 metros de altura [45].

3.1.1.3 Microcontrolador

O ESP32 é um MCU desenvolvido pela Espressif Systems. Este MCU é composto por dois núcleos XTensa LX6, que operam com uma frequência de relógio de duzentos e quarenta megahertz. O ESP32 possui internamente portos e registos dedicados para protocolos de comunicação como o UART, I2C e SPI, facilitando a interação entre dispositivos externos. Para além dos protocolos de comunicação, este MCU possui um sistema de interrupções rápido, registos contadores e vários portos de *Pulse Width Modulation* (PWM). Adicionalmente, contém internamente conversores *Analog to Digital Converter* (ADC) e *Digital to Analog Converter* (DAC). Este MCU contém também *interfaces* que possibilitam a comunicação através dos protocolos Wi-Fi e Bluetooth [20].

Para além do seu excecional processamento computacional, as funcionalidades de Wi-Fi integradas no ESP32 eliminam a necessidade de módulos externos adicionais, reduzindo o custo geral e a complexidade do projeto. Para além disso, o seu baixo consumo energético, suporte para vários modos de *sleep* e preço extremamente competitivo, tornam-no no MCU indicado para este projeto [20].

Neste projeto empregaram-se dois MCU ESP32. Um dos microcontroladores está responsável por recolher, aglomerar e reencaminhar a informação oriunda da EM e de todos os sensores à entrada da ZHC. O MCU responsável por controlar e monitorizar o

processo à entrada da ZHC está também responsável por recolher a informação do caudal devolvida pela bomba peristáltica. O ESP32 à saída é responsável por recolher, aglomerar e reencaminhar a informação dos sensores à saída da ZHC. O MCU presente na saída do processo está também responsável por atuar a bomba submergível à saída, consoante a informação dos sensores de nível.

As duas seguintes figuras (Figura 3.1 e Figura 3.2) ilustram superficialmente as funcionalidades dos MCU na entrada e saída da ZHC. É de realçar que a numeração dos pinos no MCU nas figuras é meramente ilustrativa.

A Figura 3.1 mostra um diagrama de blocos correspondente ao MCU presente á entrada da ZHC. Este ESP32 é responsável por fazer o pedido das leituras aos sensores de pH, EC, Turbidez, carga orgânica e EM e de remeter os dados recebidos para um sistema de supervisão e controlo explicado mais adiante (secção 3.2). Para além das leituras dos sensores, este MCU é responsável por controlar e ler as rotações por minuto da bomba peristáltica.

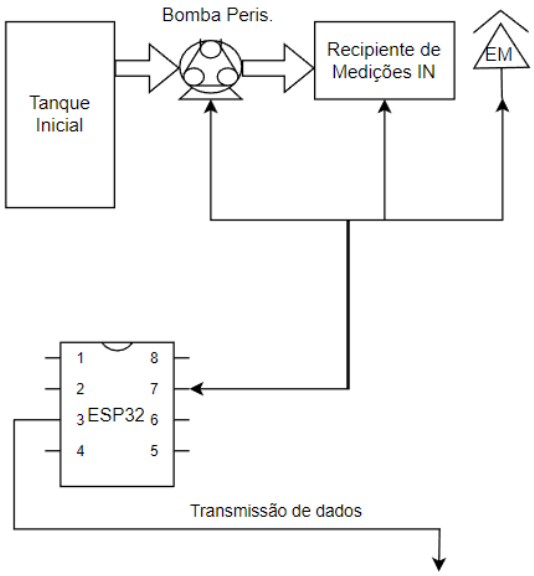


Figura 3.1 – Diagrama do MCU na entrada do processo.

O MCU à saída da ZHC é encarregue por solicitar e receber as leituras dos sensores submersos. O ESP32 também é encarregue de controlar uma bomba submersa com base nos dados dos sensores de nível. A informação do número de vezes que a bomba foi acionada é também remetida para o MCP através de protocolos *wireless*. A Figura 3.2 mostra os blocos associados ao MCU na saída do processo.

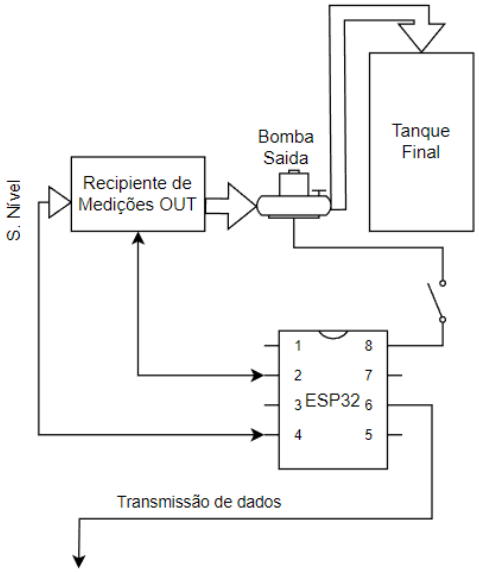


Figura 3.2 – Diagrama funcional do MCU na saída do processo.

3.1.1.4 Eletrónica auxiliar

Utilizou-se uma grande variedade de componentes eletrónicos para auxiliar diversos processos no âmbito deste projeto.

O módulo TEL007 [48], baseado no IC MAX485, permite a conversão dos sinais e níveis lógicos do protocolo UART para o protocolo ModBus com o esquema de ligação RS-485. Este módulo é fundamental neste projeto uma vez que vários sensores utilizados comunicam através do protocolo ModBus e o MCU escolhido não dispõe, nativamente, de uma *interface* ModBus.

Utilizaram-se relés G5V-2-H1 [49] para controlar a bomba submergível. Escolheram-se estes relés, pois a tensão de excitação da bobine é adequada aos níveis de tensão disponíveis. Para além da tensão de excitação, a escolha também passou pela corrente máxima, sendo esta suficiente para comutar a bomba submergível.

De forma a controlar os relés com o MCU, escolheram-se transístores de junção do tipo NPN BC547 [50] para atuarem como um interruptor. Ao utilizar estes transístores é garantido a tensão e corrente necessárias para controlar os relés, sem comprometer os pinos do MCU.

3.2 Projeto e desenvolvimento do MT

No seio deste projeto, o MT é representado por todos os equipamentos e métodos responsáveis pela transmissão de dados entre dispositivos. O MT é um sistema composto por um dispositivo *edge* ou *gateway* e a plataforma de visualização de dados na nuvem, responsável por representar e armazenar todos os parâmetros recolhidos. Além do mais, o MT é responsável pela utilização e gestão dos protocolos de comunicação de dados entre todos módulos que compõem este projeto.

3.2.1 Componentes

3.2.1.1 Dispositivo edge/gateway

O Raspberry Pi (RasPi) é um MCP equipado com uma plataforma robusta, capaz de funcionar sobre vários OS, sendo os mais utilizados os OS baseados em Linux como o Raspian ou o Unbutu [51]. A interação com o RasPi pode ser feita através de um GUI ou de um terminal SSH na sua rede local. Este MCP permite criar e executar diversas rotinas em várias linguagens de programação, sendo a sua versatilidade um ponto chave para este projeto.

Este MCP permite também expandir as suas capacidades através do *download* de pacotes com várias funcionalidades em repositórios de acesso gratuito. Esta funcionalidade providencia uma coleção de ferramentas de *software* quase ilimitada.

Para este projeto, optou-se pela utilização de um RasPi em vez de outro MCP, uma vez que este já se encontrava disponível no IPT e já se tinha trabalhado com este MCP anteriormente.

Neste projeto, utilizou-se um RasPi com o sistema operativo Raspian Bullseye 11 para servir como dispositivo *edge* entre o processo e o sistema remoto de visualização e armazenamento de dados. Tal como o nome indica este dispositivo encontra-se entre a fronteira do próprio processo físico e da rede de dados. Por outras palavras, o RasPi serve de intermediário entre os dados capturados nas diferentes fases da ZHC e o sistema de monitorização. O RasPi é responsável por receber os parâmetros dos sensores remetidos pelos MCU à entrada e saída da ZHC, juntar todos os dados em formato *JavaScript Object Notation* (JSON), e enviá-los para o sistema remoto de visualização e armazenamento de

dados através de protocolos de comunicação *wireless*. A Figura 3.3 mostra o diagrama fundamental correspondente ao RasPi.

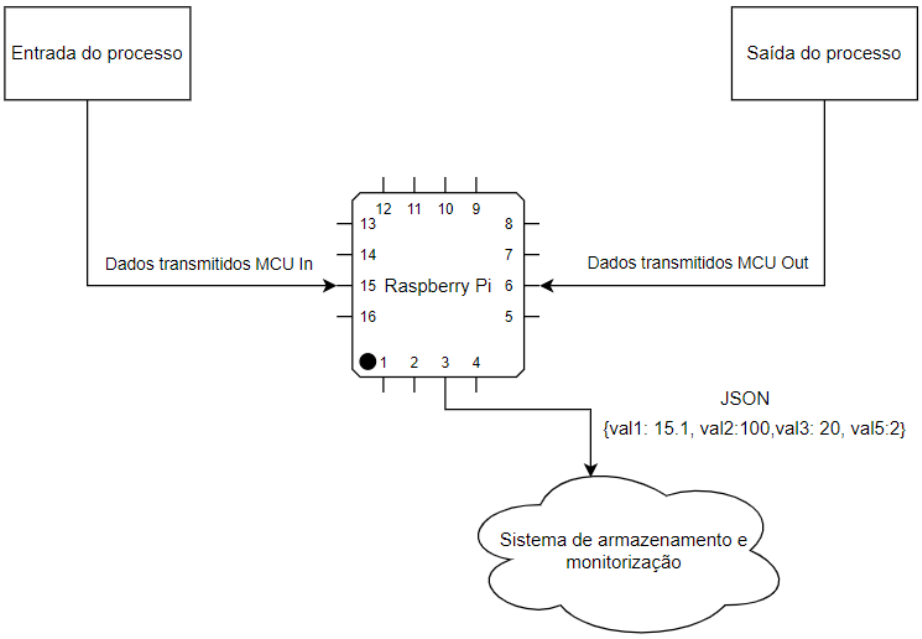


Figura 3.3 – Diagrama funcional correspondente ao MCP.

3.2.2 Sistemas de armazenamento e monitorização de dados

O MRIPTA é uma plataforma de visualização de dados que foi criado e dimensionado por alunos da Licenciatura em Engenharia Informática do IPT [52]. Esta plataforma engloba uma base de dados onde são guardados os dados e uma *Application Programming Interface* (API) onde os diferentes parâmetros podem ser visualizados. A Figura 3.4 mostra a página inicial da plataforma MRIPTA.

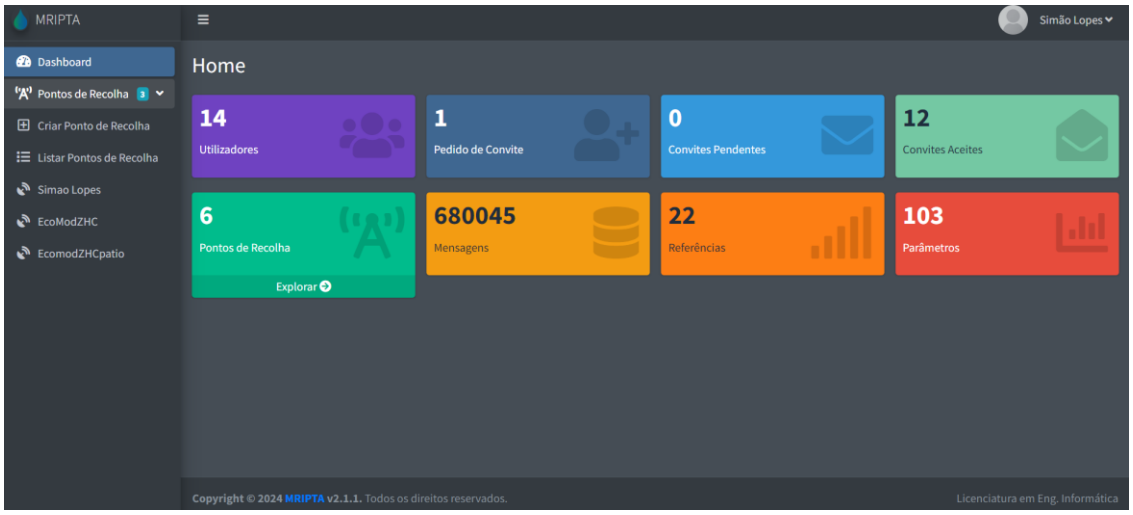


Figura 3.4 - Homepage da plataforma MRIPTA.

Os dados guardados no sistema de armazenamento do MRIPTA podem ser visualizados, atualmente, nas formas de gráficos e tabelas. Os parâmetros transmitidos pelo MCP podem ser também analisados em tempo real, isto é, o MRIPTA encarrega-se de atualizar os gráficos e tabelas assim que surgem novos dados. A base de dados encontra-se num servidor presente na rede interna do IPT. A interação com a plataforma por parte dos sistemas responsáveis por acompanhar o processo, pode ser feita através do protocolo de comunicação MQTT.

Tomou-se a decisão de utilizar este sistema de armazenamento e monitorização de dados, uma vez que existe um controlo total sobre a plataforma, evitando dependências de *softwares* de terceiros. Desta forma, o MRIPTA pode sempre sofrer atualizações para melhorar a sua interação com o utilizador e o próprio processo. A Figura 3.5 mostra um diagrama funcional da interação que os dispositivos têm com a plataforma MRIPTA.

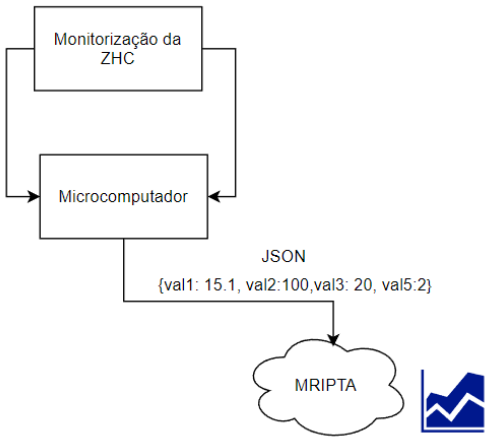


Figura 3.5 - Diagrama funcional correspondente ao MRIPTA.

3.2.3 Protocolos de comunicação utilizados no projeto

Os protocolos de comunicação permitem estabelecer a comunicação entre vários dispositivos. Neste projeto os protocolos de comunicação foram empregues para estabelecer a comunicação entre o processo físico e o utilizador.

A escolha dos protocolos a utilizar passou por analisar a sua viabilidade entre os dispositivos e se estes disponham do interface necessário. Deu-se sempre primazia aos protocolos *wireless* uma vez que evitam o uso de cabos e outros componentes.

Os sensores de pH e EC escolhidos permitem comunicar através dos protocolos I2C e UART. Para facilitar a comunicação entre os sensores e os MCU, decidiu-se evitar a implementação do protocolo UART. Como já foi referido anteriormente (secção 2.2.5.2), o protocolo UART exige um conjunto de dois pinos dedicados (RX/TX) para comunicar com cada sensor individualmente. Esta escolha resultaria na alocação de quatro pinos do MCU, o que, e numa perspetiva futura, poderia restringir o número de pinos disponíveis do MCU, prejudicando a possível expansão do sistema de monitorização. Desta forma, optou-se por adotar o protocolo I2C para a comunicação entre os sensores de pH e EC com o MCU, sendo apenas necessário reservar dois pinos do MCU para estabelecer os barramentos SDA e SCL (capítulo 2.2.5.3). O protocolo I2C aumenta significativamente o número de dispositivos que podem ser acoplados ao barramento I2C sem comprometer o número de pinos disponíveis.

A comunicação entre os sensores de carga orgânica, turbidez e estação meteorológica foi estabelecida através do protocolo ModBus. Este conjunto de sensores não apresentava uma variedade de protocolos de comunicação além do ModBus com o esquema de ligação RS-485. É importante notar que com o MCU escolhido a comunicação ModBus é feita através de um módulo que converte os sinais ModBus em UART (MAX485).

Para estabelecer as comunicações entre os MCU na entrada e na saída da ZHC, com o RasPi e a plataforma MRIPTA, escolheu-se o protocolo MQTT. A escolha do protocolo MQTT, entre os dois MCU e o RasPi, foi tomada tendo em consideração vários pontos importantes referidos no subcapítulo dedicado a este protocolo (capítulo 2.2.5.6). A facilidade em enviar novos pacotes de dados e o facto de estes estarem separados e indexados por tópicos, faz com que este protocolo seja eleito devido à sua modularidade e flexibilidade. Entre o dispositivo *edge* (RasPi) e a plataforma MRPITA, o protocolo MQTT foi mais uma vez utilizado, uma vez que a plataforma MRIPTA está configurada para comunicar através deste protocolo. Neste contexto, o RasPi atua como *broker* entre os MCU e o MRIPTA. A existência de um dispositivo intermediário é recomendada pois agiliza muito o processo de troca de pacotes MQTT. A sua existência não é obrigatória, porém é recomendada nos casos onde o volume de publicações e subscrições é elevado. Implementado o serviço de *broker* no RasPi, facilita-se a escalabilidade do sistema sem que a troca de pacotes MQTT seja comprometida [35].

Adicionalmente, utilizou-se o protocolo HTTP para criar um servidor local nos próprios MCU. Com a ajuda deste protocolo e a linguagem *HyperText Markup Language* (HTML), criou-se uma simples página que disponibiliza os dados atualizados de todos os parâmetros recolhidos pelos sensores. Desta forma, clientes HTTP como *browers* nos telemóveis ou computadores podem ligar-se ao IP do servidor alocado no MCU e realizar pedidos HTTP, fornecendo informações importantes sobre o sistema. Escolheu-se empregar este protocolo e consequentemente a página HTML para facilitar a análise do sistema de filtragem. Da mesma forma, este protocolo ajuda no *degub* de problemas no sistema, sem ter de aguardar pelo processo de comunicação até à plataforma MRIPTA.

A Figura 3.6 mostra um esquema geral de todo o sistema com os protocolos de comunicação diferenciados e evidenciados por cores. Além disso, a Figura 3.6 concilia a informação explicada nos parágrafos e subcapítulos anteriores numa só imagem. É importante notar nas setas, sendo que uma linha com uma seta no início e outra no fim

simboliza que o protocolo está a ser utilizado com funções de bidirecionalidade. Caso a linha apenas tenha uma seta, simboliza que o protocolo está a ser utilizado para transmitir dados unilateralmente. As linhas sólidas simbolizam um protocolo implementado com fios, enquanto um protocolo *wireless* é representado como uma linha a tracejado.

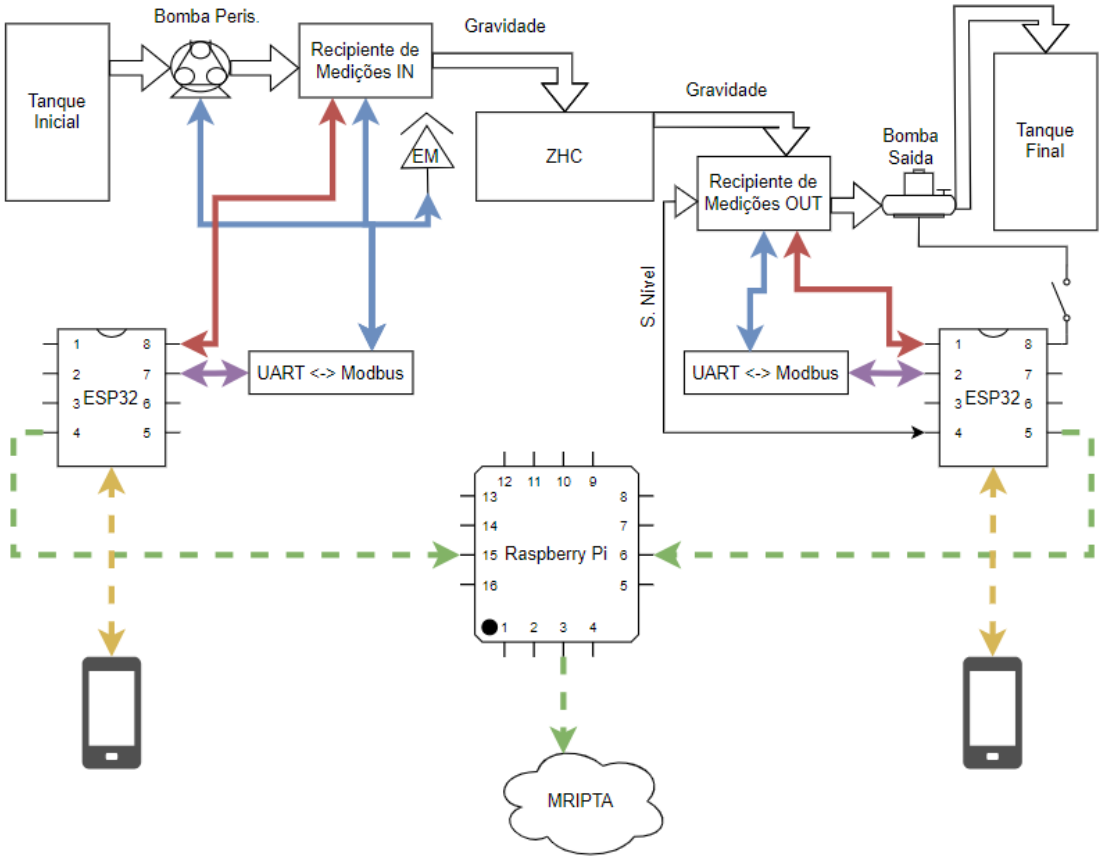


Figura 3.6 – Esquema funcional do projeto com os protocolos de comunicação empregues.

A Tabela 3.1 ajuda na interpretação dos protocolos de comunicação mostrados da Figura 3.6.

Tabela 3.1 – Informações de apoio à interpretação da figura Figura 3.6.

Nome do Protocolo	Implementação com fios	Cor correspondente
ModBus	Sim	
I2C	Sim	

UART	Sim	
HTTP	Não	
MQTT	Não	

3.3 Integração da programação no projeto

A programação desempenha um papel fundamental neste projeto, entrelaçando intrinsecamente as diferentes funcionalidades de todos os módulos que compõem o projeto. Neste subcapítulo encontram-se explicadas as rotinas e programação efetuadas para ambos os MCU ESP32 e o MCP RasPi. Para além da explicação do código implementado, justificam-se também as escolhas de linguagens e ambientes de programação em cada caso específico.

Todo o código desenvolvido e apresentado nos subcapítulos seguintes, pode ser acedido na sua íntegra através do repositório online presente em [53].

3.3.1 ESP32

Programaram-se os MCU ESP32 com o auxílio do IDE Visual Studio Code (VSC) com a extensão PlatformIO. A escolha do IDE passou essencialmente pela familiaridade com o *software* e a documentação disponível.

O ESP32 é usualmente programado através de duas plataformas: A *Espressif IoT Development Framework* (ESP-IDF) ou a *framework* Arduino [55]. Estas *frameworks* são implementadas sobre a camada de funções correspondente ao FreeRTOS, que tal como foi referido anteriormente (secção 2.2.4), é o sistema de tempo real pré-definido pelo fabricante deste MCU. É ainda importante notar que as linguagens de programação de ambas as plataformas são C ou C++. ESP-IDF é a plataforma oficial do fabricante. Esta plataforma fornece uma série de funções de baixo nível que permite aceder a funcionalidades especificas do MCU. Por outro lado, plataforma Arduino fornece um conjunto de funções mais acessíveis e *user-friendly*.

Escolheu-se utilizar uma mistura das duas *frameworks* para este projeto. Programou-se a maioria das rotinas na *framework* Arduino, devido às suas vastas bibliotecas e comunidade ativa. Utilizou-se, porém a *framework* IDF para compor rotinas especificas do MCU ESP32.

Paralelamente, decidiu-se programar estes dispositivos com a linguagem de programação C++ devido à sua versatilidade e capacidade de secionar e modular código, conseguindo ao mesmo tempo aceder a funções de mais baixo nível do tipo C.

3.3.1.1 Estrutura dos ficheiros e serviços

Optou-se por dividir o projeto em diferentes diretorias e subdiretorias, organizando e separando de forma coerente o código. O programa escrito encontra-se dividido em diferentes serviços, sendo que a cada serviço está atribuída uma subdiretoria que contém um ficheiro do tipo C *plus plus* (.cpp) e *Header plus plus* (.hpp) responsáveis por descrever as instruções que o MCU deve executar.

Para além das subdiretorias referentes a cada serviço, na diretoria *source* (src) surgem os ficheiros main.cpp e env.hpp. O ficheiro main.cpp contém as primeiras instruções que o MCU executa, enquanto o ficheiro env.hpp contém dados constantes importantes, como endereços I2C, ModBus, endereços *web* entre outros.

A pasta com o nome de *web* contém um ficheiro do tipo .html, com o código necessário para gerar uma página *web* em conjunto com o protocolo HTTP. A informação deste ficheiro não é compilada pelo IDE, porém é embutida na memória Flash do MCU e acedida posteriormente por um serviço.

A cada serviço atribui-se uma tarefa do FreeRTOS, sendo que a sua função e as funções auxiliares à tarefa encontram-se descrita nos respetivos ficheiros do tipo *.cpp* e *.hpp*. Desta forma, existe uma clara separação entre cada serviço que o ESP32 tem de processar, facilitando a sua programação e possíveis *debugs*.

A Figura 3.7 expõe a estrutura dos ficheiros anteriormente explicados que compõem o conjunto de serviços do MCU presente à saída da ZHC. A pasta src contém todas as subpastas de todos os serviços, a pasta web, os ficheiros main.cpp e env.hpp.

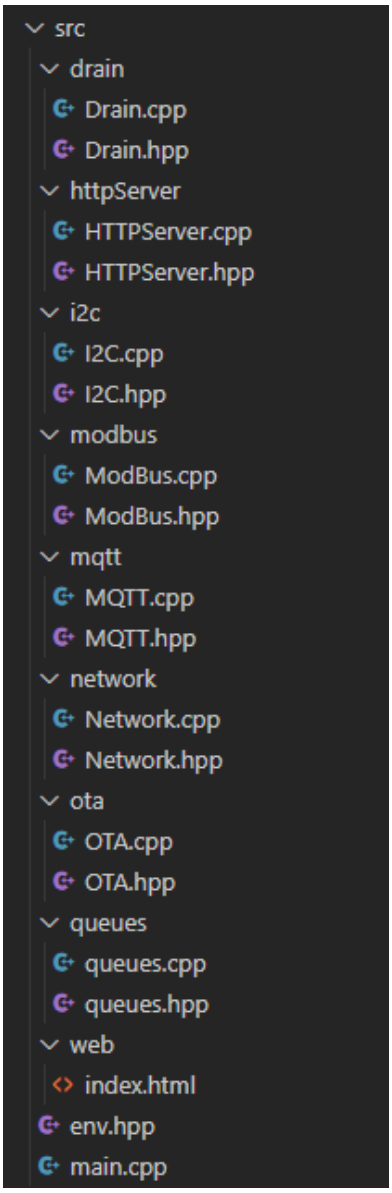


Figura 3.7 - Estrutura da diretoria src.

Fora da diretoria src surgem alguns ficheiros e diretorias adicionais que têm como função definir alguns parâmetros importantes. Estes parâmetros podem ser referentes ao IDE, compilador C++, Git ou do próprio MCU. A Figura 3.8 mostra de uma forma geral as diretorias e ficheiros que compõem o projeto de programação dos ESP32.

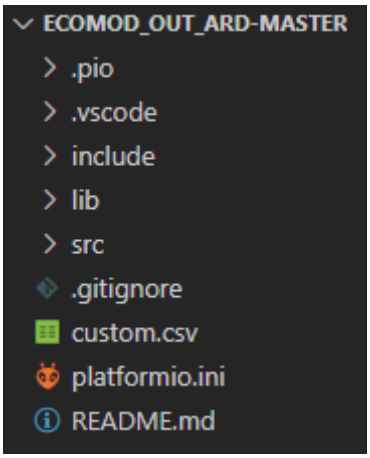


Figura 3.8 – Estrutura das diretorias e ficheiros do projeto de programação.

Um dos ficheiros importantes a referir é a tabela de partições (custom.csv). Neste ficheiro encontram-se definidas quais e qual o tamanho das partições da memória Flash do ESP32. Editou-se este ficheiro para possibilitar as atualizações *Over The Air* (OTA), possibilitando que a Flash do MCU consiga guardar dois binários distintos (partições ota_0 e ota_1). Para além das partições dos próprios binários, são definidas as partições que guardam as credenciais da rede Wi-Fi (*phy_init*), dados sobre o binário que se encontra a ser executado (*otadata*) e uma partição que permite ao programador guardar quaisquer parâmetros não voláteis (*nvs*) [20, 55]. A Figura 3.9 mostra os campos que compõe a tabela de partições composta para este projeto.

```
# Special partition table for unit test app_update
# Name,     Type, SubType, Offset,   Size, Flags
nvs,        data, nvs,     ,        0x4000
otadata,    data, ota,     ,        0x2000
phy_init,   data, phy,     ,        0x1000
ota_0,      0,    ota_0,   ,        1500K
ota_1,      0,    ota_1,   ,        1500K
```

Figura 3.9 - Tabela de partições.

Adicionalmente, o ficheiro *platformio.ini* é utilizado pela extensão IDE PlatformIO e tem como objetivo a configuração de diversos parâmetros, tais como, a definição do MCU para que se destina o código, qual a *framework*, a velocidade do monitor serial, a diretoria dos ficheiros a embutir na memória flash, a tabela de partições personalizada e as bibliotecas externas. Os diferentes campos definidos pelo ficheiro platformio.ini encontram-se expostos na Figura 3.10.

```
[env:esp32doit-devkit-v1]
platform = espressif32
board = esp32doit-devkit-v1
framework = arduino
monitor_speed = 115200
board_build.embed_txtfiles = src/web/index.html
board_build.partitions = custom.csv
lib_deps =
    knolleary/PubSubClient@^2.8
    bertmelis/esp32ModbusRTU@^0.0.2
    emelianov/modbus-esp8266@^4.1.0
```

Figura 3.10 – Ficheiro platformio.ini.

3.3.1.2 Algoritmos, funções e estruturas de dados

As tarefas do FreeRTOS são compostas pelas funções principais de cada serviço que constituem o projeto de programação. As funcionalidades principais de cada serviço são:

- HTTP *Server*

  Este serviço tem como função principal carregar a página HTLM descrita no ficheiro *index.html* embutido na Flash do ESP32 e executar todos os pedidos efetuados por clientes HTTP. Assim que um cliente HTTP deseje aceder aos dados mais atualizados dos sensores, este serviço acede às filas do FreeRTOS que contêm estes dados e disponibiliza-os na página web [54];

- I2C

  O serviço I2C executa todas as funções necessárias para que o ESP32 comunique através do protocolo I2C, com os dispositivos correspondentes. Adicionalmente, este serviço aglomera todos os parâmetros recebidos e coloca-os numa fila do FreeRTOS;

- ModBus

  Este serviço tem como função executar todas as instruções necessárias para que o ESP32 consiga estabelecer a comunicação com os dispositivos que comunicam através do protocolo ModBus. Adicionalmente, este serviço junta todos os parâmetros recebidos e coloca-os numa fila do FreeRTOS;

- MQTT

O serviço MQTT acede, através das filas do FreeRTOS, todos os parâmetros recolhidos dos serviços I2C e ModBus, agrupa-os em tópicos e transmite-os para o RasPi através do protocolo MQTT;

- *Network*

As instruções responsáveis por conectar o ESP32 à rede Wi-Fi local (modo *Station* (STA)) encontram-se programadas neste serviço. Paralelamente, o serviço *Network* é responsável por executar todas as funções que permitam com que o ESP32 crie a sua rede própria local (modo *Access Point* (AP)) [56];

- *Over The Air* (OTA)

Para que seja possível fazer um *update* de *firmware* através do Wi-Fi programou-se o serviço OTA. Na necessidade de se efetuar um *update firmware* sem que seja necessário conectar o MCU ao computador, este serviço permite que seja carregado um ficheiro binário para o ESP32 e que o novo *firmware* seja carregado após um reboot do sistema [56];

- *Queues*

Este conjunto de funções tem como objetivo declarar e inicializar todas as *queues* necessárias para que o *firmware* funcione corretamente sobre o FreeRTOS. É importante referir que este conjunto de funções são executadas apenas no início do programa.

- *Drain*

O serviço *drain* é responsável por controlar a bomba submergível presente num recipiente à saída da ZHC, através de sinais de *input* dos sensores de nível. Ao mesmo tempo, este serviço envia, para o serviço MQTT, através de uma fila do FreeRTOS o número de vezes que a bomba foi acionada, para que esta informação seja remetida e posteriormente analisada. É importante notar que, ao contrário dos restantes serviços, este serviço encontra-se presente apenas no MCU responsável por monitorizar e controlar o sistema à saída da ZHC.

A seguinte Figura 3.11 mostra um digrama correspondente às diferentes instruções que os MCU devem processar. Assim que o ESP32 é alimentado pela primeira vez, é executado o *boot* automaticamente. Durante o *boot* o ESP32 configura internamente alguns parâmetros importantes, como a definição da frequência de relógio, a configuração das partições da memória Flash segundo a tabela de partições, a inicialização do *scheduler* do FreeRTOS, entre outros parâmetros [20]. Posteriormente, o programa desenvolvido e escrito na memória de programa começa a ser executado, e em simultâneo são declaradas e iniciadas as filas e tarefas FreeRTOS. Cada tarefa é definida pela função principal de cada serviço, sendo que todas as tarefas são executadas paralelamente e infinitamente. É importante notar que as prioridades, núcleo de processamento e tempos de bloqueio de cada tarefa foram escolhidos e programados consoante a criticidade e a cadência de recolha de dados necessária, no caso das tarefas que comunicam com sensores. As tarefas com baixa prioridade apresentam um tempo de bloqueio menor e vice-versa. Na Figura 3.11, a verde, amarelo e violeta encontram-se os dados recolhidos dos sensores através do protocolo I2C, ModBus e contagem de descargas à saída da ZHC, respetivamente. Todos estes dados são colocados em filas FreeRTOS unitárias, do estilo *mailbox*. Estas filas são acedidas pelas tarefas MQTT e HTTPServer, de modo a processar os parâmetros recolhidos pelos sensores.

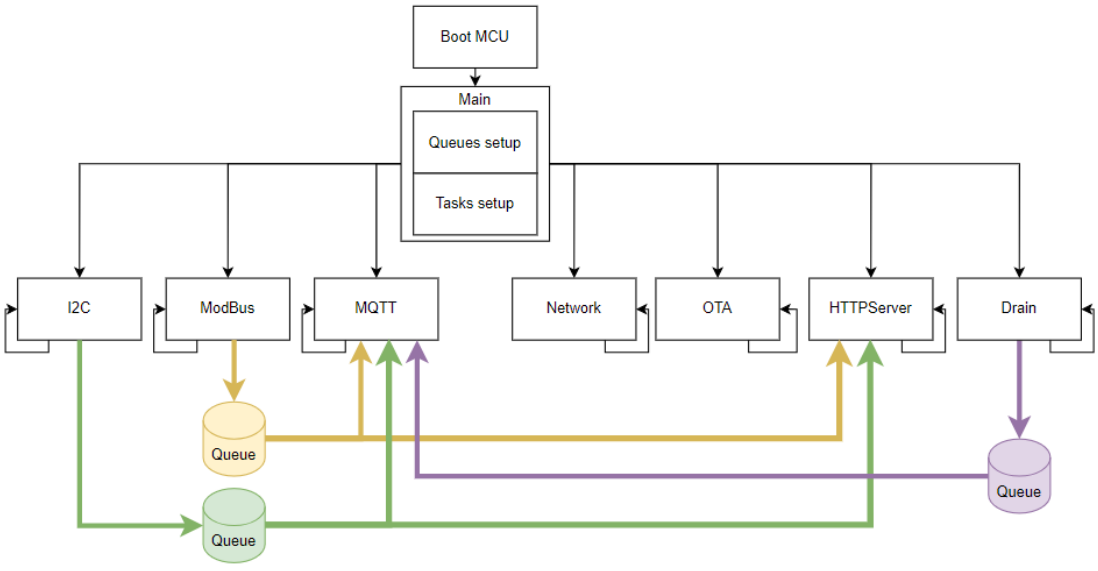


Figura 3.11 - Diagrama funcional do firmware do ESP32.

É possível obter uma melhor perceção de cada tarefa, analisando os diagramas de pseudocódigo no anexo 8.1.

As primeiras instruções executadas pelos MCU encontram-se no ficheiro main.cpp, na função *setup()*. Nesta função são inicializadas e criadas as filas e tarefas do FreeRTOS que constituem o projeto de programação. As filas são inicializadas no seu próprio serviço (Figura 3.12), posteriormente a cada tarefa é endereçada a função principal de cada serviço.

```
#include "queues.hpp"

namespace queues
{

    QueueHandle_t data; // Mailbox
    QueueHandle_t sta_cred;
    QueueHandle_t i2c_readings; // Mailbox
    QueueHandle_t modbus_readings;//Mailbox
    QueueHandle_t ncycles; // Mailbox

    void setup()
    {
        data = xQueueCreate(1, sizeof(uint8_t)); // Example queue

        sta_cred = xQueueCreate(2, sizeof(STA_cred_t));
        i2c_readings = xQueueCreate(1, sizeof(I2C_readings_t));
        modbus_readings = xQueueCreate(1, sizeof(Modbus_readings_t));
        ncycles = xQueueCreate(1,sizeof(uint16_t));
    }
}
```

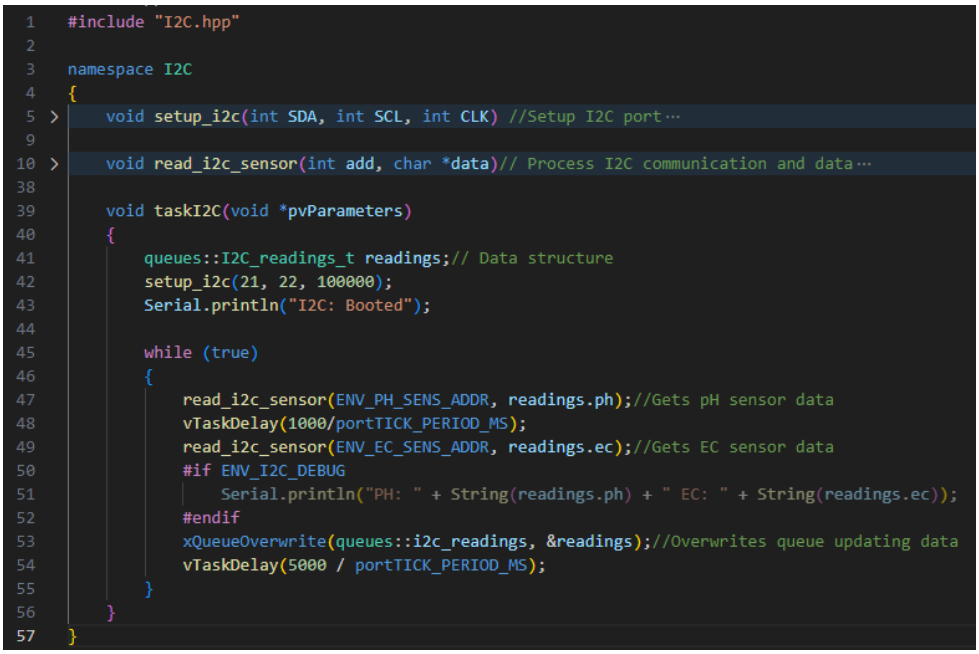
Figura 3.12 - Serviço Queues. Ficheiro queues.cpp.

A Figura 3.13 mostra o conjunto de instruções que constituem a função *setup()* no ficheiro main.cpp.

```
21  void setup()
22  {
23    Serial.begin(115200);
24
25    queues::setup();
26
27    #if ENV_TASK_NETWORK
28    xTaskCreate(Network::taskNetwork, "TaskNetwork", 5 * 1024, NULL, 1, &TaskNetwork);
29    #endif
30
31    #if ENV_TASK_MQTT
32    xTaskCreate(MQTT::taskMQTT, "TaskMQTT", 5 * 1024, NULL, 2, &TaskMQTT);
33    #endif
34
35    #if ENV_TASK_HTTPSERVER
36    xTaskCreate(HTTPServer::taskHTTPServer, "TaskHTTPServer", 5 * 1024, NULL, 1, &TaskHTTPSever);
37    #endif
38
39    #if ENV_TASK_OTA
40    xTaskCreate(OTA::taskOTA, "TaskOTA", 5 * 1024, NULL, 1, &TaskOTA);
41    #endif
42
43    #if ENV_TASK_I2C
44    xTaskCreate(I2C::taskI2C, "TaskI2C", 5 * 1024, NULL, 1, &TaskI2C);
45    #endif
46
47    #if ENV_TASK_MODBUS
48    xTaskCreate(ModBus::taskModbus, "TaskModbus", 5 * 1024, NULL, 1, &TaskModBus);
49    #endif
50
51    #if ENV_TASK_DRAIN
52    xTaskCreate(Drain::taskDrain, "TaskDrain", 3 * 1024, NULL, 1, &TaskModBus);
53    #endif
54  }
```

Figura 3.13 - Função setup(). Ficheiro main.cpp.

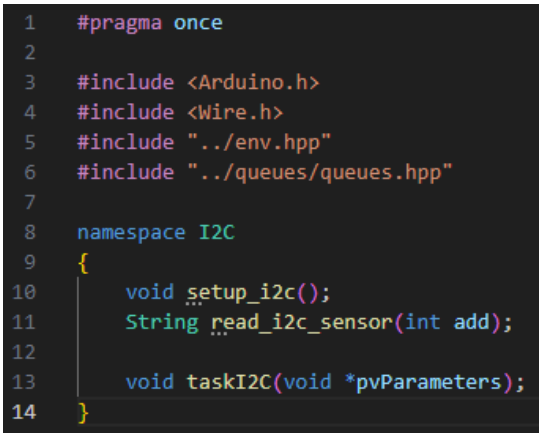
Todos os serviços seguem a mesma estrutura. A cada serviço é atribuído um *namespace* que permite seccionar e endereçar certas funções e tipos de variáveis com maior facilidade. O nome do *namespace* é igual ao nome do serviço. Englobadas pelo *namespace* encontram-se todas as funções que definem um serviço. No início da função que define a tarefa FreeRTOS, são inicializados todos os parâmetros necessários, como porto I2C, UART, Wi-Fi, pinos digitais, entre outros. Posteriormente, a tarefa entra no seu ciclo infinito processando dados importantes para o projeto com uma certa cadência. A Figura 3.14 mostra o código que define o serviço I2C. A função *taskI2C()* define a tarefa deste serviço, sendo que as restantes funções são responsáveis por ativar o porto I2C e recolher os dados dos sensores.



```cpp
#include "I2C.hpp"

namespace I2C
{
    void setup_i2c(int SDA, int SCL, int CLK) //Setup I2C port···

    void read_i2c_sensor(int add, char *data)// Process I2C communication and data···

    void taskI2C(void *pvParameters)
    {
        queues::I2C_readings_t readings;// Data structure
        setup_i2c(21, 22, 100000);
        Serial.println("I2C: Booted");

        while (true)
        {
            read_i2c_sensor(ENV_PH_SENS_ADDR, readings.ph);//Gets pH sensor data
            vTaskDelay(1000/portTICK_PERIOD_MS);
            read_i2c_sensor(ENV_EC_SENS_ADDR, readings.ec);//Gets EC sensor data
            #if ENV_I2C_DEBUG
                Serial.println("PH: " + String(readings.ph) + " EC: " + String(readings.ec));
            #endif
            xQueueOverwrite(queues::i2c_readings, &readings);//Overwrites queue updating data
            vTaskDelay(5000 / portTICK_PERIOD_MS);
        }
    }
}
```

Figura 3.14 - Serviço I2C. Ficheiro I2C.cpp.

Tal como já foi referido anteriormente, dentro da diretoria de cada serviço, para além do ficheiro *.cpp*, encontra-se o ficheiro *.hpp*. O ficheiro *.hpp* incluí todas as bibliotecas e outros ficheiros do tipo *header* (*.hpp*) que o serviço necessite. Adicionalmente e englobado pelo *namespace* do serviço são definidos os cabeçalhos de todas as funções presentes no ficheiro *.cpp*. O conteúdo programado no ficheiro *.hpp* do serviço I2C encontra-se exposto na Figura 3.15.

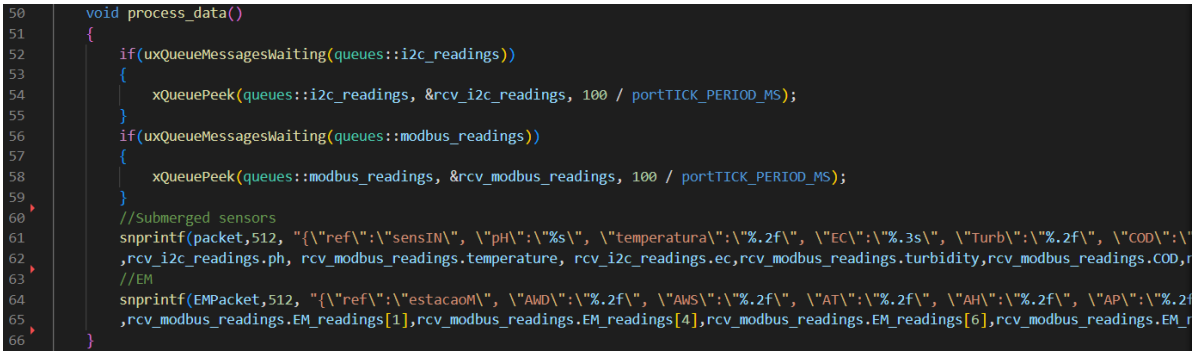


```cpp
#pragma once

#include <Arduino.h>
#include <Wire.h>
#include "../env.hpp"
#include "../queues/queues.hpp"

namespace I2C
{
    void setup_i2c();
    String read_i2c_sensor(int add);

    void taskI2C(void *pvParameters);
}
```

Figura 3.15 - Serviço I2C. Ficheiro I2C.hpp

Os dados das filas são recebidos de semelhante forma pelos serviços MQTT e HTTPServer. No caso do serviço MQTT os parâmetros dos sensores são recolhidos das filas correspondentes na função *process_data()*. Na mesma função, são construídas as

Strings em formato JSON que constituem as mensagens MQTT a enviar para o RasPi. A Figura 3.16 mostra a função anteriormente explicada.



```
50  void process_data()
51  {
52      if(uxQueueMessagesWaiting(queues::i2c_readings))
53      {
54          xQueuePeek(queues::i2c_readings, &rcv_i2c_readings, 100 / portTICK_PERIOD_MS);
55      }
56      if(uxQueueMessagesWaiting(queues::modbus_readings))
57      {
58          xQueuePeek(queues::modbus_readings, &rcv_modbus_readings, 100 / portTICK_PERIOD_MS);
59      }
60      //Submerged sensors
61      snprintf(packet,512, "{\"ref\":\"sensIN\", \"pH\":\"%s\", \"temperatura\":\"%.2f\", \"EC\":\"%.3s\", \"Turb\":\"%.2f\", \"COD\":\"
62      ,rcv_i2c_readings.ph, rcv_modbus_readings.temperature, rcv_i2c_readings.ec,rcv_modbus_readings.turbidity,rcv_modbus_readings.COD,r
63      //EM
64      snprintf(EMPacket,512, "{\"ref\":\"estacaoM\", \"AWD\":\"%.2f\", \"AWS\":\"%.2f\", \"AT\":\"%.2f\", \"AH\":\"%.2f\", \"AP\":\"%.2f
65      ,rcv_modbus_readings.EM_readings[1],rcv_modbus_readings.EM_readings[4],rcv_modbus_readings.EM_readings[6],rcv_modbus_readings.EM_r
66  }
```

Figura 3.16 – Função process_data(). Serviço MQTT. Ficheiro MQTT.cpp.

A tarefa FreeRTOS correspondente ao serviço MQTT é definida pela função *taskMQTT*() (Figura 3.17). Esta função começa por inicializar e conectar o ESP32 ao broker MQTT (RasPi) via o protocolo MQTT. Esta tarefa recolhe os dados dos sensores através da função *process_data()* e de seguida publica a mensagem construída sobre um tópico específico.

```
90  void taskMQTT(void *pvParameters) // Task de envio de parametros para o broker
91  {
92      vTaskDelay(1000 / portTICK_PERIOD_MS);
93      client.setServer(ENV_MQTT_BROKER, 1883);
94      client.setCallback(callback);
95      initialize_values();
96      Serial.println("MQTT: Booted");
97      while (true)
98      {
99          if (!client.connected())
100         {
101             reconnect();
102         }
103         client.loop();
104
105         process_data();
106         client.publish("sensors/input", packet);
107         client.publish("sensors/EM", EMPacket);
108         #if ENV_MQTT_DEBUG
109             Serial.printf("MQTT: Packet sent: %s",packet);
110             Serial.printf("MQTT: Packet sent: %s",EMPacket);
111         #endif
112         vTaskDelay(ENV_SEND_PERIOD_SEC * 1000 / portTICK_PERIOD_MS);
113     }
114 }
115 }
```

Figura 3.17 – Função taskMQTT(). Serviço MQTT. Ficheiro MQTT.cpp.

3.3.1.3 Página auxiliar Web

Desenvolveu-se uma página web que permite um acesso rápido e direto aos dados dos sensores ligados aos ESP32. Com o auxílio do serviço HTTPServer, o ESP32 funciona como um servidor HTTP. Isto permite que o MCU processe pedidos de cliente HTTP que se encontrem na rede AP criada pelo ESP32.

O serviço HTTPServer começa por definir os parâmetros do servidor HTTP e o seu arranque. De seguida são definidos os *end points* que processam os pedidos HTTP feitos pelos clientes. A Figura 3.18 mostra o conjunto de instruções que compõe a função que define o serviço HTTPServer.

```
116     void taskHTTPServer(void *pvParameters)
117     {
118         vTaskDelay(2 * 1000 / portTICK_PERIOD_MS);
119         httpd_config_t config = HTTPD_DEFAULT_CONFIG();
120         ESP_ERROR_CHECK(httpd_start(&http_server, &config));
121         registerEndpoints();
122 
123         Serial.println("HTTPServer: Booted");
124         while (true)
125         {
126             vTaskDelay(5000 / portTICK_PERIOD_MS);
127         }
128     }
129 }
```

Figura 3.18 – Função taskHTTPServer(). Serviço HTTPServer. Ficheiro HTTPServer.cpp.

A cada *end point* é indexado uma função que tem como propósito processar o pedido HTTP referente. Algumas declarações de *end points* do serviço HTTPServer podem ser analisadas através da Figura 3.19.

```
78      void registerEndpoints()
79      {
80          const httpd_uri_t index_get = {
81              .uri = "/",
82              .method = HTTP_GET,
83              .handler = index_get_handler,
84              .user_ctx = NULL};
85          ESP_ERROR_CHECK(httpd_register_uri_handler(http_server, &index_get));
86
87          const httpd_uri_t update_post_fw = {
88              .uri = "/update/firmware",
89              .method = HTTP_POST,
90              .handler = OTA::ota_update_post_handler,
91              .user_ctx = NULL};
92          ESP_ERROR_CHECK(httpd_register_uri_handler(http_server, &update_post_fw));
93
94          const httpd_uri_t update_post_cred = {
95              .uri = "/update/credentials",
96              .method = HTTP_POST,
97              .handler = update_post_handler,
98              .user_ctx = NULL};
99          ESP_ERROR_CHECK(httpd_register_uri_handler(http_server, &update_post_cred));
```

Figura 3.19 - Função registerEndpoints(). Serviço HTTPServer. Ficheiro HTTPServer.cpp.

O código HTML que descreve a página web encontra-se embutido no binário do próprio programa, sendo o binário alocado na memória Flash do MCU. De modo a carregar o código HTML, foi necessário utilizar algumas instruções da linguagem Assembly, responsáveis por aceder diretamente aos dados embutidos no binário [58]. Desta forma, assegurou-se que o conjunto de instruções HTML não sobrecarregava a memória RAM do MCU e a programação da página web tornou-se muito mais direta. Com o auxílio das instruções em Assembly, todo o código HTML é enviado como resposta ao respetivo pedido HTTP, gerando assim a página web. A Figura 3.20 mostra a função que é executada pelo MCU responsável por gerar a página web.

```
28      static esp_err_t index_get_handler(httpd_req_t *req)
29      {
30          extern const uint8_t index_html_start[] asm("_binary_src_web_index_html_start");
31          extern const uint8_t index_html_end[] asm("_binary_src_web_index_html_end");
32          httpd_resp_send(req, (const char *)index_html_start, index_html_end - index_html_start);
33          return ESP_OK;
34      }
```

Figura 3.20 – Função index_get_handler. Serviço HTTPServer. Ficheiro HTTPServer.cpp.

A página web permite alterar as credenciais da rede que o ESP32 se pretende conectar, carregar um ficheiro binário para efetuar uma atualização de *firmware*, analisar os dados mais recentes dos sensores e caso seja necessário efetuar um *reboot* do MCU. A página web alocada no MCU presente à entrada do sistema, permite ainda atualizar, manualmente o tempo de espera entre envio de dados para o RasPi. A Figura 3.21 mostra o aspeto da página *web*.

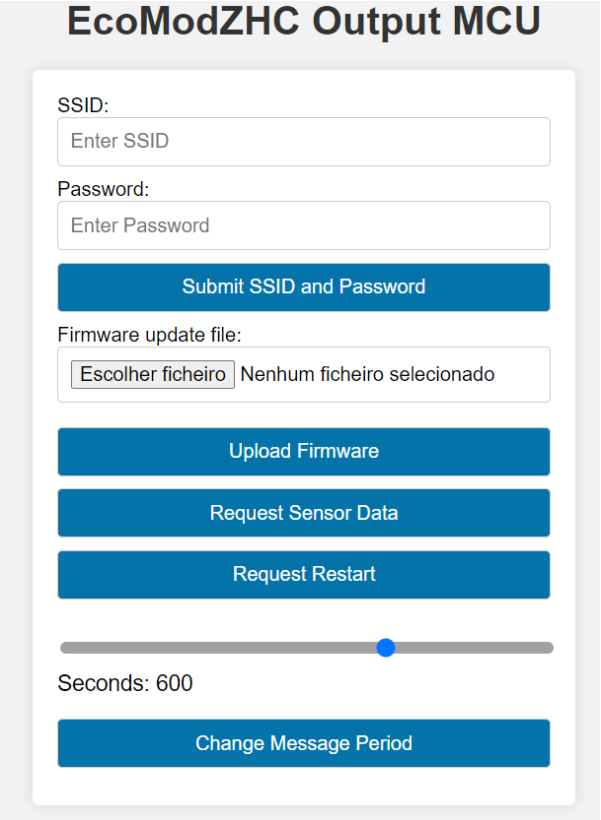


Figura 3.21 – Layout da página web.

3.3.2 RasPi

Optou-se por escrever as rotinas presentes no RasPi, responsáveis por receber, formatar e remeter as informações transmitidas pelos dois MCU, para a plataforma MRIPTA com o auxílio da linguagem de programação Python. A escolha passou pela familiaridade com a linguagem de programação, a sua versatilidade e a facilidade com que o Python apresenta em processar vastos volumes de dados em diferentes formatos.

Adicionalmente, utilizaram-se alguns comandos Bash para definir o *script* Python programado como um serviço. Desta forma o RasPi executa autonomamente o *script* Python no *boot* do sistema.

É importante notar que a interação com o RasPi foi sempre feita através do terminal *Secure Shell* (SSL), desta forma, não se utilizou qualquer ambiente de programação na escrita das rotinas presentes neste dispositivo.

3.3.2.1 MQTT Broker

Tal como já foi referido anteriormente (secção 3.2.1.1), o RasPi atua como um intermediário de dados MQTT (*broker*). Com o auxílio da plataforma Mosquitto, programou-se um script Python responsável por receber os tópicos MQTT oriundos dos MCU, formatar os dados recebidos e posteriormente publicá-los sobre um novo tópico na plataforma MRIPTA.

A Figura 3.22 mostra as linhas de código escritas que compõem a rotina Python responsável por tornar o RasPi num *broker* MQTT. Inicialmente são importadas as bibliotecas de funções necessárias para receber/transmitir tópicos MQTT e tratar dados no formato JSON. A função definida como *on_message()* é executada sempre que uma mensagem sobre um tópico subscrito pelo *broker* é recebida. Esta função extrai a mensagem efetiva do pacote MQTT, imprime a mensagem para efeitos de *debug*, analisa e processa a mensagem JSON. Por fim a função *on_message()* conecta o RasPi à plataforma MRIPTA e publica a informação recebida sobre um novo tópico. Fora da função *on_message()*, são subscritos dois tópicos MQTT, sendo estes enviados pelos dois MCU ESP32 que monitorizam a ZHC. É ainda importante notar que este *script* é executado infinitamente, sendo a cadência das mensagens enviadas para a plataforma MRIPTA definida pelos MCU.

```python
import paho.mqtt.client as mqtt
import json

def on_message(client, userdata, message):
    received_message = message.payload.decode()
    print(f"Received message '{received_message}' on topic '{message.topic}'")

    # Forward the message
    parsed_topic = json.loads(received_message)
    forward_topic = parsed_topic.get("ref")
    forward_message = received_message

    forward_client = mqtt.Client()
    forward_client.username_pw_set("ecomodzhc","ecomodzhc")
    forward_client.connect("mripta.ci2.ipt.pt", 45647, 60)  # Connect to MRIPTA
    forward_client.publish(forward_topic, forward_message)  # Publish the message
    print(f"Sent with topic: '{forward_topic}'")
    forward_client.disconnect()  # Disconnect from MRIPTA

client = mqtt.Client()
client.on_message = on_message
client.connect("localhost", 1883, 60)  # Connect to the local broker

source_topicI = "sensors/input"
source_topicII = "sensors/output"


client.subscribe(source_topicI)  # Subscribe to the source topic
client.subscribe(source_topicII)  # Subscribe to the source topic

client.loop_forever()  # Keep the script running to receive and forward messages
```

Figura 3.22 - Script Python executado periodicamente pelo RasPi.

3.4 Desenvolvimento das placas de circuito impresso

De modo a agregar a ligação de todos os sensores e os seus barramentos aos MCU, dimensionaram-se placas de circuito impresso, em inglês PCB com a ajuda do programa Eagle da Autodesk. As PCB têm como função minimizar o espaço (comparando com os circuitos em placas de teste), diminuir o ruído eletromagnético e aumentar consideravelmente a robustez e segurança dos circuitos.

3.4.1 Desenvolvimento do circuito

Dimensionou-se primeiramente o circuito eletrónico com todos os componentes necessários para o bom funcionamento dos sistemas eletrónicos que compõe este projeto.

Como os valores das tensões de alimentação dos sensores e dos restantes dispositivos não são todos iguais adicionaram-se, ao circuito, conversores de tensão comutados DC/DC do tipo *Buck* do fabricante Traco Power [59]. Desta forma, possibilita-se a alimentação de todos os sistemas presentes na PCB com apenas uma tensão. Ficando os conversores de tensão comutados responsáveis por regularizar as tensões que são aplicadas aos respetivos dispositivos.

Colocaram-se condensadores de desacoplamento nas alimentações do MCU e dos conversores comutados para mitigar a interferência do ruído gerado [60] e ainda um LED (*Ligh Emitting Diode*) que informa o utilizador se a PCB se encontra alimentada. Adicionou-se, por segurança um fusível de proteção, para o caso de ocorrer algum curto-circuito inesperado.

Para comandar a bomba submersa da saída, escolheu-se comutar com um relé e não com um MOSFET (*Metal Oxide Semiconductor Field Effect Transistor*), uma vez que o relé é uma solução mais universal, isto é, comuta com tensões continuas ou alternadas, e é galvanicamente isolado. De modo que a atuação do relé seja feita com o MCU e uma vez que a corrente máxima fornecida pelos pinos do mesmo não é suficiente para atuar a bobine do relé, foi necessário dimensionar um circuito auxiliar com um transístor bipolar. Adicionalmente, colocou-se um LED em paralelo com a bobine do relé para informar quando é que este se encontra ativado. A Figura 3.23 mostra o circuito projetado para o propósito enunciado neste parágrafo. É importante constatar que a numeração dos pinos do MCU, na Figura 3.23 é meramente ilustrativa.

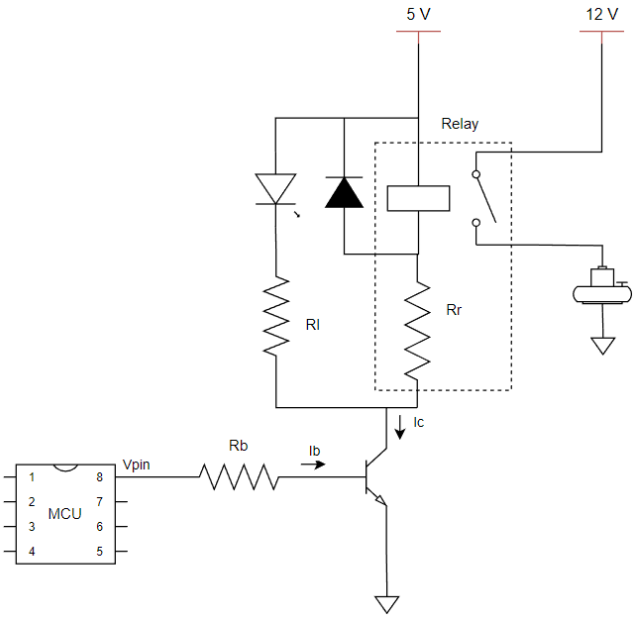


Figura 3.23- Circuito auxiliar para controlar a bomba submergível.

Com a ajuda de um transístor bipolar a funcionar na zona de saturação, ou seja, com um comportamento semelhante a um interruptor, é possível comandar a corrente necessária para atuar a bobine do relé com uma corrente β (ganho do transístor) vezes inferior. É ainda importante notar o díodo em antiparalelo com o relé, também conhecido como díodo *flywheel*, colocado de modo a dissipar a energia guardada no campo magnético da bobine.

Procedeu-se ao cálculo do valor da resistência colocada entre a base do transístor e o pino digital do MCU (Resistência Rb na Figura 3.23), com o auxílio das equações seguintes (Equação 1 e Equação 2).

$$Ib = \frac{Ic}{\beta} = \frac{130\ mA}{420} = 0{,}3095\ mA \tag{1}$$

$$Rb = \frac{Vpin - Vbe}{Ib} = \frac{3{,}3\ V - 0{,}77\ V}{0{,}3095\ mA} = 8{,}4\ k\Omega \tag{2}$$

A Figura 3.24 mostra o circuito dimensionado, de forma simplificada. No canto superior esquerdo encontra-se representado a regularização das tensões de alimentação de todos

os sensores, sistemas e subsistemas, sendo que a tensão de entrada é de doze volt. Para além dos componentes eletrónicos e subsistemas, no circuito encontram-se expostas as ligações onde os sensores e atuadores são conectados. É importante notar que a numeração dos pinos do MCU presentes na figura são apenas ilustrativos e não representam a numeração real.

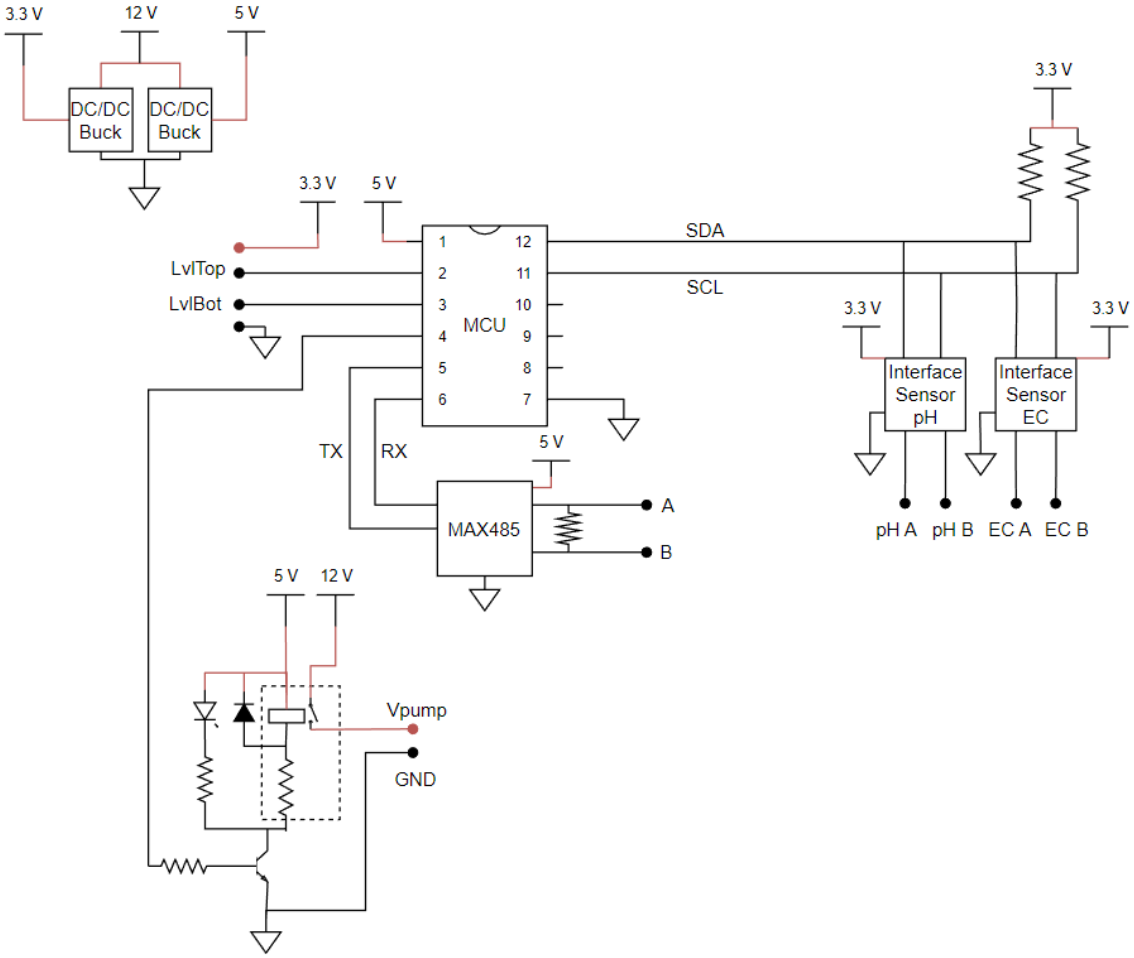


Figura 3.24 – Circuito simplificado do sistema de monitorização projetado.

O circuito projetado no programa Eagle pode ser analisado com detalhe consultando o anexo 8.2.

3.4.2 PCB

Na disposição dos componentes na própria PCB não foi necessário nenhum cuidado especial, uma vez que apenas fluem pelas pistas sinais digitais e correntes de intensidade reduzida. Houve, no entanto, necessidade de colocar os condensadores de

desacoplamento o mais próximo possível dos componentes críticos para evitar sinais e tensões de alimentação ruidosos.

O *layout* dos componentes na PCB dimensionada com a ajuda do programa Eagle pode ser analisado com detalhe consultando o anexo 8.2.

A Figura 3.25 exibe a PCB após ser impressa, já com todos os componentes soldados e algumas ligações externas já efetuadas.

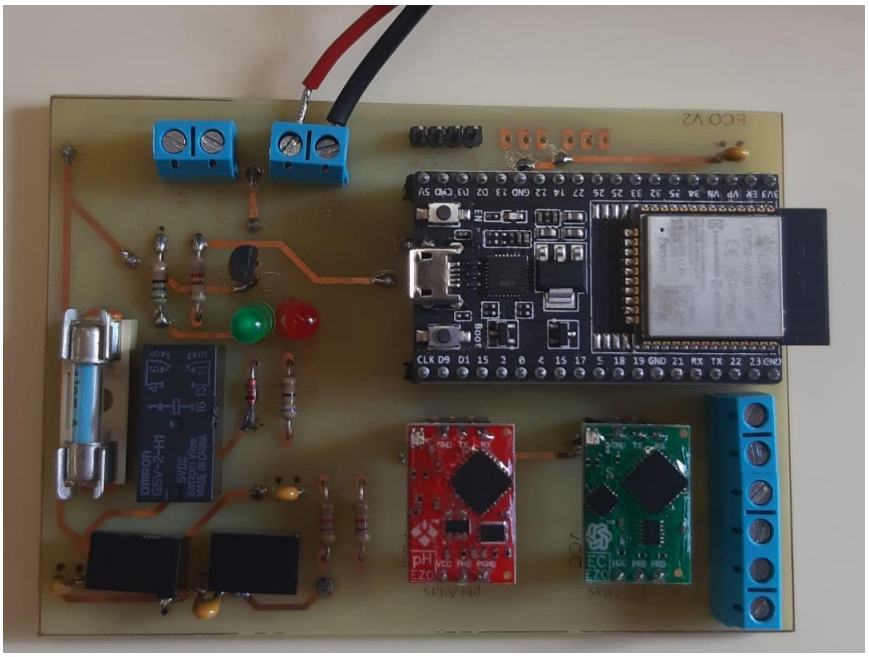


Figura 3.25 - PCB impressa e com todos os componentes.

3.5 Conclusões do capítulo

Neste capítulo foram descritos os procedimentos tomados na implementação do sistema de monitorização tendo em conta as necessidades do projeto EcoModZHC.

A implementação do sistema UCAD responsável pelo controlo e monitorização dos processos inerentes á ZHC, foi realizado em várias fases. Foram apresentadas as opções de projeto da unidade monitorização e controlo dos processos, descreveu-se o projeto do módulo MT, o dimensionamento das PCB e o desenho e o projeto de firmware do sistema.

Em suma, foram cuidadosamente selecionados dispositivos, protocolos e procedimentos, com o intuito de assegurar que o projeto seja o máximo modular e escalável. Cada escolha

foi ponderada para garantir não apenas a eficiência imediata, mas também a flexibilidade necessária para enfrentar desafios futuros.

4 Testes e ensaios

A importância de ensaiar, de testar e verificar o desempenho dos controladores e transferência de dados, prende-se com o objetivo de despistar possíveis falhas e erros no desenvolvimento dos sistemas desenvolvidos, sem que se causem possíveis danos maiores. Neste capítulo descrevem-se os testes e ensaios finais dos vários módulos que constituem este projeto (UCAD e MT). O objetivo é avaliar o desempenho do sistema cujo desenvolvimento e realização foram descritos nas secções anteriores.

4.1 Testes ao sistema UCAD

O sistema UCAD é composto essencialmente por controladores presentes à entrada e à saída da ZHC. Estes controladores recolhem dados com a ajuda de sensores e remetem os mesmos para o MCP. Paralelamente o sistema UCAD é responsável por controlar a bomba peristáltica à entrada e a bomba submergível à saída da ZHC.

Os testes e ensaios efetuados ao sistema UCAD passaram por verificar se os valores recolhidos pelos sensores estavam dentro dos parâmetros expectáveis. Estes testes foram efetuados usando soluções aquosas com composição conhecida e com equipamento usual de laboratório de química com o intuito de averiguar a exatidão e precisão de todos os sensores. A Figura 4.1 mostra o *setup* temporário montado para efetuar testes aos sensores.

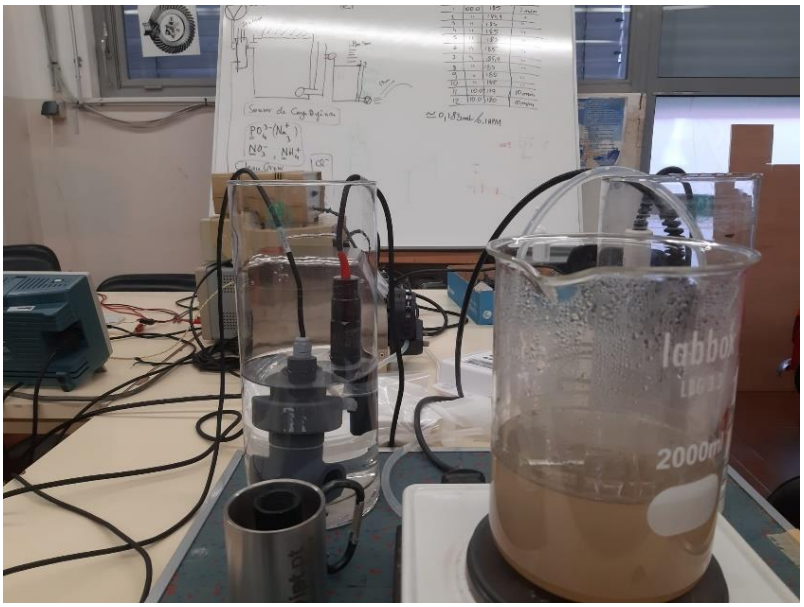


Figura 4.1 Testes ao sistema UCAD.

Para além dos ensaios aos sensores, efetuaram-se testes à bomba peristáltica e bomba submersa para confirmar o seu bom e correto funcionamento.

4.2 Testes ao MT

O MT é responsável por garantir a transmissão de dados entre o processo e o utilizador final. Faz parte do MT o MCP, sendo este o dispositivo *edge* responsável por encaminhar os dados para o MRIPTA.

Os testes efetuados ao MT passaram por assegurar que todos os dados eram corretamente transmitidos entre o processo físico e o sistema de monitorização. A Figura 4.2 mostra um dos testes efetuado no MCP. O teste passa por aferir se os pacotes MQTT e mensagens em formato JSON se encontram corretos.

```
Received message '{"ref":"sensIN", "pH":"7.1", "temperatura":"14.2", "EC":"285", "Turb":"1.25", "COD":"5.2",
Sent with topic: 'sensIN'
Received message '{"ref":"EM", "AWD":"0", "AWS":"23", "AT":"17", "AH":"55", "AP":"50", "RF":"0", "UV":"27",
```

Figura 4.2 - Dados recebidos e transmitidos pelo MCP.

A Figura 4.3 mostra um exemplo da visualização dos dados no MRIPTA. Sempre que a visualização em tempo real dos dados é possível, pode-se afirmar que o MT se encontra a funcionar corretamente.

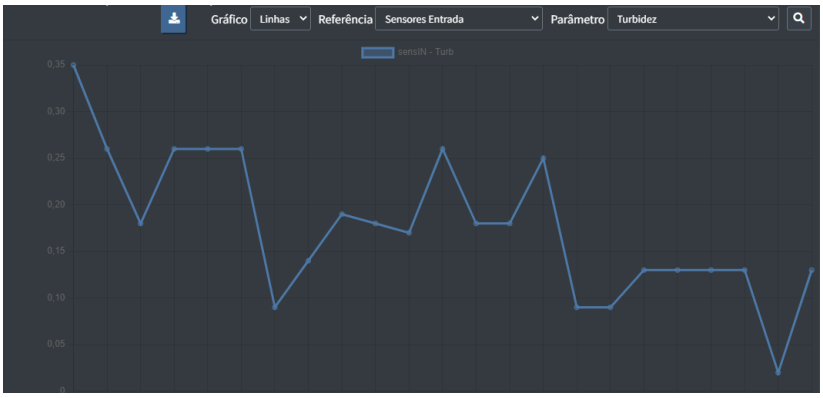


Figura 4.3 - Testes à monitorização do sistema de teste. MRIPTA.

4.3 Conclusões do capítulo

Neste capítulo foram descritos os testes e ensaios realizados na implementação de um sistema de monitorização tendo em conta as necessidades do projeto EcoModZHC.

Os testes realizados ao sistema UCAD, responsável pelo controlo e monitorização dos processos inerentes á ZHC, foram realizados de modo a aferir a precisão e exatidão dos valores recolhidos. Os ensaios efetuados ao MT passaram por garantir que o fluxo de dados envidados e recolhidos, desde os sensores, MCU, MCP e MRIPTA estava a funcionar corretamente.

Concluindo, os ensaios realizados a ambos os sistemas e módulos permitiram garantir que todo o sistema de monitorização se encontrava a funcionar corretamente e dentro das condições pretendidas.

5 Implementação do sistema de monitorização

Neste capítulo, encontram-se detalhados os procedimentos práticos e técnicas envolvidas na integração do sistema na ZHC.

Nesta etapa final do projeto procedeu-se à implementação dos módulos e sistemas desenvolvidos no terreno.

5.1 Instalação caixas de ligação dos sensores

Para efetuar todas as ligações e manter as PCB e módulos acoplados robustos utilizaram-se caixas de ligação. Foram preparadas duas caixas de ligação uma para monitorizar o processo à entrada e outra para monitorizar o processo à saída. Nestas caixas efetuaram-se todas as ligações necessárias dos sensores, atuadores e alimentações. As caixas de ligações podem ser aferidas com o auxílio da Figura 5.1.

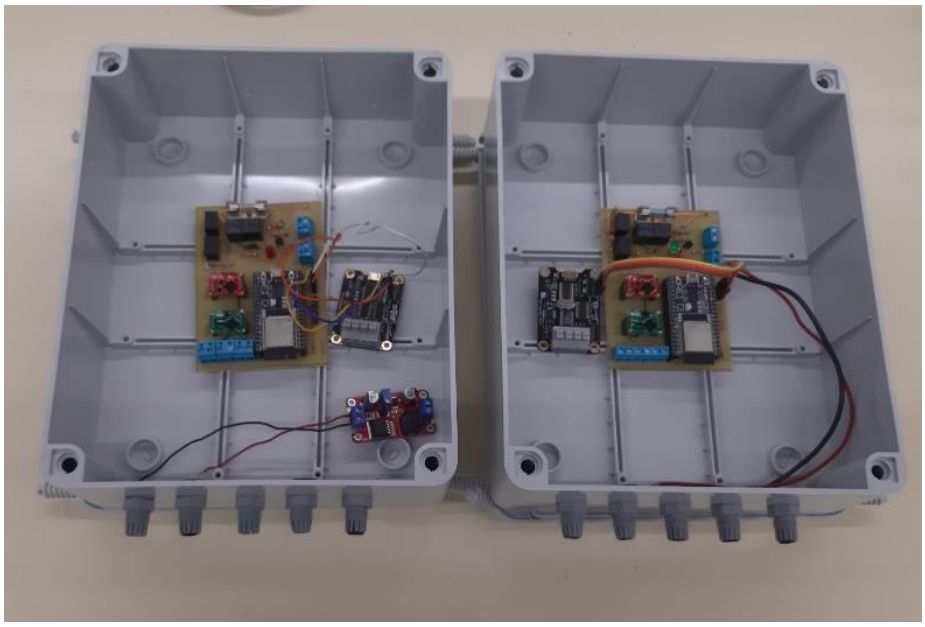


Figura 5.1 - Caixas de ligação.

5.2 Instalação dos componentes na instalação ZHC piloto

A ZHC com os respetivos recipientes de entrada e saída pode ser observada na Figura 5.2. O recipiente, em cima à esquerda recebe a água residual e aloja os sensores de

entrada. A água após ser filtrada pela ZHC flui para o recipiente de saída (em baixo à direita).

Figura 5.2 – ZHC e recipientes.

Um dos recipientes que aloja os sensores pode ser aferido com a ajuda da Figura 5.3. É com o auxílio destes recipientes que a água residual é analisada à entrada e á saída da ZHC.

Figura 5.3 – Recipiente com os sensores.

A caixa com sistema de monitorização da água no recipiente da entrada encontra-se fixada por baixo do mesmo. Este pode ser aferido consultado a Figura 5.4.

Figura 5.4 – Caixa de ligação para o sistema da entrada.

A caixa com o sistema de monitorização da saída encontra-se dentro de um abrigo. Este abrigo alberga, ao mesmo tempo, a bomba peristáltica e um sistema fotovoltaico (baterias e controladores de carga), sendo este responsável por fornecer energia para todos os sistemas projetados. A Figura 5.5 mostra o abrigo observando-se com uma das caixas de ligação, bomba peristáltica e estação meteorológica.

Figura 5.5 – Abrigo.

5.3 Transferência de dados para a plataforma MRIPTA

Finalmente, após a implementação de todos o sistema, começaram a ser canalizados dados relevantes para a plataforma MRIPTA. As figuras seguintes (Figura 5.6 e Figura 5.7) mostram as diferentes formas de monitorização dos vários parâmetros analisados, com o auxílio de gráficos de linhas e tabelas simples.

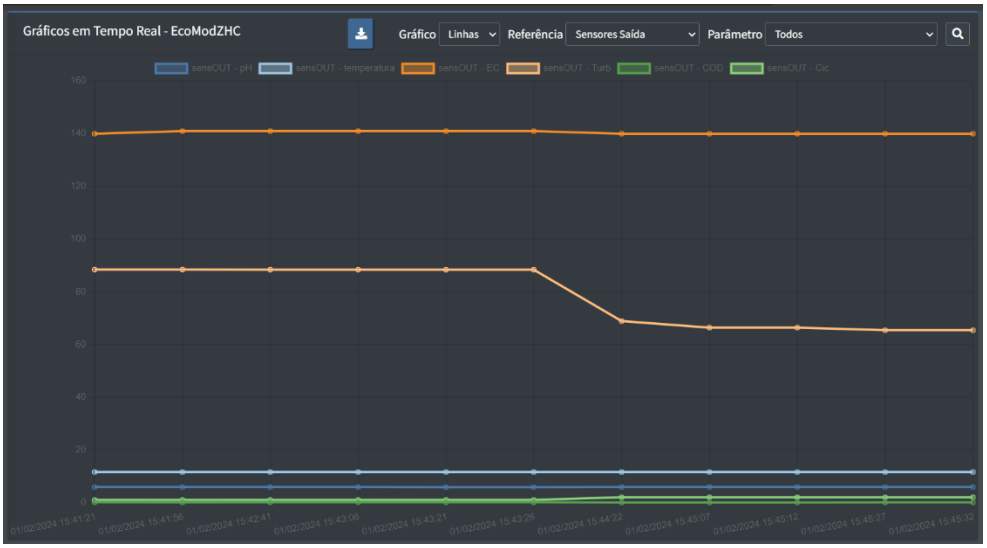


Figura 5.6 - Visualização de dados no MRIPTA. Gráfico de linhas.

sensIN	{"ref":"sensIN", "pH":"7.8", "temperatura":"5.96", "EC":"554", "Turb":"2.25", "COD":"28.25", "RPM":"0" }	1707214040
EM	{"ref":"EM", "AWD":"329.00", "AWS":"0.00", "AT":"12.00", "AH":"79.00", "AP":"1022.00", "RF":"0.00", "UV":"0.00", "Rad":"469.00" }	1707213975
sensOUT	{"ref":"sensOUT", "pH":"8.799", "temperatura":"8.30", "EC":"133", "Turb":"5.59", "COD":"0.00", "Cic":"0" }	1707213854

Figura 5.7 - Visualização dos dados no MRIPTA. Tabela.

A Figura 5.8 mostra um exemplo da evolução temporal da carga orgânica à saída do sistema.

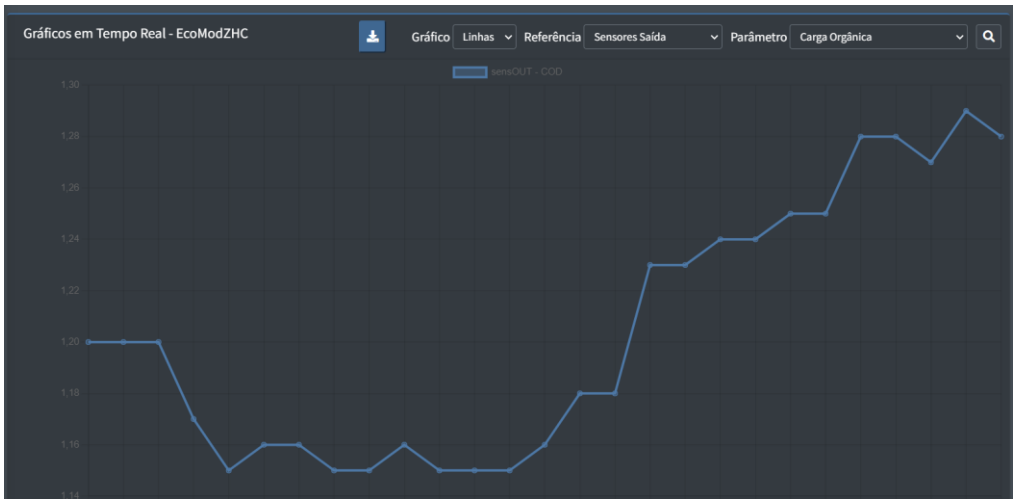


Figura 5.8 – Visualização de dados no MRIPTA. Parâmetro isolado.

A Figura 5.9 mostra um exemplo da funcionalidade do MRIPTA de cruzar informação sobre diversos parâmetros. O exemplo seguinte mostra a evolução durante um dia dos parâmetros de condutividade elétrica à entrada (magenta) e à saída (laranja).

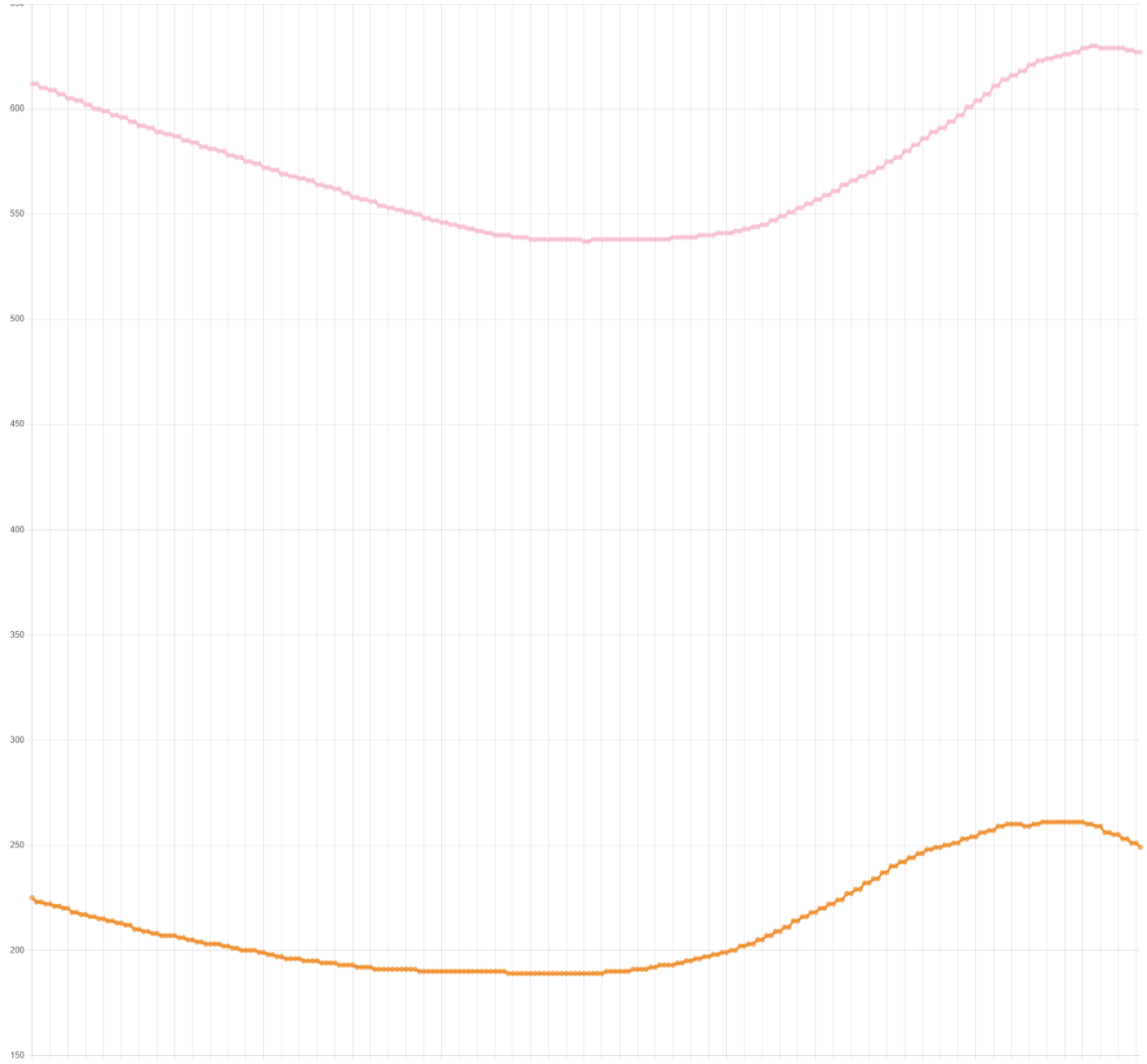


Figura 5.9 - Visualização de dados no MRIPTA. Parâmetros cruzados (EC).

A Figura 5.10 mostra a evolução durante um dia dos parâmetros de carga orgânica à entrada (verde) e à saída (rosa).

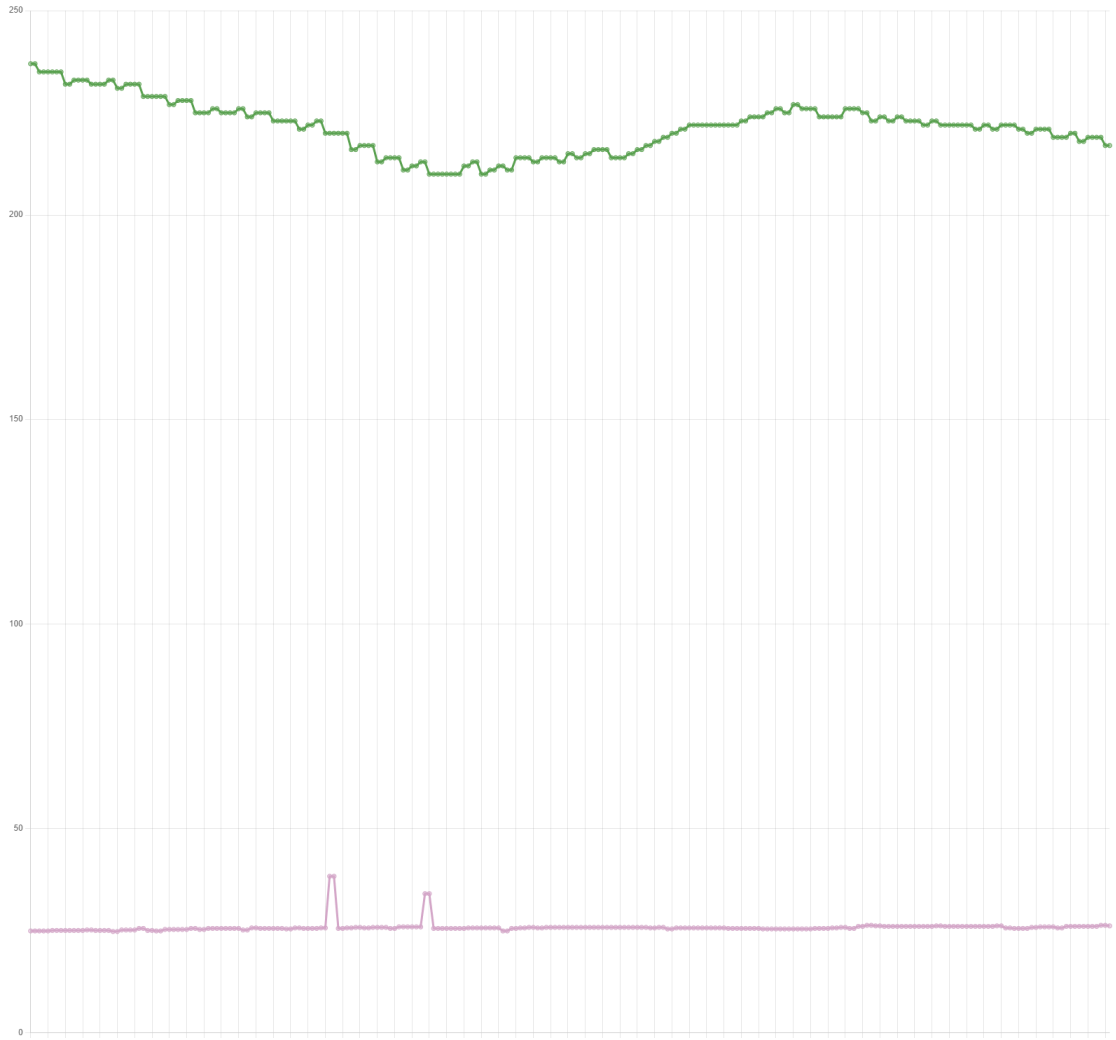


Figura 5.10 - Visualização de dados no MRIPTA. Parâmetros cruzados (Carga Orgânica).

A Figura 5.11 mostra a evolução durante um dia dos parâmetros de temperatura da água à entrada (azul), à saída (magenta) e no ar (amarelo).

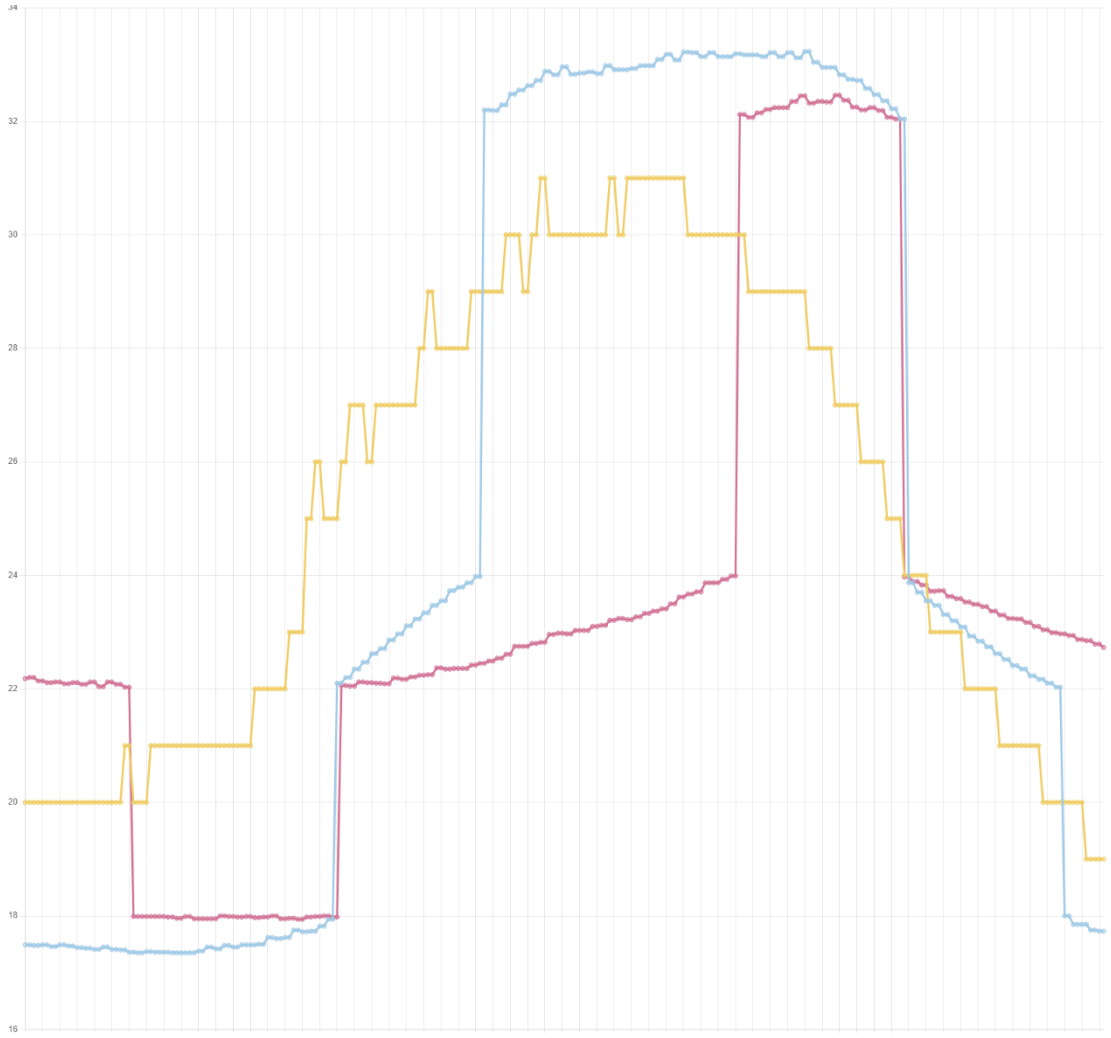


Figura 5.11 - Visualização de dados no MRIPTA. Parâmetros cruzados (Temperatura).

5.4 Conclusões do capítulo

Neste capítulo foram expostos os procedimentos efetuados a fim de implementar todos os sistemas de monitorização e atuação junto da própria ZHC.

A implementação contou com alguns desafios. Nomeadamente, com a disposição dos recipientes, organização das caixas e construção de estruturas para resguardar e fixar os sistemas.

Com a implementação dos sistemas é agora possível começar a aferir dados reais de todo o processo de filtragem da ZHC e cruzar os mesmos com dados meteorológicos vinte e quatro horas por dia.

Concluindo, a última etapa deste projeto permite a inicialização de novos estudos sobre o comportamento das ZHC. Os dados remetidos, guardados e expostos no MRIPTA possibilitaram certamente novos estudos sobre o tema da filtragem de águas residuais com o auxílio de ZHC.

6 Conclusões

O trabalho desenvolvido no âmbito deste projeto consistiu na realização e implementação de um sistema de monitorização automatizado impulsionado principalmente por avanços nos sistemas embebidos, tecnologias de comunicação e de microssistemas, com a finalidade de introduzir melhorias de desempenho e de produtividade do processo de monitorização das águas residuais processadas pela ZHC.

Durante o seu desenvolvimento optou-se por projetar sistemas modulares e flexíveis, de modo a facilitar a futura evolução deste projeto. Deu-se primazia aos protocolos de comunicação *wireless* porque oferecem maior flexibilidade e facilidade de instalação. Deu-se também especial atenção à escolha das técnicas de programação utilizadas para programar os MCUs e MCP para permitir que a sua leitura e consequente evolução fosse intuitiva e o mais simplificada possível.

A implementação do sistema de monitorização passou por uma longa fase de desenvolvimento e testes funcionais descritas ao longo do relatório. Contudo, uma das fases mais críticas revelou-se ser o teste de funcionalidade e calibração das sondas usadas no sistema. Para este efeito, projetou-se uma bancada de testes com todos os sensores disponíveis, permitindo assim a verificação e a veracidade dos dados apresentados.

Depois de calibrar o sistema de medição comparando os dados medidos com valores de referência, todo o sistema foi montado na sua localização final, onde se encontra até ao momento a recolher e transmitir dados sobre a ZHC. Alguns ajustes tiveram de ser efetuados relativos à fixação das caixas que alojam os sistemas, uma vez que a disposição dos tanques e outros componentes foram alteradas. Testes de longa duração permitiram detetar ainda alguns problemas com os dados recolhidos por alguns sensores. Concluiu-se que a possível solução para este problema deverá passar por criar rotinas de calibração e limpeza dos sensores.

Infelizmente, não foi possível implementar no espaço final o sistema de drenagem e contagem de caudal no recipiente de saída. Isto aconteceu uma vez que não se encontrou uma forma robusta o suficiente para fixar os sensores de nível. Não obstante, este sistema provou estar funcional nas atividades da bolsa de investigação [5] e artigo científico [6].

Adicionalmente foi criada uma página web de acesso local, fazendo um *bypass* a todo o processo de envio de dados para o MRIPTA, para facilitar de forma rápida o *debug* de possíveis problemas. Com o passar do tempo, as funcionalidades desta página foram crescendo para se adaptarem às necessidades dos especialistas de análises dos dados.

Com o tempo surgiu também a necessidade de adicionar uma nova funcionalidade ao sistema implementado. Veio-se a verificar que seria muito vantajoso permitir a atualização remota das unidades com novos atributos ou novo *firmware*. Desta forma novos binários podem ser embutidos nos MCUs sem a necessidade de aceder ao hardware. Tomou-se também iniciativa de criar um repositório de código online [53] para que todas as alterações do código fonte sejam documentadas e guardadas. Facilitando o trabalho a futuros colegas e/ou investigadores que decidam continuar e melhorar o sistema de monitorização.

Para trabalho futuro recomenda-se que seja desenvolvida uma rotina de calibração para todos os sensores, uma vez, que se verificou que alguns dos sensores utilizados neste projeto, com o passar do tempo, requerem uma recalibração. Ao mesmo tempo, recomenda-se que a plataforma MRIPTA deve ser melhorada de forma que seja possível enviar comandos para o MCP e MCUs, como a alteração do RPM da bomba peristáltica, possibilitando assim uma comunicação bilateral.

Adicionalmente, sugere-se a inclusão de LEDs de estado acoplados às caixas que contém os sistemas de monitorização, para transmitirem informações relevantes ao utilizador, como o estado da alimentação do sistema, informação da qualidade da rede Wi-Fi e dos próprios sensores. Em paralelo, recomenda-se ainda a recolha dos dados disponibilizados pelo sistema de painéis fotovoltaicos através do protocolo Bluetooth e submissão dessa informação para a plataforma MRIPTA. E por último, recomenda-se uma revisão geral de forma a aumentar a robustez total do sistema.

Em suma, o sistema de monitorização autónomo agora implementado recolhe, atua e remete dados relevantes sobre uma ZHC para a plataforma de monitorização MRIPTA, possibilitando a análise e comparação de diversos parâmetros em tempo real. Apesar das enumeras recomendações de possíveis melhoramentos, todos os objetivos propostos foram alcançados, com exceção do sistema de monitorização de caudal da saída. Este trabalho possibilitou, fundamentalmente, a criação de uma base de conceitos diversos que

permitiram melhorar os futuros projetos e/ou artigos elaborados a partir do conceito das ZHC.

7 Referências

1. Pinho, H; Mateus, D; Ferreira, Carlos; Barros, F.M. "*Smart Monitoring of Constructed Wetlands to Improve Efficiency and Water Quality*". In: Xu, H. (eds) Proceedings of the 5th International Symposium on Water Resource and Environmental Management. WREM 2022. Environmental Science and Engineering, 189-197 https://doi.org/10.1007/978-3-031-31289-2_15. (12/2022)
2. Instituto Politécnico de Tomar. Economia Circular de Água e Materiais através de Zonas Húmidas Construídas Modulares. Disponível em http://www.ecomodzhc.ipt.pt. (06/2023)
3. Kadlec, R.H., & Wallace, S. (2008). Treatment Wetlands (2nd ed.). CRC Press. https://doi.org/10.1201/9781420012514. (05/2011)
4. Pinho, H. J. O., & Mateus, D. M. R. "*Valorization of solid waste in subsurface flow constructed wetlands based on renewable modular structures: A contribution to a circular economy. In Circular Economy and Sustainability*" (pp. 215–233). Elsevier. https://doi.org/10.1016/b978-0-12-821664-4.00019-4. (09/2022)
5. Lopes, S. Relatório da Bolsa de Iniciação à Investigação Científica (BII) EcoModZHC - Eletrotécnica. "Rede de Sensores e Sistema Distribuído de Controlo ". Centro de Investigação em Cidades Inteligentes. (06/2023)
6. Lopes, S., Barros, M., Ferreira, C., Mateus, D., Matos, P., Neves, P., & Pinho, H. (2023). "*REMOTE MONITORING OF ENERGY-AUTONOMOUS CONSTRUCTED WETLANDS*". In Ecology & Safety (Vol. 17, Issue 1, pp. 1–15). Science Events Ltd. https://doi.org/10.62991/es1996214429. (03/2024)
7. Sousa, J. Dissertação de Mestrado. "Monitorização do Desempenho e Zonas Húmidas Construídas Validação de IdC". Instituto Politécnico de Tomar. (03/2023)
8. Pinho, H. J. O., & Mateus, D. M. R. (2022). "*Contribution of Constructed Wetlands for Reclaimed Water Production: A Review*". In IOP Conference Series: Earth and Environmental Science (Vol. 1006, Issue 1, p. 012008). IOP Publishing. https://doi.org/10.1088/1755-1315/1006/1/012008. (09/2021)
9. Koskiaho, J., & Puustinen, M. (2019). "*Suspended solids and nutrient retention in two constructed wetlands as determined from continuous data recorded with*

sensors". In Ecological Engineering (Vol. 137, pp. 65–75). Elsevier BV. https://doi.org/10.1016/j.ecoleng.2019.04.006. (03/2019)

10. Chasmer, L., Cobbaert, D., Mahoney, C., Millard, K., Peters, D., Devito, K., Brisco, B., Hopkinson, C., Merchant, M., Montgomery, J., Nelson, K., & Niemann, O. "*Remote Sensing of Boreal Wetlands 1: Data Use for Policy and Management. Remote Sensing"*, 12(8), 1320". https://doi.org/10.3390/rs12081320. (06/202)
11. Xiaoying, S., & Huanyan, Q. "*Design of Wetland Monitoring System Based on the Internet of Things. Procedia Environmental Sciences*", *10*, 1046–1051. https://doi.org/10.1016/j.proenv.2011.09.167. (05/2018)
12. Razzaghmanesh, M., & Borst, M. (2018). "*Investigation clogging dynamic of permeable pavement systems using embedded sensors. In Journal of Hydrology*" (Vol. 557, pp. 887–896). Elsevier BV. https://doi.org/10.1016/j.jhydrol.2018.01.012. (03/2018)
13. Dian, J; Vahidnia, R. "*IoT Use Cases and Technologies*" British Columbia Institute of Technology, Vancouver, Canada. (01/2020)
14. Eloranta, Veli-Pekka; Koskinen, Johannes; Leppänen, Marko; Reijonen, Ville. "*Designing distributed control systems: a pattern language approach. Wiley series in software design patterns*". (06/2014)
15. Innovic. "*Distributed Control System (DCS)*". Disponível em: http://www.innovicindia.com. (08/2023)
16. Gridling, G; Weiss, B. "*Introduction to Microcontrollers, courses 182.064 & 182.074*". Vienna University of Technology (02/2007).
17. Yousif, S. "*Introduction of Microcontrollers and Microcontroller Design*". Al Mustaqbal University. (12/2021)
18. R. Chéour, S. Khriji, M. abid and O. Kanoun, "*Microcontrollers for IoT: Optimizations, Computing Paradigms, and Future Directions*" 2020 IEEE 6th World Forum on Internet of Things (WF-IoT), New Orleans, LA, USA, 2020, pp. 1-7, doi: 10.1109/WF-IoT48130.2020.9221219. (09/2023)
19. Arduino. "*Arduino UNO R3, Product Reference Manual*". (09/2023)
20. Espressif Systems. "*ESP32 Series, Datasheet*". (07/2023)
21. STMicroeletronics. "*STM32WBA52 Series, Datasheet*". (06/2023)

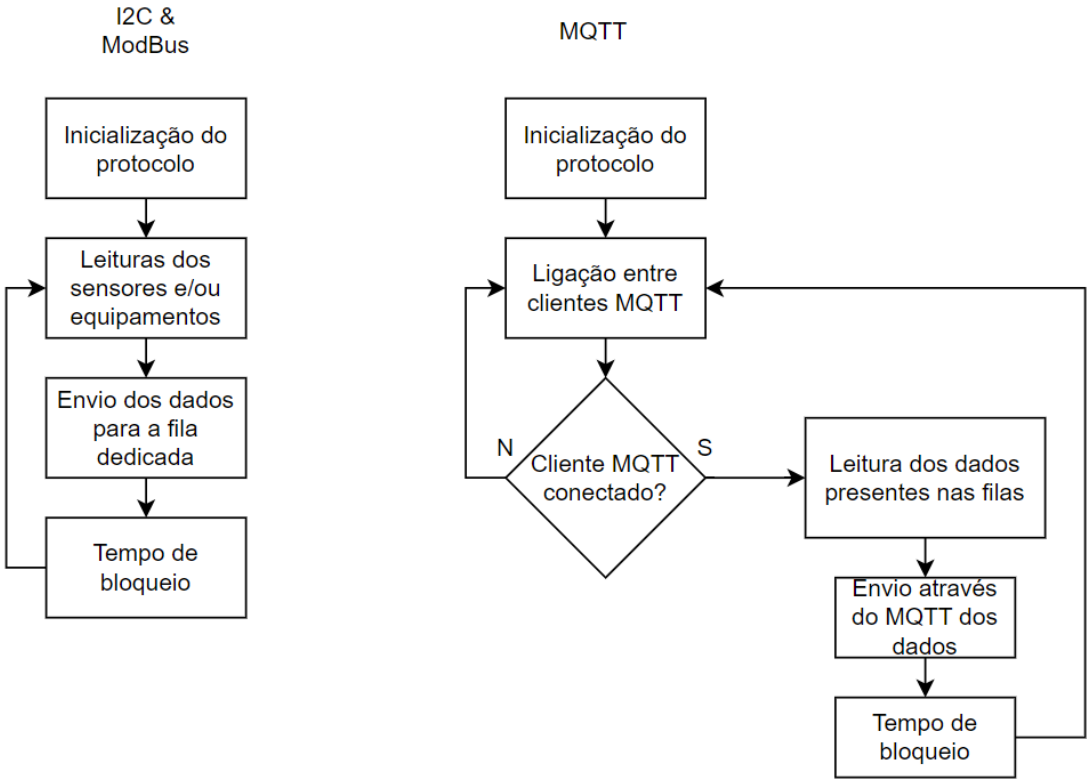
22. Texas Instruments. “*AM243 Siatra Microcontrollers, Datasheet*”. (01/2023)
23. Benzinga. "*Real-time Operating Systems (RTOS)*". (09/2023)
24. Samek, M. "*Programming embedded systems: RTOS – what is real-time?*". (05/2023)
25. Espressif Systems. “*FreeRTOS Overview*”. Disponível em: https://docs.espressif.com/projects/espidf/en/latest/esp32/apireference/system/freertos.html#freertos-overview. (08/2023)
26. WindRiver. “*VxWorks Cert Edition*”. (10/2022)
27. Barry, R. “*Mastering the FreeRTOS Real Time Kernel, Hands-On Tutorial Guide*”. (01/2016)
28. Global Knowledge. “*The OSI Model: Understanding the Seven Layers of Computer Networks*”. (05/2006)
29. Analog Devices. “*UART: A Hardware Communication Protocol Understanding Universal Asynchronous Receiver/Transmitter*”. (07/2020)
30. Texas Instruments. “*A Basic Guide to I2C*”. (09/2022)
31. Analog Devices. “*Introduction to SPI Interface*”. (09/2028)
32. Modicon. “*Modbus Protocol Reference Guide*”. (06/1996)
33. Texas Instruments. “*RS-485 Reference Guide*”. (05/2014)
34. Yassein, M; Shatnawi, M. “*Internet of Things: Survey and open issues of MQTT Protocol*”. (07/2017)
35. Kawaguchi, R; Bandai, M. “*A Distributed MQTT Broker System for Location-based IoT Application*”. (09/2019)
36. Gourley, D; Totty, B. “*HTTP: Definitive Guide*”. Disponível em: https://books.google.pt/books?hl=en&lr=&id=3EybAgAAQBAJ&oi=fnd&pg=PT4&dq=http&ots=X67ZSbiTVq&sig=ydH9THUY_OR9oJVdZJjBd0oeK40&redir_esc=y#v=onepage&q&f=false. (08/2023)
37. The Automatization.com. “*Best HMI software list in Industrial Automation*”. Disponível em: https://theautomization.com/hmi-software-list/. (08/2023)
38. Chooruang, K; Meekul, K. “*Design of an IoT Energy Monitoring System*”. (03/2018)
39. Ministério do Ambiente. “Diário da República – I Série-A – Decreto – Lei nº. 152/97”. (06/1997)

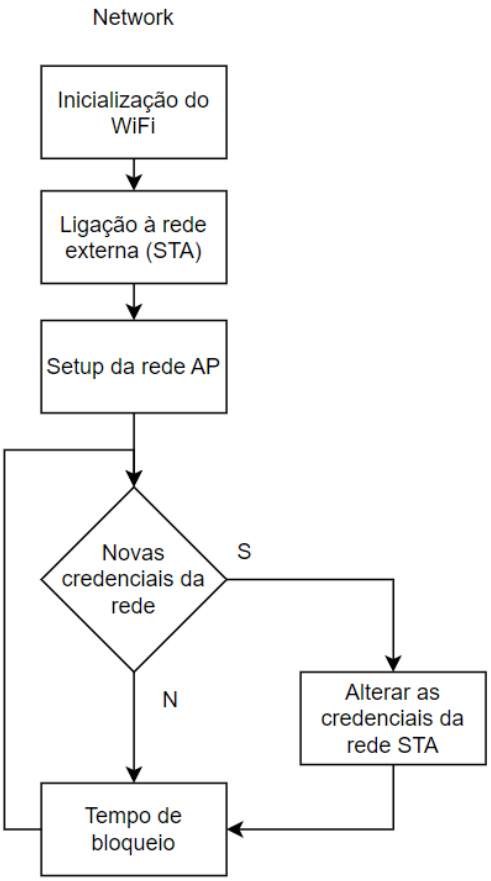
40. AtlasScientific. "*EZO-pH, Embedded pH Circuit, Datasheet*". (02/2024)
41. AtlasScientific. "*EZO-EC, Embeded EC Circuit, Datasheet*". (02/2024)
42. Aqualabo Ponsel. "*NTU Numerical Sensor, User Manual*". (08/2023)
43. Zata International Limited. "*Zata ZWQ-COD1 COD Sensor, User Guide*". (01/2023)
44. Golander Pump. "*BT100S-1 Basic Variable-Speed Peristaltic Pump, Operational Manual*". (02/2023)
45. DFRobot. "*Immersible Pump & Water Tube*". Disponível em: https://www.dfrobot.com/product-667.html (03/2023)
46. Zata International Limited. "*ZWS ZATA, Operation Manual*". (02/2024)
47. DFRobot. "*Gravity: Non-contact Digital Liquid Level Sensor*". Disponível em: https://www.dfrobot.com/product-1493.html. (01/2024)
48. DFRobot. "*TEL0070 Multi USB RES232 RS485 TTL Converter*". Disponível em: https://wiki.dfrobot.com/Multi_USB_RS232_RS485_TTL_Converter__SKU_TEL0070_. (03/2023)
49. Omron. "*PCB Relay G5V-2 – Miniature Relay for Signal Circuits*". (03/2023)
50. Fairchild Semiconductor. "*BC547 Switching Low Noise Silicon Transistor*". (08/2002)
51. Raspberry Pi. "*Raspberry Pi 4 Model B, Product Reference Manual*". (06/2018)
52. Henrique Bernardo, João Canoso, Monitorização Remota de Infraestrutura Piloto de Tratamento de Águas Residuais por via Biológica - MRIPTA, Projeto de Licenciatura em Engenharia Informática, Instituto Politécnico de Tomar. (07/2021)
53. GitHub. "*EcoModZHC Project's Source Code Repository*". Disponível em: https://github.com/Simao-Lopes98. (01/2024)
54. Espressif Systems. "*HTTP Server Documentation*". Disponivél em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/protocols/esp_http_server.html. (01/2024)
55. Espressif Systems. "*Get Started Documentation*". Disponível em: https://docs.espressif.com/projects/esp-idf/en/latest/esp32/get-started/index.html. (03/2024)

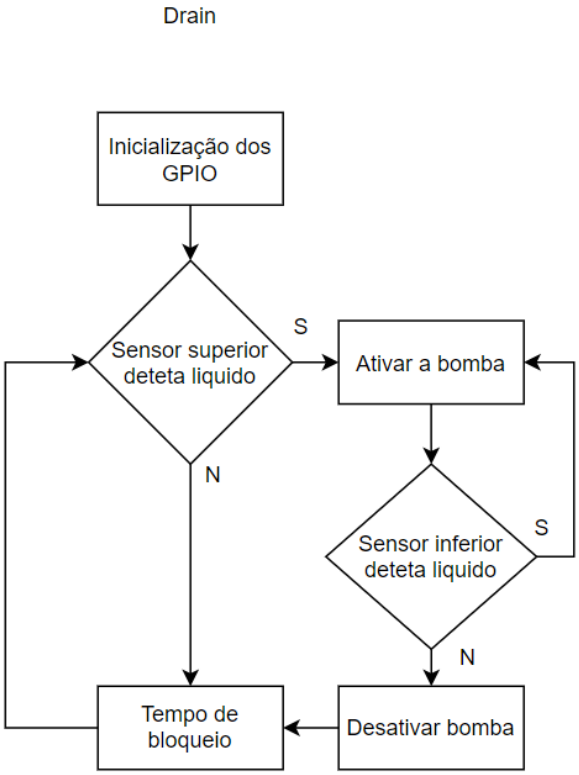
56. Espressif Systems. “*Networking APIs Documentation*”. Disponível em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/network/index.html. (01/2024)
57. Espressif Systems. “*Over The Air Updates (OTA) Documentation*”. Disponível em: https://docs.espressif.com/projects/esp-idf/en/stable/esp32/api-reference/system/ota.html. (01/2024)
58. GCC GNU Compiler. “*How to Use Inline Assembly Language in C Code*”. Disponível em: https://gcc.gnu.org/onlinedocs/gcc/extensions-to-the-c-language-family/how-to-use-inline-assembly-language-in-c-code.html. (02/2024)
59. Traco Power. “*Non-Isolated DC/DC Converter Datasheet*”. (03/2023)
60. Li Wern Chew, "*The effect of higher order model decoupling capacitors in the design of a power delivery network*" 2012 4th Asia Symposium on Quality Electronic Design (ASQED), Penang, 2012, pp. 117-122, doi: 10.1109/ACQED.2012.6320486. (02/2012)

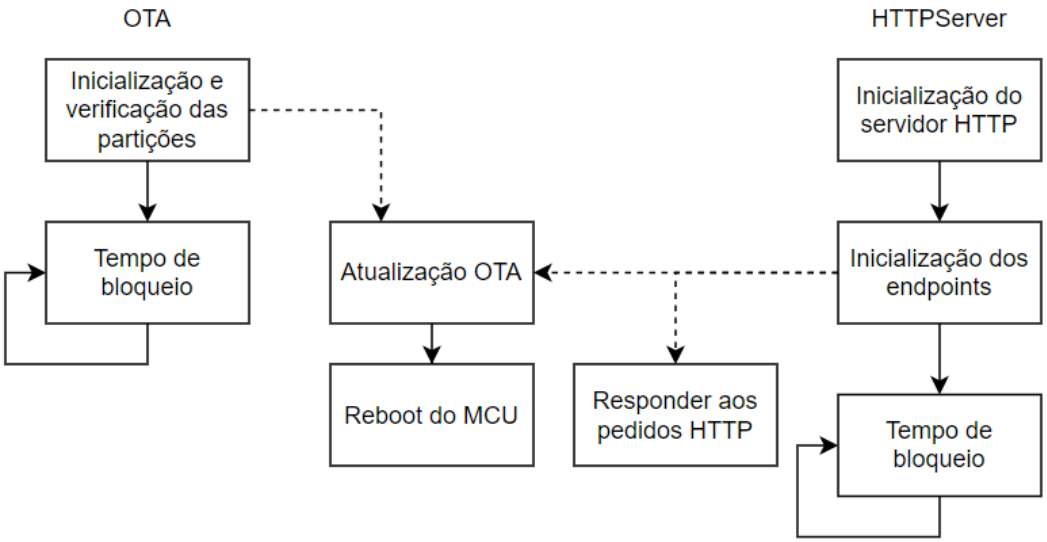
8 Anexos

8.1 Secção A – Fluxogramas das tarefas implementadas

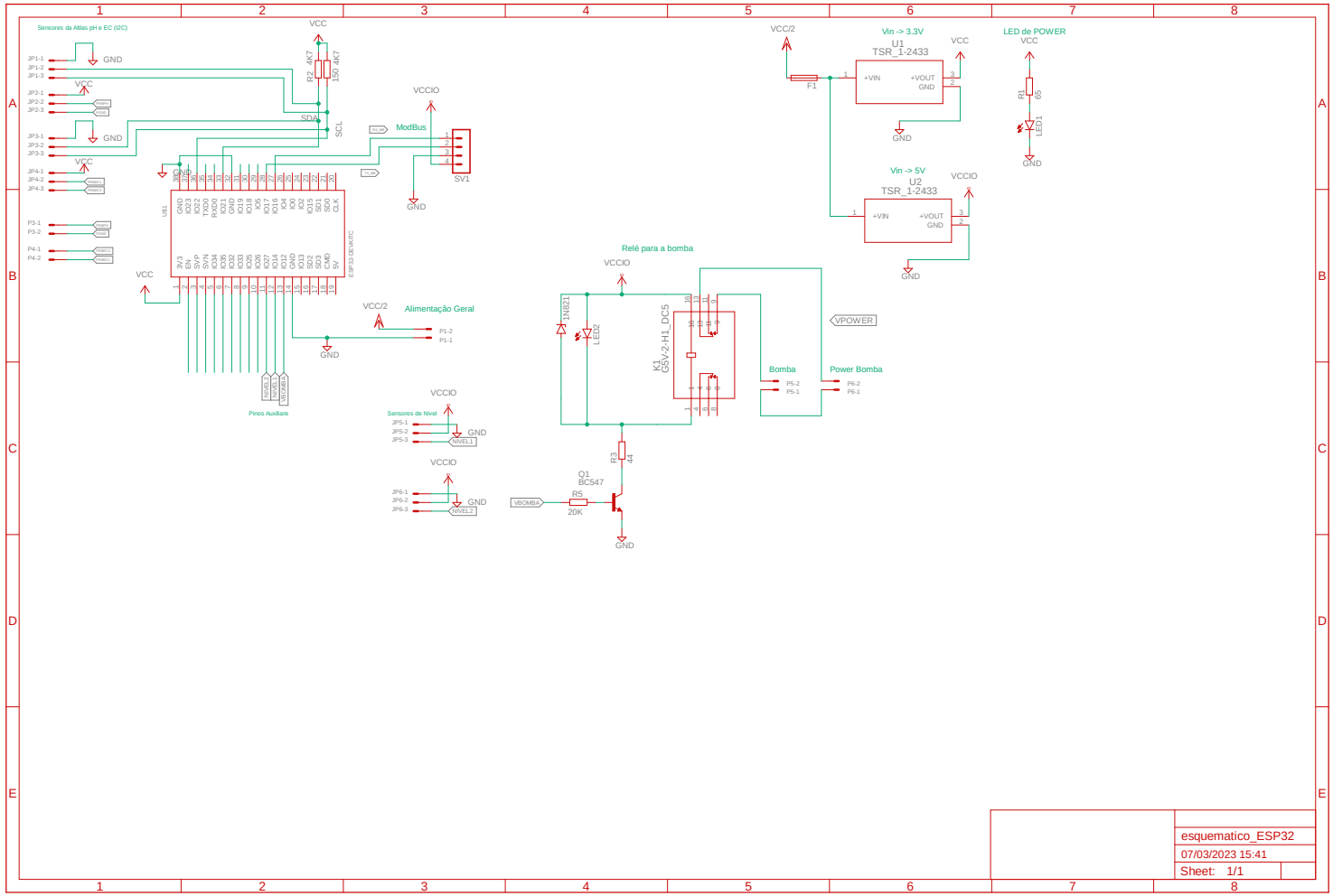


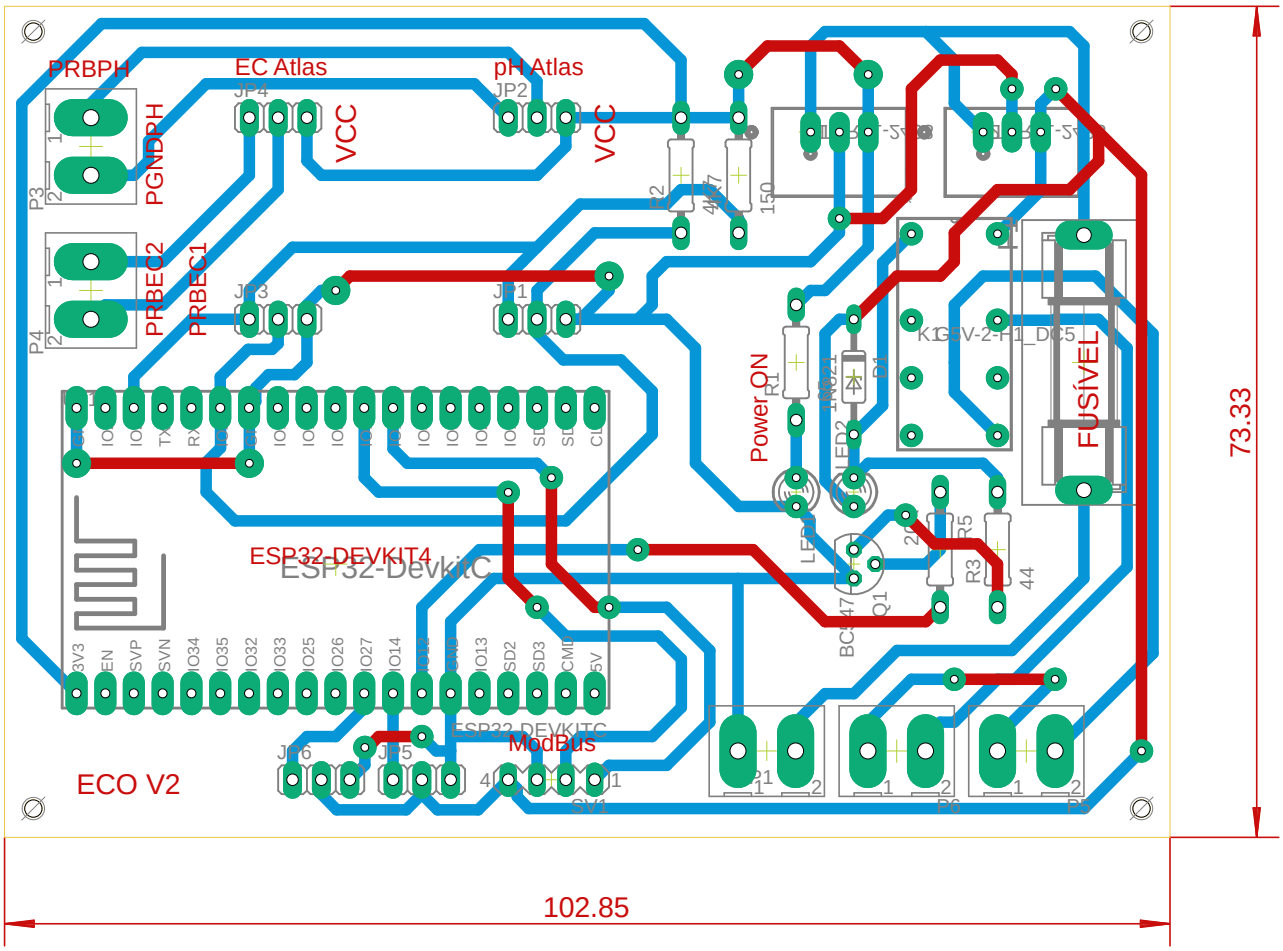

Network
Inicialização do WiFi
Ligação à rede externa (STA)
Setup da rede AP
Novas credenciais da rede
S
N
Alterar as credenciais da rede STA
Tempo de bloqueio


Drain
Inicialização dos GPIO
Sensor superior deteta liquido
S
N
Ativar a bomba
Sensor inferior deteta liquido
S
N
Tempo de bloqueio
Desativar bomba


OTA
Inicialização e verificação das partições
Tempo de bloqueio
Atualização OTA
Reboot do MCU
HTTPServer
Inicialização do servidor HTTP
Inicialização dos endpoints
Responder aos pedidos HTTP
Tempo de bloqueio

8.2 Secção B - PCB


Sensores da Atlas pH e EC (I2C)
VCC
GND
R2 4K7
R4 4K7
SDA
SCL
VCCIO
ModBus
SV1
U$1
ESP32-DEVKITC
Pinos Auxiliare
VCC/2
Alimentação Geral
P1-2
P1-1
Sensores de Nivel
NIVEL1
NIVEL2
Relé para a bomba
1N821
LED2
K1
G5V-2-H1_DC5
VPOWER
Bomba
Power Bomba
P5-2
P5-1
P6-2
P6-1
R3
44
Q1
BC547
R5
20K
VBOMBA
F1
Vin -> 3.3V
U1
TSR_1-2433
+VIN
+VOUT
Vin -> 5V
U2
LED de POWER
R1
68
LED1
esquematico_ESP32
07/03/2023 15:41
Sheet: 1/1


PRBPH
EC Atlas
pH Atlas
JP4
JP2
VCC
VCC
PGNDPH
P3
PRBEC2
PRBEC1
P4
JP3
JP1
Power ON
FUSÍVEL
ESP32-DEVKIT4
ESP32-DevkitC
ESP32 DEVKITC
ModBus
ECO V2
JP6
JP5
SV1
P1
P6
P5
73.33
102.85