UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

LUCAS MACENA LIMA DA SILVA

CONTROLE INDUSTRIAL COM MODBUS/TCP E MICROCONTROLADOR ESP8266

FORTALEZA

2022

LUCAS MACENA LIMA DA SILVA

CONTROLE INDUSTRIAL COM MODBUS/TCP E MICROCONTROLADOR ESP8266

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientador: Prof. Dr. Wilkley Bezerra Correia

FORTALEZA

2022

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Biblioteca Universitária
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

S581c Silva, Lucas Macena Lima da.
Controle Industrial com Modbus/TCP e Microcontrolador ESP8266 / Lucas Macena Lima da Silva. – 2022.
95 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2022.
Orientação: Prof. Dr. Wilkley Bezerra Correia.

1. Modbus. 2. Controle Industrial. 3. Internet das coisas. 4. ESP8266. 5. IoT. I. Título.

CDD 621.3

LUCAS MACENA LIMA DA SILVA

CONTROLE INDUSTRIAL COM MODBUS/TCP E MICROCONTROLADOR ESP8266

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dr. Wilkley Bezerra Correia (Orientador)
Universidade Federal do Ceará (UFC)

Msc. Davi Nunes Oliveira

Prof. Dr. Victor de Paula Brandão Aguiar
Universidade Federal Rural do Semi-Árido
(UFERSA)

Aos meus amigos que acompanharam cada dificuldade no meu caminho e me ajudaram a superá-las. Em memória de Kiara, por ter me ensinado o significado de lealdade.

AGRADECIMENTOS

Ao meu orientador, Prof. Dr. Wilkley Bezerra Correia. Obrigado por sempre pedir o melhor de mim e me mostrar quando eu estava errado.

Ao meu supervisor de estágio, o Eng. Luiz Eduardo Pereira Formiga, por ter me dado o primeiro contato com uma rede de comunicação entre equipamentos. Sua orientação e apoio fizeram toda a diferença no meu desenvolvimento profissional.

À *Altput Robótica*, em especial ao meu amigo Eng. Matheus Sampaio, por me ensinar tudo o que sei de eletrônica e por sempre acreditar no meu potencial.

Ao Grupo de Desenvolvimento Aeroespacial da UFC, em especial ao subsistema da eletrônica, por serem a minha família na graduação. As risadas e conversas na casinha de plástico me salvaram muitas vezes.

À minha amiga, Eng. Grazielly Alves, por sempre ser o meu reforço, sem sua amizade por todos esses anos, nada disso seria possível.

À minha amiga, Lorena Karen, por estar comigo nos piores momentos. Gratidão pelo que me ensinou. Você me mostrou como ser mais gentil e paciente todos os dias.

Aos meus amigos Amanda Bricio, Igor Natanael e Raul Kelvin, por nunca desistirem de mim apesar de todo o drama e da distância ao longo dos anos. Espero que possamos continuar com nossas brincadeiras até na velhice.

À minha namorada, Jâmily Karen, nossos momentos e sua companhia me proporcionaram uma enorme alegria no final dessa jornada. Espero sempre ter você comigo daqui para frente nessa nova fase.

À minha irmã de coração, Vitória Cholanda, sei que apesar da distância, às vezes, sempre poderei contar com você. Você tem um lugar especial na minha vida, e quero sempre ter você na minha trajetória.

À minha amiga, Ana Carolina, por ser essa companhia nos últimos anos de faculdade. Você me superestima às vezes, mas seu apoio me ajudou a levantar nos momentos que eu caí.

Aos meus amigos de colégio, Madson Ivens, Teófilo Ravel, João André e Caio Gabriel. As idas e vindas da vida nunca foram suficientes para nos afastar. Nossa amizade permanece tão forte quanto na infância.

Ao meu irmão Marcos Filho e ao meu primo Alan Jefferson. Vocês afastam os males e problemas de mim quando estou triste. Eu sempre estarei com vocês.

À minha mãe, Osmarina Macena, por tudo que sacrificou para me dar o melhor na vida. Você é um exemplo de gentileza e altruísmo para mim. Ao meu pai, Marcos Antônio, por seu trabalho duro e perseverança. Ao meu avô, Antônio Manuel, por me inspirar a seguir na engenharia elétrica e a definir o rumo profissional da minha vida.

Ao Doutorando em Engenharia Elétrica, Ednardo Moreira Rodrigues, e seu assistente, Alan Batista de Oliveira, aluno de graduação em Engenharia Elétrica, pela adequação do *template* utilizado neste trabalho para que o mesmo ficasse de acordo com as normas da biblioteca da Universidade Federal do Ceará (UFC).

“Para todos que tiveram um momento de fraqueza. Não vai doer para sempre, então não deixe isso afetar o que há de melhor em você.”

(J. A. Redmerski)

RESUMO

O seguinte trabalho de conclusão de curso implementa comunicação remota à aplicação de controle discreto de uma planta industrial por meio do microcontrolador ESP8266 embarcado na placa de desenvolvimento Wemos D1 mini. A comunicação remota é realizada pelo protocolo *Message Queue Telemetry Transport* (MQTT), responsável pelo envio de comandos e recebimento de informação do sistema, até mesmo em diferentes redes *wi-fi*. A Wemos D1 mini permite a realização uma interface remota-física entre o sistema motriz, onde o controlador discreto está sendo aplicado, e uma unidade de supervisão remota. A implementação e a comunicação é realizada pelo protocolo *Modbus*/TCP. O controlador executado pelo microprocessador é um Proporcional Integral Derivativo (PID) discreto devidamente projetado e contemplando a dinâmica de atraso de pacotes vinculada à comunicação via *Transport Control Protocol* / Protocolo de controle de transporte (TCP) sem fio. Devido ao fato do controlador não ter a capacidade de antecipar as inconstâncias do roteamento de pacotes, até mesmo em rede local, um filtro discreto passa-baixa foi implementado e aplicado na realimentação da malha de controle. Este trabalho apresenta resultados que evidenciam a possibilidade de estabelecer comunicação sem fio entre dispositivos inteligentes com uma placa de desenvolvimento de baixo custo e de fácil implementação, sendo aplicável em *smart homes*, monitoramento e alarme remoto.

Palavras-chave: Modbus. Controle Industrial. Internet das coisas. ESP8266. IoT. SCADA. MQTT.

ABSTRACT

The following course conclusion work implements remote communication to the discrete control application of an industrial plant through the ESP8266 microcontroller embedded in the Wemos D1 mini development board. Remote communication is carried out by the Message Queue Telemetry Transport (MQTT) protocol, responsible for sending commands and receiving information from the system, even in different Wi-Fi networks. The Wemos D1 mini allows the realization of a physical-remote interface between the driving system, where the discrete controller is being applied, and a remote supervision unit. The implementation and communication is performed by the *Modbus*/TCP protocol. The controller executed by the microprocessor is a discrete proportional integral derivative (PID) properly designed and contemplating the dynamics of delay of packets linked to the communication via Transport Control Protocol (TCP) wireless. packets, even in a local network, a discrete low-pass filter was implemented and applied to the control loop feedback. This work presents results that show the possibility of establishing wireless communication between smart devices with a low cost and easy to implement development board, being applicable in smart homes, monitoring and remote alarm.

Keywords: Modbus. Industrial Control. Internet of things. ESP8266. IoT. SCADA. MQTT.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE CÓDIGOS-FONTE

LISTA DE ABREVIATURAS E SIGLAS

ADC	*Analogic Digital Converter* / Conversor analógico digital
CLP	Controlador lógico programável
CSV	*Comma separated values* / Valores separados por vírgula
ERP	*Enterprise Resource Planning* / Planejamento dos Recursos Corporativos
HTTP	*Hyper Text Transport Protocol*
I^2C	*Inter-Integrated Circuit* / Circuito Inter-Integrado
IHM	Interface Homem Máquina
IoT	*Internet of Things* / Internet das coisas
ISO	*International Organization for Standardization* / Organização Internacional para Padronização
KPI	*Keys Performance Indicators* / Indicadores Chaves de Performance
LAN	*Local Area Network* / Rede de Area Local
MES	*Manufacturing Execution System* / Sistema de Execução da Produção
MQTT	*Message Queue Telemetry Transport*
MTBF	*Mean Time Between Failures* / Tempo médio entre falhas
MTTR	*Mean Time To Repair* / Tempo médio de reparo
OSI	*Open Systems Interconnection* / Interconexão de Sistemas Abertos
OTA	*Over The Air* / Programação Através do Ar
PID	Proporcional Integral Derivativo
PWM	*Pulse-width modulation* / Modulação por largura de pulso
RTU	*Remote Terminal Unit* / Unidade Terminal Remota
SED	Subestação de Distribuição
SPI	*Serial Peripheral Interface* / Interface periférica serial
TCP	*Transport Control Protocol* / Protocolo de controle de transporte
UART	*Universal Asynchronous Receiver/Transmitter* / Transmissor/Receptor universal assíncrono
UDP	*User Datagram Protocol* / Protocolo de datagrama do usuário
UFC	Universidade Federal do Ceará

SUMÁRIO

1 INTRODUÇÃO

O conceito de *Internet of Things* / Internet das coisas (IoT) simboliza a idéia de um mundo onde não só computadores e *smartphones* estejam ligados, mas sim objetos cotidianos, aqueles que utilizamos em nossas vidas diariamente para as tarefas mais comuns e domésticas. (CIRANI *et al.*, 2019).

Com o aumento das soluções IoT, surge a possibilidade da integração residencial para o usuário comum. Automação em sistemas de iluminação, temperatura, ventilação, segurança e controle permitem hoje que sejam criados residências e prédios inteligentes. Toda essa conectividade fornece informações como consumo de energia em um mundo onde a demanda por economia de energia aumenta a cada dia. Além da possibilidade de configuração de blocos inteiros de equipamentos e cômodos com um único comando (LIANG *et al.*, 2020).

A chegada dos circuitos integrados e microprocessadores causou certo receio na população operária. A ideia de que uma máquina realizasse a atividade de um ser humano de maneira contínua e mais rápido, levou os trabalhadores a temer por seus empregos. Contudo, um sistema automático ainda precisa de operadores e manutenções periódicas (RIBEIRO, 2001).

Nesse aspecto, a automação também é uma fonte de garantia e estabilidade do emprego. Essa perspectiva é muito clara ao se comparar a probabilidade de uma empresa sem automação e uma empresa com processos automáticos (e por consequência, mais eficazes) de continuar no mercado competitivo. Logo, um funcionário de uma empresa moderna e supervisionada sistematicamente possui um emprego mais estável. Do ponto de vista operativo, a automação na indústria trouxe incontáveis possibilidades e funcionalidades, entre elas a possibilidade de monitoramento remoto, registro de alarmes, controle de produção, segurança e a implementação de projetos de controle discreto à plantas mais sensíveis (RIBEIRO, 2001).

Devido ao tempo de vida restante de várias soluções industriais, vinculado à crescente emergência de tecnologias de comunicação sem fio na indústria e à demanda por sistemas de menor custo, a eficiência e produtividade vem se tornando prioridades cada vez maiores. O conceito de diálogo entre os processos industriais possui vantagens em comparação as já consolidadas soluções cabeadas. Talvez uma das características mais importantes é a de auto organização de um sistema em tempo real capaz de manter sua integridade dispensando interfaces físicas (KIM; TRAN-DANG, 2019).

1.1 Motivação do trabalho

Ao imaginar um chão de fábrica, a ideia inicial que vem à mente da maioria das pessoas é de um local barulhento, cheio de máquinas e fios. E todo esse cenário é uma visão muito sólida e imutável, onde não há espaço para mudança. Em escritórios, é comum haver uma readequação do espaço físico com o objetivo de obter uma mudança nos ares ou alterar os processos em funcionamento.

Até o início do século 21, a integração necessária entre máquinas e sistemas sustentava essa ideia imutável. Isso se deve à demasiada necessidade de cabos ligando cada planta, sensor e computador do local. A simples indagação do quão complexo seria alterar esse cabeamento, por muitas vezes, era um obstáculo frente à onda de mudanças emergentes.

Ao utilizar soluções de conectividade sem fio entre os processos, o sensoriamento e informações ligados à planta podem ser facilmente entregues aos supervisórios, em tempo real, sem a necessidade de adequação da fiação.

Devido à pandemia de *Covid-19*, as empresas foram forçadas à enviar muitos funcionários para o regime de trabalho *home office*. Mas como manter a supervisão de uma fábrica enquanto está em casa? A solução é simples, a possibilidade de receber essas informações sem a necessidade de se deslocar até o local, ou seja, informação e conectividade a qualquer hora e em qualquer lugar. A Interface Homem Máquina (IHM), sensores e sistema produtivo passam a ser ligados não mais por chicotes de condutores anilhados mas sim por sinais digitais transmitidos via rede.

1.2 Justificativa do trabalho

A evolução do processo industrial sempre foi limitado por barreiras tecnológicas. O crescente aumento das plantas industriais e complexidade dos processos, fez com que os sistemas necessitassem de uma integração cada vez maior.

Com as tecnologias emergentes dos últimos anos, houve a popularização dos equipamentos IoT e, juntamente à eles, a criação de ambientes inteligentes, tanto residenciais quanto comerciais. Desse modo, existe a necessidade de atualização das soluções de automação industrial datadas da 3ª revolução industrial, para o atual cenário produtivo conhecido como *Indústria 4.0*.

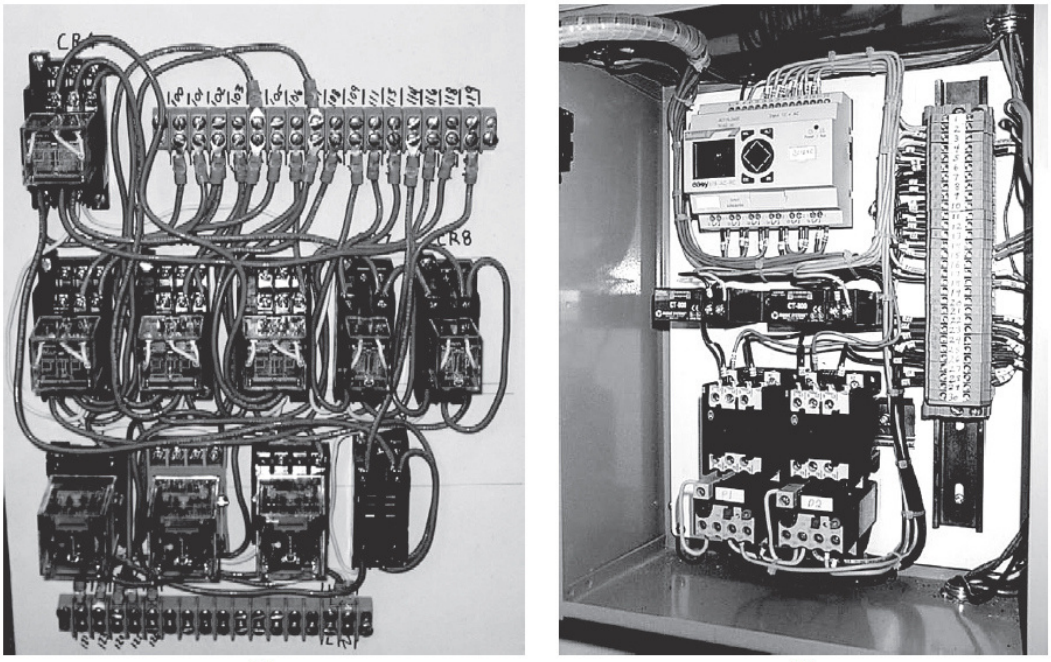
1.3 Organização do trabalho

Este documento é dividido em 5 capítulos principais. Os dois primeiros capítulos compõem uma introdução à evolução do cenário industrial e às soluções para a comunicação e automação nas fábricas implementadas na 3ª revolução industrial. O capítulo 4 trata de todo o projeto desenvolvido, explicado em detalhes. Todos os aspectos e informações necessárias para a implementação prática da aplicação proposta se encontram neste documento e no repositório *git* no endereço: **https://github.com/lucas-macena/TCC_codes**. Com a implementação da aplicação, os resultados foram adquiridos e resumidos no capítulo 5, juntamente com uma avaliação destes com o esperado. Finalmente, o capítulo 6 conclui o texto com as expectativas alcançadas com a solução, assim como trabalhos e melhorias futuras à solução desenvolvida.

2 CONTROLE E SUPERVISÃO

Em meados de 1970, a chegada dos CLPs foi ganhando espaço na indústria e vantagem quanto às outras soluções de controle industrial. Estes comumente conhecidos como computadores industriais em miniatura, são desenvolvidos especificamente para a utilização em fábrica. Eles possuem múltiplas entradas e saídas,são capazes de suportar grandes variações de temperatura e possuem resistência à vibração e impactos (GUPTA *et al.*, 2016, p. 395).

Figura 1 – (a) Painel de controle baseado em relés; (b) painel de controle baseado em CLP.



(*a*) (*b*)

Fonte: (PETRUZELLA, 2014, p. 2)

Os sistemas industriais convencionais se baseiam na conexão direta e física, por meio de cabos, entre todos os dispositivos de controle. Esse tipo de aplicação é ilustrada na figura 1 (a). O advento dos CLPs permitiu que todas as conexões diretas entre os dispositivos, fosse centralizada no CLP. Este por sua vez, realiza a conexão entre as entradas e saídas digitalmente através de sua programação. Desse modo, os quadros e sistemas são implementados como por exemplo na figura 1 (b) (GUPTA *et al.*, 2016, p. 396).

O CLP, além de funcionar como um equipamento centralizador do processamento da indústria, possui a versatilidade de ser programado de cinco maneiras diferentes sendo estas regidas pela IEC 61131-3 (PETRUZELLA, 2014, p. 76):

1. Lista de instruções (IL).

2. Texto Estruturado (ST).
3. Diagrama Ladder (LD).
4. Diagrama de blocos funcionais (FBD).
5. Mapa de Função Sequencial (SFC).

Estes dispositivos foram utilizados inicialmente como solução para o empecilho da modificação das instalações baseadas em relés, sempre que era necessária uma alteração em seu funcionamento. Juntamente ao fato de serem facilmente programados, os CLPs possuem as vantagens de alta confiabilidade e flexibilidade para a instalação onde este se encontra (PETRUZELLA, 2014, p. 2).

> O CLP é projetado para arranjos de múltiplas entradas e saídas, faixas de temperatura ampliadas, imunidade a ruído elétrico e resistência à vibração e impacto. Programas para controle e operação de equipamentos de processos de fabricação e mecanismo normalmente são armazenados em memória não volátil ou com bateria incorporada. Um CLP é um exemplo de um sistema em tempo real, considerando que a saída do sistema controlado por ele depende das condições da entrada. (PETRUZELLA, 2014, p. 1)

2.1 Surgimento dos supervisórios

A implementação de automação em diferentes plantas industriais nos anos 70 e 80, mudou a produção em diferentes segmentos industriais. Partindo do exemplo onde um processo químico depende de informação de pressão e temperatura dentro de uma caldeira em determinado ponto da instalação, e a sala de controle da fábrica. Existe a necessidade de implementação de sensores inteligentes ao processo (Anderson Mott, 2021).

A conexão entre os processos passa então a ser realizado por meio da *Local Area Network* / Rede de Area Local (LAN) da indústria de modo a possibilitar o incremento nas funções dos sistemas SCADA. A utilização da rede local nesses sistema possibilitou sua utilização por mais de um acesso simultâneo e o acesso aos bancos de dados criados a partir das informações em tempo real (BOYER, 2004).

Os sistemas automáticos de supervisão centralizam um conjunto de informações e sinais que auxiliam na tomada de decisão administrativa, de operação ou de manutenção. Podem ser divididos seguindo o diagrama da figura 2.

Figura 2 – Pirâmide de automação.

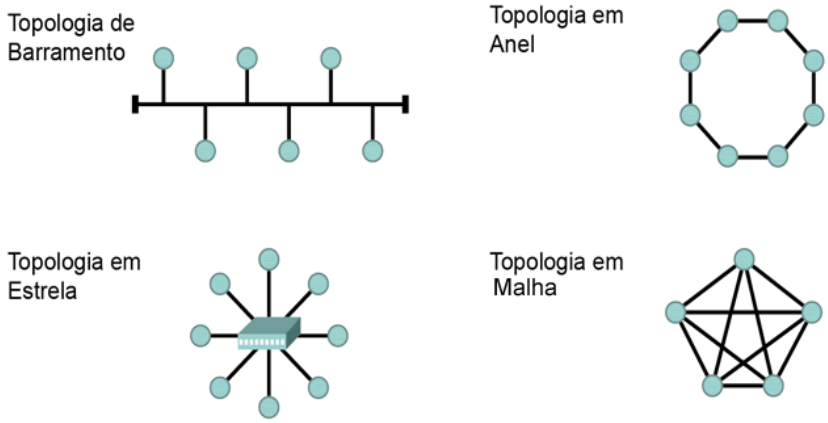
Fonte: (Brian Harrison, 2020)

Onde:

- ERP: *Enterprise Resource Planning* / Planejamento dos Recursos Corporativos. É definido como o sistema de união de todos os dados e informações relevantes para a tomada de decisão de um negócio. Os gestores, bem como os colaboradores, tem acesso aos dados em tempo real e sem o risco da existência de dados desatualizados.
- MES: *Manufacturing Execution System* / Sistema de Execução da Produção. Sistema dedicado ao controle de processos de produtividade. Por meio de seus relatórios e monitoramento em tempo real da produtividade e do estoque, permite melhorar a produção.
- SCADA: *Supervisory Control and Data Acquisition* / Supervisório de Controle e Aquisição de Dados. É uma forma de monitoramento das variáveis e estados do sistema como um todo em tempo real. Suas informações são diretamente coletadas pelos dados enviados pelos equipamentos.
- *Field Level*: Nível de acesso e informações no campo. No exemplo de uma subestação de distribuição de energia, seria o sensoriamento e interfaces localizadas dentro da SED, mas em um local afastado do equipamento.
- *The Process*: Nível local diretamente no equipamento ou máquina supervisionada.

Após a caracterização dos sistemas SCADA, as possíveis topologias de redes são exemplificadas na figura 3. Cada uma das topologias tem objetivos específicos e devem ser utilizados da melhor maneira para obter uma rede mais eficaz.

Figura 3 – Topologias de rede.



Fonte: (FabioBmed, 2012)

Onde:

- Barramento: É utilizado o mesmo meio físico para a comunicação entre todos os dispositivos. Permite uma comunicação bidirecional, mas com uma limitação do número de derivações e comprimento do meio físico.
- Anel: O fluxo de informação acontece apenas em um sentido, onde cada vez que passa por um novo dispositivo, as informações dele são adicionadas ao fluxo e enviadas ao próximo equipamento.
- Estrela: Topologia encontrada em redes locais domésticas por exemplo. Consiste em um elemento central conhecido como *Gateway*, sendo este responsável pela troca de pacotes de forma bidirecional entre os dispositivos.
- *Peer to peer*: Comunicação bidirecional entre dois dispositivos. Também pode ser chamada de M2M (*Machine to Machine*).
- Malha: É um conjunto de conexões *peer-to-peer* uqe integra todos os dispositivos, permitindo a continuidade da operação mesmo em caso de uma falha pontual.

2.2 Planejamento e controle da manutenção

Uma das vantagens da aplicação da integração dos equipamentos em uma rede sem fio é o controle das ações de manutenção. Hoje, em um cenário de aplicação de sistemas ERP, a utilização de sistemas computacionais automáticos auxilia na integração das informações. Esses dados variam, desde cadastros de equipamentos até retornos das equipes de campo durante uma manutenção (VIANA, 2002, p. 178).

A atividade de manutenção possui o objetivo de evitar o dano parcial ou total das

instalações. Na manutenção, é necessário, por parte da gestão, apurar o panorama atual e traçar um objeto através de meios definidos. Nessa ação, são utilizados indicadores específicos para cada atividade, os chamados *Keys Performance Indicators* / Indicadores Chaves de Performances (KPIs) (VIANA, 2002, p. 155).

> O controle da manutenção é feito através da criação e da gestão dos indicadores de manutenção, que servirão como base para a tomada de decisões e desenho de estratégias. Sem os indicadores de manutenção, fica impossível saber se as decisões tomadas são certas ou erradas, assim como em qualquer outra área de atuação. (Jhonata Teles, 2021)

Essa indisponibilidade é calculada tomando como base dois indicadores importantes para o planejamento e controle de produção. Eles são o *Mean Time Between Failures* / Tempo médio entre falhas (MTBF) e o *Mean Time To Repair* / Tempo médio de reparo (MTTR) e podem ser calculados tomando as equações 2.1 e 2.2

$$MTBF = \frac{\sum \text{Tempo em funcionamento (horas)}}{\text{nº de manutenções corretivas}} \tag{2.1}$$

$$MTTR = \frac{\sum \text{Tempo de reparo (horas)}}{\text{nº de intervenções}} \tag{2.2}$$

$$\%Indisponibilidade = (1 - \frac{MTBF}{MTBF + MTTR}) * 100 \tag{2.3}$$

As atividades de manutenção foram criadas durante a primeira revolução industrial e se mostraram presentes em toda a evolução do processo produtivo. O serviço de manutenção pode ser caracterizado de três maneiras diferentes seguindo a natureza pela qual ela é impulsionada (VIANA, 2002, p. 17, 25-32).

- **Manutenção corretiva:** Caracteriza-se como a correção de uma falha ou defeito já presente no objeto de manutenção. Esse tipo de manutenção apresenta alto custo de indisponibilidade e de correção das peças defeituosas.
- **Manutenção preventiva:** É a constante prevenção de falhas, de modo que com um acompanhamento constante, seja possível evitar uma indisponibilidade ou perca completa do objeto de manutenção. Contudo, se for mal dimensionada, pode apresentar alto custo operativo.

– **Manutenção preditiva:** Por meio de constante acompanhamento de parâmetros de campo, procura por problemas e pode antecipar problemas. Permite um ajuste mais fino da Manutenção Preventiva, diminuindo os custos e permitindo uma disponibilidade do sistema pelo maior tempo possível.

> Contrariamente à manutenção preventiva dos dias de hoje, a manutenção preditiva oferece melhorias em termos de performance dos equipamentos. Acrescentando sensores, um *monitoring* constante e registando diversos valores de grande importância, é possível pensarmos numa assistência remota ao equipamento. (BORLIDO, 2017)

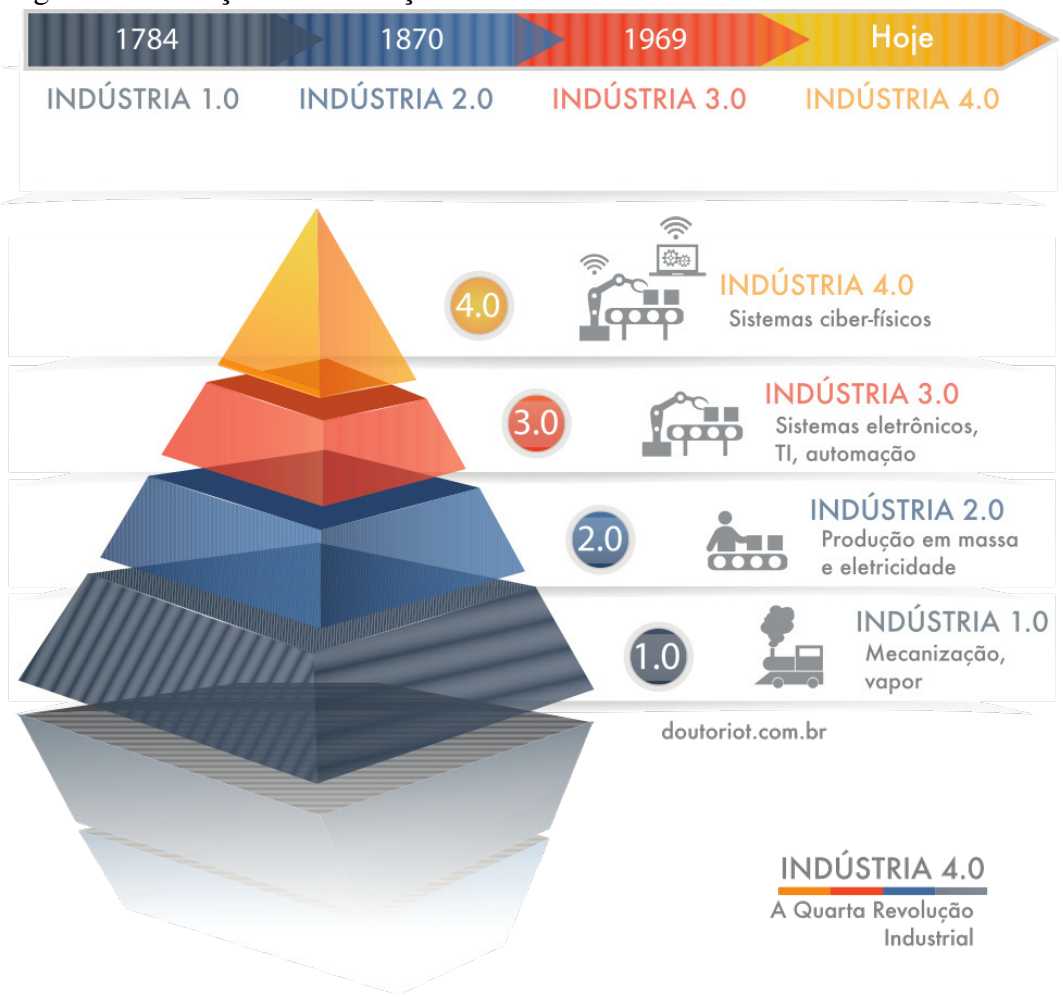
Por meio desse breve contexto no planejamento de manutenção, é notável a importância dos dispositivos IoT no cenário atual da indústria. Sem a existência destes equipamentos, a integração em rede das máquinas e processos não seria possível em larga escala. A tendência então, é de que as soluções até então consolidadas de automação serão inevitavelmente substituídas por processadores mais modernos e inteligentes.

No contexto das concessionárias de distribuição de energia, por exemplo, a informação fornecida por meio de sistemas SCADA podem ser comparadas ao coração do processo de manutenção. Um parque de distribuição usualmente possui mais de 4300 equipamentos em linhas de média tensão com telecontrole. Essas chaves e religadores estão sempre em comunicação com o sistema principal de supervisão e este, indica o estado em tempo real do parque de distribuição. Em um conjunto tão grande e disperso, as estratégias de manutenção preventiva se tornam ineficazes e com alto custo, uma vez que é inviável o planejamento de visitas periódicas para o parque completo visando a garantia de funcionamento de cada equipamento. As informações e alarmes coletados, servem então como guia para o planejamento da manutenção preditiva e na tomada de decisão estratégica de quais equipamentos devem ser priorizados.

2.3 Revolução industrial

Os processos industriais já passaram por diversas mudanças ao longo dos anos sendo palco das mais diversas inovações tecnológicas. Visando alcançar o máximo de eficiência e a redução de custos, o mercado sempre procurou o que há de mais sofisticado nas máquinas e plantas industriais. Desse modo, a constante criação de tecnologias cresce desde os tempos da primeira revolução industrial. Esse padrão segue até hoje rumo à *Indústria 4.0*. A figura 4 traz uma breve linha do tempo quanto às revoluções industriais.

Figura 4 – Evolução das revoluções industriais.



Fonte: (Gabriel Martins Dias, 2021)

– **1ª revolução industrial:** Ocorreu no final do século XVIII e foi marcada pela substituição de processo manuais por máquinas a vapor e hidráulica. As manutenções corretivas que eram realizadas eram bem simples e ocorriam apenas mediante falha.
– **2ª revolução industrial:** Iniciada no final do século XIX, foi marcada pela chegada da eletricidade no cenário industrial e os modelos de produção em massa em linhas de produção. Nesse período foi iniciado o conceito de manutenção preventiva, uma vez que o tempo de indisponibilidade de processos industriais significava uma linha de produção inteira parada. Por consequência, havia um enorme receio da ocorrência de falhas, aumentando a frequência desse tipo de supervisão.
– **3ª revolução industrial:** Tendo início nos anos 70, sua marca é a implementação da eletrônica e dos sistemas de informação. Desses sistemas, juntamente com a ampliação dos sistemas industriais, tanto em complexidade, quanto na necessidade de mão de obra qualificada surgiu a necessidade de um ajuste mais sofisticado no modelo de manutenção

preventiva, assim, se inicia a rotina de manutenção preditiva.

Uma vez de posse da evolução tecnológica que possibilitasse a implementação de algorítimos mais complexos, era então possível a utilização de sistemas de controle discreto em plantas industriais.

Estes controladores discretos, em sua forma mais simplista, consistem de uma série de cálculos matemáticos tomando como base o erro encontrado entre uma determinada referência que deve ser seguida, e a leitura na saída da planta. Dessa maneira, assim como nos CLPs, é possível implementar algo semelhante utilizando microcontroladores modernos ou placas de desenvolvimento como a Wemos D1 mini, configurando assim, uma possível evolução das já consolidadas soluções de automação.

2.3.1 Indústria 4.0

A 4ª revolução industrial foi chamada de *Indústria 4.0* pela primeira vez em 2011 na Feira Industrial de Hanover na Alemanha. A ocasião marcou o início da corrida por sistemas de informação buscando a implementação das chamadas *Smart Factory*.

É um constante erro popular em achar que uma indústria 4.0 é aquela onde os processos estão completamente automatizados. Na realidade, a 4ª revolução industrial é, além da automação, a interconexão informativa entre as máquinas e o sistema. Essa constante troca de informações possibilita a comunicação e cooperação entre as máquinas.

Uma vez que as variáveis e situação em tempo real do sistema podem ser obtidas por meio de sistemas supervisórios, tornou-se pela primeira vez na história o conceito de trabalho remoto em atividades industriais.

Nos anos 2000 havia a necessidade de operadores no chão de fábrica anotando manualmente as informações de cada máquina da indústria em intervalos periódicos. Agora, com o crescimento dos dispositivos IoT, pode-se dizer que essa revolução é marcada pela aplicação de tais equipamentos nas redes industriais.

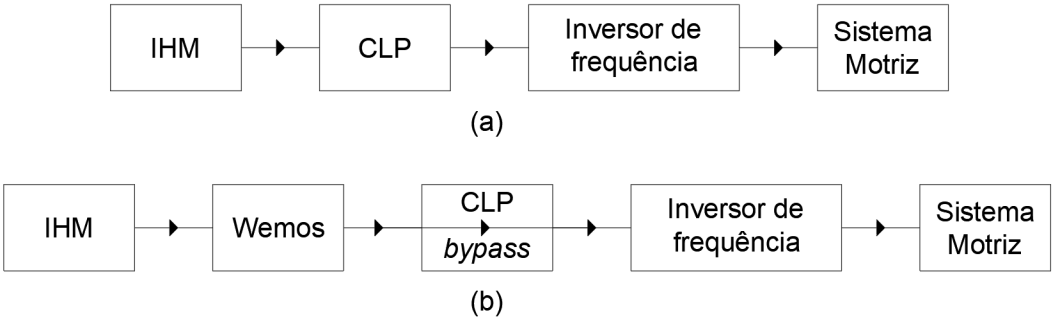
2.4 Aplicação ao Lamotriz

O trabalho aqui descrito faz parte de uma proposta de reestruturação de rede do Laboratório de Máquinas Motrizes da Universidade Federal do Ceará (UFC) (Lamotriz). A estrutura atual de cada processo do laboratório consiste de uma planta motriz, um inversor e um

elemento concentrador (CLP).

A proposta é a operação do CLP como um elemento concentrador das entradas e saídas da planta. Sua função será a de transferir as informações das entradas para as saídas, sem realizar nenhuma operação nesse intervalo. Já as suas responsabilidades e lógica de programação serão transferidas para um microcontrolador dotado de comunicação sem fio. O atual cenário operacional dos sistemas do laboratório seguem o padrão esquemático da figura 5. A sub-figura (a) representa a lógica atual utilizada, enquanto que a sub-figura (b) descreve de maneira simplificada a alteração proposta. É essa mudança que abre as possibilidades do IoT nas plantas do Lamotriz.

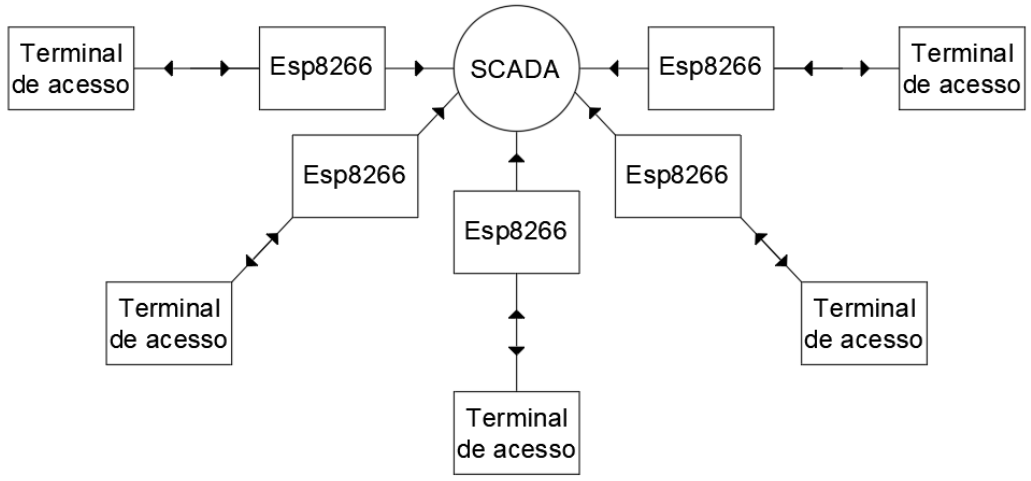
Figura 5 – Esquema operacional do Lamotriz.



Fonte: Autoria própria

Ao se basear nas topologias apresentadas em 2.1, uma possível topologia aplicável é demonstrada na figura 6. Ela é configurada como uma rede em topologia estrela onde os dispositivos a ela conectados, estão com canal de comunicação aberto em topologia *peer-to-peer* com terminais de acesso individuais.

Figura 6 – Topologia proposta para o Lamotriz.



Fonte: Autoria própria

O diagrama completo proposto inclui os protocolos de comunicação e estratégias de integração que serão descritas no capítulo 4.

3 COMUNICAÇÃO EM REDES INDUSTRIAIS

Com a crescente evolução, os sistemas de automação baseados em relés foram sendo substituídos por aqueles baseados em CLPs, e assim, não só a automação na indústria foi crescendo, mas também as funções e capacidade destes dispositivos. Hoje, eles estão mais otimizados, rápidos e possuem a habilidade de comunicação via rede (PETRUZELLA, 2014, p. VII).

Diante desse cenário, e com mais inovações tecnológicas, foram criadas as primeiras comunicações entre dispositivos industriais baseados nas topologias de rede já existentes entre computadores.

3.1 Protocolos de comunicação

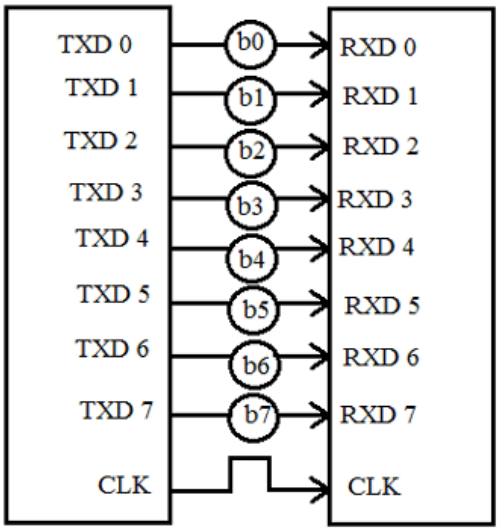
Diante da necessidade de transferir informações entre dispositivos, a busca por métodos de comunicação foi evoluindo e criando padrões específicos. Esses padrões, assim como seus conjuntos de regras, são os chamados protocolos de comunicação. Ao longo dos anos, inúmeros protocolos de rede foram criados com padrões específicos e por na maioria das vezes, incompatíveis entre si.

3.1.1 Comunicação digital

Dois tipos de transmissão digital de informações via CLP podem ser identificadas sendo elas via *Paralela* ou *Serial*. A transmissão *Paralela* é menos comum em comparação com a *Serial*, e era mais comum em dispositivos mais antigos, como impressoras. Na sequência, são listadas as principais características desse tipo de comunicação:

- Os 8 *bit*s de cada *byte* são enviados ao mesmo tempo por 8 caminhos distintos.
- Necessários 7 caminhos adicionais em relação à Comunicação Serial.
- Transmissão mais rápida que a *Serial* (1 *byte* por *clock*).

Figura 7 – Comunicação Paralela

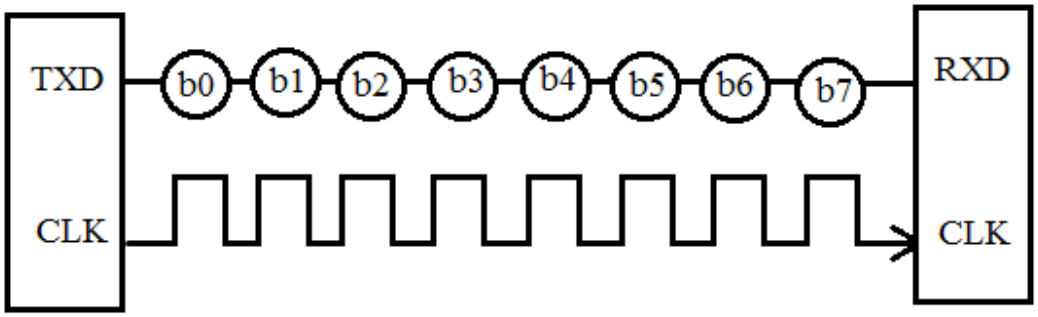


Fonte: (cacpnrj, 2020)

As principais características da comunicação *Serial* são:

- Apenas um caminho é necessário para transmissão dos dados.
- É necessário um *bit* de *start* e um *bit* de *stop* para indicar o início e fim da transmissão, além de um *bit* de paridade para verificação de erros.
- Transmissão 8 vezes mais lenta que a *Paralela* por enviar apenas um *bit* por sinal de *clock*.

Figura 8 – Comunicação Serial



Fonte: (cacpnrj, 2020)

Os sistemas de transmissão digital *duplex* são aqueles onde há, o envio e recebimento de informações por ambos os dispositivos. Ele ainda é dividido em sistemas:

- ***full-duplex*:** Envio e recebimento de dados no mesmo instante por ambos os dispositivos. Esse tipo é utilizado em comunicações *peer-to-peer* e *Serial Peripheral Interface* / Interface periférica serial (SPI).
- ***half-duplex*:** nesse tipo, apenas é possível a troca de dados em uma direção por vez,

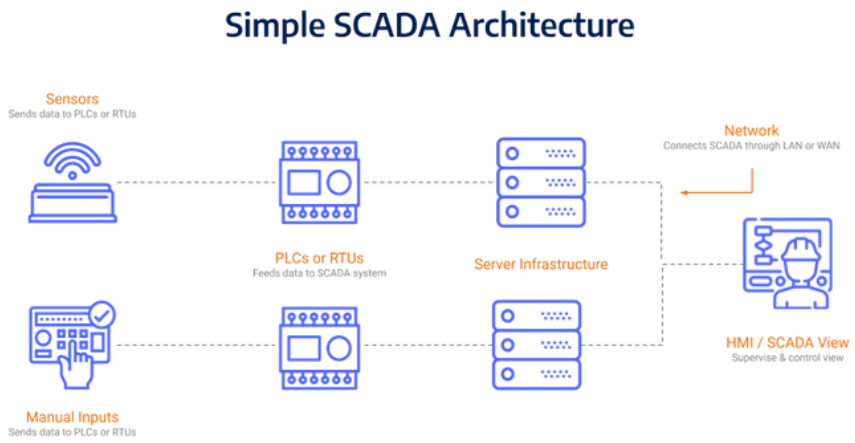
devendo esperar a troca de direção para mudar o sentido da transmissão. Esse tipo é utilizado em redes Mestre/Escravo e *Inter-Integrated Circuit* / Circuito Inter-Integrado (I^2C).

3.1.2 Comunicação via rede

Ao longo dos anos, e com a evolução e crescimento das indústrias, a interconexão entre os sistemas e máquinas foi se tornando uma demanda cada vez maior. Contudo, os modelos apresentados anteriormente de Comunicação Serial e Comunicação Paralela, mostraram-se vulneráveis à características físicas (distância) e eletromagnéticas (interferência de rádio e campos magnéticos).

Os supervisórios são compostos de sistemas conectados que conversam e enviam informações desde os sensores da figura 9, através de um CLP ou *Remote Terminal Unit* / Unidade Terminal Remota (RTU), passando pelos servidores que recebem os dados e chegam finalmente à tela do operador ou supervisor através de um IHM.

Figura 9 – Arquitetura simplificada de um sistema SCADA



Fonte: (Solis PLC, 2021)

Ao focarmos a atenção apenas no processo supervisório realizado pelo SCADA, é necessário uma organização no transporte de informações desde o chão de fábrica até o centro de controle de informação do sistema. Esse conjunto de regras e padrões pré estabelecidos configuram um protocolo de comunicação.

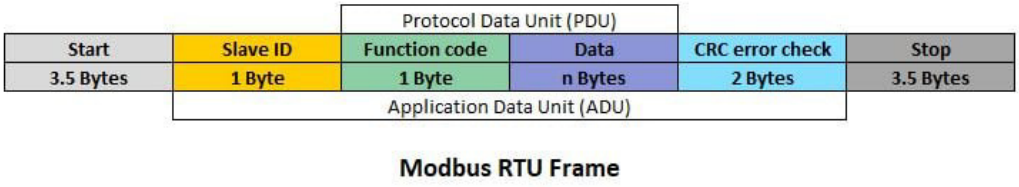
Algumas empresas foram desenvolvendo seus próprios protocolos, enquanto que outras utilizavam protocolos em código aberto, ou seja, qualquer pessoa pode utilizar. Dentre os protocolos que surgiram pode-se citar o *EtherNet/IP* que permite a interligação dos equi-

pamentos sem a necessidade de dispositivos intermediários. Outros popularizaram devido à sua simplicidade ou velocidade de transmissão. Exemplos destes são o *Fieldbus*, *Profibus* e, o utilizado na aplicação proposta, *Modbus*.

Criado pela *Modicom*, o protocolo *Modbus* consiste em um conjunto de regras *open source* para Comunicação Serial. Ele é baseado em uma arquitetura Mestre/Escravo, onde o requerente da informação é definido como o Mestre, enquanto que o requerido é denominado de Escravo.

O próximo passo se baseia na comunicação de dispositivos *Modbus* na rede local de comunicação das indústrias e foi no primeiro momento realizado pelo protocolo *Modbus/RTU* que utiliza geralmente o padrão de Comunicação Serial *RS-485*.

Figura 10 – Formato das mensagens *Modbus/RTU*



Fonte: (pico Technology, 2021)

Diferente dos métodos de transmissão digitais apresentados, o *Modbus/RTU* é um protocolo de comunicação de rede. Durante a comunicação entre dois dispositivos, uma mensagem *Modbus* deve ser enviada seguindo o padrão apresentado na figura 10, onde:

- ***Start* e *Stop*:** indicações para início e fim da transmissão de *frames* via rede.
- ***Slave ID*:** Identificação de 1 *byte* do Escravo referente à mensagem. Pode assumir valores entre 1 e 247, sendo os valores restantes dedicados à funções específicas, como o endereço 0 utilizado para mensagens de *broadcast*.
- ***Function Code*:** Número da função tabelada, podendo assumir valores entre 1 a 255. Cada funções possui um objetivo específico e dita o formato das informações do campo *Data* do *frame*.
- ***Data*:** O número de *byte*s depende da função definida na mensagem, podendo enviar desde um único *bit*, até valores de texto em registradores.
- ***CRC error check*:** Checagem de erro na transmissão de mensagens. É um código de 16 *bit*s gerado pelo remente da mensagem com base em seu conteúdo e que deve ser exatamente igual ao gerado pelo destinatário ao receber a mensagem.

3.2 Modelo OSI

No final dos anos 70, a *International Organization for Standardization* / Organização Internacional para Padronização (ISO) criou um modelo padronizado de organização das camadas da *internet*. Esse modelo veio a ser conhecido como modelo *Open Systems Interconnection* / Interconexão de Sistemas Abertos (OSI)

> O modelo OSI tomou forma quando os protocolos que iriam se tornar protocolos da Internet estavam em sua infância e eram um dos muitos conjuntos em desenvolvimento; na verdade, os inventores do modelo OSI original provavelmente não tinham a Internet em mente ao criá-lo. (KUROSE; ROSS, 2013, p. 39)

O modelo é constituído de sete camadas com funções bem definidas com o objetivo de enviar e receber informações entre dois hospedeiros, isto é, dois dispositivos diferentes.

1. **Aplicação:** Onde residem as aplicações e seus protocolos (HTTP - web, SMTP - email e o FTP - transferência de arquivos). Os sistemas de DNS também operam nessa camada. O pacote de informação é chamada da *Mensagem*.
2. **Apresentação :** Prover serviços que permitam que as APIs de comunicação interpretem os dados trocados. Compressão e codificação de dados, assim como a descrição de dados
3. **Sessão :** É responsável pela delimitação e sincronização da troca de dados.
4. **Transporte :** Carrega as mensagens entre o cliente o servidor. Os protocolos dessa camada são o TCP e o *User Datagram Protocol* / Protocolo de datagrama do usuário (UDP). O pacote dessa camada é chamado de *Segmento*.
5. **Rede :** Recebe um segmento e um endereço de destino e então provê o serviço de entrega do segmento à camada de transporte no hospedeiro de destino. O protocolo IP opera nessa camada. Realiza a movimentação entre os hospedeiros de pacotes da camada de rede, conhecidos como *Datagramas*.
6. **Enlace :** Roteia um datagrama por meio de uma série de roteadores entre origem e destino levando um pacote entre os nós de comunicação. Os pacotes são chamados de *Quadros*.
7. **Físico :** Movimenta os bits individuais dos quadros de um nó para o próximo.

3.3 IoT e Redes 5G

Em um futuro onde o anseio pela integração de todos os dispositivos caseiros, desde o controle dos itens em sua geladeira até a própria cafeteira utilizada diariamente, estejam

conectados de modo a manter todas as informações da residência atualizadas, é necessária uma forma de comunicação extremamente eficiente.

Os principais pontos limitantes, quando se fala em aplicações embarcadas IoT, convergem sempre para a autonomia de baterias e o alcance de conexão. Por se tratarem geralmente de sistemas com alimentação isolada da rede elétrica, o consumo de energia deve ser gerenciado da melhor forma. A maior parcela desse consumo é ligado ao tipo de comunicação implementado. Sistemas como *bluetooth* ou *RFID* tomam a dianteira por apresentarem um consumo extremamente baixo. Contudo, possuem o sacrifício do alcance de conexão, que por vezes é um fator decisivo dependendo da aplicação remota. Por outro lado, a implementação de comunicação *wifi*, apesar do maior alcance, exige um esforço computacional maior, e por consequência, maior consumo de energia.

> Os sistemas 1G-4G anteriores contam com o chamado acesso múltiplo ortogonal. Tome como exemplo o acesso múltiplo por divisão de tempo usado pelo 2G: dividimos um segundo em vários intervalos de tempo com curta duração. Em seguida, alocamos um determinado intervalo de tempo para cada usuário, e um usuário não pode acessar um canal alocado para outros. Esse acesso múltiplo ortogonal será difícil de suportar para futuras aplicações de IoT. Teremos muitos dispositivos e teremos que alocar slots de tempo dedicados a cada um deles. Mas, no final das contas, isso é um luxo que não podemos pagar, uma vez que o número de slots de tempo e recursos de largura de banda disponíveis serão insuficientes. É por isso que o acesso ortogonal múltiplo não funcionará para o 5G. (Zhiguo Ding, 2015)

Com os conceitos de topologias de rede bem estruturados, é possível iniciar a criação da topologia de rede ideal para a aplicação proposta. Nesse sentido, a evolução da tecnologia microprocessada permite uma atualização de sistemas utilizados na indústria.

Um desses exemplos é a placa de desenvolvimento Wemos D1 mini que, além de apresentar excelente custo-benefício, consegue aplicar conexões *wireless* de uma maneira amigável ao usuário. Vale ressaltar que atualmente essas soluções são implementadas em sua grande maioria apenas com suporte ao IPv4. Com a iminente tomada das comunicações pelo IPv6, algumas soluções e adaptações intermediárias surgirão para manter os sistemas IoT já existentes em funcionamento.

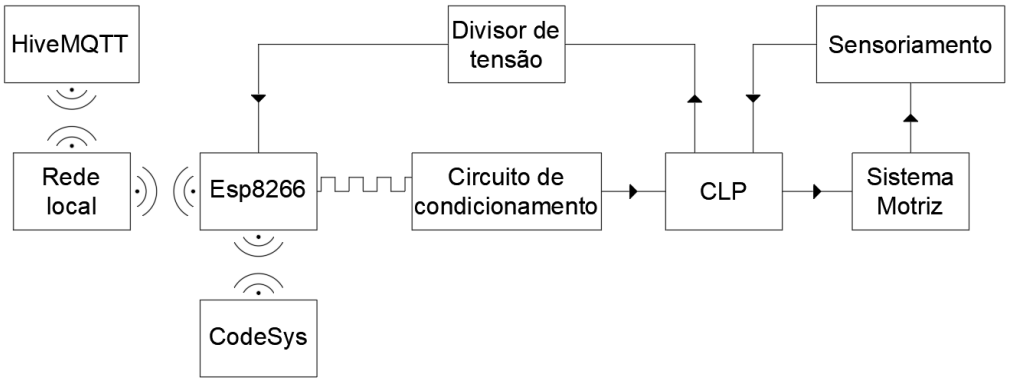
4 PROJETO *AD HOC* DE CONTROLE E COMUNICAÇÃO

O estudo de caso aplicado trata-se de uma implementação de comunicação sem fio para o acionamento e controle de um motor DC com caixa de redução. Essa aplicação serve de base para a inclusão de uma placa de desenvolvimento Wemos D1 mini, e seu respectivo circuito de condicionamento, em um sistema automático baseado em CLP existente no Laboratório de Máquinas Motrizes da UFC, *campus* do Pici.

O esquema mostrado na figura 11 descreve cada etapa dos protocolos e comunicação envolvidos ao microcontrolador responsável por acionar o motor. Esse diagrama é o modelo completo em relação àquele apresentado na seção 2.4. Nessa aplicação, o CLP funciona como elemento concentrador de informações, servindo como interface de entrada e saída de dados, sem alterar o funcionamento lógico aplicado pelos outros componentes. Esse tipo de *bypass* lógico é descrito na figura 5.

Com a implementação descrita, é possível controlar o comportamento e referência do motor em questão de forma remota, através dos protocolos de comunicação descritos abaixo, possibilitando estudos de controladores diferentes e supervisão remota da planta adotada.

Figura 11 – Diagrama completo do sistema



Fonte: Autoria própria

Uma vez que o operador envia a referência desejada no terminal onde o *client* MQTT está sendo executado, este envia o valor em formato de texto para o *broker* que por sua vez o distribui para os clientes inscritos no tópico em questão.

Ao receber a referência via protocolo MQTT, o microcontrolador ESP8266, realiza um pré processamento da mensagem recebida, de modo a determinar qual o objetivo específico desta. Para que a placa entenda a mensagem como uma referência a ser seguida pelo controlador, é necessário que o primeiro caractere da cadeira de texto seja a letra "**R**". Tal critério é importante

para que novas funções possam ser desenvolvidas e aplicadas na placa sem a necessidade da implementação de lógicas elaboradas ou grandes alterações no código, bastando apenas, incluir um critério de mensagem na função de *callback* do MQTT.

Com a referência corretamente recebida, esta é enviada para o *software CodeSys* via protocolo *Modbus* para que sejam realizados os cálculos do controlador e enviados de volta à ESP8266.

O sinal de referência é inserido na saída de *Pulse-width modulation* / Modulação por largura de pulso (PWM) de 0 à 3,3V que então passa por um circuito integrador e de elevação para o intervalo de 0 a 10V de modo que possa ser lido corretamente pelo CLP, encarregado de acionar o sistema motriz em questão. Essa interface é o chamado circuito de condicionamento.

Acoplado ao CLP, um sistema de sensoriamento é utilizado para realizar o fechamento da malha de controle. Esta medição passa pelo CLP, e é então enviado para a ESP8266 através de um divisor de tensão, sendo possível assim, seu envio via *Modbus* para a realimentação do controlador.

4.1 O protocolo Modbus/TCP

O *Modbus/TCP* é uma evolução do antigo protocolo *Modbus/RTU* possuindo características de comunicação através de interfaces físicas de rede. No contexto dos requerimentos realizados entre um dispositivo Mestre e seus Escravos, a estrutura de solicitação e resposta é semelhante com a arquitetura Cliente/Servidor utilizada em servidores *Hyper Text Transport Protocol* (HTTP).

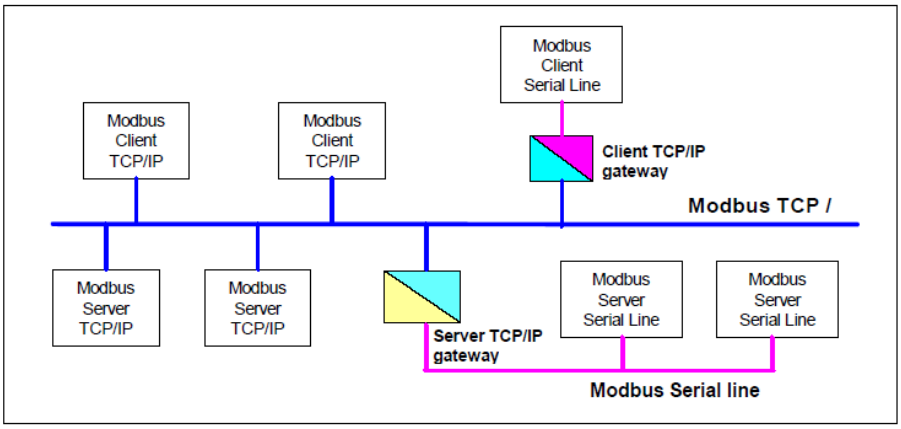
> A rede mestre/escravo é aquela em que um controlador -mestre controla todas as comunicações vindas de outros controladores. Seu funcionamento pode ser resumido da seguinte maneira:
>
> - O controlador-mestre envia os dados para os controladores- escravos.
> - Quando o mestre necessitar dos dados de um escravo, ele solicitará (endereço) ao escravo e esperará por uma resposta.
> - Nenhuma comunicação ocorre sem o mestre iniciá-la.
>
> (PETRUZELLA, 2014, p. 306) modificado.

Nesse tipo de comunicação, o cliente *Modbus* inicia uma requisição criando uma unidade de aplicação de dados conhecido como **ADU** (ORGANIZATION, 2006, p. 9). Essas solicitações são enviadas para o servidor *Modbus* que possui endereço e porta (que por padrão é a porta 502) conhecidos. A resposta à essas solicitações é então enviada através da camada de

transporte para o endereço e porta relacionados ao *socket* que foi criado no estabelecimento da comunicação (KUROSE; ROSS, 2013).

> A função de um servidor MODBUS é providenciar acesso aos objetos e serviços da aplicação aos clientes remotos MODBUS.
>
> Ele também deve mapear os objetos da aplicação entre leitura ou escrita de modo a obter ou alterar os atributos dos objetos.
>
> O servidor MODBUS tem que analisar uma requisição MODBUS recebida, processar a ação requerida e então enviar uma resposta MODBUS.
>
> (ORGANIZATION, 2006)

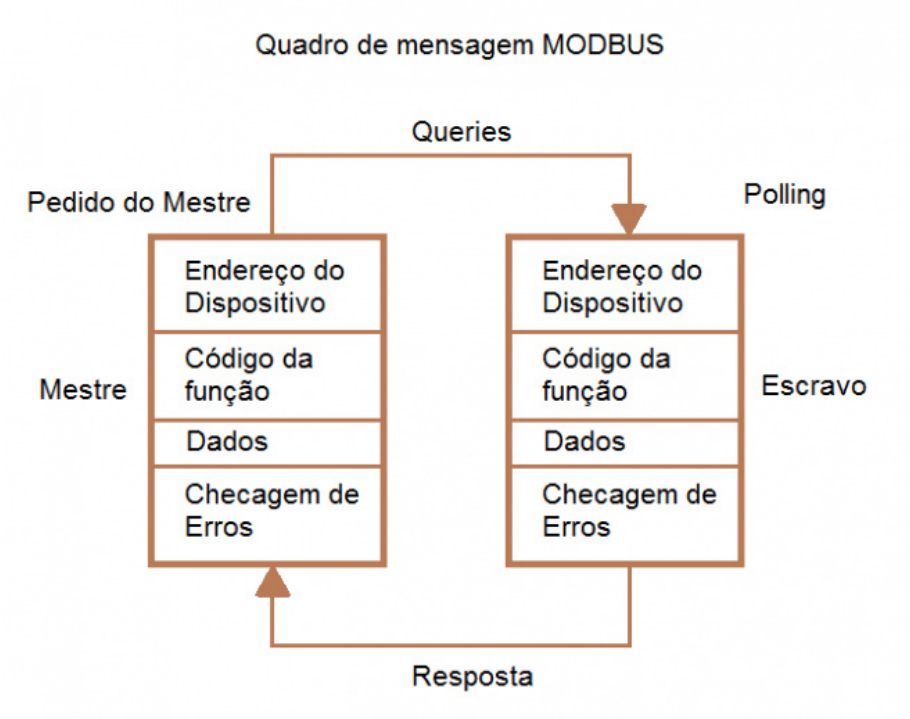
Figura 12 – Topologia do protocolo Modbus/TCP



Fonte: (ORGANIZATION, 2006)

A topologia de uma rede de equipamentos *Modbus* é definida na figura 12, onde os equipamentos se comunicam diretamente com o roteador ou com o *switch* da rede local, que por sua vez envia as informações para os supervisórios em computadores ou dispositivos remotos. Assim, como um cliente HTTP, o mestre *Modbus* envia requisições ao servidor definindo o tipo de solicitação e alguns dados na solicitação.

Figura 13 – Descrição dos quadros *Modbus*



Fonte: (Carlos Márcio Freitas, 2014)

A figura 13 descreve as informações enviadas nas solicitações *Modbus* pelo mestre, sendo o endereço do dispositivo da requisição conhecido, assim como o identificador a ele atribuído. O código da função determina o tipo de ação que deverá ser executada, sendo estas tabeladas e descritas na seção 4.1.2. No cabeçalho de dados e seu respectivo tipo é definido segundo a solicitação realizada e está relacionado com o tipo de registrador que se deseja realizar a leitura ou escrita. A descrição destes registradores e do endereçamento é descrito na seção 4.1.1 (ORGANIZATION, 2006, p. 9).

4.1.1 Registradores e endereçamento Modbus

Durante a comunicação com o protocolo, é necessário se referir aos endereços requisitados para leitura ou escrita. Esse endereçamento é padronizado para CLPs e segue a seguinte estrutura:

1. Utilização do símbolo de porcentagem (%).
2. Tipo de armazenamento, este podendo ser uma entrada (I), saída (Q) ou um espaço de memória (M).
3. Tamanho da informação. Esta é classificada quanto à quantidade de bits e é descrito na Tabela 1

4. Indexar registradores semelhantes (mais à esquerda), e se necessário, especificação da posição dos dados (para se referir à um bit específico)(mais à direita).

Tabela 1 – Descrição de tipos de dados em endereços de CLPs

<data-size>	Common Name	Number of Bits	Elementary Data Types
X	Bit	1	BOOL
B	Byte	8	BYTE, SINT, USINT
W	Word	16	WORD, INT, UINT
D	Double word	32	DWORD, DINT, UDINT, FLOAT
L	Long word	64	LWORD, LINT, ULINT, DOUBLE

Fonte: openPLC project (Thiago Alves, 2021b) Adaptado.

Para o funcionamento do código escrito para o ESP8266, os únicos registradores necessários são demonstrados na Tabela 2 e servem também como forma de exemplo de endereçamento. Vale lembrar que a quantidade máxima de registradores, sejam estes de qualquer tipo, é determinada na biblioteca *modbus.h*, conforme a seção 4.4.4.

Tabela 2 – Registros utilizados da ESP8266

Registrador	Tipo	Dado	Descrição
%IW1	Entrada	Word	Leitura de velocidade ou referência
%QW1	Saída	Word	Saída do PWM
%IX0.0	Entrada	Bit	*Flag new_ref*
%QX0.1	Saída	Bit	*Flag new_ref_ack*

Fonte: Autoria própria

4.1.2 Funções Modbus

O protocolo utiliza uma série de funções pré-estabelecidas para determinar o tipo de solicitação que o mestre deseja efetuar. Essas funções são divididas nas funções de leitura e/ou escrita, e para cada uma é determinado o tipo de variável relacionada. A Tabela 3 descreve todas as funções *Modbus* determinadas na criação do protocolo.

Tabela 3 – Código e descrição das funções *Modbus*

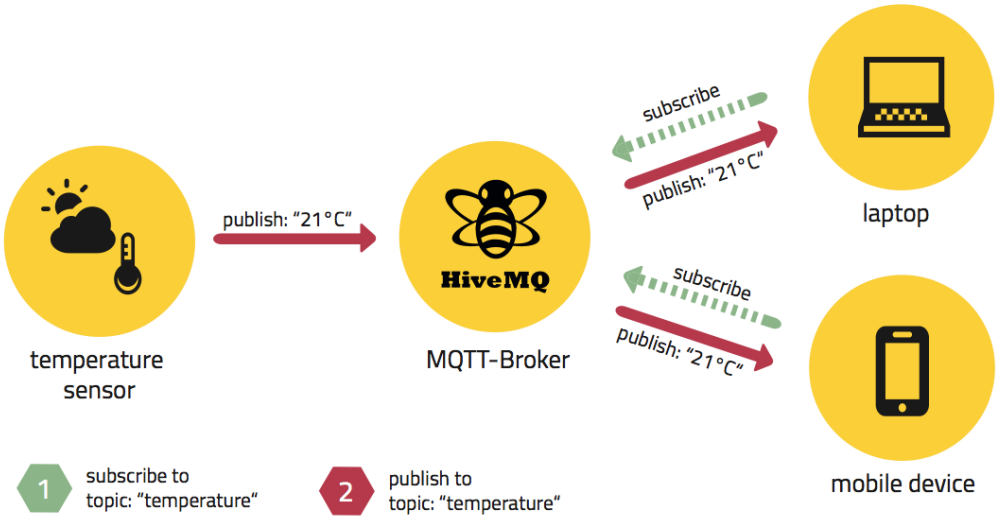
Código da função	Descrição
1	Leitura de bloco de bits do tipo coil(saída discreta).
2	Leitura de bloco de bits do tipo entradas discretas.
3	Leitura de bloco de registradores do tipo holding.
4	Leitura de bloco de registradores do tipo input.
5	Escrita em um único bit do tipo coil(saída discreta).
6	Escrita em um único registrador do tipo holding.
7	Ler o conteúdo de 8 estados de exceção.
8	Prover uma série de testes para verificação da comunicação e erro internos.
11	Modbus: Obter o contador de eventos.
12	Modbus: Obter um relatório de eventos.
15	Escrita em bloco de bits do tipo coil(saída discreta).
16	Escrita em bloco de registradores do tipo holding.
17	Ler algumas informações do dispositivo.
20	Ler informações de um arquivo.
21	Escrever informações em um arquivo.
22	Modificar o conteúdo de registradores de espera através de operações lógicas.
23	Combina ler e escrever em registradores numa única transação.
24	Ler o conteúdo da fila FIFO de registradores.
43	Identificação do modelo do dispositivo.

Fonte: Embarcados(Carlos Márcio Freitas, 2014) Adaptado.

4.2 O protocolo MQTT

O protocolo MQTT utiliza de um sistema de inscrição e publicação em tópicos únicos para gerenciar o envio de informações por parte de dispositivos, e quais sistemas finais desejam receber esses dados para tratamento e supervisão. É utilizado em indústrias automotivas, manufaturas, telecomunicação, etc. Esse protocolo utiliza uma largura de banda reduzida e é destinado em aplicações de IoT. A figura 14 ilustra um esquema simples de comunicação MQTT com a utilização do *broker* da *HiveMQ* (MQTT Org, 2022).

Figura 14 – Dinâmica de um *broker* MQTT



Fonte: Eclipse.org MQTT 101(Christian Götz, 2014)

Inicialmente, tanto o computador quanto o dispositivo móvel conectam-se ao *broker* e realizam uma inscrição no tópico "*temperature*". Ao realizar a atualização dos registros de leitura do sensor de temperatura, a informação é publicada no tópico anteriormente inscrito e é enviada para o servidor responsável. A informação é então repassada para todos os dispositivos inscritos no tópico em questão e que estejam *online*.

Tabela 4 – Detalhes de configuração do MQTT na ESP8266.

Publish	LMT01Equip
Subscribe	LMT01Server
broker	broker.hivemq.com
Porta	1883

Fonte: Autoria própria

Caso algum dispositivo esteja *offline*, a formação de um *buffer* de fila de espera é realizada e a informação posteriormente repassada. Os detalhes para a comunicação com o *broker* MQTT via TCP foram obtidos no endereço oficial da *HiveMQ*: https://www.hivemq.com/public-mqtt-broker/

4.3 Linguagem Ladder

A linguagem de desenvolvimento de CLPs utilizada no desenvolvimento do algorítimo utiliza uma arquitetura em formato de linhas horizontais com o auxílio visual de contatos e

bobinas elétricas para a criação da lógica necessária.

O código desenvolvido no *ladder* do *Codesys* é composto de 3 pequenos grupos funcionais conforme descrito abaixo e se encontra no Apêndice A

1. Recebimento e confirmação da *flag new_ref*. A confirmação de recebimento é de suma importância e deve ser enviada para o ESP8266 para que este volte a *resetar* a *flag* e passe a enviar novamente o valor da velocidade.
2. Multiplexação da leitura do registrador %IW1. Responsável por determinar se o valor lido pela função *Modbus* 4 será passado para a referência do controlador ou para a leitura de velocidade.
3. Bloco do PID e escrita do registrador %QW1.

Algoritmo 1: Algorítmo do *Ladder*

input : ref: int, RPM: int, %IX0.0: bool,

output : PWM: int, %QX0.1: bool

```
1  while True do
2      new_ref=%IX0.0;
3      if new_ref == True then
4          new_ref_ack=True;
5          Atualiza a referência ref;
6          Envia a confirmação new_ref_ack;
7          %QX0.1=new_ref_ack;
8      else
9          Recebe a velocidade RPM;
10         Calcula a saída da planta em malha fechada;
11         Envia o novo sinal de PWM;
12     end
13 end
```

O algoritmo 1 descreve a lógica de funcionamento do código mestre do protocolo escrito. Ao receber as respostas das requisições *Modbus*, o *CodeSys* realiza a leitura do *bit* 0 do registrador %IX0.0 (conforme descritos os padrões de registradores *Modbus* na seção 4.1.2) e atribui este *booleano* à variável *new_ref*, sendo esta responsável pela multiplexação do valor lido no registrador %IW1 entre as variáveis *leitura_A0* e *ref*.

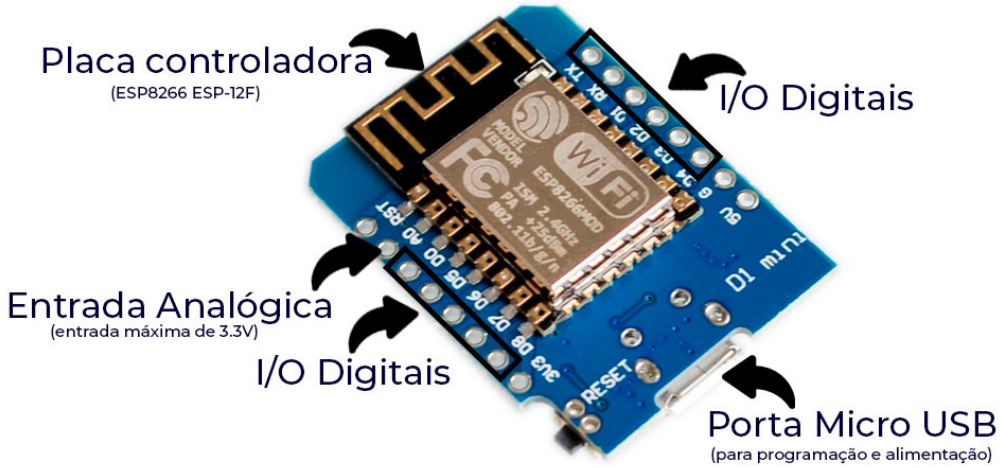
Contudo, o valor lido pela função 4 do protocolo *Modbus* é do tipo *WORD* e portanto, deve ser convertida para o tipo *REAL* antes de ser inserida no bloco PID. Da mesma forma, uma

vez que a saída do bloco de controle PID_0.Y trabalha com valores reais, esta deve ser convertida novamente no tipo *WORD* antes de ser movida para o registrador %QW1 e enviada através da função 16.

4.4 O microprocessador Esp8266

A placa de desenvolvimento Wemos D1 mini é baseada no microprocessador, de baixo consumo da *ESPRESSIF*, ESP8266X com uma arquitetura RISC de 32 bits que pode chegar a uma frequência máxima de 160 MHz. A figura 15 descreve as partes fundamentais do corpo físico da placa, cujo tamanho é um fator positivo para implementações IoT.

Figura 15 – Wemos D1 mini



Fonte: Blog Master Walker Shop(Greici Oliveira, 2020)

Dentre as características físicas e construtivas da placa de desenvolvimento, as que mais se destacam estão descritas abaixo.

- Nível lógico de 3.3V.
- 8 pinos digitais.
- PWM de 10 bits.
- 1 *Analogic Digital Converter* / Conversor analógico digital (ADC) de 10 bits.
- WiFi 802.11 b/g/n.
- 4MB de memória *flash*.
- Frequência de 80 a 160 MHz.

– SPI, I^2C e *Universal Asynchronous Receiver/Transmitter* / Transmissor/Receptor universal assíncrono (UART).

De posse dessas informações uma breve análise é suficiente para concluir que a Wemos D1 mini configura uma opção a baixo custo em comparação com placas mais simples como as baseadas em Arduíno e as alternativas mais avançadas como o NodeMCU ESP32, modelo também produzido pela *ESPRESSIF*. A tabela 5 faz uma breve comparação das características gerais das opções apresentadas.

Tabela 5 – Características das placas Arduino Nano, Wemos D1 mini e NodeMCU ESP32

Placa de desenvolvimento	Arduino Mega	Wemos D1 Mini	Node MCU ESP32
Microprocessador	ATmega328	ESP8266X	ESP32
Nível lógico	5V	3.3V	3.3V
Portas digitais	22	8	36
ADC	8	1x 10 bits	18x 12 bits
DAC	0	0	2x 8 bits
PWM	6x 8 bits	1x 10 bits	16x 16 bits
Frequência	16 MHz	80 MHz	160 MHz
Wifi	Não	Sim	Sim
Bluetooth	Não	Não	Sim
Memória flash	32 kb	4 MB	4 MB

Fonte: Autoria própria

O *firmware* desenvolvido está modularizado de acordo com funções específicas, sendo estes:

– *main.ino*. Programa principal. Responsável por unir as bibliotecas desenvolvidas e realizar a configuração e *loop* principais.
– *wifi.h*. Definição das variáveis de rede e funções de inicialização, reconexão e comunicação da rede.
– *MQTT.h*. Configurações do *broker* MQTT e do IO de mensagens.
– *Modbus.h*. Tratativa dos buffers e mensagens recebidas pelo protocolo.
– *codesys.h*. Funções de atualização dos registros Modbus para serem enviados e lidos.
– *encoder.h* Biblioteca auxiliar para a leitura da velocidade do motor.

4.4.1 main.ino

O *firmware* da aplicação está subdividido em três partes principais, tais quais todo programa principal desenvolvido para ser compilado e programado via IDE do Arduíno.

4.4.1.1 Definições

A importação das bibliotecas necessárias para o funcionamento do código de maneira mais limpa e eficiente é realizada logo no início para garantir a integridade dos objetos e funções desenvolvidas posteriormente.

São criados quatro objetos principais responsáveis por todos os protocolos de comunicação necessários, sendo eles:

1. *espClient* da classe *WiFiClient*, criação do objeto de cliente TCP.
2. *MQTT* da classe *PubSubClient*, objeto responsável por manter aberta a comunicação com o *broker* MQTT. Recebe como parâmetro o objeto de cliente TCP *espClient*.
3. *server* da classe *WiFiServer*, instância de um servidor aberto na porta 502 no aguardo de comunicações via protocolo *Modbus*.
4. *motor* da classe *encoder*, objeto de controle e leitura dos valores de velocidade oriundas do *encoder* e que serão enviadas pelo *Modbus* e MQTT.

4.4.1.2 void Setup

Chamada das funções de inicialização dos protocolos de comunicação utilizados e dos pinos da Wemos D1 mini com base nas máscaras informadas na biblioteca *codesys.h*.

- Serial.begin(115200): Inicialização da comunicação com o monitor serial utilizado para monitoramento das informações.
- pinConfig(): configuração dos pinos das máscaras como entradas ou saídas digitais ou analógicas.
- conectarWiFi(): estabelece comunicação com a rede local seguindo os parâmetros informados na seção 4.4.2
- server.begin(): inicialização do servidor *Modbus* na porta 502.
- initMQTT(): estabelecimento da comunicação com o *broker* MQTT.

4.4.1.3 void Loop

O código principal depende do estabelecimento da comunicação *Modbus* por um cliente TCP na porta 502. Com a criação de uma conexão, o programa aguarda até que haja uma requisição por parte do cliente a ser recebida pelo *buffer*. Após a leitura do *buffer Modbus*, os registradores de entrada e saída são devidamente atualizados e então transmitidos segundo as

respectivas requisições recebidas.

A comunicação MQTT é então mantida aberta por meio de uma conexão TCP persistente e que é utilizada para o envio de informações a cada período "**T**"de tempo.

4.4.2 wifi.h

Nesta biblioteca são fornecidas as informações de rede para conexão com o roteador local com IP fixo. A fim de manter um registro em sistema local e documentação a respeito das plantas, é necessário fixar o IP da placa para evitar o problema de descobrir se o IP da planta mudou ou não durante as atividades futuras, eliminando um espaço para potencial falha.

Deste modo, as informações requisitadas são:

- **SSID e PASSWORD**: Nome e senha da rede locais as quais se deseja conectar.
- **IP**: Definição do IP que se deseja fixar. A escolha do IP fixo deve é claro respeitar as regras de endereçamento da máscara de rede da conexão local.
- **DNS**: Endereço do servidor DNS da conexão deve ser definido segundo a rede local.
- ***gateway***: O *Gateway* padrão do roteador local.
- ***subnet***: A máscara de sub-rede (Máscara de subrede) definida no roteador local.

Como sugestão, é possível descobrir estes 3 últimos ao utilizar o comando "*ipconfig /all*"no *prompt* de comando do *Windows*.

Também são definidos os parâmetros para comunicação TCP básico com um servidor *python*. Sendo assim, é necessário conhecer previamente o IP do *host* servidor e a porta aberta para estabelecimento de comunicação. Como regra, o número da porta para comunicações em serviços que não sejam "bem conhecidos"deve ser estar fora do intervalo de 0 a 1023 (KUROSE; ROSS, 2013).

4.4.3 MQTT.h

Assim como nas definições básicas de rede da seção 4.4.2, nas configurações básicas do MQTT são inseridas as informações já apresentadas na Tabela 4. Também é nessa biblioteca que são definidas dois tipos de funções.

1. Funções de criação e manutenção.
2. Funções de tratamento da informação.

As funções de criação e manutenção são responsáveis por iniciar a comunicação com o *broker* MQTT e por manter a conexão ativa, além de escutar o recebimento e gerenciar o

envio de mensagens.

Já as funções de tratamento da informação são criadas para decidir o que fazer com a mensagem MQTT recebida e são chamadas durante o *callback* da conexão. Exemplos desse tipo de função são a função de *nova_referência* e a função *reiniciar_esp*.

4.4.4 modbus.h

O módulo é dedicado exclusivamente para a tratativa de envio, recebimento e análise de erros no *buffer* de mensagem a ser executada. Deste modo, as funções do protocolo estão mapeadas e definidas tanto em tamanho de janela de mensagem a ser enviada, como em ordenamento e valores dos bits da mensagem de 2 *bytes*.

O código fonte pode ser obtido no endereço do projeto *openPLC* (Thiago Alves, 2021a), assim como as funções originais de atualização dos registradores de entrada e saída que se encontram na seção 4.4.5.

4.4.5 codesys.h

O conjunto responsável pela atualização de registradores foi adaptado do projeto *openPLC* para servir às limitações e dinâmicas do estudo de caso. São definidos os pinos digitais da ESP, assim como as máscaras de pinos para as funções de entradas e saídas digitais e analógicas. No programa implementado foram definidos apenas os registradores das funções 2, 4, 15 e 16.

A principal função deste módulo é a *updateIO* que é composta por 4 estruturas de repetição *for*, uma para cada registrador definido. Partindo do código original, três pequenas mudanças foram realizadas para implementação adequada.

1. Na estrutura de repetição do registrador de entrada analógica da função 4, um conjunto condicional com base na *flag new_ref* foi implementada para definir se o valor a ser enviado será o de referência recebida ou o de velocidade calculada.
2. Teste condicional se o bit 1 do registrador de escrita digital da função 15 é verdadeiro. Nessa condição, a *flag new_ref* é atualizada como falsa. Essa condição é importante pois garante que a nova referência foi enviada e processada pelo protocolo, funcionando como um *handshaking* do protocolo TCP.
3. O bit zero do registrador de entradas digitais da função 2 é sempre atualizado com a *flag new_ref*, não devendo ser utilizado para outros fins.

4.4.6 encoder.h

O objetivo desse módulo é o de definição das funções e variáveis específicas da planta estudada. No estudo de caso em questão, as constantes do disco do *encoder*, variáveis de controle, interrupções e funções de leitura.

Com a definição da interrupção por mudança de estado lógico no pino digital da ESP, a cada vez que o foto interruptor for atravessado por um orifício do disco perfurado, um pulso digital é enviado e a função *ISR_encoder* é chamada, incrementando o parâmetro *count* do objeto da classe motor.

Durante o ciclo *Modbus*, na chamada da função 2 (Leitura de bloco de bits do tipo entradas discretas), o valor em RPM é calculado e a contagem é zerada novamente. A velocidade em RPM deve ser lida neste ponto, pois deve-se certificar que ao enviar a *flag* referente à uma nova referência, o valor de RPM seja enviado corretamente.

4.5 Eletrônica básica

Com a apresentação das características da Wemos D1 mini na seção 4.4, é possível definir os componentes e módulos necessários para a implementação da planta em questão em bancada.

Uma vez que é necessário o acionamento de um motor DC de eixo duplo, se faz necessário a inclusão de um *driver* de corrente para garantir o fornecimento de potência requerida pela planta. Nesse contexto, é comum a utilização de Ponte H que, para este projeto, empregou-se o módulo L298n, já que possui a característica de fornecer uma regulação de saída de 5V para alimentação do circuito de controle. A alimentação do módulo é fornecida por um conector Jack do tipo P4 a ser utilizado com uma fonte externa de 9V/600mA.

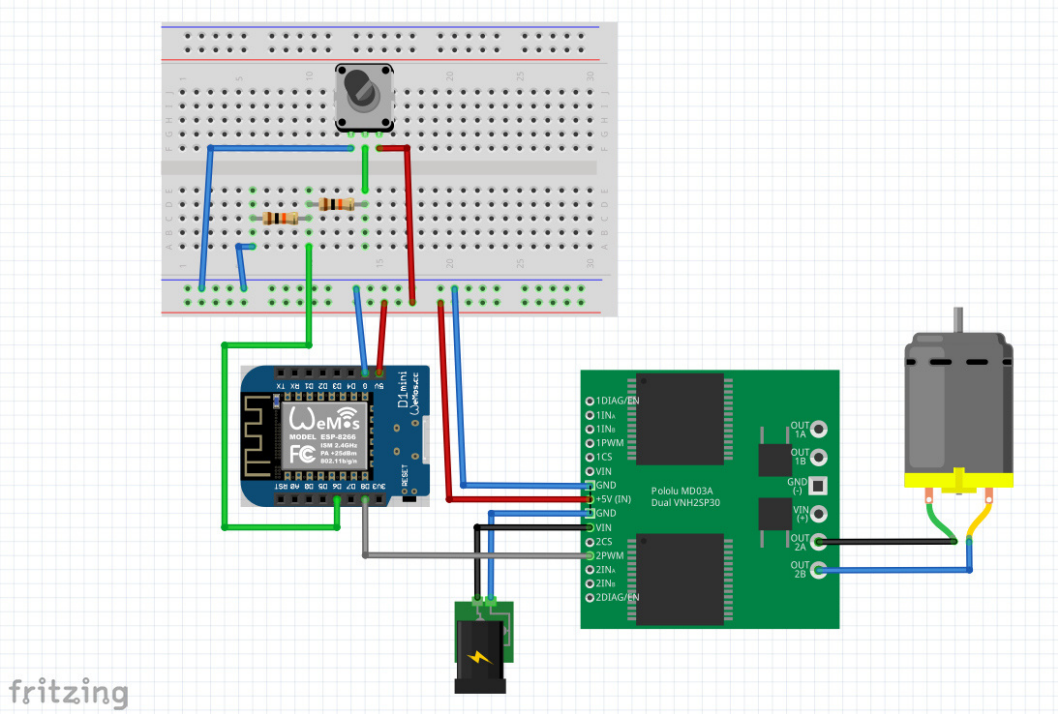
De modo a receber as informações de velocidade do motor, é utilizado um *Encoder* opto interruptor que ao perceber uma mudança no disco perfurado acoplado ao eixo do motor, envia pulsos lógicos para a Wemos D1 mini passando antes por um divisor de tensão, uma vez que a tensão de operação do módulo é 5V e a da ESP é 3,3V.

Os materiais utilizados para a montagem da planta foram:

- 1x Conector Jack para *plug* P4.
- 1x Fonte externa de 9V, 600mA com *plug* P4.
- 1x Módulo Ponte H L298n com regulação de tensão interna.

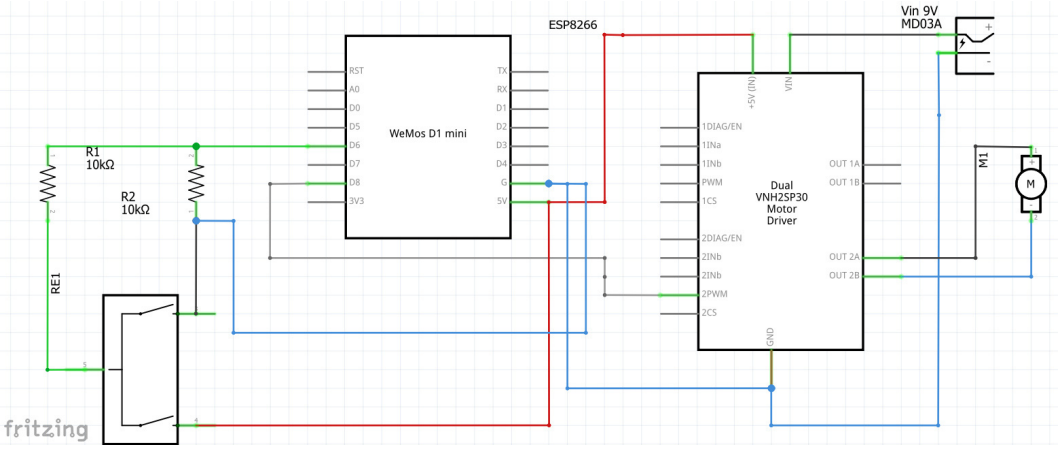
– 1x Motor DC 6V com caixa de redução, eixo duplo e roda.
– 1x Disco *encoder* de 20 furos.
– 1x Placa de desenvolvimento Wemos D1 mini baseado no microcontrolador ESP8266.
– 2x Resistores de 10kΩ 0,25W.
– 1x *protoboard*.
– *jumpers* variados.

Figura 16 – Diagrama de conexões da planta



Fonte: Autoria própria utilizando o *software Fritzing*

Figura 17 – Diagrama esquemático da planta



Fonte: Autoria própria utilizando o *software Fritzing*

As figuras 16 e 17 detalham as conexões físicas para reprodução em bancada do

estudo de caso escolhido onde o esquema de cores para os condutores de conexão são definidos abaixo.

- **Preto**. Tensão nominal de 9V para alimentação da Ponte H e dos motores.
- **Vermelho**. Tensão nominal de 5V para alimentação do ESP8266 e do módulo do *encoder*.
- **Azul**. Referência de alimentação do circuito.
- **Cinza**. Sinal de PWM enviado para a Ponte H.
- **Verde**. Sinal lógico nominal de 2.5V responsável envio da onda pulsante do *encoder* para o ESP.

4.6 Projeto de controle

Com o interesse de manter a planta seguindo a referência informada pelo usuário remoto via MQTT, foi projetado um controlador afim de garantir uma performance aceitável para seja dispensável um operador verificando se o sistema está dentro dos conformes.

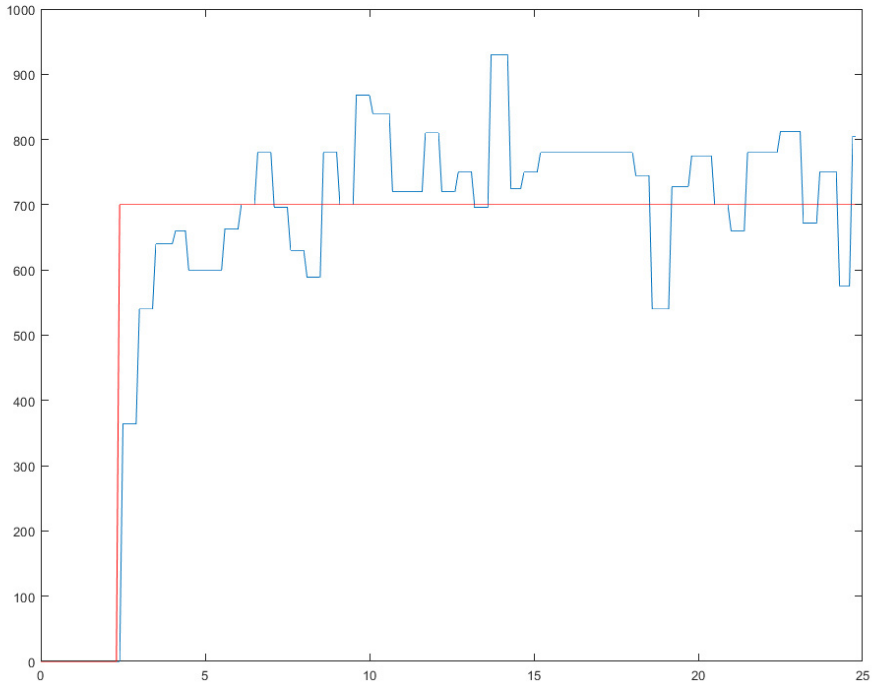
4.6.1 Identificação da planta

O primeiro passo foi a identificação da planta via comunicação Comunicação Serial com um Arduíno Uno. Para isso, um tempo de amostragem de 100ms foi utilizado com um algorítmo simples de amostragem na placa e coletada via monitor Serial.

Contudo, após uma breve análise, foi constatada a necessidade de incluir o comportamento e dinâmica da rede e processamento de pacotes na identificação. Para isso, um objeto *Trace* foi criado na aplicação do *Codesys* e então relacionada à variável de leitura da velocidade recebida pelo ESP8266.

Ao aplicar um degrau unitário com valor de 700 RPM no registrador %QW1 e com as leituras realizadas pelo *encoder*, a resposta ao degrau unitário foi obtida através da exportação do objeto *trace* do *codesys* e salvo como um arquivo *Comma separated values* / Valores separados por vírgula (CSV). Os dados foram então inseridos no *Matlab* para a discretização da planta e obtenção da figura 18.

Figura 18 – Resposta ao degrau unitário em malha aberta



Fonte: Autoria própria

4.6.2 Discretização da planta Gd(z)

Para a discretização da planta, os dados obtidos em 4.6.1 foram importados no *software Matlab*, cujo código se encontra no apêndice B. Com o cálculo aproximado da constante de tempo τ por meio da regra de 63% do valor nominal e com a utilização do *Matlab* o cálculo aproximado do ganho em malha aberta, a equação de transferência no tempo contínuo da planta é descrito na equação 4.1

$$G(s) = \frac{1.241}{0.21s + 1} \tag{4.1}$$

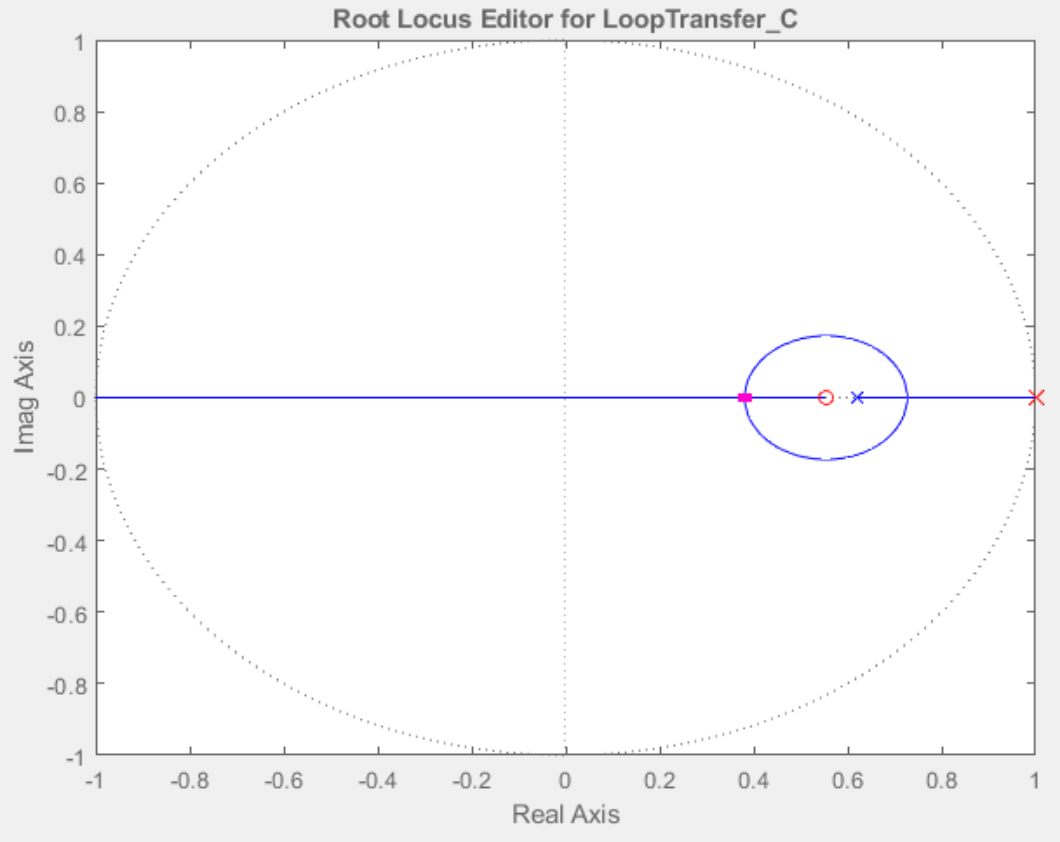
Com a equação de transferência em tempo contínuo obtida, a função *c2d* do *Matlab* foi utilizada para a discretização, encontrando a equação 4.2

$$Gd(z) = \frac{0.4703}{z - 0.6211} \tag{4.2}$$

4.6.3 Projeto do controlador sem filtro C(z)

Com a equação de transferência da planta devidamente identificada na equação 4.2, a ferramenta *Root Locus* do *Matlab* é chamada passando como argumento G_d.

Figura 19 – Lugar geométrico das raízes do controlador



Fonte: Autoria própria

Ao analisar a figura 19, adiciona-se um integrador ao controlador **C** de modo que, em regime permanente ($z \to 1$ na Função de Transferência em Malha Fechada) garante-se erro nulo no seguimento de referência do tipo degrau. Além disso, também é inserido um zero real no controlador no interior do círculo de raio unitário e no semi-plano direito (SPD) à esquerda do polo da planta. Dessa forma, tem-se que o zero do controlador não torna-se dominante em relação aos polos da planta, diminuindo o sobressinal. Em seguida, as raízes do controlador são ajustadas para permanecerem no plano real e convergindo de modo a manter a estabilidade em malha fechada.

$$C(z) = \frac{1.8274 * (z - 0.554)}{z - 1} \tag{4.3}$$

Desse modo, a equação no tempo discreto do controlador é definido pela equação

4.3. Contudo, de modo a manter a compatibilidade com o bloco de PID do *CodeSys*, é necessário calcular as componentes proporcional (K_p) e integral (K_i) da mesma. As equações 4.4 a 4.9 descrevem o passo a passo matemático de calculo das componentes.

$$C(z) = K_p + \frac{K_i * (z+1)}{z-1} \tag{4.4}$$

$$C(z) = \frac{(K_p + K_i) * z + (K_p - K - i)}{z-1} \tag{4.5}$$

$$K_p + K_i = 1.8274 \tag{4.6}$$

$$K_p - K_i = 1.8274 * 0.554 \tag{4.7}$$

$$K_p = \frac{1.8274 * (1+0.554)}{2} = 1.42 \tag{4.8}$$

$$K_i = 1.8274 - K_p = 0.407 \tag{4.9}$$

4.6.4 Fechamento da malha sem filtro

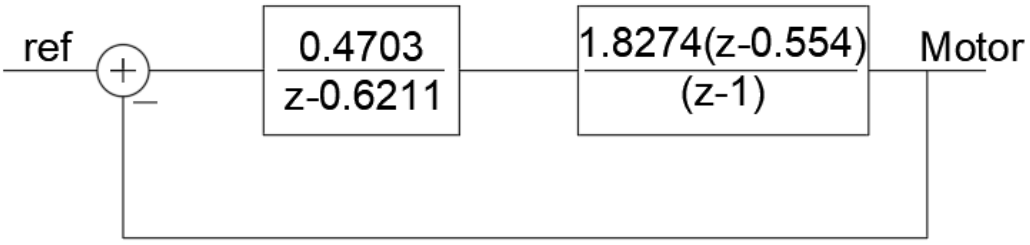
Com o projeto do controlador bem estruturado, o mesmo é colocado em série com a planta ***Gd(z)*** e em seguida é realizado o fechamento da malha com realimentação negativa conforme ilustrado na Figura 20.

A equação de transferência da planta em malha fechada ***Y(z)*** é então identificada segundo as equações 4.10 e 4.11.

$$Y(z) = \frac{Gd(z) * C(z)}{1 - Gd(z) * C(z)} \tag{4.10}$$

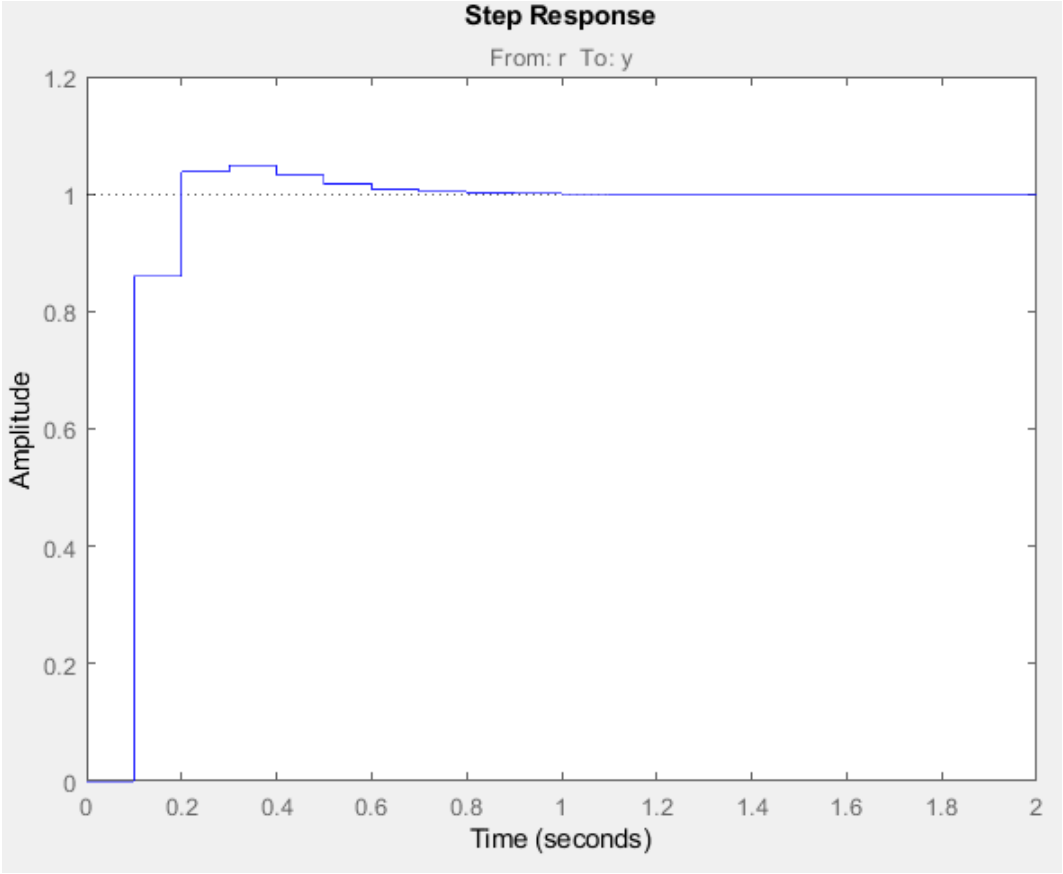
$$Y(z) = \frac{0.85939 * (z-0.554)}{(z-1) * (z-0.6211)} \tag{4.11}$$

Figura 20 – Diagrama da planta em malha fechada



Fonte: Autoria própria

Figura 21 – Resposta ao degrau unitário em malha fechada



Fonte: Autoria própria

O último passo é então o levantamento do comportamento da planta em malha fechada para um pulso unitário.

Observando a figura 21 é possível notar um *overshoot* de 4,9% mas que está dentro do limite de 5%. Esse *overshoot* é decorrente da proximidade entre o zero e o polo do controlador discreto conforme a figura 19, onde mesmo que as raízes do controlador estejam na extremidade esquerda, o tamanho do local das raízes causa essa ultrapassagem do valor de referência.

4.6.5 Projeto do filtro passa-baixa H(z)

Com o intuito de refinar os sinais de leitura realizadas pelo *Encoder*, um filtro discreto projetado para rejeitar altas frequências é projetado com o formato da equação 4.12.

$$H(z) = \frac{1-p}{z-p} \tag{4.12}$$

Uma vez que a planta entre em regime permanente, ou seja, com $z \to 1$, o ganho do filtro tende ao valor unitário conforme a equação 4.13

$$\lim_{z\to 1} H(z) = \frac{1-p}{1-p} = 1 \tag{4.13}$$

Tomando como base a equação de transferência discreta no domínio da frequência do filtro ***H(z)***, uma manipulação matemática é realizada para determinar a equação amostral do filtro a ser implementada no código da Wemos D1 mini conforme demonstrado nas equações 4.14 a 4.17. Nas demonstrações que se seguem, $\omega(z)$ e $Y(z)$ são respectivamente as saídas filtrada e pré-filtrada.

$$H(z) = \frac{\omega(z)}{Y(z)} = \frac{1-p}{z-p} * \frac{z^{-1}}{z^{-1}} \tag{4.14}$$

$$\omega(z) * z * z^{-1} - \omega(z) * p * z^{-1} = Y(z) * (1-p) * z^{-1} \tag{4.15}$$

$$\omega(z) = (1-p) * Y(z) * z^{-1} + p * \omega(z) * z^{-1} \tag{4.16}$$

A equação 4.16 é então transformada para o domínio amostral k.

$$\omega[k] = (1-p) * y[k-1] + p * \omega[k-1] \tag{4.17}$$

Ao analisar a forma de filtro encontrada em 4.17, é evidente que a escolha do polo ***p*** determina a dinâmica do filtro. Ao escolher um valor de ***p*** próximo ao valor de ***z*** em regime permanente, ou seja, o valor unitário, o comportamento de ***H(z)*** faz com que a leitura atual da

saída ***y[k-1]*** tenha uma contribuição pequena quando comparada com o último valor filtrado $\omega[k-1]$. Desse modo, a dinâmica da planta em malha fechada tem um comportamento mais lento, levando mais tempo para alcançar o regime permanente. No projeto realizado, o valor de ***p*** escolhido foi *0.9* e assim, o filtro toma a forma mostrada na equação 4.18 e a linha de código escrita na Wemos D1 mini deve seguir a lógica da equação 4.19.

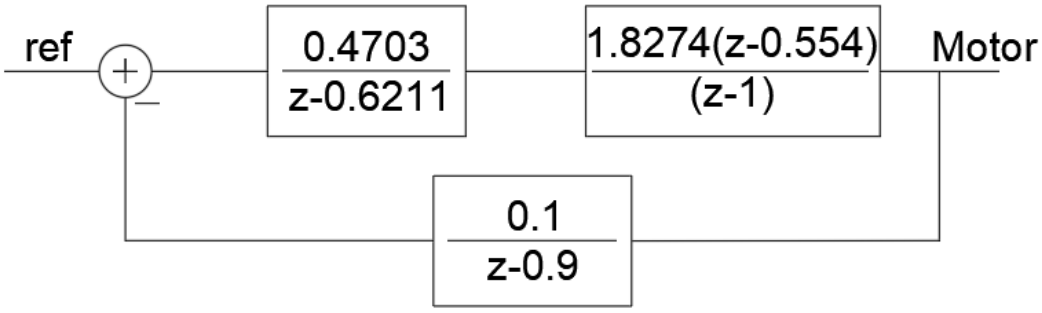
$$H(z) = \frac{1-0.9}{z-0.9} = \frac{0.1}{z-0.9} \tag{4.18}$$

$$\omega[k] = 0.1 * y[k-1] + 0.9 * \omega[k-1] \tag{4.19}$$

4.6.6 Fechamento da malha com filtro

Da mesma maneira demonstrada na seção 4.6.4, a malha é fechada com uma realimentação negativa conforme a figura 22.

Figura 22 – Diagrama da planta em malha fechada com filtro



Fonte: Autoria própria

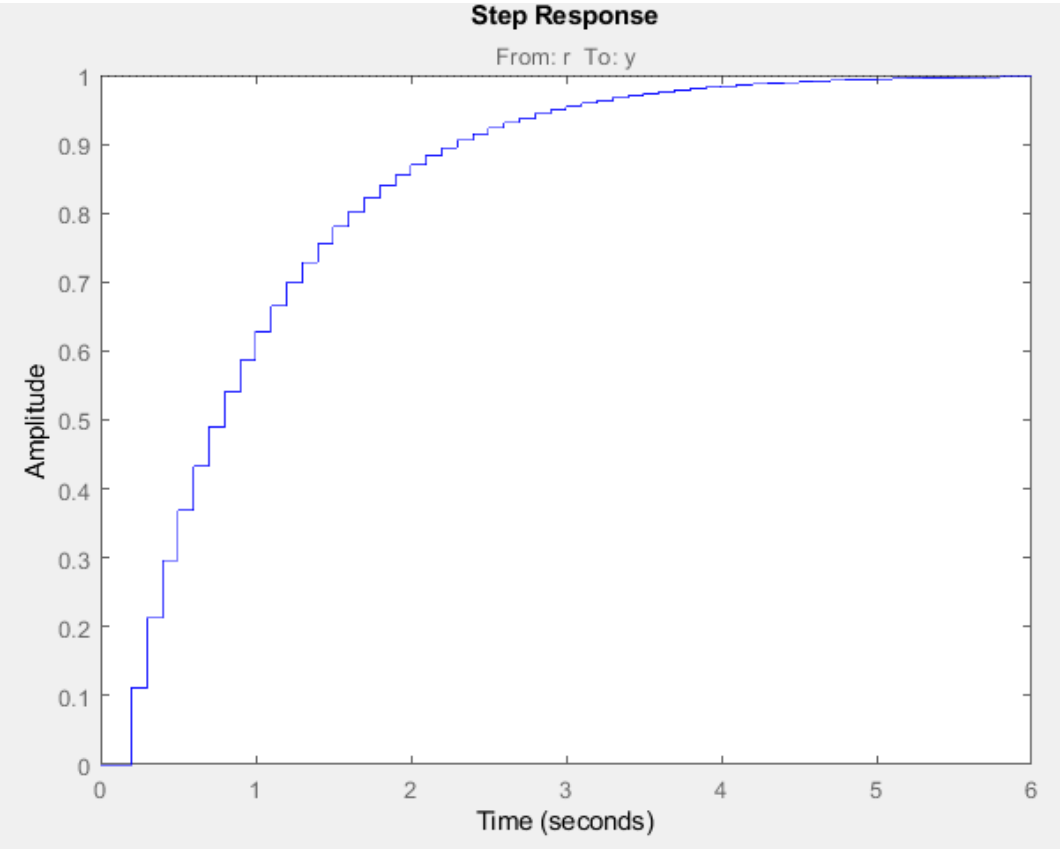
A equação de transferência ***Y(z)*** toma a forma das equações 4.20 e 4.21 configurando assim, uma planta de segunda ordem em malha fechada.

$$Y(z) = \frac{Gd(z) * H(z) * C(z)}{1 - Gd(z) * H(z) * C(z)} \tag{4.20}$$

$$Y(z) = \frac{0.85939 * (z-0.554) * (z-0.6211) * (z-0.9) * (z-1)}{(z-0.6211) * (z-1)(z-1.262) * (z^2 - 1.259z + 0.4052)} \tag{4.21}$$

Com o filtro ***H(z)*** inserido na dinâmica da planta, o comportamento da mesma fica mais lenta, levando assim, mais tempo para alcançar o regime contínuo assim como explicado na seção 4.6.5.

Figura 23 – Resposta ao degrau unitário em malha fechada com filtro



Fonte: Autoria própria

Como indicado na figura 23, o tempo necessário para que a planta em malha fechada ***Y(z)*** alcance o regime permanente está próximo de 5 segundos, enquanto que sem o filtro, segundo a figura 21, leva algo próximo de 1 segundo.

5 RESULTADOS EXPERIMENTAIS

Após um amplo esforço na implementação da comunicação *Modbus* da Wemos D1 mini com o projeto *openPLC* ficou evidente que tanto o *ladder* disponível quanto estrutura inflexível da solução se tornaram empecilhos em uma comunicação aberta às mudanças necessárias. Ao alterar o código base fornecido para a comunicação do microcontrolador, a implementação e integração com *softwares* como o *CodeSys* e o *Elipse* foi uma tarefa simples.

5.1 Análise da comunicação Modbus/TCP

O comportamento dinâmico da rede de dados tem influência direta no processamento de dados, tanto no código de automação e controle em *ladder* quanto no terminal remoto, ou seja, a própria ESP. Dessa forma, o comportamento dinâmico da planta pode ser afetado em função de possíveis atrasos de transporte, causados por perdas de pacotes de rede. Isso pode levar a oscilações de velocidade, no caso da aplicação de controle de velocidade de um motor.

Uma vez que o protocolo *Modbus* realiza apenas uma solicitação por vez, e considerando o tempo de ciclo do protocolo padrão do *Codesys* de 100ms, na ocorrência de uma perda de pacotes, é perceptível aos sentidos humanos a mudança de desempenho do controlador.

Figura 24 – *Handshaking* e ciclo Modbus.

Time	Source	Destination	Protocol	Length	Info
2021-09-24 17:33:47,743937	192.168.11.13	192.168.11.100	TCP	66	63824 → 502 [SYN] Seq=0 Win=64240 Len=0 MSS=1460 WS=256 SACK_PERM=1
2021-09-24 17:33:47,915368	192.168.11.100	192.168.11.13	TCP	62	502 → 63824 [SYN, ACK] Seq=0 Ack=1 Win=2144 Len=0 MSS=536 SACK_PERM=1
2021-09-24 17:33:47,915499	192.168.11.13	192.168.11.100	TCP	54	63824 → 502 [ACK] Seq=1 Ack=1 Win=65392 Len=0
2021-09-24 17:33:47,943852	192.168.11.13	192.168.11.100	Modbus/TCP	68	Query: Trans: 1; Unit: 255, Func: 15: Write Multiple Coils
2021-09-24 17:33:47,960027	192.168.11.100	192.168.11.13	Modbus/TCP	66	Response: Trans: 1; Unit: 255, Func: 15: Write Multiple Coils
2021-09-24 17:33:47,963819	192.168.11.13	192.168.11.100	Modbus/TCP	66	Query: Trans: 2; Unit: 255, Func: 4: Read Input Registers
2021-09-24 17:33:48,122555	192.168.11.100	192.168.11.13	TCP	54	502 → 63824 [ACK] Seq=13 Ack=27 Win=2118 Len=0
2021-09-24 17:33:48,523839	192.168.11.100	192.168.11.13	Modbus/TCP	65	Response: Trans: 2; Unit: 255, Func: 4: Read Input Registers
2021-09-24 17:33:48,543922	192.168.11.13	192.168.11.100	Modbus/TCP	69	Query: Trans: 3; Unit: 255, Func: 16: Write Multiple Registers
2021-09-24 17:33:48,577551	192.168.11.100	192.168.11.13	Modbus/TCP	66	Response: Trans: 3; Unit: 255, Func: 16: Write Multiple Registers
2021-09-24 17:33:48,583844	192.168.11.13	192.168.11.100	Modbus/TCP	66	Query: Trans: 4; Unit: 255, Func: 2: Read Discrete Inputs
2021-09-24 17:33:48,592160	192.168.11.100	192.168.11.13	Modbus/TCP	64	Response: Trans: 4; Unit: 255, Func: 2: Read Discrete Inputs
2021-09-24 17:33:48,603839	192.168.11.13	192.168.11.100	Modbus/TCP	68	Query: Trans: 5; Unit: 255, Func: 15: Write Multiple Coils
2021-09-24 17:33:48,641866	192.168.11.100	192.168.11.13	Modbus/TCP	66	Response: Trans: 5; Unit: 255, Func: 15: Write Multiple Coils

Fonte: Autoria própria

A figura 24 ilustra o comportamento completo obtido pela comunicação *Modbus* com a ESP8266. Uma das vantagens da utilização do *Modbus* está na sua fácil leitura e compreenssão por se basear em pacotes escritos em Código ASCII ao invés de *bytes* como o TCP por exemplo.

Deste modo, assim como descrito na tabela 3, ao abrir um pacote de requisição da função 15 e selecionar o cabeçalho *Data*, será encontrado um valor em base decimal, que ao ser convertido para a base binária, descreverá como os 4 bits do registrador, isto é, os pinos digitais de saída da ESP8266 devem se comportar.

Já ao analisar os pacotes de requisição da função 16 e a resposta da função 4, no

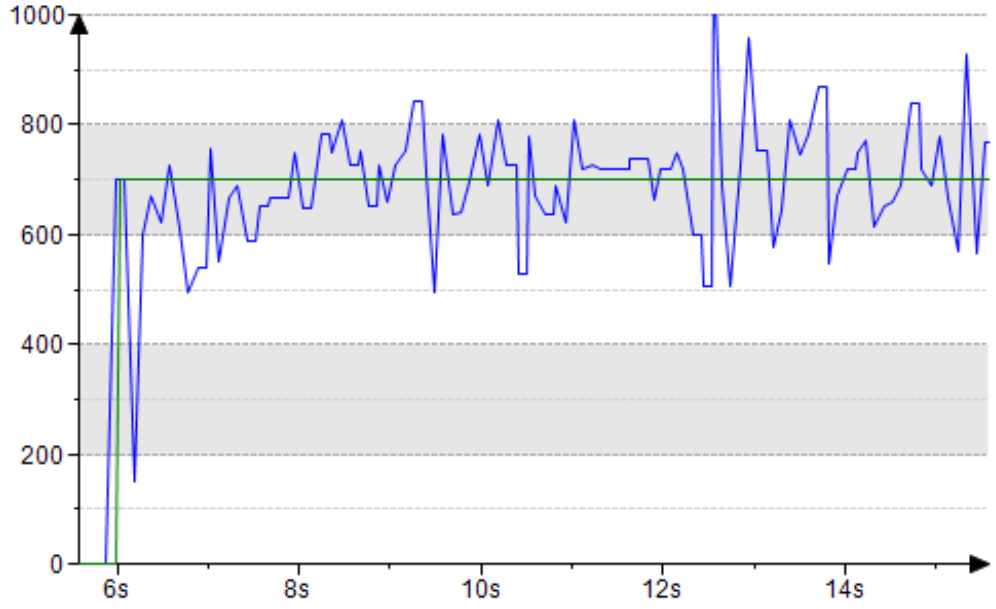
cabeçalho *Register Value*, existe um valor em UINT16 que descreve o valor que será escrito na saída de PWM e o valor de velocidade enviado ao servidor *Modbus*

Por último, a resposta da função 2 possui um cabeçalho para cada um dos 8 bits do registrador de entradas digitais (4 bits para os pinos e 4 bits não utilizados).

5.2 Desempenho do controlador

Com o auxílio do *Codesys*, é possível visualizar os dados e comportamento do *ladder* de forma *online*, desse modo, uma análise em tempo real do comportamento da planta pode ser feito afim de manter uma supervisão na mesma.

Figura 25 – Comportamento do controlador sem filtro

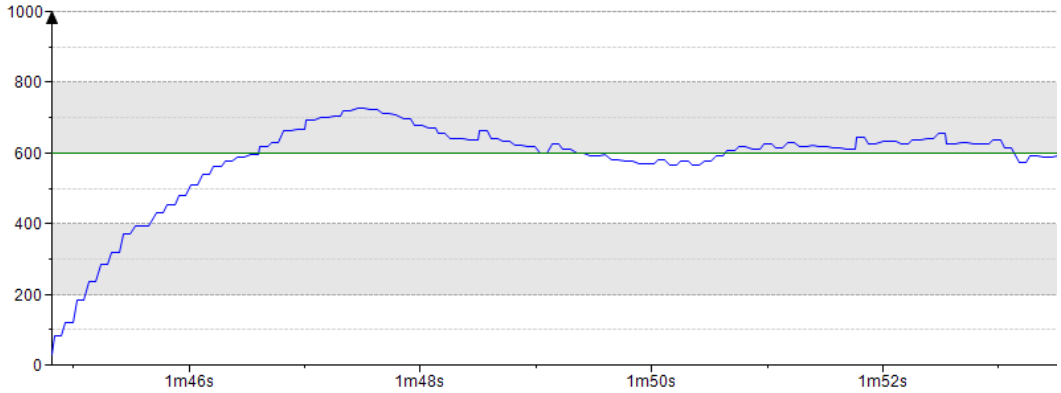


Fonte: Autoria própria

A figura 25 é um gráfico comparativo das leituras recebidas pelo mestre e a referência informada pelo MQTT. Existe um ruído nas leituras de velocidade do *encoder* decorrente de falha na leitura dos pulsos digitais pela interrupção da ESP8266.

Afim de reduzir o ruído de leitura, o filtro passa-baixa *H(z)* foi projetado e incluído no projeto de controle. Conforme a Figura 26 demonstra, a implementação do filtro na realimentação do fechamento da malha de controle trouxe estabilidade à dinâmica da planta.

Figura 26 – Comportamento do controlador com filtro



Fonte: Autoria própria

Contudo, apesar do notório aumento de eficiência do controlador, este permanece um dos mais simples tipo de controle discreto. Deste modo, é incapaz de compensar e incluir a dinâmica da rede e troca de pacotes por completo. Na ocorrência de uma retransmissão de pacote TCP.

6 CONCLUSÕES E TRABALHOS FUTUROS

A partir dos resultados de desempenho do controlador obtidos com a utilização do filtro ***H(z)***, a eficiência do sistema desenvolvido é um pontapé inicial em uma ampla gama de projetos de controlador a serem implementados para trabalhos em modalidade remota.

É necessário um estudo mais profundo quanto às peculiaridades da rede e dinâmica de perdas de pacotes e/ou atrasos de transporte. Em condições constantes de comportamento de rede, o sistema descrito na figura 11 possui resultados satisfatórios no controle de velocidade do motor DC. Contudo, na ocorrência de uma perda de pacote e/ou retransmissões TCP, é perceptível a instabilidade do sistema, à medida que o controlador tenta acompanhar a dinâmica do sistema com leituras amostrais atrasadas.

Apesar disso, uma vez que o filtro implementado se encontra diretamente na Wemos D1 mini, sua performance não é afetada por tais oscilações transitórias de comunicação. Desse modo, o projeto de um controlador capaz de prever e compensar o dinamismo do transporte e roteamento de pacotes levaria à um grau de eficiência muito mais elevado que o proposto atualmente. Outra possibilidade seria a utilização de uma rede com maior estabilidade e qualidade de serviço.

O potencial educativo a ser explorado consiste na possibilidade de alterações remotas de parâmetros das variáveis do *ladder* que funciona no *CodeSys*. Os alunos podem então alterar as constantes do controlador PID e comparar os resultados exibidos graficamente em tempo real no terminal em que o *Modbus* está se comunicando. Além disso, também é possível implementar controladores mais robustos por meio dos blocos matemáticos presentes no *software*.

É importante ressaltar que existem várias formas de melhorar o *firmware* implementado na Wemos D1 mini. Uma dessas possibilidades consiste na possibilidade de atualização do programa de maneira remota (Programação *Over The Air* / Programação Através do Ar (OTA)) possibilitando uma facilidade ainda maior na implementação de plantas com características específicas ou na inclusão de novos comandos de tratamento de mensagens do MQTT.

Um exemplo de melhoria nesse sentido seria a inclusão de uma lista de comandos pré programados na Wemos D1 mini de modo a permitir a mudança de variáveis chave pelo MQTT, como por exemplo as constantes de um PID, ou na inclusão de comandos de difusão através da rede de equipamentos MQTT do laboratório possibilitando enviar a mesma mensagem para todos ao mesmo tempo, semelhante à um IP de *broadcast* na rede local.

Como medida de segurança de que há alguém responsável operando a planta em

caráter experimental, é necessário que uma comunicação *Modbus* esteja aberta e funcionando em tempo real no *CodeSys*. Contudo, em uma aplicação prática na indústria, isso nem sempre é possível. Assim, é necessário alterar as condicionais que funcionam no laço principal do *firmware* para que a Wemos D1 mini funcione de maneira *offline*.

Com a implementação de trabalho *offline* por parte da Wemos D1 mini, a porta 502 destinada ao protocolo *Modbus* fica disponível para ser utilizada por outros dispositivos. É possível então manter o monitoramento de toda a rede *Modbus* por meio de *softwares* SCADA utilizando o programa *Elipse* por exemplo.

Uma alternativa ao procedimento descrito acima seria a abertura de outra porta TCP na Wemos D1 mini de modo a permitir a comunicação entre ambos, sistema SCADA e *CodeSys*. Deve-se avaliar então, a velocidade e otimização do processamento do código principal para permitir a inclusão de mais um processo de comunicação, mesmo que este, *Modbus*, já esteja implementado, porém, em outra porta TCP.

REFERÊNCIAS

Anderson Mott. **O que são Sistemas Supervisórios?** 2021. https://www.automacaoindustrial.info/o-que-sao-sistemas-supervisorios/. Online; acessado em 02 de Fevereiro de 2022.

BORLIDO, D. J. A. **Indústria 4.0 - Aplicação a Sistemas de Manutenção**. Dissertação (Mestrado em Engenharia Mecânica) – Faculdade de Engenharia, Universidade do Porto., 2017.

BOYER, S. A. **Supervisory Control and Data Acquisition**. -: The Instrumentation, Systems, and Automation Society, 2004. -.

Brian Harrison. **Maintenance automation: create a new recipe for data integration**. 2020. https://www.controleng.com/articles/maintenance-automation-create-a-new-recipe-for-data-integration/. Online; acessado em 03 de Dezembro de 2021.

cacpnrj. **Protocolo de comunicação serial RS232: Noções básicas, funcionamento e especificações**. 2020. https://capsistema.com.br/index.php/2020/12/18/protocolo-de-comunicacao-serial-rs232-nocoes-basicas-funcionamento-e-especificacoes/. Online; acessado em 28 de Novembro de 2021.

Carlos Márcio Freitas. **Protocolo Modbus: Fundamentos e Aplicações**. 2014. https://www.embarcados.com.br/protocolo-modbus/. Online; acessado em 23 de Setembro de 2021.

Christian Götz. **MQTT 101 – How to Get Started with the lightweight IoT Protocol**. 2014. https://www.eclipse.org/community/eclipse_newsletter/2014/october/article2.php. Online; acessado em 23 de Setembro de 2021.

CIRANI, S.; FERRARI, G.; PICONE, M.; VELTRI, L. **INTERNET of THINGS. ARCHITECTURES, PROTOCOLS AND STANDARTS**. -: John Wiley & Sons Ltd, 2019. -.

FabioBmed. **Topologia em redes**. 2012. https://www.fabiobmed.com.br/site/topologia-em-redes/. Online; acessado em 02 de Fevereiro de 2022.

Gabriel Martins Dias. **O que é Indústria 4.0?** 2021. https://www.doutoriot.com.br/negocios/industria-40/o-que-e/. Online; acessado em 04 de Dezembro de 2021.

Greici Oliveira. **Conheça o Wemos D1 Mini**. 2020. https://blogmasterwalkershop.com.br/embarcados/wemos/conheca-wemos-d1-mini. Online; acessado em 23 de Setembro de 2021.

GUPTA, A.; ARORA, S.; WESTCOTT, J. R. **Industrial Automation And Robotics**. -: Mercury Learning And Information, 2016. -.

Jhonata Teles. **Indicadores de Manutenção: Conheça os principais KPI's para Gestão da Manutenção!** 2021. https://engeteles.com.br/indicadores-de-manutencao/. Online; acessado em 04 de Dezembro de 2021.

KIM, D.-S.; TRAN-DANG, H. **Industrial Sensors and Controls in Communication Networks**. Switzerland: Springer Nature, 2019. -.

KUROSE, J. F.; ROSS, K. W. **Redes de computadores e a internet: uma abordagem top-down**. [*S. l.*]: Pearson, 2013.

LIANG, R.; ZHAO, L.; WANG, P. Performance evaluations of lora wireless communication in building environments. MDPI, 2020.

MQTT Org. **MQTT: The Standard for IoT Messaging**. 2022. https://mqtt.org/. Online; acessado em 03 de Fevereiro de 2022.

ORGANIZATION, M. **MODBUS Messaging on TCP/IP Implementation Guide V1.0b**. -: Modbus Organization, 2006. -.

PETRUZELLA, F. D. **Controladores Lógicos Programáveis**. Porto Alegre: AMGH, 2014. -.

pico Technology. **Modbus® serial protocol decoding**. 2021. https://www.picotech.com/library/oscilloscopes/modbus-serial-protocol-decoding. Online; acessado em 28 de Novembro de 2021.

RIBEIRO, M. A. **Automação Industrial**. Salvador, Bahia: Tek Treinamento & Consultoria, 2001. -.

Solis PLC. **Introduction to SCADA | What does SCADA Stand for?** 2021. https://www.solisplc.com/scada. Online; acessado em 03 de Dezembro de 2021.

Thiago Alves. **OPENPLC ON ESP8266**. 2021. https://www.openplcproject.com/runtime/esp8266/. Online; acessado em 06 de Outubro de 2021.

Thiago Alves. **PLC ADDRESSING**. 2021. https://www.openplcproject.com/reference/plc-addressing/. Online; acessado em 27 de Setembro de 2021.

VIANA, H. R. G. **PCM, Planejamento e Controle da Manutenção**. Rio de Janeiro: Quality Mark Editora, 2002. -.

Zhiguo Ding. **Why IoT Needs 5G**. 2015. https://spectrum.ieee.org/5g-taking-stock. Online; acessado em 02 de Dezembro de 2021.

GLOSSÁRIO

A

Arduíno: É uma placa de desenvolvimento de código aberto baseada em processadores ATmega. Sua facilidade de programação de robustez atrai muitos iniciantes em eletrônica e aumenta sua presença no ensino de programação.

B

***bit*:** Informação singular de 0 ou 1 utilizada para representar um sinal booleano.

booleano: É um valor representado de maneira binária, isto é, com apenas duas possibilidades, Verdadeiro ou Falso.

***broadcast*:** Em redes de computadores, se refere à um endereço ou definição de uma maneira de difundir um pacote entre todos os dispositivo conectados àquele serviço ou rede.

***broker*:** É o responsável por gerir as mensagens entre os tópicos MQTT.

***buffer*:** Armazenamento temporário de informações que foram recebidas, mas ainda não foram processadas, ficando assim em uma espécie de fila de espera.

***byte*:** Conjunto de 8 bits utilizado para representar uma informação alfanumérica.

C

***callback*:** Função de retorno. É um conjunto de código que é executado na ocorrência de algum evento.

Cliente/Servidor: Arquitetura de servidores HTTP bastante utilizada que consiste em um servidor esperando requisições enviadas por um cliente que realiza alguma solicitação e espera uma resposta.

***clock*:** Responsável por determinar a base de tempo funcional que rede as instruções de um microprocessador.

***CodeSys*:** *software* de automação IEC 61131-3 para sistemas de engenharia de controle.

Comunicação Paralela: Transmissão de informações digitais através do método Paralelo, isto é, 8 bits de cada vez em apenas 1 pulso de *clock*.

Comunicação Serial: Transmissão de informações digitais através do método Serial, isto é, um bit de cada vez em 8 pulsos de *clock*.

Código ASCII: *American Standard Code for Information Interchange*. É o padrão alfanumérico de codificação para comunicação eletrônica.

D

DNS: *Domain Name System* ou Sistema de nomes de domínios. É o conjunto de registros e servidores em união que relacionam um determinado nome de endereço à um endereço IP.

E

***Elipse*:** *Software* desenvolvido para o controle de informações e supervisão de processos industriais em tempo real.

***Encoder*:** Sensor de leitura da velocidade do motor com base no número de pulsos em um intervalo de tempo.

ESP8266: Microprocessador desenvolvido pela ESPRESSIF de baixo consumo com base em uma arquitetura RISC de 32 bits.

G

***Gateway*:** Funciona como um portão de acesso entre duas LANs. Todas as informações de comunicação entre um dispositivo em uma rede e um equipamento de outra rede passam por essa porta de acesso, que redireciona as mensagens.

H

***handshaking*:** Apresentação em 3 vias utilizado pelo protocolo TCP da camada de transporte.

I

IP: *Internet Protocol* ou Protocolo de Internet, é o protocolo que funciona na camada de rede da internet. Responsável por Estabelecer uma relação entre hospedeiros remetentes e destinatários de um determinado pacote.

IPv4: Versão 4 do protocolo de Internet. Cada endereço é formado por 32 *bit*s. Versão do protocolo aplicado atualmente em quase todos as comunicações. Porém, sua migração para a versão 6 acontecerá num breve futuro.

IPv6: Versão 6 do protocolo de Internet. Cada endereço é formado por 128 *bit*s. Possui algumas mudanças em relação à versão 4, como por exemplo um cabeçalho aprimorado de 40 *byte*s, fluxo de prioridade, classe de tráfego, entre outros.

L

***ladder*:** Linguagem desenvolvida como auxílio gráfico na programação de CLPs. Atende por este nome por utilizar diagramas horizontais semelhantes à uma escada.

M

***Matlab*:** *Software* matemático desenvolvido pela *MathWorks* amplamente utilizado na academia.

Mestre/Escravo: Arquitetura de comunicação entre dispositivos, onde existe um Mestre que comanda e realiza as solicitações à todos os Escravos.

***Modbus*:** Protocolo aberto desenvolvido pela Modicom para comunicação entre CLPs.

***Modbus/RTU*:** Implementação em rede do protocolo Modbus. A interface de conexão é realizada de maneira Serial entre um dispositivo e a RTU que por sua vez, envia as informações pela rede.

multiplexação: O conceito de multiplexação consiste em múltiplas entradas disputando um mesmo canal de transmissão onde apenas um pode ser enviado por vez. Desse modo, uma espécie de seletora eletrônica envia uma entrada por vez de maneira seletiva.

Máscara de subrede: Consiste no processo de sub-dividir a rede principal em fatias menores, de modo à diminuir o trafego de pacotes e facilitar a administração.

N

NodeMCU ESP32: Placa de desenvolvimento baseada no microcontrolador ESP32. Trata-se de uma versão otimizada e com melhorias em relação ao modelo mais simples, NodeMCU ESP8266.

O

***overshoot*:** É a ocorrência de uma ultrapassagem de um sinal do seu valor de referência.

P

***peer-to-peer*:** Comunicação entre um par de dispositivos, onde estes trocam informações por um canal dedicado à eles.

Ponte H: Módulo eletrônico responsável por realizar a interface de acionamento entre microcontroladores de baixa potência e motores acionados por estres microcontroladores.

protocolo: Conjunto de regras pré-estabelecidas entre dispositivos de modo a garantir a comunicação e troca de informações funcional entre eles.

R

***REAL*:** Tipagem de variável para assumir valores Reais.

RISC: *Reduced Instruction Set Computer*, é um tipo de arquitetura para programação de microprocessadores baseada em instruções compactas e curtas para uma implementação mais

otimizada.

***RS-485*:** Padrão de transmissão de dados via cabo e padroniza os detalhes físicos como nível de tensão distância e conector.

S

SCADA: Sistema supervisório de controle e aquisição de dados. É utilizado no monitoramento de um conjunto de equipamentos que comunicam com o servidor.

T

TCP: *Transport Control Protocol* / Protocolo de Controle de Transporte. É responsável por realizar o estabelecimento da comunicação entre equipamentos com a determinação de um IP e uma porta para cada um.

U

UINT16: Tipagem de variável compacta do tipo *Unsigned Int 16 bits*, ou seja, valores inteiros sem sinal que possam ser representados em até 16 bits, 0 a 65535.

W

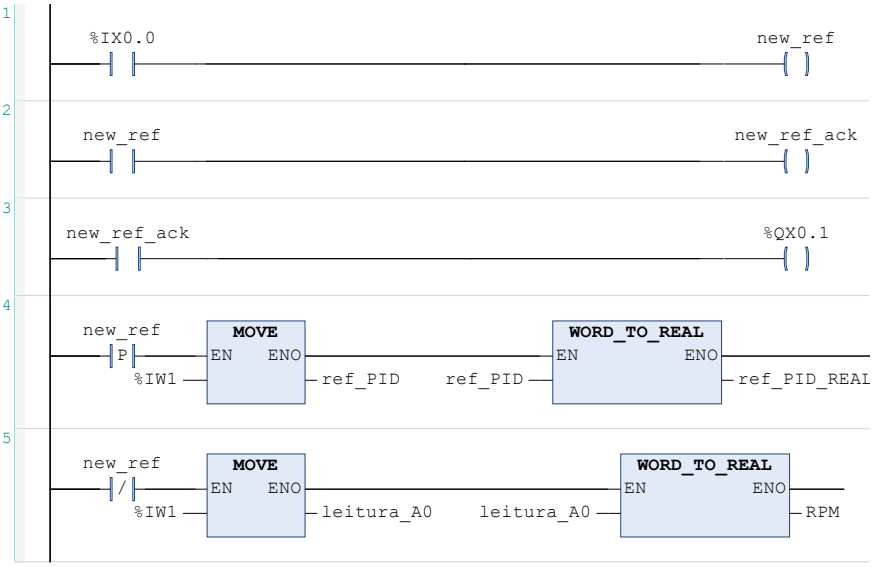
Wemos D1 mini: Placa de desenvolvimento baseada no microprocessador, de baixo consumo ESP8266X.

***WORD*:** Tipagem de variável para assumir valores de cadeias de Texto.

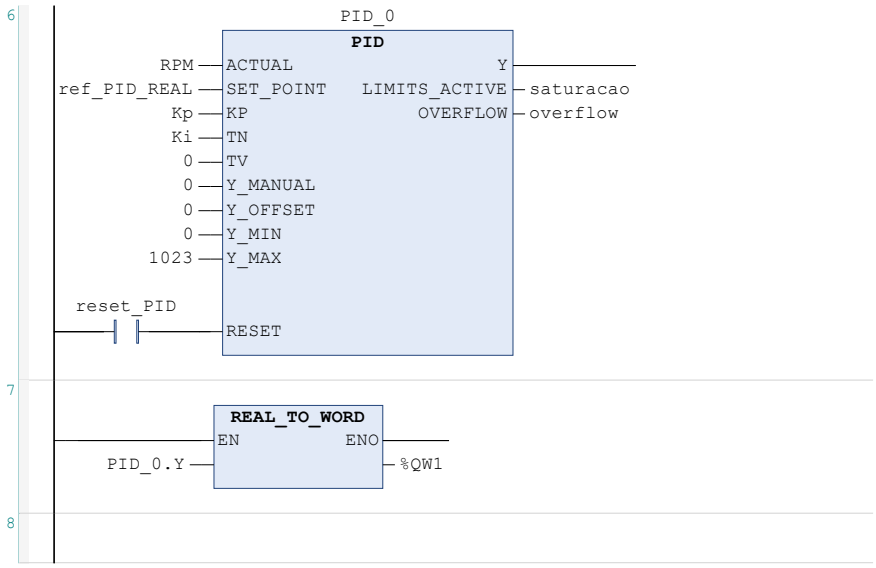
APÊNDICE A – CÓDIGO DO *LADDER*

POU: PLC_PRG

```
1   PROGRAM PLC_PRG
2   VAR
3       ref_PID : WORD ;
4       new_ref : BOOL ;
5       new_ref_ack : BOOL ;
6
7       Kp : REAL := 1.42 ;
8       Ki : REAL := 0.407 ;
9       PWM : REAL ;
10      PID_0 : PID ;
11      reset_PID : BOOL ;
12      overflow : BOOL ;
13      saturacao : BOOL ;
14      leitura_A0 : WORD ;
15      ref_PID_REAL : REAL ;
16      RPM : REAL ;
17  END_VAR
18
19
```



POU: PLC_PRG


6
PID_0
PID
RPM
ACTUAL
Y
ref_PID_REAL
SET_POINT
LIMITS_ACTIVE
saturacao
Kp
KP
OVERFLOW
overflow
Ki
TN
0
TV
0
Y_MANUAL
0
Y_OFFSET
0
Y_MIN
1023
Y_MAX
reset_PID
RESET
7
REAL_TO_WORD
EN
ENO
PID_0.Y
%QW1
8

APÊNDICE B – CÓDIGO DE IDENTIFICAÇÃO E DISCRETIZAÇÃO DO CONTROLADOR

27/09/21 21:23 C:\Users\lucas\D...\identificacao_100ms.m 1 of 1

```
clear all
close all
clc

T = readtable('dados_rpm_codesys.xlsx');

eof = 241; % ultima linha com valores úteis de rpm

rpm=table2array(T(2:eof,3));
Ts = 0.1; % período de amostragem

u = 600*ones(1,length(rpm));
t = [0:length(rpm)-1]*Ts;

K = mean(rpm(end-30:end))/u(1);
tau = 0.21;

G = tf(K,[tau 1]);
Gd = c2d(G,Ts);

yest = lsim(Gd,u,t);
plot(t,rpm,t,yest,'r');

disp('Modelo do sistema:')
Gd


rltool(Gd)
```

APÊNDICE C – CÓDIGO PRINCIPAL E BIBLIOTECAS DESENVOLVIDAS

Código-fonte 1 – main.ino

```
1
2  #include <ESP8266WiFi.h>
3  #include <PubSubClient.h> // Importa a Biblioteca
       PubSubClient
4  /*********NETWORK CONFIGURATION*********/
5  #include "wifi.h"
6  #include "modbus.h"
7  #include "codesys.h"
8
9  #include "MQTT.h"
10 #include "encoder.h"
11 /*************************************/
12 #define motorA NODE_PIN_D0
13 #define motorB NODE_PIN_D1
14 #define encoderPin NODE_PIN_D5
15
16 #define T 2000
17
18 unsigned long int t=0;
19
20 // MQTT
21
22
23 //Variaveis e objetos globais
24 WiFiClient espClient; // Cria o objeto espClient
25 PubSubClient MQTT(espClient); // Instancia o Cliente MQTT
       passando o objeto espClient
26
27
```

```
28 WiFiServer server(502);
29
30 encoder motor(motorA , motorB, encoderPin);
31
32
33 void setup()
34 {
35     Serial.begin(115200);
36     delay(10);
37
38     pinConfig();
39
40     // Connect to WiFi network
41     conectarWiFi();
42
43     // Start the server
44     server.begin();
45     Serial.println("Aguardando comunicacao Modbus");
46
47     updateIO();
48
49     initMQTT();
50 //     TCPconnect();
51 }
52
53
54
55 void loop()
56 {
57
58
59     WiFiClient client = server.available();
```

```
60      if (!client)
61
62          return;
63
64      Serial.println("Nova conexao Modbus");
65
66      while(client.connected())
67      {
68          // Wait until the client sends some data
69          while(!client.available())
70          {
71
72              delay(1);
73              if (!client.connected())
74                  return;
75          }
76
77          int i = 0;
78          while(client.available())
79          {
80              modbus_buffer[i] = client.read();
81              i++;
82
83              if (i == 100)
84                  break;
85          }
86
87          //DEBUG
88          /*
89          Serial.print("Received MB frame: ");
90          PrintHex(modbus_buffer, i);
91          */
```

```
92          MQTT.loop();
93          updateIO();
94          unsigned int return_length = processModbusMessage(
               modbus_buffer, i);
95          client.write((const uint8_t *)modbus_buffer,
               return_length);
96
97
98
99          VerificaConexoesWiFIEMQTT();
100
101         //envia o status de todos os outputs para o Broker no
                protocolo esperado
102         if (millis()-t>T){
103           EnviaEstadoOutputMQTT();
104           t = millis();
105         }
106 //          TCPSend();
107 //          MQTT.loop();
108     }
109
110     Serial.println("Modbus Desconectado");
111
112 }
```

Código-fonte 2 – wifi.h

```
1
2  #ifndef _wifi_H_
3  #define _wifi_H_
4
5
6
7  const char* SSID = "SSID"; // SSID / nome da rede WI-FI que
       deseja se conectar
8  const char* PASSWORD = "password"; // Senha da rede WI-FI que
        deseja se conectar
9  IPAddress ip(192,168,0,1); //COLOQUE UMA FAIXA DE IP
       DISPONIVEL DO SEU ROTEADOR. EX: 192.168.1.110 **** ISSO
       VARIA, NO MEU CASO E: 192.168.0.175
10 IPAddress dns(192,168,0,1);
11 IPAddress gateway(192,168,0,1); //GATEWAY DE CONEXAO (ALTERE
       PARA O GATEWAY DO SEU ROTEADOR)
12 IPAddress subnet(255,255,255,0); //MASCARA DE REDE
13 #endif
14
15 unsigned long timeout = 0;
16 const char* host = "192.168.0.10";
17 const uint16_t port = 10001;
18
19 void conectarWiFi();
20
21 // Use WiFiClient class to create TCP connections
22 WiFiClient TCPClient;
23 bool tryToConnect=true;
24
25 void TCPconnect();
26 void TCPSend();
```

```
27 #endif
```

Código-fonte 3 – wifi.ino

```
#include "wifi.h"

void conectarWiFi()
{
    //se ja esta conectado a rede WI-FI, nada e feito.
    //Caso contrario, sao efetuadas tentativas de conexao
    if (WiFi.status() == WL_CONNECTED)
        return;

    Serial.println();
    Serial.println();
    Serial.print("Connecting to ");
    Serial.println(SSID);

    if (!WiFi.config(ip, dns, gateway, subnet)) {
        Serial.println("Falha ao configurar a rede!");
    }


    WiFi.begin(SSID, PASSWORD); // Conecta na rede WI-FI

    while (WiFi.status() != WL_CONNECTED)
    {
        delay(100);
        Serial.print(".");
    }

    Serial.println();
    Serial.print("Conectado com sucesso na rede ");
    Serial.print(SSID);
    Serial.println();
```

```
32      Serial.print("IP local: ");
33      Serial.println(WiFi.localIP());
34      Serial.print("Gateway padrao: ");
35      Serial.println(WiFi.gatewayIP());
36      Serial.print("Mascara de subrede: ");
37      Serial.println(WiFi.subnetMask());
38      Serial.print("DNS: ");
39      Serial.println(WiFi.dnsIP());
40      Serial.println();
41  }
42
43  void TCPconnect(){
44
45    Serial.print("Conectando ao servidor ");
46    Serial.print(host);
47    Serial.print(':');
48    Serial.println(port);
49
50    if (!TCPClient.connect(host, port)) {
51        Serial.println("Falha de conexao com o Python");
52        TCPClient.stop();
53        return;
54      }
55    else{
56        Serial.println("Conectado ao servidor Python");
57
58        return;
59    }
60  }
61
62  void TCPSend(){
63
```

```
64      if (millis() - timeout < 100) {return;}
65
66      timeout = millis();
67
68      if (TCPClient.connected() && tryToConnect) {
69
70        TCPClient.print(ref);
71
72        if (TCPClient.available()) {
73          char ch = static_cast<char>(TCPClient.read());
74          Serial.print(ch);
75
76        }
77  //        TCPClient.stop();
78      }
79      else{
80        if ((!TCPClient.connect(host, port)) && tryToConnect) {
81            Serial.println("Falha de conexao com o Python");
82            TCPClient.stop();
83            tryToConnect=false;
84            return;
85          }
86          Serial.println("oi");
87        }
88        return;
89  }
```

Código-fonte 4 – MQTT.h

```
1
2  #ifndef _MQTT_H_
3  #define _MQTT_H_
4
5  #define TOPICO_SUBSCRIBE "topicoEnviar"     //topico MQTT de
       escuta
6  #define TOPICO_PUBLISH   "topicoReceber"    //topico MQTT de
       envio de informacoes para Broker
7  #define ID_MQTT  "meuID"     //id mqtt (para identificacao de
        sessao)
8
9  const char* BROKER_MQTT = "broker.hivemq.com"; //URL do
       broker MQTT que se deseja utilizar
10 int BROKER_PORT = 1883; // Porta do Broker MQTT
11
12 unsigned int rpmOut;
13
14 int ref=0;
15
16 bool new_ref = false;
17
18 void initMQTT();
19 void EnviaEstadoOutputMQTT(void);
20 void reconnectMQTT();
21 void VerificaConexoesWiFIEMQTT(void);
22
23 void mqtt_callback(char* topic, byte* payload, unsigned int
       length);
24 void nova_referencia(String msg,int tamanho);
25 void reiniciar_esp();
26
```

```
27 #endif
```

Código-fonte 5 – MQTT.ino

```
1  #include "MQTT.h"
2
3  void initMQTT()
4  {
5      MQTT.setServer(BROKER_MQTT, BROKER_PORT);   //informa
           qual broker e porta deve ser conectado
6      MQTT.setCallback(mqtt_callback);            //atribui
           funcao de callback (funcao chamada quando qualquer
           informacao de um dos topicos subescritos chega)
7      MQTT.subscribe(TOPICO_SUBSCRIBE);
8  }
9
10 void EnviaEstadoOutputMQTT(void)
11 {
12
13     rpmOut = motor.RPM;
14
15     char rpmMQTT[16];
16     itoa(rpmOut,rpmMQTT,10);
17     MQTT.publish(TOPICO_PUBLISH, rpmMQTT);
18
19 }
20
21 void reconnectMQTT()
22 {
23     while (!MQTT.connected())
24     {
25         Serial.print("* Tentando se conectar ao Broker MQTT:
               ");
26         Serial.println(BROKER_MQTT);
27         if (MQTT.connect(ID_MQTT))
```

```
28          {
29              Serial.println("Conectado com sucesso ao broker
                   MQTT!");
30              MQTT.subscribe(TOPICO_SUBSCRIBE);
31          }
32          else
33          {
34              Serial.println("Falha ao reconectar no broker.");
35              Serial.println("Havera nova tentativa de conexao
                   em 2s");
36
37          }
38      }
39  }
40
41  void VerificaConexoesWiFIEMQTT(void)
42  {
43      if (!MQTT.connected())
44          reconnectMQTT(); //se nao ha conexao com o Broker, a
               conexao e refeita
45
46       conectarWiFi(); //se nao ha conexao com o WiFI, a
             conexao e refeita
47  }
48
49  void mqtt_callback(char* topic, byte* payload, unsigned int
       length)
50  {
51      String msg;
52      String comando;
53
54
```

```
55
56      for(int i = 0; i < length; i++)
57      {
58
59         char c = (char)payload[i];
60         msg += c;
61
62      }
63     comando += msg[0];
64
65
66      if (msg.equals("RST")){reiniciar_esp();}
67
68      if (comando.equals("R")){nova_referencia(msg,length);}
69      if (comando.equals("C")){tryToConnect=true;}
70  }
71
72
73  void nova_referencia(String msg,int tamanho){
74    Serial.print("Nova referencia: ");
75
76    ref = 0;
77    int mult = 1;
78    for(int i = tamanho-1; i > 0;i--){
79      ref += (int)(msg[i]-48)*mult;
80      mult = mult*10;
81
82      }
83
84    new_ref=true;
85    Serial.println(ref);
86
```

```
87 }
88
89 void reiniciar_esp(){
90 //  Serial.println("Comando de reset");
91   analogWrite(pinMask_AOUT[0], 0);
92   ESP.restart();
93 }
```

Código-fonte 6 – codesys.h

```
1
2  #ifndef _codesys_H_
3  #define _codesys_H_
4
5  #define NODE_PIN_D0    16
6  #define NODE_PIN_D1    5
7  #define NODE_PIN_D2    4
8  #define NODE_PIN_D3    0
9  #define NODE_PIN_D4    2
10 #define NODE_PIN_D5    14
11 #define NODE_PIN_D6    12
12 #define NODE_PIN_D7    13
13 #define NODE_PIN_D8    15
14
15 uint8_t pinMask_DIN[] = { NODE_PIN_D4, NODE_PIN_D5,
      NODE_PIN_D6, NODE_PIN_D7 };
16 uint8_t pinMask_DOUT[] = { NODE_PIN_D0, NODE_PIN_D1,
      NODE_PIN_D2, NODE_PIN_D3 };
17 uint8_t pinMask_AIN[] = { A0 };
18 uint8_t pinMask_AOUT[] = { NODE_PIN_D8 };
19
20 unsigned char modbus_buffer[100];
21 int processModbusMessage(unsigned char *buffer, int
      bufferSize);
22
23 extern bool mb_discrete_input[MAX_DISCRETE_INPUT];
24 extern bool mb_coils[MAX_COILS];
25 extern uint16_t mb_input_regs[MAX_INP_REGS];
26 extern uint16_t mb_holding_regs[MAX_HOLD_REGS];
27
28 void pinConfig();
```

```
29 void PrintHex(uint8_t *data, uint8_t length);
30 void updateIO();
31
32 #endif
```

Código-fonte 7 – codesys.ino

```
#include "codesys.h"

void pinConfig()
{
    pinMode(NODE_PIN_D0, OUTPUT);
    pinMode(NODE_PIN_D1, OUTPUT);
    pinMode(NODE_PIN_D2, OUTPUT);
    pinMode(NODE_PIN_D3, OUTPUT);

    pinMode(NODE_PIN_D4, INPUT);
    pinMode(NODE_PIN_D5, INPUT);
    pinMode(NODE_PIN_D6, INPUT);
    pinMode(NODE_PIN_D7, INPUT);

    pinMode(NODE_PIN_D8, OUTPUT);
}

void PrintHex(uint8_t *data, uint8_t length) // prints 8-bit
   data in hex with leading zeroes
{
    for (int i=0; i<length; i++)
    {
        if (data[i] < 0x10)
            Serial.print("0");
        Serial.print(data[i], HEX); Serial.print(" ");
    }
    Serial.println();
}

void updateIO()
{
```

```
31
32      for (int i = 1; i < sizeof(pinMask_DIN); i++)
33      {
34          mb_discrete_input[i] = digitalRead(pinMask_DIN[i]);
35      }
36
37
38      for (int i = 0; i < sizeof(pinMask_DOUT); i++)
39      {
40          digitalWrite(pinMask_DOUT[i], mb_coils[i]);
41      }
42
43      for (int i = 0; i < sizeof(pinMask_AIN); i++)
44      {
45
46          if(modbus_buffer[7] == MB_FC_READ_INPUTS){motor.
               lerVelocidade();}
47          if(new_ref){mb_input_regs[i] = (ref);}
48
49
50          else{mb_input_regs[i] = motor.RPM;}
51
52      }
53
54      for (int i = 0; i < sizeof(pinMask_AOUT); i++)
55      {
56          analogWrite(pinMask_AOUT[i], mb_holding_regs[i]);
57      }
58
59
60
61      if(mb_coils[1]==true){
```

```
62          new_ref=false;
63          }
64      mb_discrete_input[0]=new_ref;
65
66  }
```

Código-fonte 8 – encoder.h

```
1
2  #ifndef _encoder_H_
3  #define _encoder_H_
4
5  #define CASA
6
7  class encoder{
8    public:
9    short int pinA;
10   short int pinB;
11   short int sensor;
12
13
14   encoder(short int pinA, short int pinB, short int sensor);
15
16   int lerVelocidade();
17   volatile int count = 0;
18   unsigned int dT = 100;
19   unsigned int pulsos_rot = 20;
20   unsigned long int tempo_encoder = 0;
21
22 //  int k = (60*1000)/(dT*pulsos_rot);
23
24   int RPM = 0;
25
26   int RPM0 = 0;
27 };
28
29
30 void ISR_encoder();
31
```

```
32
33 #endif
```

Código-fonte 9 – encoder.ino

```
1  #include "encoder.h"
2
3  encoder::encoder(short int pinA, short int pinB, short int
       sensor){
4    this->pinA = pinA;
5    this->pinB = pinB;
6    this->sensor = sensor;
7
8
9    pinMode(pinA,OUTPUT);
10   pinMode(pinB,OUTPUT);
11   pinMode(sensor,INPUT);
12
13   bool sentido = false;
14   digitalWrite(pinA,sentido);
15   digitalWrite(pinB,!sentido);
16
17   attachInterrupt(digitalPinToInterrupt(sensor), ISR_encoder,
         RISING);
18
19 }
20
21 void ICACHE_RAM_ATTR ISR_encoder() {
22   motor.count++;
23 }
24
25
26 int encoder::lerVelocidade(){
27   dT = (millis() - tempo_encoder);
28
29
```

```
30    detachInterrupt(digitalPinToInterrupt(sensor));
31    RPM0 = (3000/dT)*count;
32    RPM = 0.1*RPM0 + 0.9*RPM;
33    count = 0;
34    tempo_encoder = millis();
35
36    attachInterrupt(digitalPinToInterrupt(sensor), ISR_encoder,
          RISING);
37    return RPM;
38  }
```