UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO, DE CIÊNCIAS EXATAS E EDUCAÇÃO
DEPARTAMENTO DE ENG. DE CONTROLE, AUTOMAÇÃO E COMPUTAÇÃO
CURSO DE GRADUAÇÃO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Pedro Henrique Kohler

Proposta de uma arquitetura de identificação de caminhões para acionamento de canhões de névoa.

Blumenau
2023

Pedro Henrique Kohler

Proposta de uma arquitetura de identificação de caminhões para acionamento de canhões de névoa.

Trabalho de Conclusão de Curso de Graduação em Engenharia de Controle e Automação do Centro Tecnológico, de Ciências Exatas e Educação da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Engenheiro de Controle e Automação.
Orientador: Prof. Orientador, Dr. Adão Boava

Blumenau
2023

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Kohler, Pedro Henrique
Proposta de uma arquitetura de identificação de caminhões para acionamento de canhões de névoa. / Pedro Henrique Kohler ; orientador, Adão Boava, 2023.
74 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Campus Blumenau, Graduação em Engenharia de Controle e Automação, Blumenau, 2023.

Inclui referências.

1. Engenharia de Controle e Automação. 2. Redes wireless. 3. Modbus. 4. Sensor radar. 5. Máquina de estados. I. Boava, Adão. II. Universidade Federal de Santa Catarina. Graduação em Engenharia de Controle e Automação. III. Título.

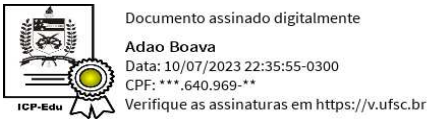
Pedro Henrique Kohler

Proposta de uma arquitetura de identificação de caminhões para acionamento de canhões de névoa.

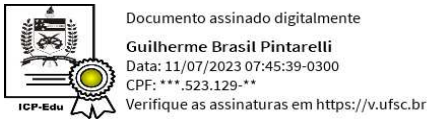
Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Engenheiro de Controle e Automação" e aprovado em sua forma final pelo Curso de Graduação em Engenharia de Controle e Automação.

Blumenau, 06 de Julho de 2023.

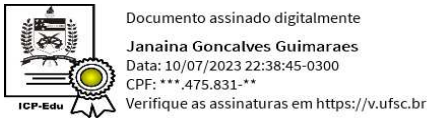
Banca Examinadora:



Prof. Dr. Adão Boava
Universidade Federal de Santa Catarina



Prof. Dr. Guilherme Brasil Pintarelli
Universidade Federal de Santa Catarina



Profª. Dra. Janaina Gonçalves Guimarães
Universidade Federal de Santa Catarina

AGRADECIMENTOS

Agradeço a minha família pelo suporte e apoio durante o desenvolvimento deste trabalho. Agradeço à Camila pelo apoio e acompanhamento. Agradeço à Sensorville pela oportunidade de me colocar a frente deste projeto e pelo suporte em todas as etapas de instalação e de desenvolvimento. Por fim, agradeço ao professor Dr. Adão Boava pela orientação e pelos ensinamentos passados durante meus anos de graduação.

RESUMO

Neste trabalho de conclusão de curso é proposto uma arquitetura de identificação de caminhões para o acionamento de canhões névoa. A identificação dos caminhões é realizada a partir de um sensor radar localizado em uma posição estratégica de detecção. A arquitetura utiliza de um par de rádios *wireless* para a transmissão dos sinais digitais de acionamento dos canhões de névoa, além de um Computador Lógico Programável (CLP) o qual executa a lógica para tal. A comunicação entre o CLP e as rádios para a leitura e escrita dos dados é feita utilizado o protocolo de comunicação Modbus RTU, protocolo esse localizado no nível físico da camada OSI utilizando da interface de dois fios RS485 como meio físico. A lógica utilizada para a comunicação serial é baseada em máquinas de estado que controlam a leitura e a escrita dos dados entre os rádios e o CLP.

Palavras-chave: sensor radar; *wireless*; Modbus RTU; máquina de estados.

ABSTRACT

In this project, an architecture for truck identification is proposed for triggering fog cannons. The trucks are identified using a radar sensor located in a strategic detection position. The architecture utilizes a pair of wireless radios for transmitting the digital signals to activate the fog cannons, along with a PLC (Programmable Logic Controller) that executes the required logic. Communication between the PLC and the radios for data reading and writing is accomplished using the Modbus RTU communication protocol, which operates at the physical layer of the OSI model and employs a two-wire RS485 interface as the physical medium. The logic employed for serial communication involves state machines that control the reading and writing of data between the radios and the PLC.

Keywords: radar sensor; wireless; Modbus RTU; state machines.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

CLP	Computador Lógico Programável
CPM	Modulação de Fase Contínua
CPU	Unidade Central de Processamento
CW	Continuous Wave
DB	Data Block
DBPSK	Differential Binary Phase Shift Keying
DQPSK	Quadrature Phase Shift Keying
DSSS	Propagação de Sequência Direta do Espalhamento de Espectro
FB	Function Block
FBD	Function Block Diagram
FC	Function
FHSS	Salto de Frequência do Espectro Propagad
GFSK	Gaussian Frequency Shift Keying
LAD	Ladder
NPN	Negativo-Positivo-Negativo
OB	Organization Block
PNP	Positivo-Negativo-Positivo
RFID	Radio Frequency Identification
RTU	Remote Terminal Unit
SCL	Structured Control Language
TCP	Transmission Control Protocol

LISTA DE SÍMBOLOS

Δ	Delta

SUMÁRIO

1 DESCRIÇÃO DO PROBLEMA

A indústria de processos possui altos requisitos de aplicações em tempo real e confiabilidade da transmissão de dados. No entanto, a transmissão de dados entre os dispositivos de campo para a produção industrial no estágio atual, adota, principalmente, rede com fios. (POZAR, 1997) Nos últimos anos, o progresso de aplicações em tempo real e a confiabilidade na tecnologia de redes sem fio vêm crescendo e se tornando uma escolha indispensável na indústria de processos. A aplicação da tecnologia de redes sem fio para a automação industrial economiza custos, torna mais conveniente e flexível para instalar, ajustar e manter o dispositivo ou processo em questão. Muitas vezes a não escolha por parte da indústria de sistemas com uma tecnologia mais atual que trazem malefícios não só para o processo como para a empresa como um todo. O caso a ser estudado neste trabalho parte exatamente desta premissa.

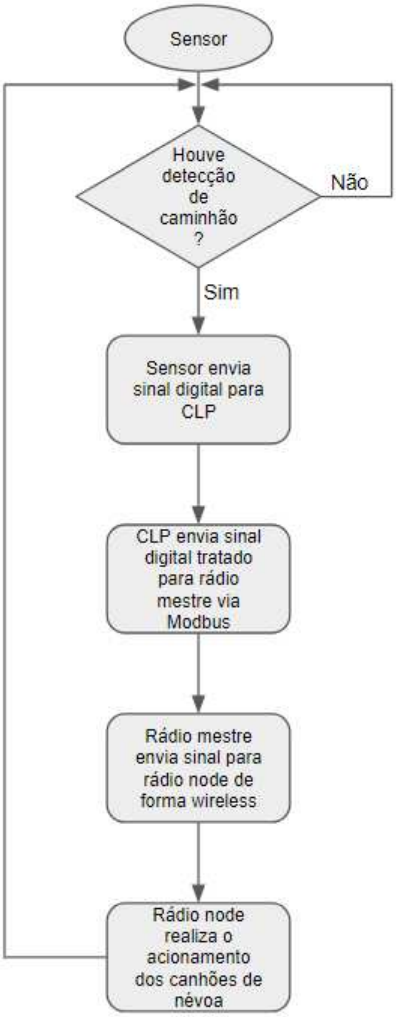
Este trabalho irá propor uma arquitetura de acionamento de canhões de névoa de forma *wireless* a fim de solucionar o problema presente em uma indústria da cidade de Joinville. Esta indústria, por se tratar de uma indústria do ramo da metalurgia, possui uma alta carga de materiais a serem descartados que não possuem mais nenhuma função em determinado processo. Uma das formas de descarte é o descarregamento destes materiais em aterros. Pelo fato da empresa possuir um fluxo alto de descarte de materiais, o descarregamento desses em aterro acarreta num elevado nível de suspensão de partículas de pó. A fim de eliminar a suspensão deste pó, foram instalados próximo ao local de descarregamento canhões de névoa capazes de minimizar a suspensão destas partículas. Seu princípio de funcionamento é a aderência da fina partícula de água às partículas de poeira, fazendo com que estas fiquem mais pesadas, dificultando que sejam carregadas pelo vento e fazendo com que elas decantem ao solo. Esta empresa não possui um sistema inteligente de acionamento destes canhões e por conta disso, havia a necessidade de deixá-los ligados durante o dia inteiro. O fato de deixá-los acionados continuamente começou a apresentar altos custos para empresa, além de não ser uma boa prática quando se trata de normas ambientais devido ao excessivo gasto de água.

Foi apresentado para a empresa em questão, uma arquitetura *wireless* de acionamento dos canhões de névoa a partir da detecção dos caminhões na área de descarte. Esta arquitetura tem como função de controlar o tempo de acionamento dos canhões e o momento exato no qual esses devem ser acionados utilizando de equipamento de rádio *wireless*, sensores e um controlador lógico programável.

A metodologia utilizada na solução foi posicionar um sensor na entrada do aterro onde acontecem os descarregamentos de materiais. Próximo ao sensor foi colocado um painel elétrico contendo um Computador Lógico Programável (CLP), o qual possui a função de executar toda a lógica de funcionamento do sistema e realizar o tratamento dos sinais digitais proveniente do sensor. Uma vez que o sensor seja acionado, ou seja, detecte

a presença de um caminhão dentro de seu raio de detecção, este envia um sinal digital para uma das entradas do Computador Lógico Programável (CLP). O CLP realiza o tratamento deste sinal, executa a lógica do sistema e envia o sinal para uma rádio mestre, também contida no painel, via protocolo de comunicação Modbus. Esta rádio mestre comunica via *wireless* com uma rádio configurada como *node*, presente em um segundo painel elétrico instalado próximo aos canhões de névoa. A rádio *node* têm a função de disponibilizar estes sinais digitais enviados pela rádio mestre a fim de realizar o acionamento dos canhões de névoa. A figura 1 mostra um fluxograma da solução proposta.

Figura 1 – Fluxograma simplificado da arquitetura prosposta



Fonte: O autor, 2022.

1.1 OBJETIVOS

Nas seções abaixo estão descritos o objetivo geral e os objetivos específicos deste TCC.

1.1.1 Objetivo Geral

O objetivo deste trabalho é propor uma arquitetura de identificação de caminhões para acionamento de canhões de névoa, utilizando a comunicação Modbus entre o CLP e as rádios, bem como a comunicação *wireless* entre as rádios para a transmissão dos sinais digitais até o ponto de acionamento dos canhões de névoa.

1.1.2 Objetivos Específicos

O objetivo específico deste trabalho é utilizar o protocolo de comunicação Modbus e sua metodologia de implementação no TIA Portal, por meio de máquinas de estado, para desenvolver uma arquitetura de identificação de caminhões e acionamento de canhões de névoa. Isso envolve a definição dos parâmetros de comunicação, configuração dos dispositivos envolvidos, estabelecimento da lógica de controle por meio de máquinas de estado e a integração adequada entre o CLP, as rádios e os canhões de névoa. O objetivo final é garantir uma comunicação confiável, eficiente e segura entre os componentes do sistema, permitindo a identificação precisa dos caminhões e o acionamento correto dos canhões de névoa.

1.2 ESTRUTURA DO DOCUMENTO

No capítulo um, é apresentada a descrição do problema, onde são expostos o contexto e os desafios enfrentados. Além disso, são estabelecidos os objetivos do projeto, tanto o objetivo geral quanto os objetivos específicos a serem alcançados. Essa seção fornece a base para as etapas de pesquisa e desenvolvimento abordadas nos capítulos seguintes.

No segundo capítulo realiza-se uma revisão bibliográfica abrangente dos principais temas relacionados ao projeto. São explorados conceitos como redes *wireless*, rádios *wireless*, salto de frequência do espectro propagado (FHSS), sensores radar e o protocolo de comunicação Modbus. Essa revisão proporciona uma compreensão teórica sólida que servirá de embasamento para as etapas subsequentes do projeto.

No capítulo três, são apresentados os materiais utilizados no projeto. Isso inclui a descrição do sensor radar empregado, bem como os rádios de performance *wireless*, como o *gateway* e o *node*. Além disso, é mencionado o controlador lógico programável utilizado, juntamente com a ferramenta de programação TIA Portal. Esse capítulo fornece uma visão geral dos componentes e ferramentas essenciais empregados no desenvolvimento do projeto.

Por fim, o capítulo quatro apresenta os resultados e discussão acerca do projeto desenvolvido. São abordadas etapas cruciais, como a detecção do caminhão, o pareamento das rádios e a atribuição de ID, além da atribuição do ID Modbus e a utilização do módulo de comunicação Modbus CB-1241. Também é explorado o software utilizado, o TIA Portal, e são descritas as funcionalidades implementadas, como a inicialização da comunicação Modbus, o controle da comunicação Modbus RTU, o uso de máquinas de estado e o controle de acionamentos. Esse capítulo oferece uma visão detalhada do desenvolvimento do projeto, enfatizando os aspectos técnicos e as soluções adotadas para a sua realização.

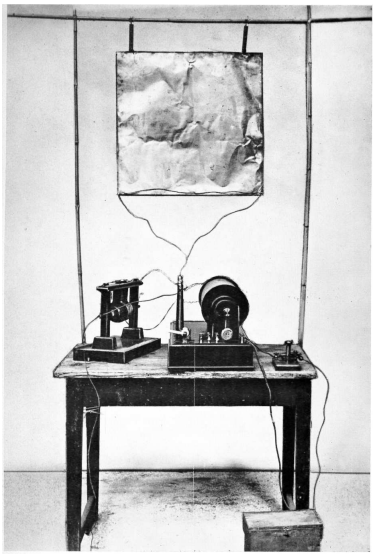
2 REVISÃO BIBLIOGRÁFICA

Neste capítulo, são apresentados os conteúdos relevantes que foram utilizados para o desenvolvimento deste trabalho.

2.1 REDE WIRELESS

As primeiras redes sem fio foram desenvolvidas na era pré-industrial. Esses sistemas transmitiam informações distâncias de linha de visão (mais tarde estendidas por telescópios) usando sinais de fumaça, sinalização por tochas, espelhos intermitentes, sinal flares ou sinalizadores de semáforo. Um elaborado conjunto de combinações de sinais foi desenvolvido para transmitir mensagens complexas com esses sinais rudimentares. As estações de observação foram construídas no topo das colinas e ao longo das estradas para transmitir essas mensagens em grandes distâncias. Essas primeiras redes de comunicação foram substituídas primeiro pela rede telegráfica, inventada por Samuel Morse em 1838, e depois pelo telefone. Em 1895, poucas décadas após a invenção do telefone, Marconi demonstrou a primeira transmissão de rádio da Ilha de Wight para um rebocador localizado a 18 milhas de distância, e assim nasceu a comunicação via rádio. O primeiro rádio transmissor proposto por Marconi é mostrado na Figura 2. A tecnologia de rádio avançou rapidamente a fim de permitir transmissões em distâncias maiores com melhor qualidade, menos energia, dispositivos menores e mais baratos, permitindo assim comunicações de rádio públicas e privadas, televisão e rede sem fio. (STANFORD UNIVERSITY, s.d.)

Figura 2 – Primeiro rádio transmissor de Marconi.



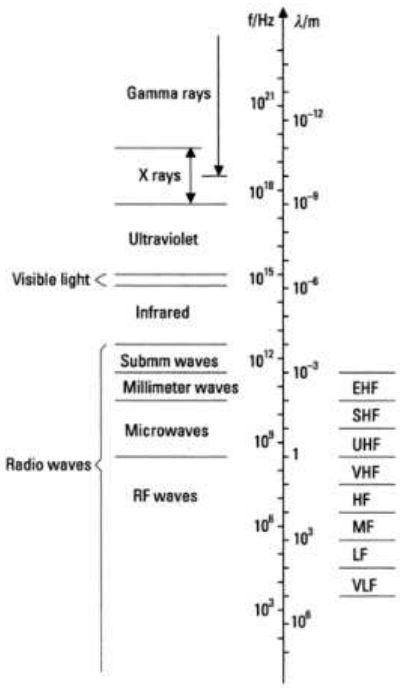
Fonte: Oxford Science Archive.

Com a constante e rápida evolução da indústria, principalmente hoje com a ascensão da indústria 4.0, cada vez mais aumenta a importância dos sistemas *wireless* em seus mais diversos ramos para o ambiente industrial. Máquinas, dispositivos, produtos e pessoas contribuem para uma transparência de informação única. Isso possibilita que as máquinas se comuniquem de forma autônoma, definindo tarefas e funções a serem desempenhadas para o correto funcionamento da fábrica (SALTIÉL, 2017). Equipamentos desta era armazenam e utilizam de informações importantes para o processo produtivo. As informações são passadas para os meios de produção de diferentes formas, sejam elas por sensores, *smart cameras*, todos conectados possibilitando uma autonomia do processo, fazendo com que a própria fábrica identifique e trate as informações, sejam elas favoráveis ou não para o processo produtivo.

2.2 RÁDIO *WIRELESS*

A palavra rádio significa técnicas utilizadas na transmissão e recepção de informação ou energia na atmosfera ou em livre espaço, ou então, linhas de transmissão das quais utilizam de ondas eletromagnéticas - também chamadas ondas de rádio - mas também os equipamentos necessários para essas (RÄISÄNEN, 2003). A Figura 3 mostra a classificação baseada nos comprimentos de onda das ondas eletromagnéticas.

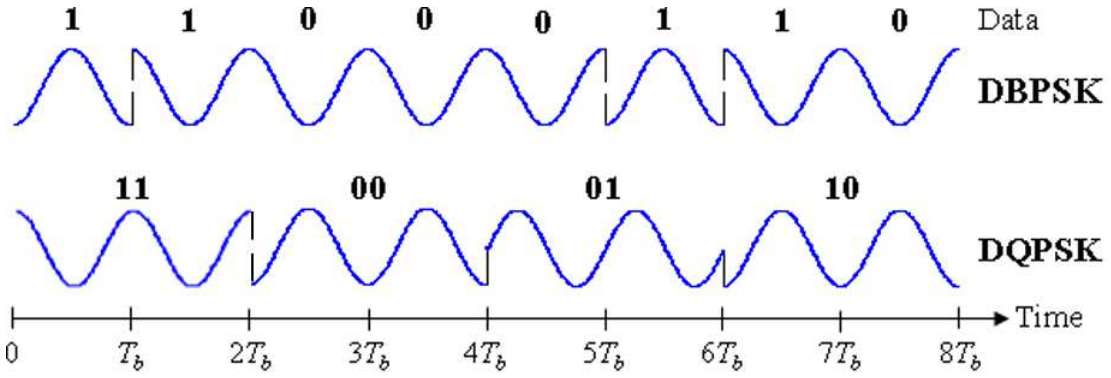
Figura 3 – Espectro eletromagnético



Fonte: (RÄISÄNEN, 2003)

Um dos grandes problemas em sistemas que utilizam de comunicação via rádio em sua topologia, é realizar o mapeamento da tecnologia a ser utilizada a fim de atender a uma variedade de requisitos para uma aplicação industrial específica. Existem vários diferentes padrões para comunicação sem fio, entre o mais importantes encontramos a LAN sem fio IEEE 802.11 (IEEE, 1999) como o padrão mais antigo e mais maduro. Esta norma especifica duas diferentes técnicas de espalhamento de espectros para a camada física, sendo estes: Propagação de Sequência Direta do Espalhamento de Espectro (DSSS) e Salto de Frequência do Espectro Propagado (FHSS). A modulação utilizada para a DSSS é chamada de *Differential Binary Phase Shift Keying* (DBPSK) ou *Differential Quadrature Phase Shift Keying* (DQPSK). No caso da FHSS (Salto de Frequência do Espectro Propagad), dois ou quatro níveis da *Gaussian Frequency Shift Keying* (GFSK) são utilizados como técnicas de modulação (WIBERG, 2001). A Figura 4 mostra um diagrama de tempo para DBPSK e DQPSK. A sequência binária encontra-se acima do sinal DBPSK. Os bits individuais do sinal DBPSK são agrupados em pares para o sinal DQPSK, o qual muda a cada $T_s = 2T_b$.

Figura 4 – Gráfico de DBPSK e DQPSK



Fonte: (CONTRIBUTORS, 2022)

Na arquitetura a ser apresentada neste projeto, são utilizados rádios *wireless* os quais adotam como técnica de troca de dados o Salto de Frequência do Espectro Propagado (FHSS). Essa troca é realizada entre duas rádios, sendo uma delas o *gateway*, ou rádio mestre, e o *node*, rádio escravo.

2.2.1 Salto de Frequência do Espectro Propagado (FHSS)

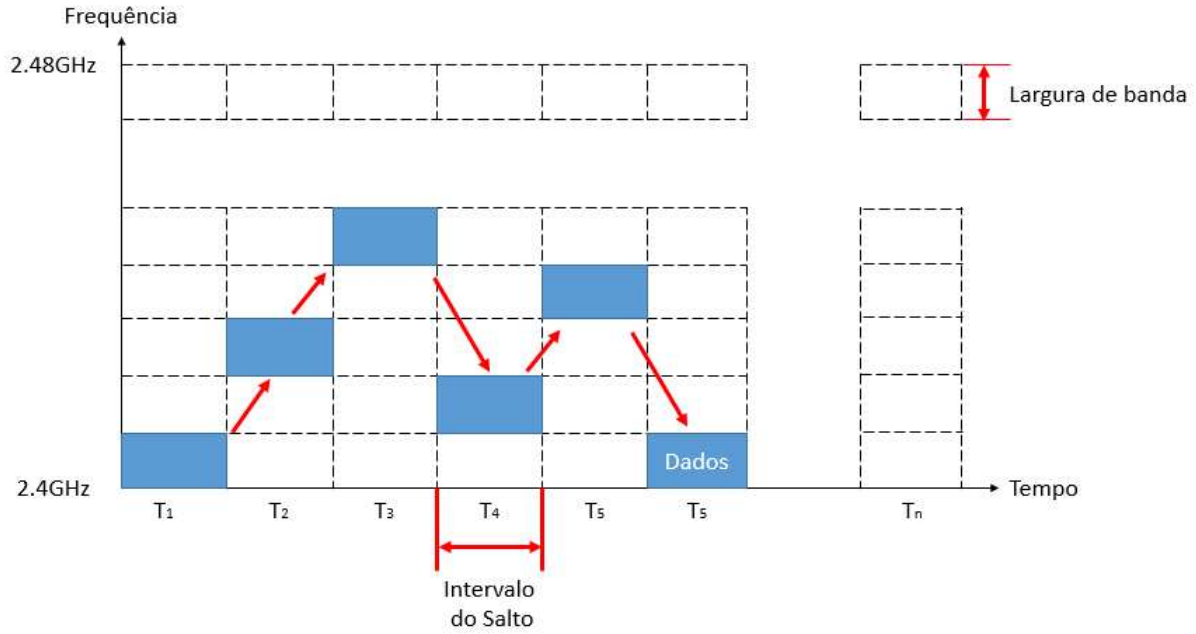
A ideia por trás do FHSS foi descoberta e redescoberta várias vezes no século 20. O físico e engenheiro elétrico alemão Jonathan Zenneck mencionou inicialmente o conceito impresso em 1908. O crédito pela técnica, no entanto, pertence à atriz Hedy Lamarr que trabalhou com o compositor George Antheil para trazer a tecnologia à existência durante a Segunda Guerra Mundial(HANNA, 2021).

Nesta técnica, o par transmissor/receptor alteram a frequência de seu canal de comunicação de uma forma pseudoaleatória (LIU, 2014).

O transmissor salta de canal a canal em uma sequência pré-determinada, mas pseudo-aleatória (Figura 5). O receptor tem uma lista idêntica de canais (o conjunto de saltos) e um gerador de sequência pseudo-aleatória idêntica ao do transmissor. Um circuito de sincronização no receptor garante que o gerador de código pseudo-aleatório no receptor seja sincronizado com o do transmissor. Quando o transmissor e receptor estão sincronizados o transmissor e o receptor estão rapidamente alterando suas frequências. No entanto, caso o receptor não esteja sincronizado com o transmissor ou um receptor convencional seja usado, nada será ouvido a menos que o transmissor salte para a frequência sintonizada do receptor. Como o transmissor de salto de frequência normalmente pula de dezenas a milhares de frequências por segundo *(Hop Rate)*, o tempo que ele permanece em um determinado canal *(Dwell Time)* é muito curto e, como resultado, o sinal apareceria como uma "explosão"de interferência (KRAUS, 2003).

Como nenhum canal é usado por muito tempo e as chances de qualquer outro transmissor estar no mesmo canal ao mesmo tempo são baixas, o FHSS é frequentemente usado como um método para permitir que vários pares de transmissores e receptores operem no mesmo espaço, canal e ao mesmo tempo (HANNA, 2021).

Figura 5 – Salto de Frequência do Espectro Propagado



Fonte: O autor.

O Salto de Frequência do Espectro Propagado fornece comunicações sem congestionamentos. A frequência de transmissão muda periodicamente em pseudo-randômicos passos discretos, sequência esta conhecida somente pelo receptor pretendido

2.2.2 *Gaussian Frequency Shift Keying* (GSFK)

GFSK é uma subclasse da Modulação de Fase Contínua (CPM) e uma modulação atrativa em muitas aplicações devido às suas propriedades espectrais aprimoradas. Em modulações em que a fase não é contínua, a eficiência espectral pobre é um problema real. Descontinuidades e mudanças abruptas de fase causam essa performance ruim, tornando a CPM um esquema mais favorecido (MORELLI *et al.*, 2003), (AULIN; SUNDBERGM, 1981). GFSK é uma modulação com memória, na qual o sinal transmitido atual depende das sequências binárias atuais e anteriores.

Em GFSK, a informação a ser transmitida é codificada em uma sequência de dígitos binários (0s e 1s), que são então usados para modular a frequência de um sinal portador. Quando um "0"é transmitido, a frequência portadora é deslocada para baixo por uma certa quantidade, enquanto quando um "1"é transmitido, a frequência portadora é deslocada para cima pela mesma quantidade. A quantidade de deslocamento de frequência é proporcional à amplitude do pulso em forma de gaussiana usado para moldar o sinal modulador.

Em DQPSK, a fase é constante ao longo de um intervalo de símbolo, enquanto em GFSK, a fase varia continuamente durante determinado período (BIN; HO, 1999). Tanto DQPSK quanto GFSK podem ser detectados de forma não coerente, o que resulta em transceptores simplificados (ANDERSON; AULIN; SUNDBERGM, 1986). O GFSK é comumente usado em sistemas de comunicação sem fio de baixa potência e baixa taxa de dados, como Bluetooth, Zigbee e RFID. Também é usado em alguns telefones digitais sem fio e mouses de computador sem fio.

2.3 SENSORES RADAR

O radar fornece um método para medir diretamente o deslocamento; no entanto, os radares são frequentemente associados a *hardware* grande e de alto custo. As aplicações típicas para esse equipamento volumoso incluem rastreamento de satélites, sensoriamento remoto e mapeamento de superfície, com agências militares e governamentais como principais usuários. Algumas aplicações em menor escala, embora ainda caras, incluem sistemas de radar meteorológico e armas de radar da polícia.

Os radares operam transmitindo ondas de rádio que são refletidas por objetos ou alvos e capturadas pelo receptor do radar. A tecnologia de radar de micro-ondas pode ser dividida em duas categorias: radar de pulso e radar de onda contínua (CW), cada um com vantagens e limitações distintas (POZAR, 1997). O radar de pulso transmite uma série de pulsos de rádio-frequência e determina a distância do alvo medindo o tempo de ida e volta para cada pulso. Além da necessidade de magnetrons volumosos para a geração de pulsos, os radares de pulso não podem ser usados para medir a velocidade de um objeto.

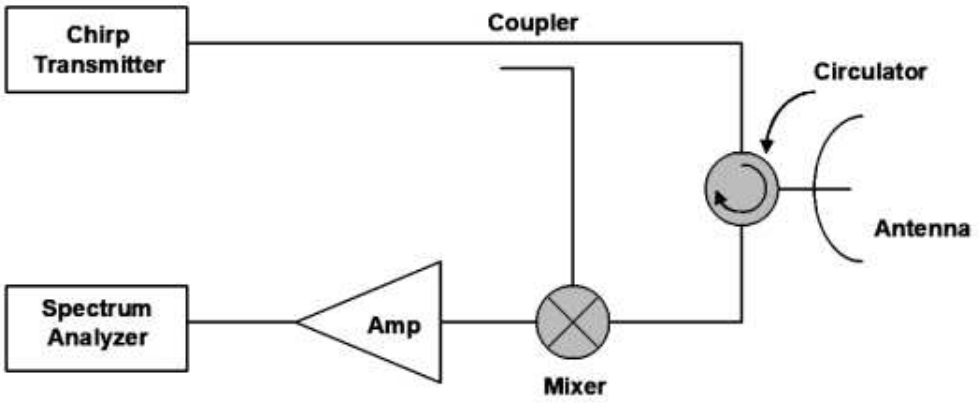
Existem dois tipos de radar CW: radar Doppler e radar de onda contínua modulada

em frequência (FMCW, do inglês Frequency Modulated Continuous Wave). O radar Doppler determina a velocidade de um objeto medindo o deslocamento da frequência Doppler da onda refletida, mas não pode determinar a distância absoluta do alvo. Mais recentemente, os radares FMCW forneceram uma alternativa aos radares de pulso para detecção de distância do alvo. Esses radares transmitem uma onda contínua controlada com uma mudança conhecida na frequência do portador ocorrendo no tempo. Medindo a diferença de frequência entre o sinal gerado atualmente e o sinal de retorno, a distância do alvo pode ser estimada com precisão (STOVE, 2004).

2.3.1 FMCW - *Frequency Modulated Continuous Wave*

A frequência desse sinal muda ao longo do tempo, geralmente em uma varredura em uma largura de banda definida. A diferença de frequência entre o sinal transmitido e o sinal recebido (refletido) é determinada misturando os dois sinais, produzindo um novo sinal que pode ser medido para determinar a distância ou a velocidade. Uma função dente de serra é o padrão mais simples e mais usado para a mudança na frequência do sinal emitido. O radar FMCW difere dos sistemas de radar pulsado clássicos porque um sinal de rádio frequência (RF) é continuamente emitido. Consequentemente, o tempo de voo para um objeto refletor não pode ser medido diretamente. Em vez disso, o radar FMCW emite um sinal RF que geralmente é varrido linearmente em frequência. O sinal recebido é então misturado com o sinal emitido e, devido ao atraso causado pelo tempo de voo para o sinal refletido, haverá uma diferença de frequência que pode ser detectada como um sinal na faixa de baixa frequência. A Figura 6 apresenta uma apresentação esquemática, mostrando um sistema de antena comum baseado em circulador.

Figura 6 – Sistema de antena comum baseado em circulador



Fonte: (MOHINDER JANKIRAMAN, 2018)

Geralmente, há duas antenas completamente separadas, uma para transmissão e outra para recepção. O problema é garantir o isolamento adequado entre as antenas.

A questão é: por que precisamos de uma função dente de serra ou qualquer tipo de modulação de frequência? Dispositivos simples de radar de onda contínua sem modulação de frequência têm a desvantagem de não poder determinar a distância do alvo porque não possuem a marcação de tempo necessária para permitir que o sistema marque com precisão, para medir os ciclos de transmissão e recepção e converter isso em distância. Uma referência de tempo para medir a distância de objetos estacionários pode ser gerada usando a modulação de frequência do sinal transmitido. Nesse método, um sinal é transmitido que aumenta ou diminui a frequência periodicamente. Quando um sinal de eco é recebido, essa mudança de frequência obtém um atraso Δt, assim como em um radar pulsado. No entanto, no radar pulsado, o tempo de execução deve ser medido diretamente. No radar FMCW, as diferenças de fase ou frequência entre o sinal efetivamente transmitido e o sinal recebido são medidas.

As características dos radares FMCW são descritas a seguir:

- A medição de distância é realizada comparando a frequência do sinal recebido com uma referência (geralmente diretamente o sinal de transmissão);
- A duração do sinal de transmissão é substancialmente maior do que o tempo de recepção necessário para o intervalo de medição de distância instalado.
- Capacidade de medir faixas muito pequenas em direção ao alvo;
- Capacidade de medir simultaneamente a distância do alvo e sua velocidade relativa;
- Alta precisão na medição de distância;
- Desempenho de processamento de sinal após a mistura em uma faixa de baixa frequência, simplificando consideravelmente a realização dos circuitos de processamento;
- Segurança pela ausência de radiação de pulso com alta potência de pico.

(MOHINDER JANKIRAMAN, 2018)

2.4 MODBUS

O protocolo Modbus foi desenvolvido pela Modicon em 1979 (MODBUS..., 2002). O Modbus é um protocolo de comunicação fundamental aplicado principalmente em indústrias. É universal, aberto e fácil de usar. Novos produtos industriais, como CLPs, PACs, dispositivos de E/S e instrumentos, podem ter uma interface Ethernet, serial ou até mesmo sem fio, mas o Modbus ainda é o protocolo mais utilizado. A principal vantagem do protocolo Modbus é que ele funciona em todos os tipos de meios de comunicação, incluindo fios de par trançado, sem fio, fibra ótica, Ethernet, etc. Os dispositivos Modbus possuem memória, onde os dados da planta são armazenados. Essa memória é dividida em quatro partes: *discrete input*, *discrete coil*, *input register* e *holding register* (MODBUS...,

2002). A *discrete input* e a *discrete coil* são de 1 bit, enquanto os *input registers* e os *holding registers* são de 16 bits. Os tipos de protocolos Modbus mais comumente usados estão listados abaixo (TAMBOLI *et al.*, 2015):

- Modbus RTU
- Modbus ASCII
- Modbus TCP

O protocolo Modbus é um protocolo Mestre-Escravo. Esse protocolo ocorre no nível 2 do modelo OSI. Um sistema do tipo mestre-escravo possui um nó (o nó mestre) que emite comandos explícitos para um dos nós "escravos"e processa as respostas (MODBUS. . . , 2002). Os nós escravos normalmente não transmitem dados sem uma solicitação do nó mestre e não se comunicam com outros escravos. No nível físico, os sistemas Modbus sobre linha serial podem usar diferentes interfaces físicas (RS485, RS232). RS-232, RS-422 e RS-485 também podem ser diferenciados com base no *baudrate* e na taxa de transporte de dados (ZHU; ZHUO; XIONG, 2012). A Tabela 1 realiza uma comparação entre os protoclos de comunicação serial destacando as características de modo de transmissão, distância e transmissão máxima.

Tabela 1 – Protocolos de comunicação com fio comuns

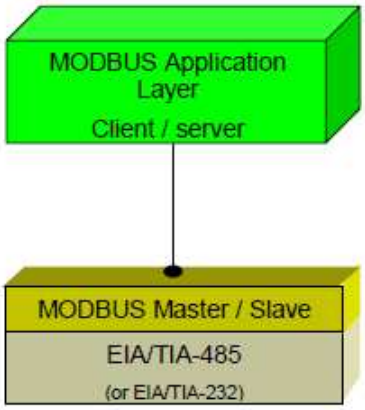
	Protocolo		
Características	**RS-232**	**RS-485**	**GB-Ethernet**
Modo de comunicação	Full-Duplex	Full/Half-Duplex	Full/Half-Duplex
Distância máxima	15 - 20 m	1200 m	100 m
Transmissão máxima	20 Kb/s	20 Mb/s	1 Gb/s

Fonte: (HERATH; ARIYATHUNGE; PRIYANKARA, 2020)

A interface de dois fios TIA/EIA-485 (RS485) é a mais comum. Como opção adicional, a interface de quatro fios RS485 também pode ser implementada. Uma interface serial TIA/EIA-232-E (RS232) também pode ser usada como interface, quando apenas a comunicação ponto a ponto curta é necessária. A figura a seguir apresenta uma representação geral da pilha de comunicação serial Modbus em comparação com as 7 camadas do modelo OSI.

Figura 7 – Sistema de antena comum baseado em circulador

Layer	ISO/OSI Model	
7	Application	MODBUS Application Protocol
6	Presentation	Empty
5	Session	Empty
4	Transport	Empty
3	Network	Empty
2	Data Link	MODBUS Serial Line Protocol
1	Physical	EIA/TIA-485 (or EIA/TIA-232)



Fonte: (MODBUS. . . , 2002)

O protocolo de mensagens Modbus da camada de aplicação, posicionado no nível 7 do modelo OSI, fornece comunicação cliente/servidor entre dispositivos conectados em barramentos ou redes. Na linha serial Modbus, o papel do cliente é fornecido pelo mestre do barramento serial e os nós escravos atuam como servidores. Apenas um mestre está conectado ao barramento por vez, e um ou vários nós escravos também estão conectados ao mesmo barramento serial. Uma comunicação Modbus é sempre iniciada pelo mestre. Os nós escravos nunca transmitirão dados sem receber uma solicitação do nó mestre e nunca se comunicarão entre si. Esse fenômeno permite um comportamento determinístico evitando colisões de pacotes de dados. (HERATH; ARIYATHUNGE; PRIYANKARA, 2020) O nó mestre inicia apenas uma transação Modbus de cada vez e emite uma solicitação Modbus para os nós escravos em dois modos:

1. No modo *unicast*, o mestre aborda um escravo individual. Após receber e processar a solicitação, o escravo retorna uma mensagem (uma "resposta") ao mestre. Nesse modo, uma transação Modbus consiste em 2 mensagens: uma solicitação do mestre e uma resposta do escravo. Cada escravo deve ter um endereço exclusivo (de 1 a 247) para que possa ser abordado independentemente de outros nós.

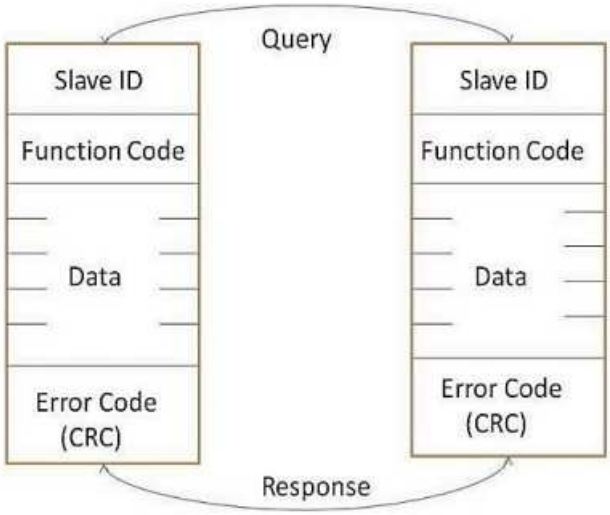
2. No modo *broadcast*, o mestre pode enviar uma solicitação para todos os escravos. Nenhuma resposta é retornada para solicitações de *broadcast* enviadas pelo mestre. As solicitações de *broadcast* são necessariamente comandos de escrita. Todos os dispositivos devem aceitar o *broadcast* para a função de escrita. O endereço 0 é reservado para identificar a troca para *broadcast*.

2.4.1 Modbus RTU

Modbus RTU é um protocolo de comunicação serial aberto ponto a ponto. É usado para desenvolver comunicação Multi-Master Slave / Servidor-Cliente entre dispositivos inteligentes. Em Modbus RTU, RS-232, RS-422 ou RS-485 podem ser usados como camada física com base na especificação da camada física. RS-232 é uma topologia duplex que transmite e recebe dados ao mesmo tempo, enquanto RS-485 é uma topologia *half-duplex* na qual o processo de transmissão e recepção é realizado um após o outro.

Em Modbus RTU, a camada física é responsável pelo endereço do escravo, bit de início e bit de parada nos dados, código CRC, tempo limite e detecção de erro de enquadramento enquanto a camada de *data link* é responsável pelo reconhecimento ou rejeição do código de função, dados ocupados ou em espera. O código de função *(Function Code)* do Modbus RTU é de ponto flutuante de 32 bits porém inteiros também são utilizados. Nos dispositivos mestres do Modbus RTU, envia-se a consulta aos dispositivos escravos e estes enviam a resposta à consulta do mestre de acordo com o código especificado. A Figura 8 mostra o formato da mensagem do Modbus RTU (TAMBOLI *et al.*, 2015).

Figura 8 – Ciclo de mensagem no campo de mensagens do protocolo Modbus RTU



Fonte: (TAMBOLI *et al.*, 2015)

2.4.1.1 Endereçamento dos tipos de dados

Modbus tem 4 tipos de dados separados. Cada tipo de dado é acessado por meio de faixas de endereço fixas. Essas faixas de endereço eram endereços de memória reais nos CLPs que originaram a tecnologia (atualmente conhecida como CLP) originais. Os tipos de dados podem armazenar dados digitais booleanos ou analógicos (registradores de 16 bits) e podem permitir acesso de leitura/gravação ou apenas leitura.

Tabela 2 – Tipos de dados Modbus

Tipo de dado	**Intervalo de endereço**	**Mapeamento de endereço**	**Acesso**
Coils	00001 até 09998	Um endereço por bit	Leitura/Escrita
Discrete Inputs	10001 até 10998	Um endereço por bit	Apenas leitura
Input Register	30001 to 39998	Um endereço por registrador	Apenas Leitura
Holding Register	40001 to 49998	Um endereço por registrador	Leitura/Escrita

Fonte: (MODBUS. . . , 2002)

2.4.1.2 Códigos de função

O campo de código de função de uma mensagem contém oito bits no Modbus RTU. Os códigos válidos estão no intervalo de 1 a 255 decimal. Destes, alguns códigos são aplicáveis a todos os controladores Modicon, enquanto outros códigos se aplicam apenas a certos modelos e outros são reservados para uso futuro. Quando uma mensagem é enviada de um mestre para um dispositivo escravo, o campo de código de função informa ao escravo que tipo de ação deve ser executada. Exemplos incluem ler os estados ligado e desligado de um grupo de *coils* ou entradas discretas, ler o conteúdo de dados de um grupo de registradores, ler o status de diagnóstico do escravo, escrever em *coils* ou registradores designados e permitir o carregamento, gravação ou verificação do programa dentro do escravo. Quando o escravo responde ao mestre, ele usa o campo de código de função para indicar uma resposta sem erros ou que ocorreu algum tipo de erro (chamado de resposta de exceção). Para uma resposta normal, o escravo simplesmente repete o código de função original. Para uma resposta de exceção, o escravo retorna um código equivalente ao código de função original com seu bit mais significativo definido como 1.

Alguns códigos de funções eram relevantes apenas para os CLPs Modicon. As implementações modernas do Modbus usam apenas uma seleção dos códigos de função mais relevantes e úteis. A Tabela três mostra os principais códigos de funções utilizados na comunicação Modbus.

Tabela 3 – Tabela de códigos de funções

Código de função	**Descrição**
01	Lê os valores de status da coil do dispositivo escravo
02	Lê os valores de estado da entrada discreta do dispositivo escravo
03	Lê os valores de dados do holding register do dispositivo escravo
04	Lê os valores de dados do registrador de entrada do dispositivo escravo
05	Escreve um único valor de estado da coil do dispositivo escravo
06	Escrve um único valor de dado de um holding register do dispositivo escravo
15	Escreve um ou mais valores de estado da coil do dispositivo escravo
16	Escreve um ou mais valores de dado de um holding register do dispositivo escravo

Fonte: (TAMBOLI *et al.*, 2015)

2.4.2 Modbus TCP/IP

O serviço de mensagens Modbus fornece uma comunicação cliente/servidor entre dispositivos conectados em uma rede Ethernet TCP/IP. Esse modelo cliente/servidor é baseado em quatro tipos de mensagens:

- Solicitação: mensagem enviada para a rede pelo cliente a fim de iniciar a transação de dados;
- Confirmação: mensagem de resposta recebida no lado do cliente
- Indicação: mensagem de requisição recebida no lado do servidor;
- Resposta: mensagem de resposta recebida no lado do servidor;

Figura 9 – Ciclo de mensagens em Modbus TCP

Fonte: (MODBUS. . . , 2002)

Um dispositivo Modbus pode fornecer uma interface Modbus cliente e/ou servidor. Uma interface de *back-end* Modbus pode ser fornecida permitindo o acesso indireto aos objetos de aplicação do usuário. Quatro áreas podem compor esta interface: entradas discretas, saídas discretas (bobinas), registradores de entrada e registradores de saída.

Tabela primária	**Tipo de objeto**	**Tipo**
Entrada discreta	Single bit	Somente leitura
Coils	Single bit	Leitura-escrita
Registradores de entrada	16-bit word	Somente leitura
Registradores de saída	16-bit word	Leitura-escrita

Fonte: (MODBUS. . . , 2002)

2.4.2.1 Cliente Modbus

O cliente Modbus permite que a aplicação do usuário controle explicitamente a troca de informações com um dispositivo remoto. O cliente Modbus constrói uma solicitação Modbus a partir dos parâmetros contidos em uma demanda enviada pela aplicação do usuário para a interface do cliente Modbus. O cliente Modbus usa uma transação Modbus cuja gestão inclui a espera e o processamento de uma confirmação Modbus.

2.4.2.2 Servidor Modbus

Ao receber uma solicitação Modbus, este módulo ativa uma ação local para ler, escrever ou realizar outras ações. O processamento dessas ações é feito totalmente de forma transparente para o programador da aplicação. As principais funções do servidor Modbus são aguardar uma solicitação Modbus na porta 502 TCP, tratar esta solicitação e, em seguida, construir uma resposta Modbus dependendo do contexto do dispositivo.

2.4.2.3 Gerenciamento de conexão

A comunicação entre um cliente e um servidor de módulo Modbus requer o uso de um módulo de gerenciamento de conexão TCP. Ele é responsável por gerenciar globalmente as conexões de mensagens TCP. Duas possibilidades são propostas para o gerenciamento de conexão: ou a própria aplicação do usuário gerencia as conexões TCP ou o gerenciamento de conexão é totalmente feito por este módulo e, portanto, é transparente para a aplicação do usuário. A última solução porém, implica menos flexibilidade.

A porta TCP de escuta 502 é reservada para comunicações Modbus. É obrigatório escutar por padrão nesta porta. No entanto, alguns mercados ou aplicações podem exigir que outra porta seja dedicada a Modbus sobre TCP. Por esse motivo, é altamente recomendável que os clientes e os servidores dêem a possibilidade ao usuário de parametrizar o número da porta Modbus sobre TCP. É importante observar que, mesmo que outra porta de servidor TCP seja configurada para o serviço Modbus em determinadas aplicações, a porta do servidor TCP 502 ainda deve estar disponível, além de quaisquer portas específicas da aplicação.

3 MATERIAIS DE PROJETO

Neste capítulo serão apresentados os materiais utilizados no desenvolvimento deste trabalho.

3.1 SENSOR RADAR

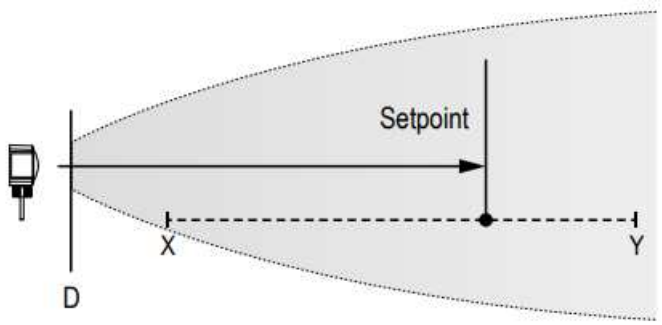
O sensor utilizado no presente projeto utiliza da tecnologia radar a fim de realizar a detecção de objetos estáticos ou em movimento que estiverem dentro do seu *range* de detecção. A tecnologia utilizada pelo sensor é a de radar de onda contínua modulada em frequência (FMCW), descrito na seção 2.3.1 deste trabalho. A Figura 10 mostra de forma visual o sensor utilizado.

Figura 10 – Sensor radar

Fonte: (BANNER, 2020a)

O sensor emite um feixe bem definido de ondas de rádio de alta frequência a partir de uma antena interna. Parte dessa energia emitida reflete de volta para a antena receptora. A eletrônica de processamento de sinal no sensor determina a distância do sensor ao objeto com base no atraso de tempo do sinal de retorno, conforme mostrado na Figura 11. O sensor pode ser configurado (via chaves DIP) para detectar objetos até uma distância específica, ignorando objetos além dessa distância (também chamada de supressão de fundo).

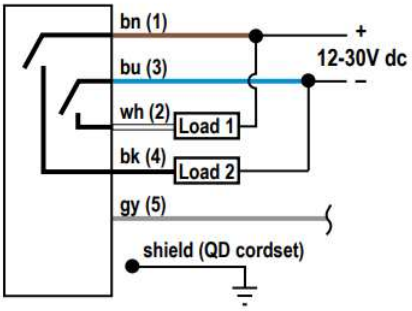
Figura 11 – Pontos de detecção do sensor radar



Fonte: (BANNER, 2020a)

As saídas do sensor são bipolares podendo-se utilizar tanto um sinal de saída NPN quanto um sinal de saída PNP, conforme o diagrama abaixo. No desenvolvimento deste trabalho, foi utilizado um sinal de saída PNP normalmente aberto.

Figura 12 – Diagrama elétrico do sensor radar



Fonte: (BANNER, 2020a)

3.2 RÁDIOS *PERFORMANCE WIRELESS*

O presente trabalho conta com rádios *wireless* da Banner Engineering responsáveis transmitir sinais digitais de um ponto a outro de forma *wireless*. As rádios utilizadas utilizam da tecnologia de salto de frequência do espectro propagado, descrito na seção 2.2.1.

3.2.1 Gateway

O rádio *gateway* age como o mestre da rede *wireless* de rádios. A rádio *gateway* utilizada neste trabalho possui capacidade de se comunicar com até 47 rádios *nodes* ou escravos. A faixa de frequência utilizada na comunicação é de 2.4Ghz.

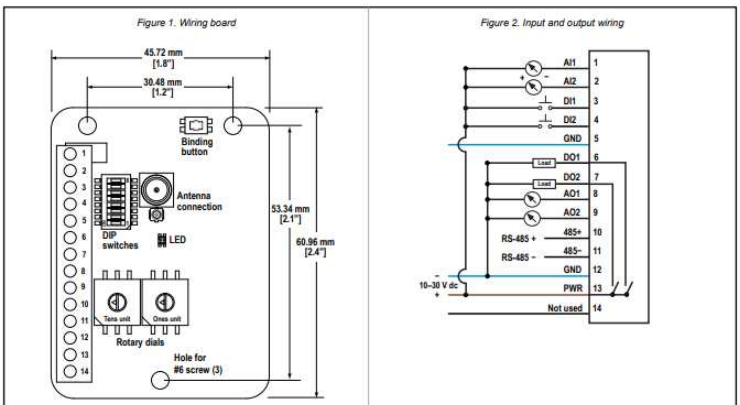
Figura 13 – Rádio *gateway*

Fonte: (BANNER, 2020c)

A rádio possui duas entradas digitais PNP, duas saídas digitais PNP, duas entradas analógicas de 4 a 20 miliamperes e 2 saídas analógicas de 4 a 20 miliamperes. Além disso, a rádio conta com entradas para comunicação Modbus RS-485 (pinos 10 e 11 da Figura 14) que serão utilizadas para comunicação com o CLP, conforme será descrito no próximo capítulo.

Quando o *gateway* utiliza de sua conexão Modbus RTU RS-485, esse passa a ser um escravo Modbus para um controlador *host* Modbus RTU. O gateway mantém os registradores Modbus de todos os dispositivos sem fio dentro da rede.

Figura 14 – Fiação do rádio *gateway*



Pin	Description	Label	Pin	Description	Label
1	Analog IN 1 (0 to 20 mA)	AI1	8	Analog OUT 1 (0 to 20 mA)	AO1
2	Analog IN 2 (0 to 20 mA)	AI2	9	Analog OUT 2 (0 to 20 mA)	AO2
3	Discrete IN 1 (PNP)	DI1	10	RS-485 + Host communication connection	485+
4	Discrete IN 2 (PNP)	DI2	11	RS-485 – Host communication connection	485–
5	Ground	GND	12	Ground	GND
6	Discrete OUT 1 (PNP)	DO1	13	10 to 30 V DC	PWR
7	Discrete OUT 2 (PNP)	DO2	14	Not used	-

Fonte: (BANNER, 2020c)

3.2.2 Node

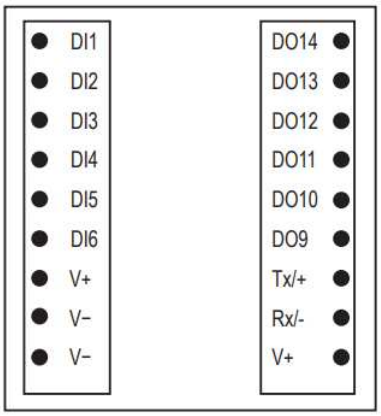
O rádio *node* age como o escravo da rede *wireless* de rádios e é o ponto final de uma rede *wireless*. Um rádio *node* possui a função de coletar e trocar dados com o o rádio *gateway*.

Figura 15 – Rádio *node*

Fonte: (BANNER, 2020b)

O rádio *node* utilizado neste trabalho possui seis entradas digitais e seis saídas digitais conforme o esquema abaixo.

Figura 16 – Fiação da rádio *node*



Fonte: (BANNER, 2020b)

3.3 *CONTROLADOR LÓGICO PROGRAMÁVEL*

O CLP é a unidade de controle central na indústria ou em um processo. A efetividade do processo e as considerações de segurança, se programadas adequadamente, pode atender com grande êxito aos objetivos requeridos. O surgimento e a aplicação de microprocessadores, microcontroladores e novas ferramentas específicas, como os CLPs, sistemas de supervisão e aquisição de dados (SCADA) e sistemas de controle distribuído (SDCD), aumentaram a produtividade, precisão, precisão e eficiência da indústria (HUDEDMANI *et al.*, 2007).

O CLP utilizado neste trabalho é um controlador da marca Siemens da família S7-1200. Este controlador possui seis entradas digitais, quatro saídas digitais e possui compatibilidade com os principais protocolos de comunicação encontrador atualmente na indústria, inclusive com Modbus RTU, protocolo utilizado neste trabalho.

A fim de estabelecer comunicação via Modbus RTU com outros dispositivos, é necessária a adição de um módulo de comunicação Modbus chamado CB 1241.

(a) Controlador Lógico Programável S7-1200 (b) Módulo de expansão Modbus

Figura 17 – Controlador Lógico Programável e módulo de expansão Modbus

A programação do CLP da Siemens é realizada a partir do software TIA Portal, software desenvolvido pela própria Siemens. Esse é amplamente difundido em grande parte das indústrias em todo o mundo pelo fato de possuir uma gama extremamente alta de funcionalidades para programação e recursos de diagnósticos. Neste software é onde são definidos os parâmetros e a configuração de comunicação entre os dispositivos do sistema, além de executar a lógica de acionamentos das saídas digitais da rádio

3.3.1 TIA Portal

De maneira geral, a programação dos controladores SIMATIC não sofreu grandes alterações com o passar do tempo. Existem as linguagens de programação conhecidas, como LAD, FBD, STL, SCL, bem como blocos de organização (OBs), blocos de função (FBs), funções (FCs) ou blocos de dados (DBs).

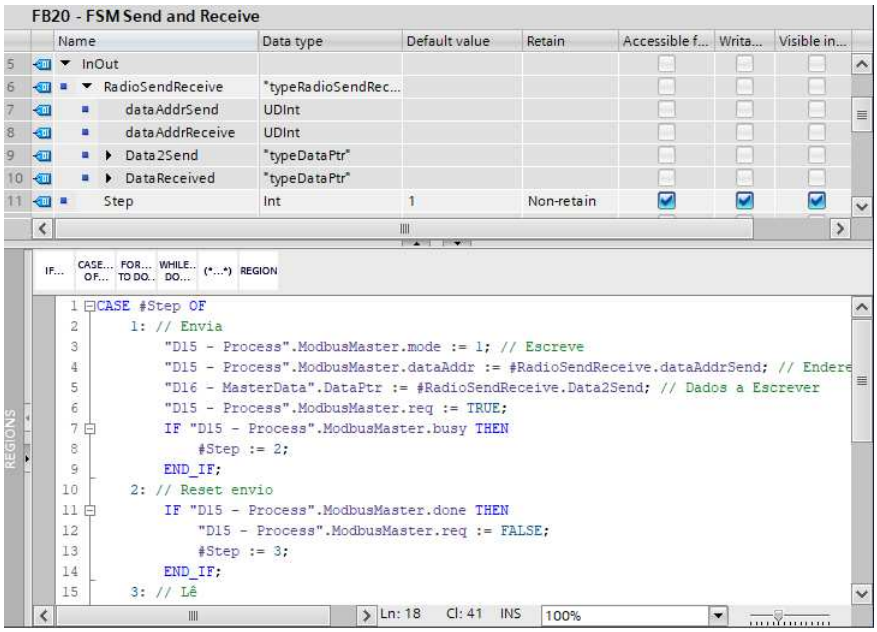
3.3.1.1 Linguagens de programação

Diferentes linguagens estão disponíveis para a programação de um código. Cada linguagem tem suas próprias vantagens que podem ser usadas de forma flexível, dependendo da aplicação. Assim, cada bloco no programa do usuário pode ser criado em qualquer linguagem de programação. As linguagens utilizadas neste projeto foram *Structured Control Language* (SCL) e *Function Block Diagram* (FBD) (IEC, 2013)

3.3.1.1.1 Structured Control Language (SCL)

Structured Control Language (SCL) é uma linguagem de programação de alto nível baseada em PASCAL para CPUs SIMATIC S7. SCL suporta a estrutura em blocos do STEP 7. Você também pode incluir blocos de programa escritos em SCL com blocos de programa escritos em LAD e FBD. As instruções SCL usam operadores padrão de programação, como para atribuição (:=), funções matemáticas (+ para adição, - para subtração, * para multiplicação e / para divisão). SCL usa operações padrão de controle de programa PASCAL, como IF-THEN-ELSE, CASE, REPEAT-UNTIL, GOTO e RETURN, como mostrado em um exemplo na Figura 18. Você pode usar qualquer referência PASCAL para elementos sintáticos da linguagem de programação SCL. Muitas outras instruções do SCL, como temporizadores e contadores, correspondem às instruções do LAD e FBD.

Figura 18 – Exemplo de SCL



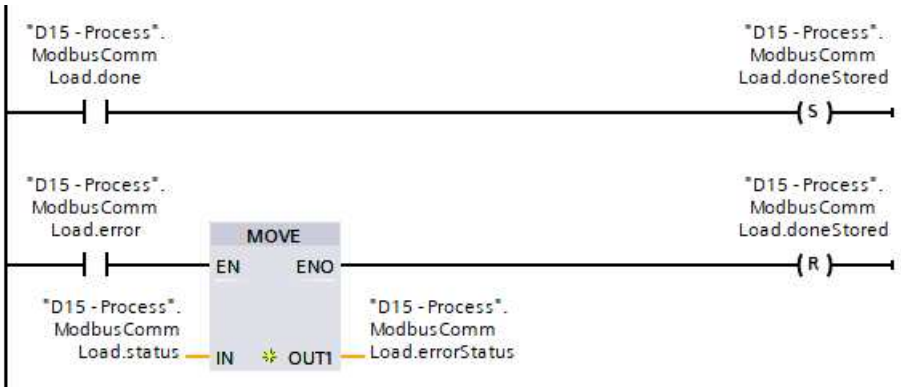
Fonte: O autor

3.3.1.1.2 Ladder (LAD)

Ladder é uma linguagem gráfica de programação a qual sua representação é baseada na representação de circuitos. Por conta da provável familiaridade do usuário com o conceito gráfico da linguagem Ladder, essa é caracterizada por sua curva de aprendizado rápida. Isso tornou a linguagem particularmente popular em aplicações as quais necessitavam de *debug* ou manutenção sem a utilização de um software externo, técnicos ou especialistas.

Na linguagem Ladder os elementor do diagrama de circuito, como contatos normalmente abertos, normalmente fechados e bobinas são interligados a fim de formarem redes.

Figura 19 – Exemplo de Ladder



Fonte: O autor

A fim de criar operações mais complexas, é possível de se adicionar ramos à rede a fim de criar circuitos paralelos. Além disso, a linguagem Ladder provê de blocos com uma grande variedade de funções a serem adicionados ao programa como, *timer*, blocos matemáticos, contadores e blocos de *move*, como mostra a Figura 19.

3.3.1.2 Blocos de programação

Dentro de um programa a ser desenvolvido no TIA Portal, o usuário tem a opção de utilizar blocos de programação a fim de proporcionar clareza e estrutura ao código.

3.3.1.2.1 Bloco de organização (OB)

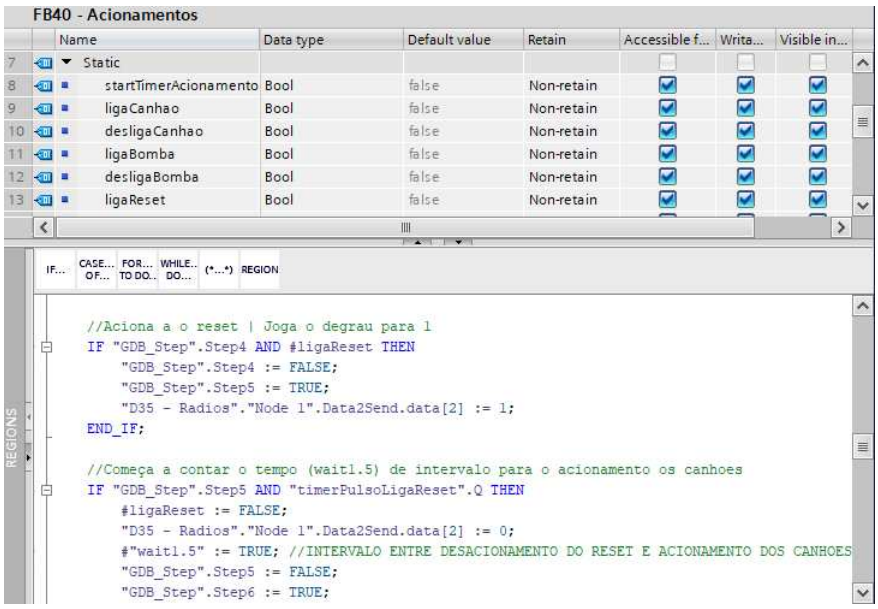
As OBs são a interface entre o sistema operacional e o programa do usuário. Elas são chamadas pelo sistema operacional e controlam processos como o comportamento de inicialização do controlador, processamento do programa cíclico, processamento do programa controlado por interrupção ou até para tratamento de erros.

3.3.1.2.2 Bloco de função (FB)

Os FBs são blocos com armazenamento cíclico de dados, nos quais seus valores são armazenados permanentemente, como mostra a Figura 20. O armazenamento cíclico é realizado em uma instância de um bloco de dados e, esta instância é somente acessada pelo respectivo bloco de função, diferentemente dos blocos de dados globais.

Uma das principais vantagens dos blocos de função é criar subprogramas e estruturar o programa. Um bloco de função também pode ser chamado várias vezes em diferentes partes do código. Isso facilita a programação de partes do programa que ocorrem com frequência.

Figura 20 – Bloco de função



Fonte: O autor

Dentro do bloco de função pode-se utilizar a linguagem de programação Ladder, SCL ou FBD.

3.3.1.2.3 Função (FC)

Os FCs são blocos sem armazenamento cíclico de dados. Por esse motivo, os valores dos parâmetros do bloco não podem ser salvos até a próxima chamada e devem ser fornecidos com parâmetros atualizados quando chamados. Para salvar permanentemente os dados de um FC, você pode utilizar as funções dos blocos de dados globais.

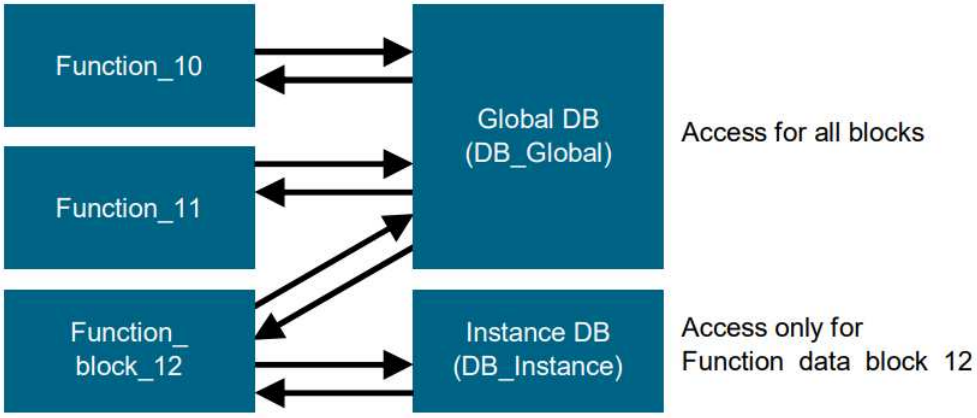
Dentro de uma função, o usuário tem a possibilidade de criar quantas saídas forem necessárias para a função além de seu valor poder ser utilizado diretamente em uma fórmula ou lógica no SCL.

3.3.1.2.4 Blocos de dados (DB)

Ao contrário dos blocos lógicos, os blocos de dados não contêm instruções. Em vez disso, eles servem como memória para os dados do usuário. Os blocos de dados contêm dados variáveis que são usados pelo programa do usuário. Você pode definir a estrutura dos blocos de dados globais conforme necessário.

Os blocos de dados globais armazenam dados que podem ser usados por todos os outros blocos, conforme mostra a Figura 21. O tamanho máximo dos blocos de dados varia dependendo da CPU utilizada.

Figura 21 – Bloco de dados



Fonte: O autor

Na maioria dos casos, os dados nos blocos de dados são armazenados de forma retentiva. Esses dados são mantidos mesmo em caso de queda de energia ou após uma parada ou inicialização da CPU.

3.3.1.2.5 *Data Types*

No TIA Portal, os *data types* (tipos de dados) referem-se aos diferentes tipos de dados que podem ser utilizados na programação de automação. Os *data types* são usados para definir o formato e a estrutura dos dados que serão armazenados e manipulados em um programa.

O TIA Portal oferece uma variedade de *data types* predefinidos que podem ser usados como booleanos, inteiros, do tipo float, char entre outros. Para este projeto, foram criadas *data types* específicas para os dados a serem transmitidos entre os dispositivos (IEC, 2013).

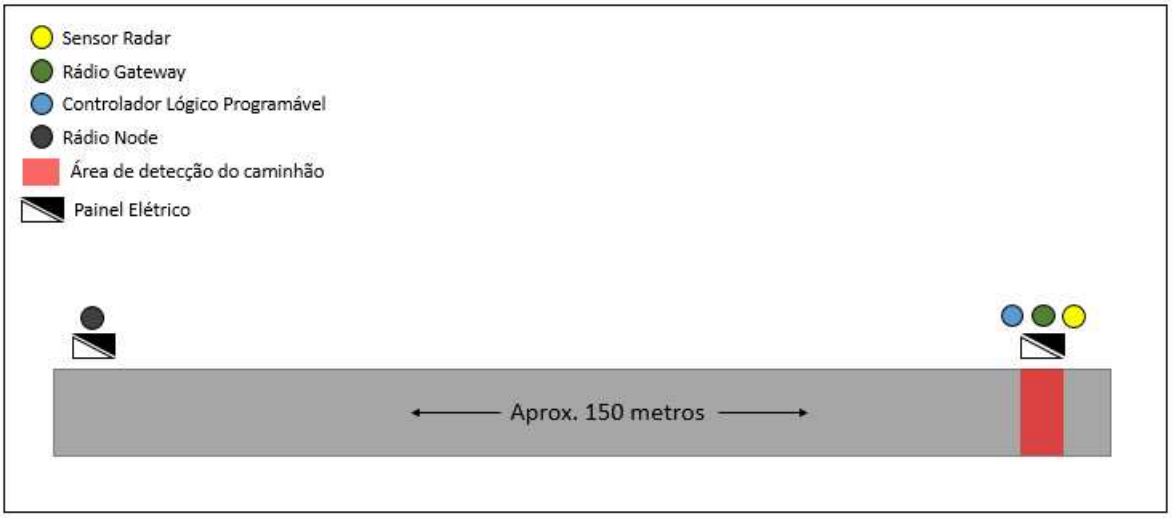
4 DESENVOLVIMENTO E RESULTADOS

Neste capítulo serão abordados características da arquitetura desenvolvida para o funcionamento do sistema de acionamento e a implementação da programação dos dispositivos contemplados no projeto.

4.1 ARQUITETURA DE PROJETO

A fim de entendermos o funcionamento e a programação do sistema de identificação de caminhões para o acionamento de canhões de névoa, devemos primeiramente entender a sua arquitetura e sua topologia. O primeiro processo do sistema é a identificação do caminhão no momento em que entra na área de detecção do sensor. Trata-se de um sensor do tipo radar posicionado a aproximadamente 150 metros da área de descarga do material. Uma vez que o caminhão é detectado pelo sensor, o mesmo envia um sinal digital ao CLP a fim de iniciar o processo de acionamento dos canhões de névoa. A arquitetura do sistema é mostrada na Figura 22.

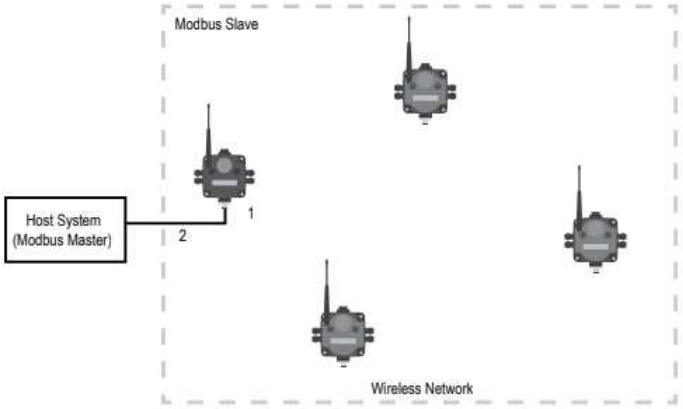
Figura 22 – Arquitetura do sistema



Fonte: O Autor

O CLP neste sistema além de tratar os sinais e executar a lógica de acionamento dos canhões de névoa, atua como um mestre Modbus do sistema e tem como escravo um rádio *gateway*. O rádio *gateway* é instalado no mesmo painél elétrico o qual contém o CLP próximo ao sensor radar. A Figura 23 mostra a topologia de comunicação Modbus entre CLP e rádio *gateway*.

Figura 23 – Arquitetura de comunicação Modbus



Fonte: (BANNER, 2020a)

Uma das principais motivações deste trabalho é o acionamento dos canhões de forma *wireless* e, por este motivo, próximo aos canhões de névoa há um segundo painel o qual possui um rádio do tipo *node*, responsável por disponibilizar saídas digitais para o acionamento dos canhões.

4.1.1 Detecção do caminhão

Como primeira etapa do sistema de acionamento dos canhões de névoa temos a detecção do caminhão. Essa ocorre a partir de um sensor radar posicionado na entrada do local onde são realizadas as descargas. A saída digital do sensor é ligada na entrada I0.1 do CLP, este sinal será utilizado na programação do acionamentos descrito na seção 4.2.4

Um dos requisitos do projeto é a detecção somente de caminhões, a detecção de pedestres, carros, bicicletas não deve ocorrer. Para tal finalidade, foram utilizados os DIP Switches localizados na parte traseira do sensor. Os DIP Switches possuem algumas configurações binárias para que o grau de sensibilidade, a distância de detecção e o tempo de resposta do sensor sejam definidos, conforme as tabelas abaixo.

Todas as configurações descritas na Tabela 4 foram realizadas a partir de testes feitos no local de atuação do sensor. No momento do teste, foi disponibilizado um caminhão pela empresa Contratante para que os testes fossem realizados de maneira robusta e confiável.

Tabela 4 – Configuração de distância do sensor radar via DIP Switches

Switch 1	**Switch 2**	**Switch 3**	**Distância (m)**
0	0	0	3.5
0	0	1	4
0	1	0	5
0	1	1	6
1	0	0	8
1	0	1	12
1	1	0	16
1	1	1	24

Fonte: (BANNER, 2020a)

A configuração utilizada para a distância do sensor foi para uma distância de 8 metros.

Tabela 5 – Configuração de sensibilidade do sensor radar via DIP Switches

Switch 4	**Switch 5**	**Sensibilidade**
0	0	4 (Muito Alta)
0	1	3 (Alta)
1	0	2 (Média)
1	1	1 (Baixa)

Fonte: (BANNER, 2020a)

A sensibilidade utilizada foi a sensibilidade média, conforme a Tabela 5. Após alguns testes utilizando carros ou pessoas posicionadas à frente do sensor, a configuração com melhor comportamento foi a de número dois.

Tabela 6 – Configuração de tempo de resposta do sensor radar via DIP Switches

Switch 7	**Switch 8**	**Liga (ms)**	**Desliga (ms)**	**Total (ms)**
0	0	30	70	100
0	1	50	300	350
1	0	30	1000	1030
1	1	120	6000	6120

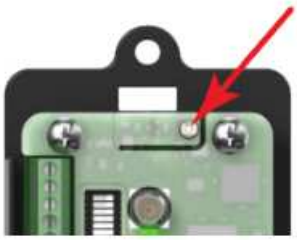
Fonte: (BANNER, 2020a)

Nesta etapa não há a necessidade de atrasos consideráveis no tempo de resposta do sensor, pois deseja-se saber o momento exato no qual ocorre a detecção do caminhão. Por este motivo, foi utilizada a configuração de menor tempo de resposta (100ms), conforme a Tabela 6.

4.1.2 Pareamento das rádios e atribuição de ID

A fim de que haja comunicação entre as rádios *gateway* e *node* do sistema, é necessário realizar o pareamento dessas. O primeiro passo do pareamento é energizar as rádios utilizando uma fonte 24V. Umas vez energizadas, deve-se entrar no modo de pareamento do *gateway* e, para isso, deve-se clicar três vezes no botão superior da rádio.

Figura 24 – Modo de pareamento da rádio *gateway*



Fonte: (BANNER, 2020c)

Com a rádio *gateway* em modo de pareamento inicia-se a configuração da rádio *node*. Para a rádio *node* é necessário primeiramente definir seu endereço, podendo ser atribuído um valor de 1 a 47. Esta etapa é importante para que os registradores Modbus sejam consultados nos endereços certos nos próximos capítulos. O endereçamento é realizado a partir dos discos rotativos presentes na parte frontal da rádio *node*. O endereço atribuído à rádio é o endereço 1. Uma vez definido o endereço, deve-se realizar o pareamento com a rádio *gateway* e, para isso, deve-se pressionar três vezes o botão 2, localizado a esquerda da parte frontal do rádio, mostrado na Figura 25.

Figura 25 – Modo de pareamento da rádio *node*

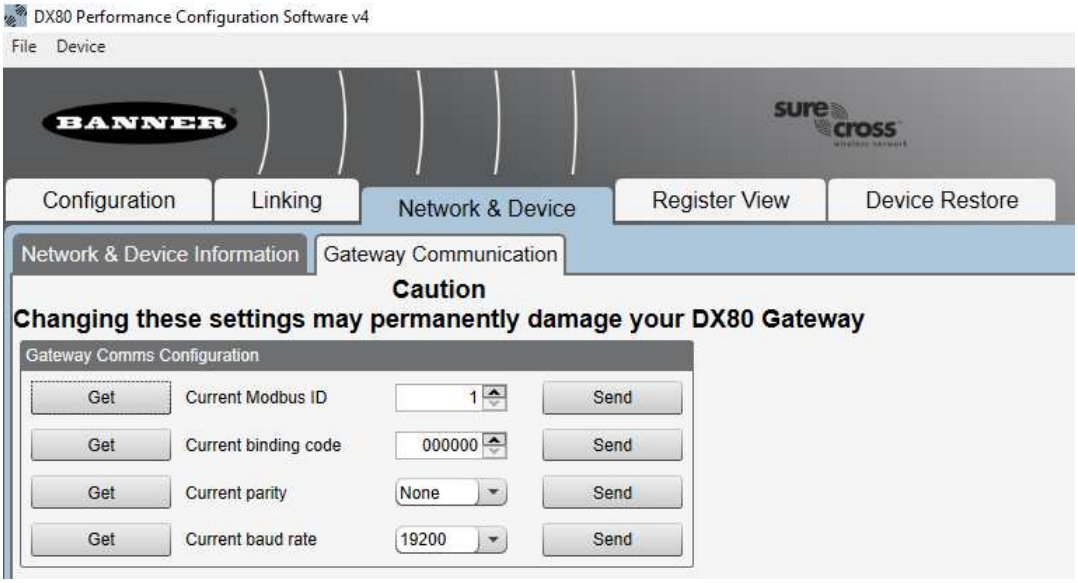
Fonte: (BANNER, 2020c)

Caso o sistema esteja pareado, o node irá piscar seus dois leds frontais quatro vezes consecutivas, indicando assim que o mesmo encontrou a rádio *gateway*.

4.1.3 Atribuição do ID Modbus

Para que seja possível a comunicação Modbus entre o CLP e a rádio *gateway*, atuante como escravo da comunicação, deve-se atribuir o ID Modbus ao escravo. A partir de um um cabo de conversão RS-485 para USB foi possível conectar a rádio *gateway* no computador e partir do software de configuração da rádio alterar seu IP Modbus. O primeiro passo no software foi selecionar a porta USB utilizada e as configurações de baudrate e paridade. Umas vez conectado ao software, na aba "Gateway Communications"é possível realizar a atribuição do ID Modbus à rádio, conforme a Figura 26.

Figura 26 – Atribuição do ID Modbus



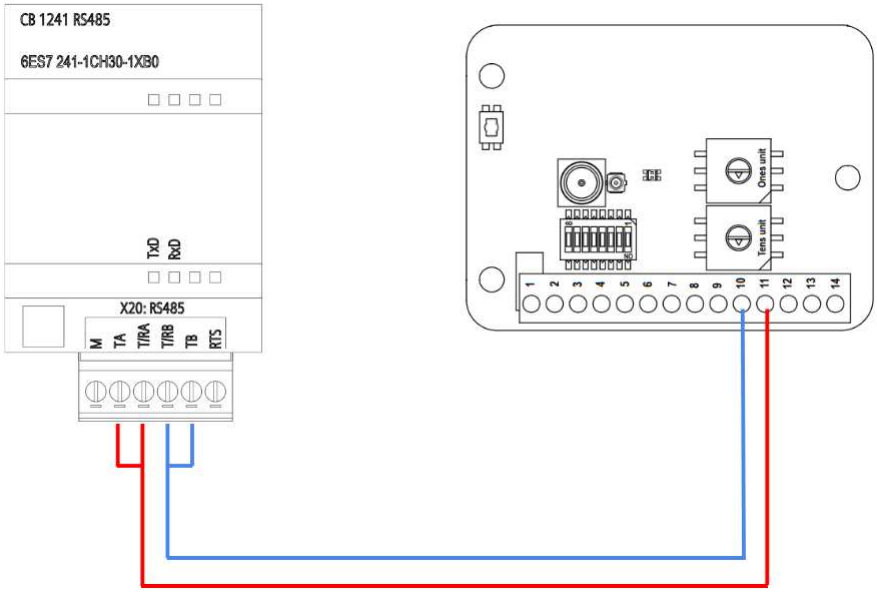
Fonte: O autor

4.1.4 Módulo de comunicação Modbus CB-1241

O CLP utilizado neste projeto não possui entradas Modbus nativas em seu hardware. Para que seja possível a comunicação Modbus, deve-se utilizar um módulo de comunicação Modbus, o qual é acoplado na parte frontal do CLP para que o barramento Modbus seja ligado aos seus terminais. O método de ligação dos fios entre o módulo de comunicação e a rádio *gateway*, dispositivo escravo do sistema Modbus, é demonstrada na figura abaixo.

Os pinos 10 e 11 correspondem ao polos positivo e negativo, respectivamente, da rádio *gateway*, conforme demonstrado na Figura 14.

Figura 27 – Ligação do módulo Modbus com a rádio *gateway*



Fonte: O autor

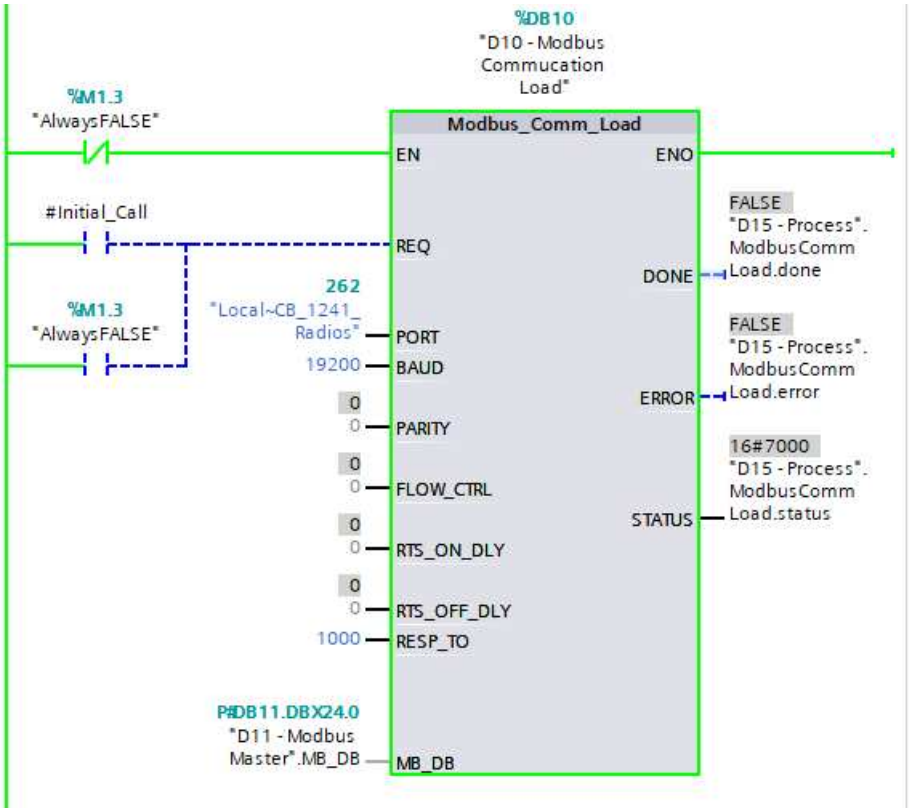
4.2 SOFTWARE - TIA PORTAL

O software responsável pelo funcionamento do sistema de acionamento dos canhões foi dividido em quatro *networks* a fim de trazer clareza e uma melhor distinção de cada processo presente no sistema. As duas primeiras *networks* tratam da inicialização e controle da comunicação Modbus do CLP com a rádio *gateway*

4.2.1 Inicialização da comunicação Modbus

A primeira *network* do programa possui a função de chamar o bloco responsável por configurar a comunicação Modbus RTU do sistema. Esse bloco é chamado de "Modbus_Comm_Load"e a partir de suas portas a configuração da comunicação Modbus RTU é realizada. O bloco possui portas de entrada e de saída nas quais serão atribuídos e coletados valores e informações referentes à comunicação.

Figura 28 – Bloco de inicialização da comunicação Modbus



Fonte: O autor

Uma instância de um bloco de dados é automaticamente criada no momento em que o bloco de comunicação é inserido no programa. Neste bloco de instância é onde é realizada a primeira configuração da comunicação Modbus. Dentro do bloco há uma variável chamada "MODE", a qual define o modo de operação da comunicação a partir do valor atribuído à variável. O modo de comunicação utilizado é Half-Duplex (RS-485) dois fios, portanto é atribuído à variável o valor 4.

Para que as entradas do bloco sejam habilitadas, a entrada EN deve estar ativa, como mostra a Figura 28. A segunda entrada chamada de requisição (REQ) tem a função de iniciar a instrução uma vez que haja uma borda de subida na entrada. Nesta entrada foi ligada uma variável de chamada inicial que, no momento em que o bloco de organização é iniciado, a variável emite um pulso de subida e portanto inicia a instrução.

A próxima entrada é utilizada para especificar o módulo de comunicação utilizado na comunicação Modbus RTU. Uma vez realizada o acoplamento do módulo, como visto na seção 4.1.4

A paridade da comunicação é configurada como *None*, portanto, adiciona-se "0"na entrada correspondente.

A próxima entrada do bloco utilizada é a entrada de tempo em milissegundos

em que o bloco Modbus_Master, o qual será abordado na próxima seção, aguarda uma resposta do escravo Modbus. Caso o escravo não dê uma resposta dentro deste tempo estipulado, o bloco repete a requisição ou então encerra a requisição com um erro.

A última entrada utilizada neste bloco é onde é referenciado a instância do bloco de dados Modbus_Master.

O bloco de comunicação disponibiliza de três saídas, entre elas estão *done*, *error* e *status*. *Done* é uma saída do tipo booleana que uma vez que um ciclo, após a última requisição ter sido feita, for completado sem erros, esta é definida como TRUE. *Error* é uma saída do tipo booleana que uma vez que um ciclo, após a última requisição ter sido feita, for completado com erros, esta é definida como TRUE. A última saída, *Status* é uma saída do tipo Word a qual exibe o código de erro encontrado durante a execução do bloco.

A fim de armazenar as informações de saída deste bloco e variáveis de controle, foi criado um bloco de dados chamado "D15 - Process", conforme mostra a Figura 29.

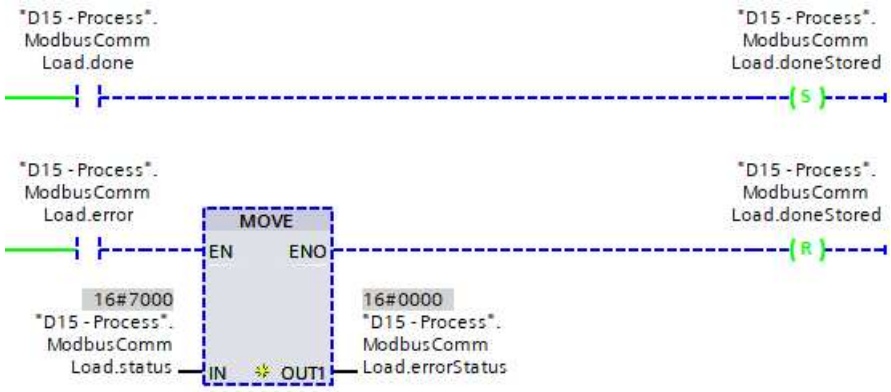
Figura 29 – Variáveis de saída do bloco de inicialização Modbus_Comm_Load

3		ModbusCommLoad	"typeModbusCom...		
4		done	Bool	FALSE	FALSE
5		error	Bool	FALSE	FALSE
6		status	Word	W#16#0	16#7000
7		errorStatus	Word	W#16#0	16#0000
8		doneStored	Bool	FALSE	TRUE

Fonte: O autor

Ao fim desta *network* é realizado o tratamento do erro, caso esse ocorra durante a execução do bloco de inicialização, conforme mostra a Figura 30. No momento em que ocorre a requisição e a saída *Done* é acionada, ou seja, definida como TRUE, a variável "ModbusCommLoad.doneStored", criada no bloco de dados "D15 - Process", é definida como TRUE. Caso haja um erro durante a execução, a variável "error"é definida como TRUE e é utilizado um bloco de instrução "MOVE"responsável por mover o conteúdo da entrada do bloco para uma variável de saída, neste caso *status* e *errorStatus* respectivamente. Uma vez que houve a transferência de conteúdo entre as variáveis a variável "ModbusCommLoad.doneStored"é reiniciada.

Figura 30 – Tratamento do erro no bloco de incialização Modbus



Fonte: O autor

4.2.2 Controle da comunicação Modbus RTU

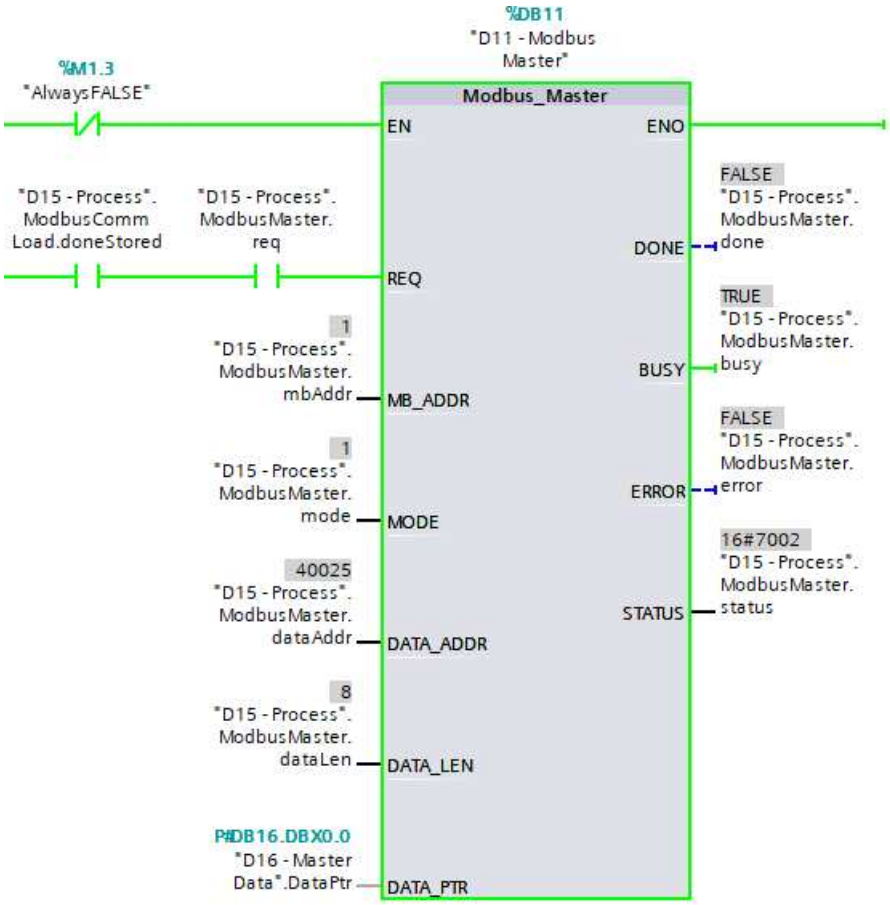
O bloco de controle da comunicação Modbus foi inserido na segunda *network* do programa, como mostra a Figura 31. Esse é responsável por controlar o envio de mensagens de leitura e escrita da comunicação Modbus RTU. O bloco de controle é chamado de "Modbus_Master"e, assim como no bloco de inicialização, sua configuração é realizada partir de suas portas de entrada.

Para que as entradas do bloco sejam habilitadas, a entrada EN deve estar ativa, assim como no bloco de inicialização. O segundo parâmetro utilizado no bloco é o de requisição. Esta entrada é acionada a partir de duas variáveis contidas dentro do bloco de dados "D15 - Process", "ModbusCommLoad.doneStored"e "ModbusMaster.req". A primeira permanece ativa a partir do momento em que a requisição no bloco de inicialização é realizada sem a presença de erros. A segunda é uma variável utilizada na máquina de estados micro responsável por habilitar requisições cíclicas de transmissão de dados entre mestre-escravo Modbus. A máquina de estados micro será abordada na terceira *network*.

O próximo parâmetro configurado no bloco é o de endereço Modbus, no qual é passado ao bloco o endereço Modbus, ou como foi chamado neste trabalho, o ID Mosbus do escravo Modbus. Como neste trabalho há somente um escravo Modbus, foi configurado o valor "1"diretamente na variável "ModbusMaster.mbAddr". Caso mais escravos fossem adicionados ao sistema, para cada requisição do bloco, o valor de endereço haveria de ser alterado para o endereço do respectivo escravo.

A variável "ModbusMaster.mode"é utilizada como entrada do próximo parâmetro para especificar se determinado dispositivo deve realizar uma operação de escrita ou leitura. Caso a variável receba valor "1"significa que o dispositivo está escrevendo no endereço pré-determinado e, caso a variável receba o valor "0", significa que o dispositivo está lendo o conteúdo do endereço Modbus pré-determinado. Esta variável será utilizada na variável

Figura 31 – Bloco de controle Modbus RTU



Fonte: O autor

micro da terceira*network*.

O próximo parâmetro do bloco é referente ao endereço do dado a ser lido ou escrito, seja pelo mestre ou pelo escravo Modbus. Os endereçamentos são configurados no bloco de dados "D35 - Radios"nas variáveis "dataAddrSend"e "dataAddrReceive", tanto do *gateway* quanto do *node*. Os endereçamentos utilizados seguem a seguinte tabela, retirada do manual de operação do *gateway*.

Figura 32 – Tabela de Holding Registers

I/O	Modbus Holding Register		I/O Type	I/O Range		Holding Register Representation	
	Gateway	Any Node		Min.	Max.	Min. (Dec.)	Max. (Dec.)
1	1	1 + (Node# × 16)	Discrete IN 1	0	1	0	1
2	2	2 + (Node# × 16)	Discrete IN 2	0	1	0	1
3	3	3 + (Node# × 16)	Discrete IN 3	0	1	0	1
4	4	4 + (Node# × 16)	Discrete IN 4	0	1	0	1
5	5	5 + (Node# × 16)	Discrete IN 5	0	1	0	1
6	6	6 + (Node# × 16)	Discrete IN 6	0	1	0	1
7	7	7 + (Node# × 16)	Reserved				
8	8	8 + (Node# × 16)	Device Message				
9	9	9 + (Node# × 16)	Discrete OUT 9	0	1	0	1
10	10	10 + (Node# × 16)	Discrete OUT 10	0	1	0	1
11	11	11 + (Node# × 16)	Discrete OUT 11	0	1	0	1
12	12	12 + (Node# × 16)	Discrete OUT 12	0	1	0	1
13	13	13 + (Node# × 16)	Discrete OUT 13	0	1	0	1
14	14	14 + (Node# × 16)	Discrete OUT 14	0	1	0	1
15	15	15 + (Node# × 16)	Control Message				
16	16	16 + (Node# × 16)	Reserved				

Fonte: O autor

A partir da tabela da imagem acima definiu-se que os valores de entrada do *gateway* são lidos a partir do valor do Holding Register 40001 e os valores de saída são escritos a partir do Holding Register 40009. Já para o Node, os valores inciais dols Holding Registers de entrada e saída são 40017 e 40025 respectivamente. Como pode-se notar, os valores de entrada e saída possuem um tamanho de 8 bits, totalizando assim 16 bits totais, conforme descrito na seção 2.4.1.1 e mostrado na Figura 33.

Figura 33 – Endereçamento ds Holding Registers de entrada e saída

D35 - Radios (snapshot created: 12/1/2021 2:09:37 PM)

	Name	Data type	Start value
1	Static		
2	Step FSM Radios	USInt	0
3	Gateway	"typeRadioSendRec...	
4	dataAddrSend	UDInt	40009
5	dataAddrReceive	UDInt	40001
6	Data2Send	"typeDataPtr"	
7	DataReceived	"typeDataPtr"	
8	Node 1	"typeRadioSendRec...	
9	dataAddrSend	UDInt	40025
10	dataAddrReceive	UDInt	40017
11	Data2Send	"typeDataPtr"	
12	DataReceived	"typeDataPtr"	

Fonte: O autor

O próximo parâmetro do bloco é onde define-se o tamanho do dado a ser transmitido. Como já foi explicado acima, o tamanho do dado é de 8 bits, portanto, o valor 8 é atribuído à variável "ModbusMaster.dataLen".

O último parâmetro de entrada é o parâmetro que aponta para o bloco de dados ou endereço de memória de um bit no qual a escrita ou leitura está sendo exeutada. Neste caso, o parâmetro aponta para o bloco de dados "D16 - MasterData". Dentro deste bloco de dados há um array de dados tamanho 8 chamado de "DataPtr", área de dados essa utilizada para comunicar com o o escravo Modbus.

Os parâmetros de saída do bloco de controle são similares aos parâmetros de saída do bloco de inicialização. Enquanto no bloco de inicialização nós temos os parâmetros "Done", "Error"e "Status", no bloco de controle temos os mesmos três parâmetros e um quarto parâmetro chamado de "Busy". Esse, indica a partir de um sinal booleano se determinada ação, seja de leitura ou escrita Modbus está em progresso.

Assim como no bloco de inicialização, todas as variáveis utilizada nas configurações dos parâmetros foram armazenadas no bloco de dados "D15 - Process", conforme mostra a Figura 34.

Figura 34 – Variáveis do bloco de controle Modbus_Master

9		ModbusMaster	"typeModbusMaster"		
10		req	Bool	FALSE	TRUE
11		mbAddr	UInt	1	1
12		mode	USInt	0	1
13		dataAddr	UDInt	0	40025
14		dataLen	UInt	8	8
15		done	Bool	FALSE	FALSE
16		busy	Bool	FALSE	TRUE
17		error	Bool	FALSE	FALSE
18		status	Word	W#16#0	16#7002
19		errorStatus	Word	W#16#0	16#0000

Fonte: O autor

4.2.3 Máquinas de estado

Uma máquina de estados finita (FSM) ou autômato de estados finito ou simplesmente uma máquina de estados é um modelo matemático de computação. É uma máquina abstrata que pode estar exatamente em um dos estados finitos em determinado momento. A FSM pode mudar de um estado para outro em resposta a determinadas entradas. A mudança de um estado para outro é chamada de transição.(WANG, 2019)

A terceira *network* é responsável por executar uma máquina de estados macro e outra máquina de estados micro. Essas controlam a comunicação com as rádios de modo geral e em específico com cada rádio respectivamente.

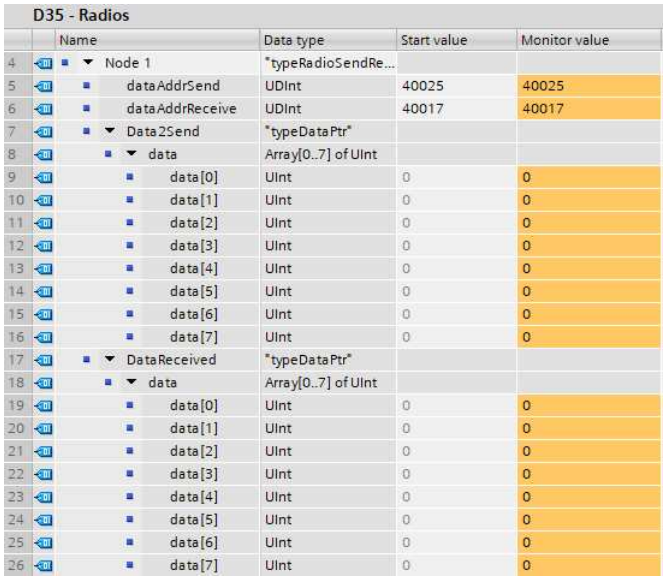
4.2.3.1 Máquina de estados macro

A máquina de estados macro tem a função de, em cada estado, chamar uma máquina de estados micro que comunica com um rádio em específico, seja ela o *gateway*

ou o *node*. Em seu primeiro estado, a máquina de estados macro chama a máquina de estados micro do gateway. A máquina de estados micro (FB FSM Send and Receive) consiste de uma máquia de quatro estados no qual o quinto estado é utilizado para sinalizar que a máquina terminou seu ciclo. No momento em que a máquina micro é chamada, o *data type* "RadioSendReceive"recebe os valores gravados no bloco de dados "D35 - Radios.Gateway". Neste momento a máquina micro estará utilizando dos valores de endereçamento, dos dados de escrita e leitura da rádio *gateway*. Uma vez que o ciclo da máquina micro se encerra, ou seja, chega ao quinto estado do ciclo, a máquina macro inicia o chamado da máquina micro do *node*. Diferentemente da situação anterior, o *data type* "RadioSendReceive"receberá os valores gravados no bloco de dados "D35 - Radios.Node 1"e portanto a máquina micro estará utilizando dos valores de endereçamento, dos dados de escrita e leitura da rádio *node*. Finalizado o ciclo da máquina micro, o ciclo reinicia, chamando assim a máquina micro do *gateway*. Os estados podem ser observado no código abaixo, assim como as estruturas dos *data types* na Figura 35.

```
1  CASE "D35 - Radios"."Step FSM Radios" OF
2      0:  // Gateway
3          "D20 - FSM Send and Receive - Mestre"(RadioSendReceive := "D35 - Radios"
       .Gateway);
4          IF "D20 - FSM Send and Receive - Mestre".Step = 5 THEN
5              "D20 - FSM Send and Receive - Mestre".Step := 1;
6              "D35 - Radios"."Step FSM Radios" := 1;
7          END_IF;
8      1:  // Node 1
9          "D21 - FSM Send and Receive - Node 1"(RadioSendReceive := "D35 - Radios"
       ."Node 1");
10         IF "D21 - FSM Send and Receive - Node 1".Step = 5 THEN
11             "D21 - FSM Send and Receive - Node 1".Step := 1;
12             "D35 - Radios"."Step FSM Radios" := 0;
13         END_IF;
14     ELSE  // Statement section ELSE
15         ;
16 END_CASE;
```

Figura 35 – DataTypes do *node*



D35 - Radios

	Name	Data type	Start value	Monitor value
4	Node 1	"typeRadioSendRe...		
5	dataAddrSend	UDInt	40025	40025
6	dataAddrReceive	UDInt	40017	40017
7	Data2Send	"typeDataPtr"		
8	data	Array[0..7] of UInt		
9	data[0]	UInt	0	0
10	data[1]	UInt	0	0
11	data[2]	UInt	0	0
12	data[3]	UInt	0	0
13	data[4]	UInt	0	0
14	data[5]	UInt	0	0
15	data[6]	UInt	0	0
16	data[7]	UInt	0	0
17	DataReceived	"typeDataPtr"		
18	data	Array[0..7] of UInt		
19	data[0]	UInt	0	0
20	data[1]	UInt	0	0
21	data[2]	UInt	0	0
22	data[3]	UInt	0	0
23	data[4]	UInt	0	0
24	data[5]	UInt	0	0
25	data[6]	UInt	0	0
26	data[7]	UInt	0	0

Fonte: O autor

4.2.3.2 Máquina de estados micro

A máquina de estados micro possui algumas funções para realizar o seu ciclo, entre elas estão definir a operação a ser realizada pelo rádio, seja de escrita ou leitura, buscar o endereço a ser lido ou escrito, escrever ou ler o dado, habilitar e desabilitar as requisições e monitorar a variável "Busy"do bloco de controle Modbus_Master.

A máquina inicia definindo o modo de operação realizado, seja ele de leitura ou escrita a partir da variável "ModbusMaster.mode"do bloco de dados "D15 - Process". Uma vez definido o tipo de operação, a máquina busca o endereço o endereço a ser utilizado. Caso a máquina esteja em modo de escrita, a variável "ModbusMaster.dataAddr"recebe o endereço de escrita presente na variável "dataAddrSend"do *dataType* "RadioSendReceive"da rádio determinada pela máquina de estados macro. A variável "DataPtr"do bloco de dados "D16 - MasterData"recebe os valores de escrita presentes no array "Data2Send"do *dataType* "RadioSendReceive". Uma vez que as variáveis de endereço e de dados a serem escritos recebem seus valores, é atribuída à variável de requisição do bloco de controle a condição "True". Caso a variável de processo "Busy"receba o valor "1", ou seja, a transmissão dos dados encontra-se em progresso, o segundo estado da máquina é chamado. No segundo estado é monitorado a variável de controle "Done", caso essa seja verdadeira, ou seja, a operação de escrita foi finalizada sem erros, é atribuído à variável de requisição a condição "False"e inicia-se o terceiro estado da máquina, o estado de leitura.

```
CASE #Step OF
    1: // Envia
        "D15 - Process".ModbusMaster.mode := 1; // Escreve
        "D15 - Process".ModbusMaster.dataAddr := #RadioSendReceive.dataAddrSend;
     // Endere o
        "D16 - MasterData".DataPtr := #RadioSendReceive.Data2Send; // Dados a
    Escrever
        "D15 - Process".ModbusMaster.req := TRUE;
        IF "D15 - Process".ModbusMaster.busy THEN
            #Step := 2;
        END_IF;
    2: // Reset envio
        IF "D15 - Process".ModbusMaster.done THEN
            "D15 - Process".ModbusMaster.req := FALSE;
            #Step := 3;
        END_IF;
```

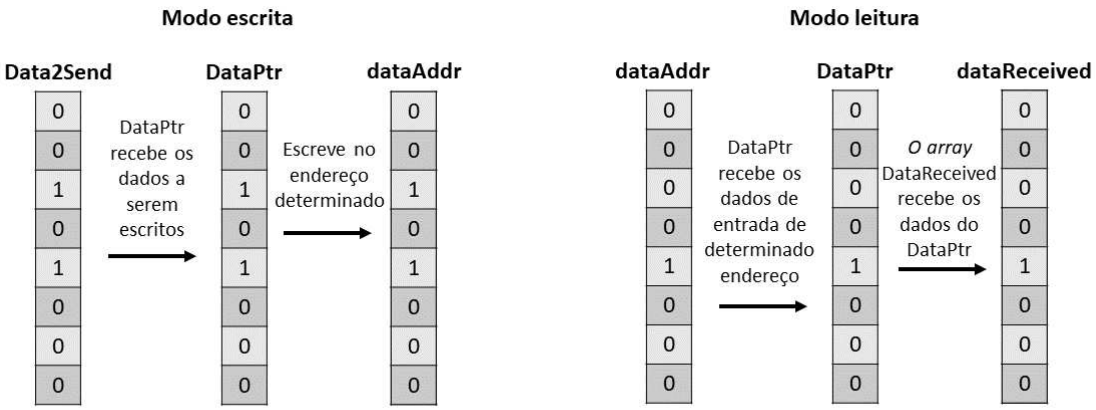
Neste estado a variável "ModbusMaster.mode"do bloco de dados "D15 - Process"recebe valor "0"indicando uma operaçao de leitura. A variável "ModbusMaster.dataAddr" recebe o endereço de leitura presente na variável "dataAddrReceive"do *dataType* "RadioSendReceive"da rádio determinada pela máquina de estados macro. Assim como no primeiro estado, é atribuído à variável de requisição do bloco de controle a condição "True". Caso a variável de processo "Busy"receba o valor "1", ou seja, a transmissão dos dados encontra-se em progresso, o quarto estado da máquina é chamado. No quarto estado é monitorado a variável de controle "Done", caso essa seja verdadeira, ou seja, a operação de leitura foi finalizada sem erros, o array "DataReceived"do *dataType* "RadioSendReceive"recebe os dados do array "DataPtr"do bloco de dados "D16 - MasterData". Após os dados de leitura serem salvos, é atribuída à variável de requisição a condição "False"e o quinto estado é chamado. O quinto estado aguarda o reinício do ciclo por parte da máquina de estados macro.

```
3: // Le
        "D15 - Process".ModbusMaster.mode := 0; // Leitura
        "D15 - Process".ModbusMaster.dataAddr := #RadioSendReceive.
    dataAddrReceive; // Endereco
        "D15 - Process".ModbusMaster.req := TRUE;
        IF "D15 - Process".ModbusMaster.busy THEN
            #Step := 4;
        END_IF;
    4: // Salvo dado lido
        IF "D15 - Process".ModbusMaster.done THEN
            #RadioSendReceive.DataReceived := "D16 - MasterData".DataPtr;
            "D15 - Process".ModbusMaster.req := FALSE;
            #Step := 5;
        END_IF;
    5:  // Aguarda reinicio externo
        ;

    ELSE  // Statement section ELSE
        #Step := 1;
END_CASE;
```

É importante observar que nos momentos de escrita e de leitura realizados na máquina micro, é utilizado o array "DataPtr"do bloco de dados "D16 - MasterData"para a transmissão dos dados. O "DataPtr"é o array resposável por carregar os valores de leitura e escrita durante a comunicação entre os dispositivos. Portanto, de uma forma resumida, quando o bloco de comunicação está em modo de escrita, os dados de transmissão são escritos no array "DataPtr"e quando o bloco está em modo de leitura os dados são lidos do array, conforme mostra a Figura 36.

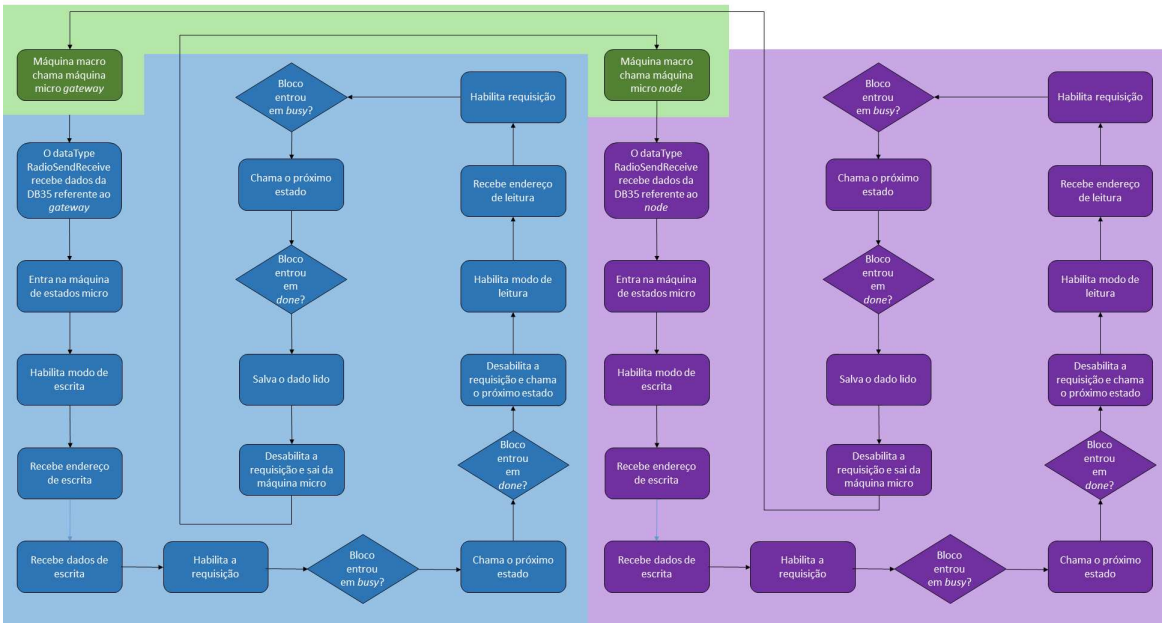
Figura 36 – Funcionamento do DataPtr



Fonte: O autor

O fluxograma da Figura 37 demonstra o funcionamento da máquina de estado macro (verde) em conjunto com a máquina de estados micro do *gateway* (azul) e com a máquina de estados micro do *node* (roxo).

Figura 37 – Sequência de funcionamento das máquinas de estado



Fonte: O autor

4.2.4 Controle de acionamentos

A última network do projeto trata-se de uma função de controle de acionamentos. Como dito ao longo deste trabalho, a rádio *gateway* possui o objetivo de disponibilizar sinais digitais enviados pelo rádio *gateway* a fim de realizar o acionamento dos canhões de névoa.

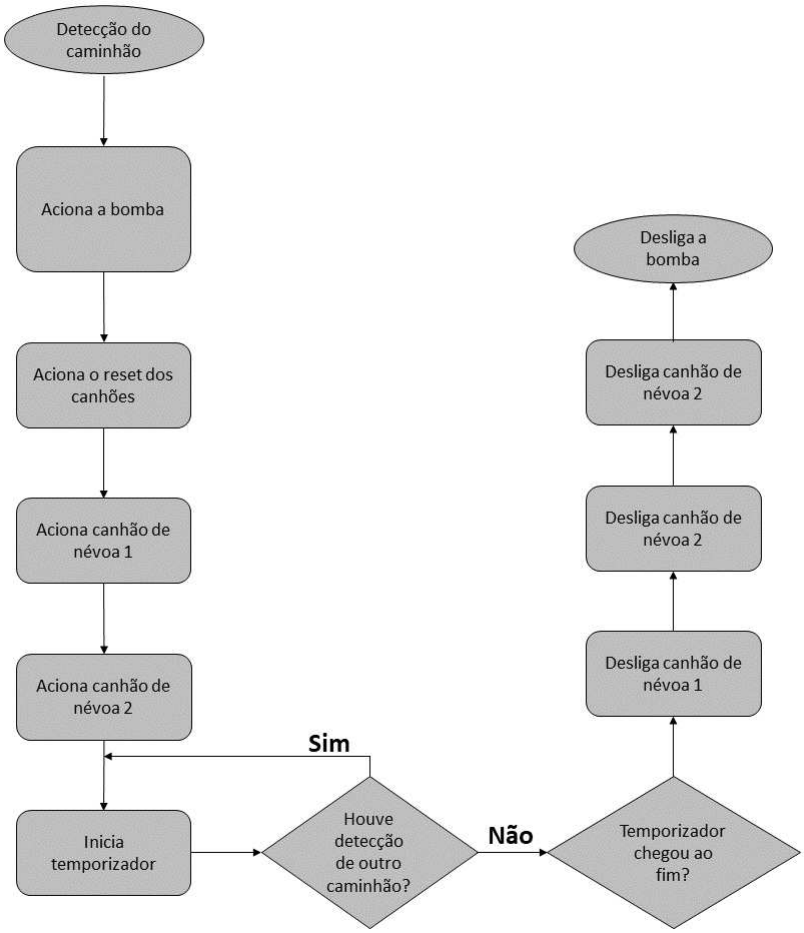
Para o correto funcionamento do sistema, foi criado, junto à equipe elétrica da empresa na qual o sistema foi instalado, uma sequência de acionamento das saídas digitais do rádio *node*. A sequência é demonstrada no fluxograma abaixo.

O algoritmo foi desenvolvido utilizando o conceito de máquina de estados, assim como o algoritmo de comunicação dos rádios. Foi pensado nesta dinâmica a fim de eximir fontes de erros entre os processos e também para um melhor diagnóstico de falhas caso haja alguma inconsistência no sistema.

O algoritmo inicia com a detecção do caminhão, conforme mostrado no fluxograma da Figura 38. O sinal digital referente ao sensor de detecção é ligado na entrada I0.1 do CLP. A fim de que este sinal possa ser monitorado no software, cria-se uma *tag* endereçada para a entrada correspondente.

Utiliza-se de um bloco de operação de tempo "TON"associado a esta variável para que o sistema seja imune à falsos acionamentos do sensor. Este bloco possui a função de somente acionar sua saída se a variável associada permanecer ativa por um tempo pré-determinado. Neste caso, foi definido um tempo de 500ms a fim de evitar que ruídos de sinal possam interferir na atuação do sistema.

Figura 38 – Sequência de acionamentos



Fonte: O autor

Figura 39 – Tag referente ao sinal digital na entrada I0.1 do CLP

PLC tags

	Name	Tag table	Data type	Address
1	sensorDetect	Standard-Variablen...	Bool	%I0.1

Fonte: O autor

Durante os acionamentos são inseridos blocos de operação de tempo entre um estado e outro. Os tempos estipulados para cada bloco foram definidos junto à equipe da empresa a fim de que as curvas de transição dos acionamentos, tanto dos canhões de névoa quanto da bomba d'água sejam respeitados.

```
1 REGION Timers
2
3     "timerOnSensor".TON(IN := "sensorDetect",
4                         PT := T#500ms);
```

```
5
6       "timerAcionamento".TON(IN := #startTimerAcionamento,
7                              PT := T#180s);
8
9       "timerPulsoLigaCanhao".TON(IN := #ligaCanhao,
10                                 PT := T#3s);
11
12      "timerPulsoDesligaCanhao".TON(IN := #desligaCanhao,
13                                    PT := T#3s);
14
15      "timerPulsoLigaBomba".TON(IN := #ligaBomba,
16                                PT := T#3s);
17
18      "timerPulsoDesligaBomba".TON(IN := #desligaBomba,
19                                   PT := T#3s);
20
21      "timerPulsoLigaReset".TON(IN := #ligaReset,
22                                PT := T#3s);
23
24      "wait1".TON(IN := #wait1,
25                  PT := T#3s);
26
27      "wait1.5".TON(IN := #"wait1.5",
28                    PT := T#5s);
29
30      "wait2".TON(IN := #wait2,
31                  PT := T#3s);
32
33  END_REGION
```

A máquina de estados responsável pelos acionamentos é composta no total por doze estados. No primeiro estado da máquina de estados, o sistema aguarda um pulso maior do que 500ms proveniente do sensor para realizar o acionamento da bomba. É criada uma condição para que o sinal do sensor não interfira nas etapas de acionamento seguintes uma vez que seu sinal já tenha sido detectado pelo sistema. O sinal do sensor só passará a ser observado novamente no nono estado do algoritmo.

```
1       IF "timerOnSensor".Q AND NOT ("GDB_Step".Step1 OR "GDB_Step".Step2 OR "
        GDB_Step".Step3 OR "GDB_Step".Step4
2           OR "GDB_Step".Step5 OR "GDB_Step".Step6 OR "GDB_Step".Step7 OR "GDB_Step
        ".Step8 OR "GDB_Step".Step9
3           OR "GDB_Step".Step10 OR "GDB_Step".Step11 OR "GDB_Step".Step12) THEN
4           "GDB_Step".Step1 := TRUE;
5           #ligaBomba := TRUE;
6       END_IF;
```

Com exceção do acionamento da bomba d'água, todos os acionamentos são realizados a partir de pulsos nas saídas digitais do rádio *node*. O tempo de cada um destes pulsos é definido nos blocos de tempo mostrados no Figura 41.

4.2.5 Resultados

O primeiro acionamento realizado é o de acionamento da bomba d'água. Para este acionamento foi designado a sexta saída do rádio *node*, ou seja, o sexto bit do *array* de dados enviados.

```
1    IF "GDB_Step".Step1 AND #ligaBomba THEN
2        "GDB_Step".Step1 := FALSE;
3        "GDB_Step".Step2 := TRUE;
4        "D35 - Radios"."Node 1".Data2Send.data[5] := 1;
5    END_IF;
```

O sistema só realiza tal acionamento caso esse não esteja em nenhum outro estado, conforme a Figura 42. O sinal digital de saída responsável pela bomba d'água permanece acionado até o fim do ciclo de acionamento.

Figura 40 – Acionamento da bomba d'água

7		Data2Send	"typeDataPtr"		
8		data	Array[0..7] of UInt		
9		data[0]	UInt	0	0
10		data[1]	UInt	0	0
11		data[2]	UInt	0	0
12		data[3]	UInt	0	0
13		data[4]	UInt	0	0
14		data[5]	UInt	0	1
15		data[6]	UInt	0	0
16		data[7]	UInt	0	0

Fonte: O autor

Passados três segundos do acionamento da bomba, o *reset* dos canhões é acionado. Para isso é emitido um pulso de 3 segundos na terceira saída do rádio *node*, conforme a Figura 41.

Figura 41 – Acionamento do botão reset

7		Data2Send	"typeDataPtr"		
8		data	Array[0..7] of UInt		
9		data[0]	UInt	0	0
10		data[1]	UInt	0	0
11		data[2]	UInt	0	1
12		data[3]	UInt	0	0
13		data[4]	UInt	0	0
14		data[5]	UInt	0	1
15		data[6]	UInt	0	0
16		data[7]	UInt	0	0

Fonte: O autor

Após 5 segundos do *reset* dos canhões, ambos são acionados ao mesmo tempo, porém utilizando saídas diferentes do rádio *node*. É utilizada a primeira e a quarta saída do rádio *node* para o acionamento do canhão 1 e do canhão 2, respectivamente. Atualmente, a ativação dos canhões ocorre simultaneamente conforme solicitado pela equipe da empresa contratante. Entretanto, em caso de mudanças futuras nessa condição, os canhões dispõem de acionamentos independentes que podem ser empregados. Assim como no *reset*, o acionamento dos canhões também é realizado a partir de pulsos de 3 segundos nas respectivas saídas do rádio *node*. No momento em que os canhões são acionados, inicia-se um temporizador o qual manterá os canhões acionados até o fim de seu tempo.

Figura 42 – Acionamento dos canhões

7		Data2Send	"typeDataPtr"		
8		data	Array[0..7] of UInt		
9		data[0]	UInt	0	1
10		data[1]	UInt	0	0
11		data[2]	UInt	0	0
12		data[3]	UInt	0	1
13		data[4]	UInt	0	0
14		data[5]	UInt	0	1
15		data[6]	UInt	0	0
16		data[7]	UInt	0	0

Fonte: O autor

Como dito anteriormente, o sinal do sensor é observado no início da rotina e no nono estado do algoritmo. O nono estado é o momento após o acionamento dos canhões. Neste momento é utilizado um bloco de operação de tempo chamado *reset timer* o qual recebe a variável de tempo de acionamento dos canhões e, neste estado é dada uma condição de que caso o temporizados de acionamento não esteja finalizado e a saída do sensor permanecer acionada por mais de 500ms o tempo de acionamento dos canhões é resetado. Deste modo, cada caminhão que passar pelo sensor possuirá o mesmo tempo de acionamento dos canhões, evitando assim desligamentos indesejados.

```
1    IF "GDB_Step".Step9 AND NOT "timerAcionamento".Q AND "timerOnSensor".Q THEN
2        RESET_TIMER(TIMER := "timerAcionamento");
3    END_IF;
```

Terminado o tempo de acionamento dos canhões, os mesmos são desligados a partir de um pulso de 3 segundos nas saídas 2 e 5 do rádio *node*, conforme mostra a Figura 43.

Figura 43 – Desacionamento dos canhões

7		Data2Send	"typeDataPtr"		
8		data	Array[0..7] of UInt		
9		data[0]	UInt	0	0
10		data[1]	UInt	0	1
11		data[2]	UInt	0	0
12		data[3]	UInt	0	0
13		data[4]	UInt	0	1
14		data[5]	UInt	0	1
15		data[6]	UInt	0	0
16		data[7]	UInt	0	0

Fonte: O autor

Por fim, desligados os canhões, é realizado o desacionamento da bomba d'água e encerra-se o ciclo.

Figura 44 – Desacionamento da bomba d'água

7		Data2Send	"typeDataPtr"		
8		data	Array[0..7] of UInt		
9		data[0]	UInt	0	0
10		data[1]	UInt	0	0
11		data[2]	UInt	0	0
12		data[3]	UInt	0	0
13		data[4]	UInt	0	0
14		data[5]	UInt	0	0
15		data[6]	UInt	0	0
16		data[7]	UInt	0	0

Fonte: O autor

5 CONCLUSÃO

Diante da análise abrangente dos temas abordados neste trabalho de conclusão de curso, foi explorada a implementação de uma solução inovadora para a identificação de caminhões e acionamento de canhões de névoa. Foi utilizado uma rede *wireless* como meio de comunicação, permitindo a transmissão dos sinais necessários de forma eficiente e flexível, dispensando a necessidade de cabos físicos. Além disso, foi adotado o protocolo Modbus RTU para a troca de informações entre o CLP e as rádios, garantindo uma comunicação padronizada e confiável.

A fim de assegurar a estabilidade e confiabilidade da comunicação *wireless*, as rádios utilizadas utilizam da técnica de transmissão chamada de salto de frequência do espectro propagado (FHSS), a qual varia a frequência de transmissão em intervalos regulares. Combinando esses elementos, foi alcançada uma arquitetura eficiente e confiável, contribuindo para a automação industrial e o controle preciso de névoa em determinados ambientes.

Através do desenvolvimento da lógica de controle da comunicação Modbus e dos acionamentos dos canhões, descrita de forma detalhada na seção 4 do trabalho, foi possível estabelecer uma solução eficiente e funcional para o propósito proposto. A utilização da rede *wireless* e das rádios *wireless* demonstrou-se adequada para viabilizar a transmissão dos sinais necessários, enquanto a aplicação do protocolo Modbus possibilitou a comunicação efetiva entre os dispositivos envolvidos.

Com base nos resultados obtidos, pode-se concluir que a arquitetura proposta oferece uma solução viável e eficaz para a identificação de caminhões e o acionamento dos canhões de névoa. Além disso, a utilização dos conceitos e tecnologias abordados no trabalho proporciona benefícios como maior flexibilidade, escalabilidade e confiabilidade no sistema.

Por fim, destaca-se a importância de aprimorar e expandir essa arquitetura, considerando possíveis variações de cenários e necessidades específicas. A continuidade dos estudos e a implementação de melhorias podem contribuir para o aperfeiçoamento contínuo do sistema, tornando-o cada vez mais eficiente e adequado às demandas do contexto em que será aplicado. Espera-se que este trabalho possa servir como base para o desenvolvimento e aprimoramento de sistemas similares, contribuindo para o avanço tecnológico e a melhoria dos processos industriais.

REFERÊNCIAS

ANDERSON, J.; AULIN, T.; SUNDBERGM, C. **Digital Phase Modulation**. [*S.l.*], 1986.

ANDREA GOLDSMITH. **WIRELESS COMMUNICATIONS**. Stanford: Cambridge University Press.

AULIN, T.; SUNDBERGM, C. **Continuous Phase Modulation-Part I: Full Response Signaling**. [*S.l.*], 1981. Disponível em: https://ieeexplore.ieee.org/document/1095001. Acesso em: 19 mar. 2023.

BANNER. **R-GAGE® QT50R-AFH Sensor**. [*S.l.*], 2020. Disponível em: https://info.bannerengineering.com/cs/groups/public/documents/literature/162359.pdf.

BANNER. **Sure Cross® DX80 Performance Wireless I/O Networks**. [*S.l.*], 2020. Disponível em: https://info.bannerengineering.com/cs/groups/public/documents/literature/132607.pdf.

BANNER. **Sure Cross® Performance PB2 Gateway Board Module**. [*S.l.*], 2020. Disponível em: https://info.bannerengineering.com/cs/groups/public/documents/literature/163211.pdf.

BIN, L.; HO, P. **Data-Aided Linear Prediction Receiver for Coherent DPSK and CPM Transmitted over Rayleigh Flat-Fading Channels**. [*S.l.*], 1999. Disponível em: https://ieeexplore.ieee.org/document/775371. Acesso em: 19 mar. 2023.

CONTRIBUTORS, Wikipedia. **Phase-shift keying**. [*S.l.*], 2022. Disponível em: https://en.wikipedia.org/w/index.php?title=Phase-shift_keying&oldid=1105658467. Acesso em: 27 set. 2022.

HANNA, Katie Terrell. **Frequency-hopping spread spectrum (FHSS)**. [*S.l.*], 2021. Disponível em: https://www.techtarget.com/searchnetworking/definition/frequency-hopping-spread-spectrum. Acesso em: 27 set. 2022.

HERATH, H.; ARIYATHUNGE, S.; PRIYANKARA, H. **Development of a Data Acquisition and Monitoring System Based on MODBUS RTU Communication Protocol**. [*S.l.*], 2020. Disponível em: https://www.researchgate.net/profile/Kasun-

Herath/publication/342513211_Development_of_a_Data_Acquisition_and_Monitoring_System_Based_on_MODBUS_RTU_Communication_Protocol/links/5ef8add6299bf18816edf2dc/Development-of-a-Data-Acquisition-and-Monitoring-System-Based-on-MODBUS-RTU-Communication-Protocol.pdf.

HUDEDMANI, Mallikarjun G.; M, Umayal R; KABBERALLI, ShivaKumar; HITTALAMANI, Raghavendra. **Programmable Logic Controller (PLC) in Automation**. [*S.l.*], 2007. Disponível em: https://journals.aijr.org/index.php/ajgr/article/view/185/77.

IEC. **IEC 61131**. [*S.l.*], 2013.

IEEE. **Wireless M Medium Access Control and Physical Layer Specification 802.11b**. [*S.l.*], 1999. Disponível em: https://standards.ieee.org/ieee/802.11/10548/. Acesso em: 27 set. 2022.

KRAUS, Daniel. **Spread spectrum direct sequence**. [*S.l.*], 2003. Disponível em: http://www.dankr.net/skola/spread_spectrum/Final20report.pdf. Acesso em: 28 set. 2022.

LIU, Feng. **Compressive Detection of Multiple Frequency-Hopping Spread Spectrum Signals**. [*S.l.*], 2014. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6824467. Acesso em: 27 set. 2022.

MODBUS. [*S.l.*], 2002. Disponível em: https://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf.

MOHINDER JANKIRAMAN. **FMCW radar design**. [*S.l.*: *s.n.*], 2018.

MORELLI, M.; BENFATTO, D.; LUCANO, D.; LUISE, M.; MENGALI, U. **Simple Non-Coherent Detectors for CPM Signals Transmitted over Rayleigh Flat-Fading Channels**. [*S.l.*], 2003. Disponível em: https://ieeexplore.ieee.org/document/1319009?arnumber=1319009. Acesso em: 19 mar. 2023.

POZAR, D.M. **Microwave engineering**. [*S.l.*], 1997.

RÄISÄNEN, Arto Letto. **Radio Engineering for Wireless Communication and Sensor Applications**. [*S.l.*], 2003. Disponível em: https://ieeexplore.ieee.org/document/9100448. Acesso em: 13 set. 2022.

SALTIÉL, Renan Mathias Ferreira. **Indústria 4.0 e Sistema Hyundai de Produção: suas interações e diferenças**. [*S.l.*], 2017. Disponível em: https://www.researchgate.net/publication/317369702_Industria_40_e_Sistema_Hyundai_de%20_Producao_suas_interacoes_e_diferencas. Acesso em: 8 set. 2022.

STOVE, A.G. **Modern FMCW radar - techniques and applications**. [*S.l.*], 2004. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1396506. Acesso em: 19 mar. 2023.

TAMBOLI; SADIK; RAWALE; MALLIKARJUN; THORAIET; RUPESH; AGASHE; SUDHIR. **Implementation of Modbus RTU and Modbus TCP Communication using Siemens S7-1200 PLC for Batch Process**. [*S.l.*], 2015. Disponível em: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7225424.

WANG, Jiacun. **Formal Methods in Computer Science**. [*S.l.*], 2019. Disponível em: https://books.google.com.br/books?id=OUyeDwAAQBAJ.

WIBERG, Bilstrup. **Wireless Technology in Industry - Applications and User Scenarios**. [*S.l.*], 2001. Disponível em: https://ieeexplore.ieee.org/document/996361. Acesso em: 27 set. 2022.

ZHU, Yixin; ZHUO, Fang; XIONG, Liansong. **Communication platform for energy management system in a master-slave control structure microgrid**. [*S.l.*], 2012. Disponível em: https://ieeexplore.ieee.org/document/6258875.

APÊNDICE A – Máquina de estados macro

```
1  CASE "D35 - Radios"."Step FSM Radios" OF
2      0:  // Gateway
3          "D20 - FSM Send and Receive - Mestre"(RadioSendReceive := "D35 - Radios"
       .Gateway);
4          IF "D20 - FSM Send and Receive - Mestre".Step = 5 THEN
5              "D20 - FSM Send and Receive - Mestre".Step := 1;
6              "D35 - Radios"."Step FSM Radios" := 1;
7          END_IF;
8      1:  // Node 1
9          "D21 - FSM Send and Receive - Node 1"(RadioSendReceive := "D35 - Radios"
       ."Node 1");
10         IF "D21 - FSM Send and Receive - Node 1".Step = 5 THEN
11             "D21 - FSM Send and Receive - Node 1".Step := 1;
12             "D35 - Radios"."Step FSM Radios" := 0;
13         END_IF;
14     ELSE  // Statement section ELSE
15         ;
16 END_CASE;
```

APÊNDICE B – Máquina de estados micro

```
CASE #Step OF
    1: // Envia
        "D15 - Process".ModbusMaster.mode := 1; // Escreve
        "D15 - Process".ModbusMaster.dataAddr := #RadioSendReceive.dataAddrSend;
     // Endere o
        "D16 - MasterData".DataPtr := #RadioSendReceive.Data2Send; // Dados a
    Escrever
        "D15 - Process".ModbusMaster.req := TRUE;
        IF "D15 - Process".ModbusMaster.busy THEN
            #Step := 2;
        END_IF;
    2: // Reset envio
        IF "D15 - Process".ModbusMaster.done THEN
            "D15 - Process".ModbusMaster.req := FALSE;
            #Step := 3;
        END_IF;
    3: // L
        "D15 - Process".ModbusMaster.mode := 0; // Leitura
        "D15 - Process".ModbusMaster.dataAddr := #RadioSendReceive.
    dataAddrReceive; // Endere o
        "D15 - Process".ModbusMaster.req := TRUE;
        IF "D15 - Process".ModbusMaster.busy THEN
            #Step := 4;
        END_IF;
    4: // Salvo dado lido
        IF "D15 - Process".ModbusMaster.done THEN
            #RadioSendReceive.DataReceived := "D16 - MasterData".DataPtr;
            "D15 - Process".ModbusMaster.req := FALSE;
            #Step := 5;
        END_IF;
    5:  // Aguarda reinicio externo
        ;

    ELSE  // Statement section ELSE
        #Step := 1;
END_CASE;
```

APÊNDICE C – Acionamentos function block

```
REGION Timers

    "timerOnSensor".TON(IN := "sensorDetect",
                        PT := T#500ms);

    "timerAcionamento".TON(IN := #startTimerAcionamento,
                           PT := T#180s);

    "timerPulsoLigaCanhao".TON(IN := #ligaCanhao,
                               PT := T#3s);

    "timerPulsoDesligaCanhao".TON(IN := #desligaCanhao,
                                  PT := T#3s);

    "timerPulsoLigaBomba".TON(IN := #ligaBomba,
                              PT := T#3s);

    "timerPulsoDesligaBomba".TON(IN := #desligaBomba,
                                 PT := T#3s);

    "timerPulsoLigaReset".TON(IN := #ligaReset,
                              PT := T#3s);

    "wait1".TON(IN := #wait1,
                PT := T#3s);

    "wait1.5".TON(IN := #"wait1.5",
                  PT := T#5s);

    "wait2".TON(IN := #wait2,
                PT := T#3s);

END_REGION

REGION Acionamentos

    // Aguarda um pulso maior que 200ms do sensor para acionar o reset dos
    canhoes
    IF "timerOnSensor".Q AND NOT ("GDB_Step".Step1 OR "GDB_Step".Step2 OR "
    GDB_Step".Step3 OR "GDB_Step".Step4
        OR "GDB_Step".Step5 OR "GDB_Step".Step6 OR "GDB_Step".Step7 OR "GDB_Step
    ".Step8 OR "GDB_Step".Step9
        OR "GDB_Step".Step10 OR "GDB_Step".Step11 OR "GDB_Step".Step12) THEN
        "GDB_Step".Step1 := TRUE;
        #ligaBomba := TRUE;
    END_IF;

    //Aciona reset | Joga o degrau para 1
    IF "GDB_Step".Step1 AND #ligaBomba THEN
        "GDB_Step".Step1 := FALSE;
        "GDB_Step".Step2 := TRUE;
        "D35 - Radios"."Node 1".Data2Send.data[5] := 1;
    END_IF;

```

```
52      //Passado o tempo de pulso do reset, joga o degrau para 0
53      IF "GDB_Step".Step2 AND "timerPulsoLigaBomba".Q THEN
54          #ligaBomba := FALSE;
55          "GDB_Step".Step2 := FALSE;
56          "GDB_Step".Step3 := TRUE;
57          #wait1 := TRUE; //INTERVALO ACIONAMENTO DA BOMBA E RESET
58      END_IF;
59
60      //Espera um tempo (wait1) para realizar o acionamento do reset
61      IF "GDB_Step".Step3 AND "wait1".Q THEN
62          #wait1 := FALSE;
63          "GDB_Step".Step3 := FALSE;
64          "GDB_Step".Step4 := TRUE;
65          #ligaReset := TRUE;
66      END_IF;
67
68      //Aciona a o reset | Joga o degrau para 1
69      IF "GDB_Step".Step4 AND #ligaReset THEN
70          "GDB_Step".Step4 := FALSE;
71          "GDB_Step".Step5 := TRUE;
72          "D35 - Radios"."Node 1".Data2Send.data[2] := 1;
73      END_IF;
74
75      //Come a a contar o tempo (wait1.5) de intervalo para o acionamento os
        canhoes
76      IF "GDB_Step".Step5 AND "timerPulsoLigaReset".Q THEN
77          #ligaReset := FALSE;
78          "D35 - Radios"."Node 1".Data2Send.data[2] := 0;
79          #"wait1.5" := TRUE; //INTERVALO ENTRE DESACIONAMENTO DO RESET E
        ACIONAMENTO DOS CANHOES
80          "GDB_Step".Step5 := FALSE;
81          "GDB_Step".Step6 := TRUE;
82      END_IF;
83
84      //Passado o tempo wait1.5 aciona variavel para ligar os canhoes
85      IF "GDB_Step".Step6 AND "wait1.5".Q THEN
86          #"wait1.5" := FALSE;
87          "GDB_Step".Step6 := FALSE;
88          "GDB_Step".Step7 := TRUE;
89          #ligaCanhao := TRUE;
90      END_IF;
91
92      //Aciona os canhoes e inicia contagem do tempo que os canhoes permanecerao
        acionados
93      IF "GDB_Step".Step7 AND #ligaCanhao THEN
94          "GDB_Step".Step7 := FALSE;
95          "GDB_Step".Step8 := TRUE;
96          #startTimerAcionamento := TRUE;
97          "D35 - Radios"."Node 1".Data2Send.data[0] := 1;
98          "D35 - Radios"."Node 1".Data2Send.data[3] := 1;
99      END_IF;
100
101     //Joga o pulso de acionamento para 0
102     IF "GDB_Step".Step8 AND "timerPulsoLigaCanhao".Q THEN
103         #ligaCanhao := FALSE;
104         "GDB_Step".Step8 := FALSE;
105         "GDB_Step".Step9 := TRUE;
```

```
106         "D35 - Radios"."Node 1".Data2Send.data[0] := 0;
107         "D35 - Radios"."Node 1".Data2Send.data[3] := 0;
108     END_IF;
109 
110     //Caso o sensor detecte outro caminhao enquanto os canhoes estejam ligados
111     //ocorre o reset no tempo de acionamento
112     IF "GDB_Step".Step9 AND NOT "timerAcionamento".Q AND "timerOnSensor".Q THEN
113         RESET_TIMER(TIMER := "timerAcionamento");
114     END_IF;
115 
116     //Aciona variavel para desligar os canhoes
117     IF "GDB_Step".Step9 AND "timerAcionamento".Q THEN
118         #startTimerAcionamento := FALSE;
119         "GDB_Step".Step9 := FALSE;
120         "GDB_Step".Step10 := TRUE;
121         #desligaCanhao := TRUE;
122     END_IF;
123 
124     //Desliga canhoes | Joga o pulso de desacionamento para 1
125     IF "GDB_Step".Step10 AND #desligaCanhao THEN
126         "GDB_Step".Step10 := FALSE;
127         "GDB_Step".Step11 := TRUE;
128         "D35 - Radios"."Node 1".Data2Send.data[1] := 1;
129         "D35 - Radios"."Node 1".Data2Send.data[4] := 1;
130     END_IF;
131 
132     //Passado o tempo do pulso, joga o pulso de desacionamento para 0
133     IF "GDB_Step".Step11 AND "timerPulsoDesligaCanhao".Q THEN
134         #desligaCanhao := FALSE;
135         "GDB_Step".Step11 := FALSE;
136         "GDB_Step".Step12 := TRUE;
137         "D35 - Radios"."Node 1".Data2Send.data[1] := 0;
138         "D35 - Radios"."Node 1".Data2Send.data[4] := 0;
139         #wait2 := TRUE;
140     END_IF;
141 
142     //Espera o tempo wait2 para realizar o desligamento da bomba
143     IF "GDB_Step".Step12 AND "wait2".Q THEN
144         "D35 - Radios"."Node 1".Data2Send.data[5] := 0;
145         "GDB_Step".Step12 := FALSE;
146         #wait2 := FALSE;
147     END_IF;
148 
149 END_REGION
```

APÊNDICE D – Main

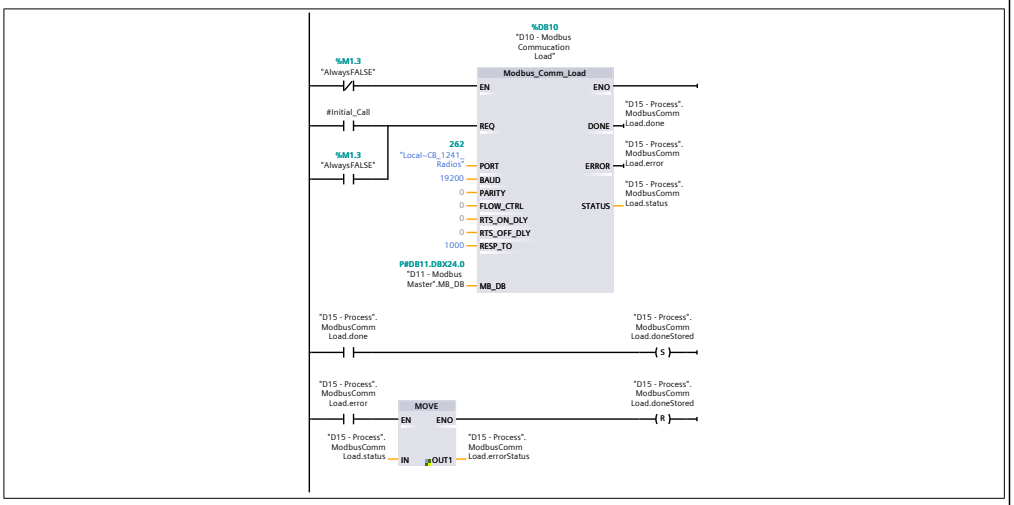
Main [OB1]

Main Properties							
General							
Name	Main	**Number**	1	**Type**	OB	**Language**	LAD
Numbering	Automatic						
Information							
Title	"Main Program Sweep (Cycle)"	**Author**		**Comment**		**Family**	
Version	0.1	**User-defined ID**					

Main			
Name	**Data type**	**Default value**	**Comment**
Input			
Initial_Call	Bool		Initial call of this OB
Remanence	Bool		=True, if remanent data are available
Temp			
Constant			

Network 1: Inicialização do Modbus

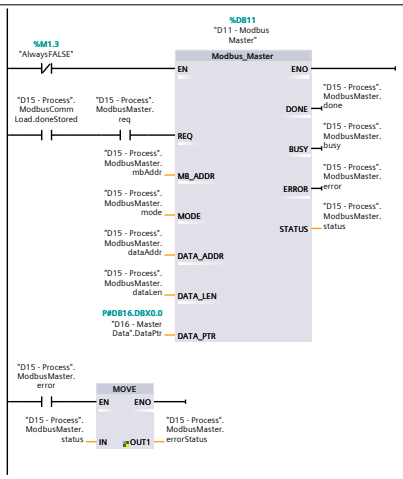
Chama o bloco que configura a comunicação Modbus RTU



Network 2: Controle do Modbus

Chama o bloco que controla o envio de mensagens de leitura e escrita via Modbus RTU

Totally Integrated Automation Portal



Network 3: Chamada da máquinas de estado que controla comunicação com a rádio

Network 4: Chamada da função de controle de acionamento