UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA – PATOS DE MINAS
ENGENHARIA ELETRÔNICA E DE TELECOMUNICAÇÕES

RAFAEL LUIZ PEREIRA DOS SANTOS

USO DA TECNOLOGIA LORA EM ROBÔS DETECTORES DE GASES

PATOS DE MINAS
2022

RAFAEL LUIZ PEREIRA DOS SANTOS

USO DA TECNOLOGIA LORA EM ROBÔS DETECTORES DE GASES

Monografia apresentada à Universidade Federal de Uberlândia como requisito para conclusão do trabalho de conclusão de curso de graduação em Engenharia Eletrônica e de Telecomunicações da Faculdade de Engenharia Elétrica.

Orientador: Prof. Dr. Daniel Costa Ramos.

PATOS DE MINAS

2022

RAFAEL LUIZ PEREIRA DOS SANTOS

USO DA TECNOLOGIA LORA EM ROBÔS DETECTORES DE GASES

Monografia apresentada à Universidade Federal de Uberlândia como requisito para conclusão do trabalho de conclusão de curso de graduação em Engenharia Eletrônica e de Telecomunicações da Faculdade de Engenharia Elétrica.

Orientador: Prof. Dr. Daniel Costa Ramos.

Patos de Minas, 27 de julho de 2022

Banca Examinadora

_____

Prof. Dr. Daniel Costa Ramos – FEELT/UFU (Orientador)

_____

Prof. Dr. Renan Alves dos Santos – FEELT/UFU (Membro 1)

_____

Prof. Dr. Karine Barbosa Carbonaro – FEELT/UFU (Membro 2)

AGRADECIMENTOS

A Deus por me ajudar a ultrapassar todos os obstáculos encontrados ao longo do curso. Aos meus familiares que me incentivaram nos momentos difíceis. Aos professores, pelas correções e ensinamentos que me permitiram apresentar um melhor desempenho no meu processo de formação. Ao Prof. Dr. Daniel Costa Ramos por financiar todos os custos do trabalho e por ter sido meu orientador e desempenhado tal função com muita dedicação.

RESUMO

Os robôs móveis têm sido utilizados para garantir mais segurança, auxiliando humanos em tarefas perigosas. A segurança é algo primordial aos trabalhadores, principalmente em ambientes perigosos onde podem estar expostos a condições extremas, como em ambientes com presença de gases tóxicos e inflamáveis. Neste cenário, a tecnologia pode tornar o trabalho em locais de alta periculosidade mais seguros por meio de uma análise de risco contínua utilizando tecnologias como a Internet das Coisas e a robótica. Este monitoramento possibilita trazer alertas para o trabalhador, que pode se adequar a medidas de prevenção e tomar as devidas providências. Neste sentido, este trabalho tem como escopo o monitoramento remoto de emissão de gases, emulando de forma controlada a situação em indústrias, para auxiliar a segurança dos trabalhadores em ambientes com presença de gases perigosos utilizando robótica móvel e tecnologia LoRa como ferramentas, dado o desafio do uso de sensores de baixo custo e de interferências na comunicação. Dentre os fatores a serem averiguados neste trabalho está a adaptabilidade do robô no ambiente, o funcionamento da comunicação em uma certa distância e caracterização dos sensores na medição de gases. Ao final deste projeto foi obtido um protótipo inicial funcional capaz de se locomover de forma autônoma no ambiente enviando informações de sua geolocalização e dos gases a serem monitorados para o usuário, para que em um trabalho futuro o mesmo possa ser inserido e avaliado em um ambiente com características industriais.

Palavras-chave: robótica móvel. IoT. LoRa. Segurança. Indústria 4.0.

ABSTRACT

Mobile robots have been used to ensure more safety by assisting humans in dangerous tasks. Safety is paramount for workers, especially in hazardous environments where they may be exposed to extreme conditions, such as in environments with the presence of toxic and flammable gases. In this scenario, technology can make work in highly hazardous locations safer through continuous risk analysis using technologies such as IoT and robotics. This monitoring makes it possible to bring alerts to the worker, who can adjust to preventive measures and take appropriate action. In this sense, this work has as scope the remote monitoring of gas emissions, emulating in a controlled way the situation in industries to assist the safety of workers in environments with the presence of hazardous gases using mobile robotics and LoRa technology as tools, given the challenge of using low-cost sensors and interference in communication. Among the factors to be investigated in this work is the adaptability of the robot in the environment, the operation of communication over a certain distance, safety, and the characterization of the sensors in the measurement of gases. At the end of this project, we achieved an initial functional prototype capable of moving autonomously in the environment sending information about its geolocation and the gases to be monitored to the user, so that in a future work it can be inserted and evaluated in an environment with industrial characteristics.

Keywords: mobile robotics. IoT. LoRa. Security. Industry 4.0.

LISTA DE SIGLAS

ADSL	*Asymmetrical Digital Subscriber Line*
API	*Application Programming Interface*
DHCP	*Dynamic Host Configuration Protocol*
DNS	*Domain Name System*
FTP	*File Transfer Protocol*
GLP	Gás Liquefeito de Petróleo
GSM	*Groupe Special Mobile*
GPS	*Global Positioning System*
GND	*Ground*
HTTP	*HyperText Transfer Protocol*
I2C	*Inter Integrated Circuit*
ISO	*International Organization for Standardization*
LIDAR	*Light Detection and Ranging*
MISO	*Master Input Slave Output*
MOSI	*Master Output Slave*
PID	*Photoionization Detectors*
RMLD	*Remote Methane Leak Detector*
RX	*Receiver*
SCK	*Serial Clock*
SMS	*Short Message Service*
SPI	*Serial Peripheral Interface*
SS	*Slave Select*
TX	*Transmitter*
UART	*Universal Asynchronous Receiver/Transmitter*

LISTA DE FIGURAS

Figura 1.1 - Robô Colossus 16
Figura 1.2 - BotCH4 22
Figura 1.3 - Robô detector de acetona misturado com ar, rBot 23
Figura 1.4 - Drone detector de gás metano 25
Figura 2.1- Robô antibomba, Cobham tEODor 29
Figura 2.2 - Robôs na linha de produção de carros 30
Figura 2.3 - Classificação dos Robôs 30
Figura 2.4 - Componentes de um Robô 30
Figura 2.5 - Robô móvel autônomo, capina 31
Figura 2.6 - Exemplos de sensores utilizados na robótica 32
Figura 2.7 - Tipos de placa Arduino 33
Figura 2.8 - Estrutura de barramento do protocolo I2C 34
Figura 2.9 - Exemplos de sensores internos 34
Figura 2.10 - Exemplos de sensores externos 35
Figura 2.11 - Esquemático do sensor eletroquímico 35
Figura 2.12 - Esquemático do sensor semicondutor 35
Figura 2.13 - Espectro dos gases 36
Figura 2.14 - Sensor MQ 37
Figura 2.15 - Encapsulamento de um sensor MQ 37
Figura 2.16 - Sensores MQ 37
Figura 2.17 - *Encoder* Óptico 39
Figura 2.18 - Princípio de funcionamento do sensor ultrassônico 39
Figura 2.19 - Arquitetura de rede cliente-servidor 40
Figura 2.20 - Arquitetura de rede P2P 41
Figura 2.21 - Modelo OSI 41
Figura 2.22 - Modelo TCP/IP 42
Figura 2.23 - Taxonomia de redes sem fio 43
Figura 2.24 - Principais tecnologias sem fio e suas características 45
Figura 2.25 - Topologia em estrela de rede LPWAN 45
Figura 2.26 - Sinal up-chirp (Amplitude x Tempo) 46
Figura 2.27 - Sinal Down-Chirp (Amplitude X Tempo) 46
Figura 2.28 - Sinal *up-chirp* (Frequência X Tempo) 47

Figura 2.29 - Sinal *down-chirp* (Frequência X Tempo) 47
Figura 2.30 - Frame LoRa 47
Figura 2.31 - Relação de cada SF com o tempo de transmissão 48
Figura 2.32 - Diferenças entre LoRa e LoRaWAN no modelo OSI 49
Figura 2.33 - Topologia LoRaWAN 49
Figura 2.34 - Arquitetura LoRaWAN 50
Figura 2.35 - Sentido e orientação do vetor deslocamento 52
Figura 2.36 - Projeção de um vetor. 52
Figura 2.37 - Deslocamento do robô 54
Figura 3.1 - Chassi que será utilizado no projeto 55
Figura 3.2 - Módulo ponte H TB6612FNG 56
Figura 3.3 - Módulo *shield* LoRa 56
Figura 3.4 - LoRa *gateway* LG01-P 57
Figura 3.5 - Logo do *ThingSpeak* 58
Figura 3.6 - Visualização de dados no *ThingSpeak* 59
Figura 3.7 - Interação entre *ThingSpeak* e Matlab 59
Figura 3.8 - Id e API do canal no *ThingSpeak* 60
Figura 4.1 - Fluxograma do projeto 62
Figura 4.2 - Chassi do Robô desmontado 63
Figura 4.3 - Chassi Montado 63
Figura 4.4 - Rodas de condução fixadas 64
Figura 4.5 - *Encoder* fixado na roda de condução 64
Figura 4.6 - Sensores ultrassônicos fixados 65
Figura 4.7 - Sensor de gás fixado 65
Figura 4.8 - Protoboard com Arduino e ponte H conectados 66
Figura 4.9 - Estrutura completamente montada 66
Figura 4.10 - Pinagens da ponte H TB6612FNG 67
Figura 4.11 - *Jumpers* soldados nas conexões dos motores 68
Figura 4.12 - Esquemático do circuito dos motores 68
Figura 4.13 - Movimento do robô para frente 70
Figura 4.14 - Movimento do robô para direita 71
Figura 4.15 - Movimento do robô para a esquerda 72
Figura 4.16 - Movimento do robô para trás 72
Figura 4.17 - Esquemático do circuito dos sensores ultrassônicos 73

Figura 4.18 - *Encoder* óptico LM293 78
Figura 4.19 - Esquemático do circuito do *encoder* óptico 78
Figura 4.20 - Sensor MQ-6 83
Figura 4.21 - Esquemático de conexão do sensor na *shield* 84
Figura 4.22 - Emitindo gás no sensor diretamente 84
Figura 4.23 - Sensor MQ-6 85
Figura 4.24 - Diagrama de blocos das funcionalidades dos Arduinos 86
Figura 4.25 - Esquemático de ligação dos pinos dos Arduinos 86
Figura 4.26 - Estrutura do pacote LoRa 90
Figura 4.27 - Conexão do cabo rj45 97
Figura 4.28 - IP padrão do *gateway* 98
Figura 4.29 - Pagina inicial de configuração do *gateway* 98
Figura 4.30 - Configuração de Internet no *gateway* 98
Figura 4.31 - Configurações do LoRa no *gateway* 99
Figura 4.32 - Download da placa dragino do *gateway* 99
Figura 4.33 - Instalação da placa na IDE Arduino 100
Figura 4.34 - Seleção da placa LG01 para programar 100
Figura 4.35 - Carregamento de um código no MCU do *gateway* 101
Figura 4.36 - Estrutura do pacote de dados recebido 103
Figura 4.37 - Mensagens enviadas pelo nó LoRa e recebidas no *gateway* 104
Figura 4.38- Página de login no *ThingSpeak* 105
Figura 4.39 - Configuração do canal no *ThingSpeak* 105
Figura 4.40 - Chaves de escrita e leitura no *ThingSpeak* 106
Figura 4.41 - Código MATLAB no *ThingSpeak* 106
Figura 4.42 - Ilustração do vetor de dados recebidos no *gateway* 107
Figura 5.1 - *Gateway* LoRa no escritório da granja 112
Figura 5.2 - Percurso esperado do robô 113
Figura 5.3 - Robô desviando do obstáculo 113
Figura 5.4 - Local final de locomoção do robô 114
Figura 5.5 - Gráfico de concentração de gás com relação ao tempo 114
Figura 5.6 - Gráfico de deslocamento em X com relação ao tempo 115
Figura 5.7 - Gráfico de deslocamento em Y com relação ao tempo 115
Figura 5.8 - Trajetória do robô 116
Figura 5.9 - Debug de pacotes LoRa recebidos 116

LISTA DE TABELAS E QUADROS

Tabela 2.1- Sensores MQ e os gases nos quais são sensíveis.....38
Tabela 2.2 - Tipos e principais características de redes sem fio.....44
Tabela 3.1 - Materiais e custos, valores de 2020.....61
Quadro 4.1 - Referência das pinagens da ponte H no código.....69
Quadro 4.2 - Código de controle dos motores.....69
Quadro 4.3 - Código de teste de funcionamento dos motores.....70
Quadro 4.4 -Código de locomoção para frente.....71
Quadro 4.5 - Código de locomoção para direita.....71
Quadro 4.6 - Código de locomoção para esquerda.....72
Quadro 4.7 - Código de locomoção para trás.....73
Quadro 4.8 - Variáveis necessárias para programar os sensores ultrassônicos.....74
Quadro 4.9 - Código das funções de leitura dos sensores ultrassônicos.....75
Quadro 4.10 - Armazenamento dos valores lidos pelos sensores ultrassônicos.....76
Quadro 4.11 - Código referente a autonomia do robô.....77
Quadro 4.12 - Código da função de interrupção.....79
Quadro 4.13 - Lógica de contagem de pulsos.....80
Quadro 4.14 - Lógica para se obter informação de sentido do robô.....80
Quadro 4.15 - Valores de partida do robô.....81
Quadro 4.16 - Código de acréscimo do contador.....81
Quadro 4.17 - Código de decréscimo do contador.....82
Quadro 4.18 - Código de conversão de contador para ângulo.....82
Quadro 4.19 - Código para zerar o ângulo.....83
Quadro 4.20 - Armazenamento da leitura do sensor de gás.....84
Quadro 4.21 - Inclusão da biblioteca para comunicação I2C.....86
Quadro 4.22 - Inicialização da comunicação I2C.....87
Quadro 4.23 - Função para enviar dados serial.....87
Quadro 4.24 - Endereço no barramento I2C.....87
Quadro 4.25 - Função a ser executada quando chegar dados via I2C.....88
Quadro 4.26 - Armazenamento dos dados recebidos via I2C.....88
Quadro 4.27 - Importação do driver RH_RF95.....88
Quadro 4.28 - Configuração da frequência de operação do LoRa.....89

Quadro 4.29 - Configuração da potência do transmissor 89
Quadro 4.30 - Verificação de inicialização do driver..... 89
Quadro 4.31 - Configuração do spreanding factor 90
Quadro 4.32 - Identificador do nó LoRa 90
Quadro 4.33 - Armazenamento dos bytes da informação do gás 91
Quadro 4.34 - Funções para obter os bytes de deslocamento..... 92
Quadro 4.35 - Função para obter dados de deslocamento do robô..... 92
Quadro 4.36 - Funções para obter CRC 93
Quadro 4.37 - Vetor de dados para envio via LoRa 94
Quadro 4.38 - Armazenamento dos dados a serem enviados 94
Quadro 4.39 - Tamanho do vetor de dados 95
Quadro 4.40 - Armazenamento de CRC..... 95
Quadro 4.41 - Logica para mostrar os dados enviados com CRC no monitor serial 95
Quadro 4.42 - Armazenamento de "data" no vetor de envio de dados via LoRa..... 95
Quadro 4.43 - Lógica para obter os bytes de CRC..... 96
Quadro 4.44 - Logica para mostrar os dados enviados sem CRC no monitor serial..... 96
Quadro 4.45 - Função de envio de dados via LoRa..... 96
Quadro 4.46 - Lógica de envio e recebimento de mensagens via LoRa 97
Quadro 4.47 - Importação do driver RH_RF95 e configurações iniciais do LoRa no *gateway* 101
Quadro 4.48 - Habilitação da comunicação entre Linux e MCU 101
Quadro 4.49 - Habilitação da biblioteca serial para mostrar dados de comunicação entre Linux e MCU 102
Quadro 4.50 - Lógica de recebimento e envio de dados via LoRa no *gateway* 102
Quadro 4.51 -Armazenamento dos dados de CRC recebidos do nó LoRa..... 103
Quadro 4.52 - Logica para obter os 16 bits de CRC recebidos 103
Quadro 4.53 - Lógica de verificação de mensagens válidas recebidas 104
Quadro 4.54 - Lógica de armazenamento dos dados recebidos do nó LoRa..... 108
Quadro 4.55 - Lógica para obter os 16 bits da informação do sensor de gás..... 108
Quadro 4.56 - Funções para obter os valores de 32 bits do deslocamento em X e em Y 109
Quadro 4.57 - Armazenamento da chave de escrita do *ThingSpeak* 109
Quadro 4.58 - Logica de concatenação dos campos para construir a URL..... 110
Quadro 4.59 - String URL com cabeçalho https 110
Quadro 4.60 - Concatenação da URL com a chave de escrita 110

Quadro 4.61 - Concatenação da URL com dataString 110
Quadro 4.62 - Adicionamento da URL como parâmetro para ser executada..... 111
Quadro 4.63 - Execução do processo com a URL..... 111
Quadro 4.64 - Lógica para mostrar mensagens no monitor serial enquanto houver saída do Linux..... 111

SUMÁRIO

1 INTRODUÇÃO 16

1.1 TEMA DO PROJETO 18
1.2 PROBLEMATIZAÇÃO 19
1.3 HIPÓTESE 20
1.4 OBJETIVOS 20
1.4.1 Objetivos Gerais 20
1.4.2 Objetivos Específicos 21
1.5 JUSTIFICATIVA 21

2 REFERENCIAL TEÓRICO 26

2.1 GASES NA INDÚSTRIA 26
2.1.1 Vazamentos e Cuidados 26
2.1.2 Principais Gases da Indústria 27
2.2 SISTEMA DE SENSORIAMENTO POR ROBÔS 29
2.2.1 Robótica Móvel 29
2.2.2 Arduino 32
2.2.3 Comunicação Serial entre Arduinos 33
2.2.4 Sensor de Gás 34
2.2.5 Sensor de Velocidade e de Distância 38
2.3 SISTEMA DE COMUNICAÇÃO SEM FIO 39
2.3.1 LoRa e LoRaWAN 46
2.4 MODELO CINEMÁTICO E ODOMETRIA 51

3 MATERIAIS E MÉTODOS 55

3.1 FERRAMENTAS UTILIZADAS 55
3.1.1 Robô do tipo esteira 55
3.1.2 Módulo ponte H TB6612FNG 55
3.1.3 Modulo LoRa *Shield* IoT (Dragino) 56
3.1.4 LG01-P LoRa Gateway (Dragino) 57
3.1.5 *ThingSpeak* 58
3.1.6 *ThingSpeak* e Matlab 59
3.2 MÉTODOS 60
3.3 CUSTOS 61

4 DESENVOLVIMENTO 62

4.1 CONSTRUÇÃO MECÂNICA 63

4.2 PROGRAMAÇÃO DOS COMPONENTES ELETRÔNICOS NOS MICROCONTROLADORES 67
4.2.1 Motores e Locomoção do Robô 67
4.2.2 Sensor ultrassônico e lógica de autonomia do robô 73
4.2.3 *Encoder* Óptico e Implementação da Interrupção 77
4.2.4 Implementação da Lógica de Deslocamento 80
4.2.5 Sensor de Gás MQ-4 e MQ-6 83
4.2.6 Comunicação entre Arduinos 85
4.2.7 *Shield* LoRa e Envio de Dados 88
4.2.8 Gateway LoRa e Recepção de Dados 97
4.2.9 Configuração do *ThingSpeak* e Matlab 105
4.2.10 Envio de Dados para o *ThingSpeak* 107
5 RESULTADOS 112
5.1 LOCOMOÇÃO E MEDIÇÃO DE GÁS EM BARRACÕES DE SUÍNOS DE TERMINAÇÃO 112
5.2 LIMITAÇÕES ENCONTRADAS 117
6 CONCLUSÃO, CONTRIBUIÇÕES E TRABALHOS FUTUROS 119
REFERÊNCIAS 120
APÊNDICE I 126
APÊNDICE II 129
APÊNDICE III 133
APÊNDICE IV 136

1 INTRODUÇÃO

Com um avanço acelerado das ferramentas de trabalho, potencializadas ainda mais pela internet, houve por consequência avanços significativos no campo das tecnologias, principalmente no setor industrial. Isso levou ao que chamamos de Indústria 4.0.

Esse termo, Industria 4.0, apareceu pela primeira vez em um artigo publicado pelo governo alemão em 2011 como sendo uma estratégia de alta tecnologia [1] baseada em automação e possue alicerces como a IoT (*Internet of Things*) e computação em nuvem [2, 3].

Esse cenário de automação que a Indústria 4.0 traz, faz com que importantes tecnologias como a robótica móvel sejam aplicadas em diversos setores, a fim de aumentar a produtividade através do trabalho cooperativo entre trabalhadores e robôs [4,5]. Além do aumento de produtividade, os robôs móveis têm sido muito utilizados para realizar tarefas que são perigosas ao homem e com isso trazer uma maior segurança aos trabalhadores, um exemplo disso é o robô Colossus (Figura 1.1) no qual foi utilizado para ajudar os bombeiros a apagar o incêndio na Catedral de Notre-Dame [6, 7].

Figura 1.1 - Robô Colossus

Fonte: [7]

Segurança é algo primordial a trabalhadores em qualquer setor, pois há muitos riscos que podem colocar a vida do trabalhador em perigo, principalmente perigos que não são perceptíveis ao sentido humano como a presença de gases tóxicos e inflamáveis. Vê-se cenários assim dentro de indústrias nas quais lidam com inúmeros processos químicos diariamente, e tem que manipular milhares de elementos e substâncias que originam gases tóxicos e inflamáveis. Neste contexto, existem muitas normas relacionadas a prevenção de explosões e incêndio em indústrias que, inclusive, estão diretamente ligadas a vazamento de gás. As normas definem por exemplo, medidas de segurança como a inspeção periódica em instalações que manipulam inflamáveis com enfoque na segurança e saúde no ambiente de trabalho [10]. A presença de gases tóxicos e inflamáveis em indústrias faz com que a preocupação de risco de vazamento de gás exija um sistema de detecção de gás para que haja um monitoramento

constante. Esse cenário traz a possibilidade de usar a Indústria 4.0 como uma ferramenta para tornar o trabalho mais seguro por meio de uma análise de risco contínua [11].

Um dos aspectos essenciais do monitoramento na Indústria 4.0, é o fornecimento de informações ao usuário ao longo do tempo, e para que isso seja efetivo, uma boa interação entre usuário e dispositivo é essencial. O avanço das tecnologias faz com que surjam discussões sobre as necessidades das empresas no ambiente de trabalho e dentre os pontos expostos, a comunicação sem fio se faz importante por causa dos seus impactos positivos, como agregar mais conectividade, mobilidade e resultados. Para isso é necessária uma infraestrutura e protocolos de comunicação, a qual tem avançado cada vez mais no contexto industrial [12, 13].

As redes sem fio surgiram com o intuito de promover mobilidade e visualização de dados independentemente de onde esteja o usuário, tendo informações transmitidas pelo ar ou espaço livre [14]. Redes sem fio estão em constante desenvolvimento, e com isso novas tecnologias e protocolos de comunicação são cada vez mais frequentes como por exemplo o Wi-Fi (IEEE 802.11 g/n), LoRaWAN, ZigBee (IEEE 802.15.4) e o *Bluetooth* (IEEE 802.15) [15].

A comunicação de dispositivos sem a necessidade de cabos é uma porta de inúmeras aplicações tanto domésticas como corporativas. Com a comunicação sem fio é possível proporcionar mais agilidade e mobilidade a fim de solucionar problemas comuns e dificuldades em produtividade, nesse sentido a IoT tem um papel importante sendo um alicerce na Indústria 4.0 e é apoiada por diversas tecnologias sem fio [12].

O termo IoT foi proposto pela primeira vez por um tecnólogo britânico chamado Kevin Ashton [16] e tem como propósito conectar dispositivos para se comunicar com um usuário e transmitir dados para uma rede. Por meio do IoT é possível detectar, controlar ou coletar informações de dispositivos remotamente [12, 17].

Vale ressaltar que dispositivos IoT apresentam necessidades de rede diferentes de computadores, telefones ou tablets. Esses dispositivos geralmente são usados para acessar dados em grandes blocos em intervalos irregulares, como por exemplo, websites e streaming de vídeos. Já no sistema IoT na maioria das aplicações, precisa enviar pequenos pacotes de dados em intervalos regulares.

A diversidade de aplicações de um sistema IoT, trouxe a possibilidade de usá-lo em chão de fábrica para processos de automação industrial. Porém, o uso de tecnologias de comunicação sem fio na qual é base de um sistema IoT em chão de fábrica, sempre foi visto com certo receio por causa da natureza não confiável e não determinística do canal de comunicação [18]. Em contrapartida a necessidade de mobilidade, modularidade, rápido desenvolvimento e redução de instalação e manutenção, fizeram com que as indústrias tivessem um grande interesse na

tecnologia sem fio. Com a evolução dessa tecnologia se proporcionou novos mecanismos para garantir confiabilidade de rede, e essas mudanças aliadas à criação de protocolos de comunicação específicos como WirelessHART, permitiu a viabilidade do uso da tecnologia sem fio nas indústrias. Esse protocolo, por exemplo, se baseia no padrão IEEE 802.15.4 e se tornou um padrão de rede industrial sem fio utilizado em controle de processos e de aplicações em indústrias [18, 19].

A grande variedade de protocolos que surgiram ao longo do tempo, trouxe a possibilidade de aumentar ainda mais o investimento de tecnologia sem fio. Devido ao ambiente industrial alguns protocolos se tornam inviáveis em chão de fábrica. Dispositivos que utilizam Zigbee, por exemplo, são vulneráveis a ruídos existentes em ambientes industriais por não possuir saltos de canal e por compartilhar mesmo canal [20], enquanto o WI-FI possui um excessivo consumo energético o que pode trazer a possibilidade de ser inviável em indústrias [18].

Em contrapartida o protocolo LoRaWAN tem se tornado popular em aplicações IoT principalmente na agricultura [21]. LoRa (*Long Range*) é uma tecnologia de rede que tem como princípio a comunicação de dados digital LPWAN (*Low Power Wide Area Network*), por meio dessa tecnologia tem-se comunicações a longas distâncias com consumo mínimo de energia e de fácil instalação, além de ter um baixo custo financeiro e possuir uma grande robustez com relação a ruídos. Para padronizar o protocolo LPWAN às especificações LoRaWAN, foi criado uma associação sem fins lucrativos chamado LoRa Alliance com o objetivo de permitir a implantação em larga escala da IoT [22].

Podemos ver que as características do protocolo LoRaWAN são semelhantes as especificações WirelessHART em quesitos de confiabilidade e eficiência para monitoramento, isso faz com que haja a possibilidade de utilizar o protocolo LoRaWAN como outra alternativa de comunicação sem fio nas indústrias.

Desta forma tem-se a possibilidade de termos um sistema IoT com tecnologia LoRa, um sistema assim pode ser atrativo a empresas de grande estrutura como indústrias caso precise de um monitoramento constante de informações.

1.1 TEMA DO PROJETO

A robótica também é um dos pilares tecnológicos da Indústria 4.0 e ao propor soluções utilizando esta tecnologia, se contribui para a modernização da indústria. Nesse contexto, este trabalho tem como objetivo o desenvolvimento de um robô móvel autônomo capaz de detectar

a presença de gases por meio de sensores com um sistema de IoT utilizando tecnologia LoRa para interação com o usuário em um ambiente fabril.

1.2 PROBLEMATIZAÇÃO

Segurança do trabalho é algo muito importante em qualquer empresa, pois é por meio de medidas de prevenção para proteção de colaboradores de uma empresa, que é possível minimizar acidentes de trabalho e doenças ocupacionais. Para se ter uma ideia, mesmo nas indústrias que possuem uma grande automação industrial, acidentes ocupacionais são um problema responsável por milhares de mortes não só no Brasil como no mundo, um exemplo disso é o desastre de Bhopal, onde ocorreu um vazamento de gás em uma fábrica de pesticidas na Índia em 1984. Esse acidente é considerado o pior desastre industrial da história [8, 9] onde teve por consequência 13 mil mortes diretas e indiretas, esse cenário de perigo que indústrias podem trazer, faz com que investimento em segurança seja algo essencial [10].

Por meio do desastre de Bhopal e outros desastres que envolveram vazamentos de gás, nota-se o quão é perigoso pessoas ficarem expostas em um ambiente que contém gases tóxicos, e ainda mais que muitos desses gases tóxicos são incolores, inodoros e insípidos aos sentidos humanos [23]. Os gases mais prejudiciais à saúde, mesmo que a pessoa fique exposta em pouco tempo, são o benzeno e amônia. Além desses também existem diversos tipos de gases que são nocivos como, por exemplo, o monóxido de carbono que além de ter toxidade, é inflamável, o que traz ainda mais a necessidade de um sistema de segurança para monitorar o ambiente.

Existem muitos dispositivos comerciais que detectam diversos tipos de gases que são utilizados em indústrias [24]. Em termos ideais, para garantir a adequada detecção de vazamento de gás em qualquer quantidade, é necessário utilizar um sensor em cada ponto onde poderia acontecer um possível vazamento, essa estratégia exige um investimento descomunal [25]. Além de precisar ter funcionários para verificar manualmente tais equipamentos várias vezes ao dia. Nesse cenário as indústrias têm apostado seus investimentos em soluções como a IoT, para se ter maior eficiência e economia de custos, além de proporcionar mais segurança aos trabalhadores e redução de riscos ambientais [26].

Uma parte fundamental em sistemas IoT são os protocolos de comunicação sem fio. No caso do ambiente industrial, existem protocolos padrão como o WirelessHART para atender as necessidades da indústria. Este protocolo oferece uma grande confiabilidade na comunicação, mas ele não é projetado especificamente para monitorar grandes áreas, como por exemplo, um campus universitário [27]. No contexto de raio de alcance de comunicação, uma tecnologia que

tem se tornado popular em sistemas IoT é o LoRa. Entretanto, a sua capacidade de transmitir informações é baixa, portanto é inapropriado fazer uma comparação direta entre os padrões WirelessHART e LoRa.

A rede LoRa tem melhor performance em aplicações que utilizam poucos dispositivos concentrados [22]. No caso de monitoramento de gases em indústrias, poderia ser inviável a utilização de uma rede LoRa onde vários sensores de gás espalhados na indústria estariam conectados. Entretanto, é possível contornar o problema do número de sensores, se for utilizado um sensor móvel para detectar algum gás específico. Nesse caso, a rede LoRa poderia se tornar uma alternativa a ser utilizada nessa aplicação, pois por meio de um sensor móvel poderia fazer verificações em todos os pontos em que se há probabilidade de vazamento de gás, de difícil acesso ou que o acesso pode trazer riscos ao ser humano. Nesse cenário, tem-se a possibilidade de usar a robótica móvel para fazer um sensoriamento móvel de gás ligado a um sistema IoT.

Por fim, considera-se que os robôs móveis já são realidade em indústrias, principalmente de manufatura, devido ao seu impacto positivo na logística, aspectos financeiros e segurança [28]. No entanto, o potencial ainda é pouco explorado, com poucos exemplos aplicados a inspeção de segurança, principalmente de forma a complementar outras formas tradicionais.

1.3 HIPÓTESE

Neste trabalho foi levantada a hipótese sobre a possibilidade de se utilizar um robô para fazer um sensoriamento móvel de gás por meio de uma rede LoRa em um ambiente fabril. Dentre as questões a serem averiguadas estão a viabilidade de se utilizar uma rede LoRa em robôs e a adaptabilidade do robô móvel para esta aplicação, levando em consideração o custo e a tecnologia da solução proposta.

1.4 OBJETIVOS

1.4.1 Objetivos Gerais

A proposta deste trabalho é estudar a viabilidade de um sistema de sensoriamento remoto de vazamento de gás por robô móvel, no qual utilizará um sistema de IoT e uma rede de comunicação com tecnologia LoRa.

1.4.2 Objetivos Específicos

Para poder realizar o estudo de viabilidade do sistema proposto, o primeiro objetivo específico a ser alcançado é obter um protótipo funcional capaz de perceber e reconhecer diversos tipos de gases.

Objetivo Específico 1: *Obter protótipo robótico funcional com sensores de gases*

O segundo objetivo a ser alcançado nesse trabalho é o de conectar o protótipo desenvolvido em uma rede LoRa, disponibilizando as informações de sensoriamento para o usuário.

Objetivo Específico 2: *Disponibilizar as informações do robô na Internet via rede LoRa*

Por fim, para atender o objetivo geral é necessário validar o sistema em um ambiente prático, mas controlado, a fim de estudar a performance do dispositivo, suas limitações e benefícios.

Objetivo Específico 3: *Validar o sistema de sensoriamento em ambiente controlado.*

1.5 JUSTIFICATIVA

Quando se instala uma indústria em alguma cidade, é prioridade fazer uma análise político-social ao invés de uma análise técnico-ambiental [29]. Isso pode se tornar um grande problema pelo fato de que na maioria das indústrias terem um grande potencial de emissão de efluentes na atmosfera. Isso faz com que coloque em risco a vida de empregados e da sociedade. Dentre os sete maiores acidentes industriais da história um deles aconteceu por causa de vazamento de gás potencialmente tóxico ao ser humano [30]. Isso mostra o quão é importante o investimento técnico-ambiental em indústrias de quaisquer setores.

Gases inflamáveis e tóxicos são muito utilizados em ambientes relativamente fechados como indústrias e residências para diversos fins, e por consequência, vazamentos de gás nesses ambientes podem se tornar realidade. Muitos trabalhos têm sido realizados a fim de criar sistemas capazes de detectar vazamentos de gás para que acidentes como explosões e intoxicação sejam evitados. Nesse contexto, foi desenvolvido em [31] um sistema inteligente de vazamento de gás GLP, o sistema foi construído utilizando um sensor MH-440D no qual é um sensor de gás do tipo infravermelho para detectar a presença do gás GLP, um microcontrolador Tiva C TM4C123G para receber e processar os dados vindos do sensor e um módulo GSM, no qual foi utilizado para notificar o usuário por meio de uma mensagem de

texto, caso ocorresse uma grande concentração de gás GLP no ambiente. Além do alarme remoto feito por meio da notificação via SMS, um alarme no ambiente foi desenvolvido utilizando um sinal luminoso feito de fitas de led e uma sirene. Para tomar uma ação depois do vazamento detectado, foi utilizado uma válvula solenoide diretamente no botijão de gás, pois caso um vazamento fosse detectado a válvula se fechava e cortava a alimentação do gás. O protótipo foi validado pois alcançou os objetivos desejadas como fazer a detecção de vazamentos de gás GLP e desenvolver um alarme local e remoto no ambiente monitorado.

Além de ambientes fechados como residências e industrias, ambientes abertos também podem trazer riscos ao ser humano por causa da presença de gases, tendo como exemplo, o aterro sanitário no qual possui grande quantidade de gás metano devido a decomposição de matéria orgânica, que apesar de não ter tanta toxidade, sua combustão parcial pode produzir substâncias tóxicas ao ser humano como monóxido de carbono [32]. Nesse sentido, foi desenvolvido em [33] um trabalho de monitoramento do gás metano em aterros sanitários utilizando um robô móvel o qual é chamado de BotCH4. O BotCH4 (Figura 1.2) é controlado remotamente por um aplicativo Android via Bluetooth e mede a concentração de gases com raio laser por meio de um sensor RMLD, sendo que os resultados da medição são enviados via SMS. Esse trabalho possui contribuições como a automatização do processo de medição do gás metano, por meio de um robô controlado à distância via *Bluetooth*, desta forma o operador pode estar em um local protegido dos riscos que o gás metano pode trazer. Apesar das contribuições mencionadas algumas limitações foram apresentadas como, por exemplo, a estrutura do robô, pois como é um robô com rodas ele se torna limitado em áreas planas com obstáculos simples. A substituição das rodas por esteiras poderia ser melhor em ambientes com tais características. Além disso, pelo fato da comunicação *Bluetooth* ter um raio de alcance pequeno, trocá-la por um tipo de comunicação que fornece um raio de alcance maior seria ideal como por exemplo o ZigBee.

Figura 1.2 - BotCH4

Fonte: [33]

Na literatura existem diversos artigos que mostram a aplicação de robôs móveis para detectar e localizar vazamentos gás. Um exemplo é o trabalho desenvolvido em [34] onde temos um robô chamado de rBot (Figura 1.3) no qual foi desenvolvido para detectar vazamento de gás em um ambiente com ar misturado com acetona. Foi utilizado rodas como mecanismo de locomoção deste robô e sensores de gás do tipo PID para detecção de gás. Já em [35] temos o desenvolvimento de um robô do tipo Braintenberg detector de fontes de odor. Um veículo Braintenberg pode se locomover automaticamente com base em suas entradas de sensor, no caso desse projeto foram utilizados dois sensores de gás TGS2600 nos quais agiam como nariz artificial. Em ambos os trabalhos, os robôs são usados em aplicações com ambientes fechados e utilizam tecnologias semelhantes, como os sensores de gás utilizados, procedimento de auto localização utilizando sensor LIDAR e execução de procedimentos de localização e mapeamento simultâneo (*SLAM – Simultaneous Localization and Mapping*) com o objetivo de se ter a localização do robô e a localização de uma possível fonte de vazamento de gás.

Figura 1.3 - Robô detector de acetona misturado com ar, rBot

Fonte: [35]

A IoT vem se destacado dentre várias tecnologias usadas para atuar em inspeções e monitoramentos constantes, devido a possibilidade da interconexão de dispositivos físicos, como por exemplo, sensores para coletar dados e informações constantemente. Muitos trabalhos têm sido desenvolvidos nessa área no contexto de monitoramento de gás. Como por exemplo o trabalho em [36], onde foi realizado o monitoramento de concentração de amônia em um galpão de aves por IoT. Esse monitoramento é importante, pois grandes concentrações de amônia podem ser prejudiciais tanto na produção avícola quanto à saúde humana. Este projeto consiste em um dispositivo capaz de medir a concentração de amônia dentro do galpão e enviar esses dados em tempo real para o sistema responsável pelo aviário, o sensor responsável pela medição de amônia utilizado foi o MQ-135 e para transmitir esses dados para o sistema um sistema web foi utilizado o módulo WiFi ESP8266. Apesar do protótipo ter

apresentado problemas devido a discrepância dos dados obtidos do sensor MQ-135, foi possível desenvolver a aplicação de maneira satisfatória.

Dentre as diversas tecnologias de rede utilizadas em sistemas IoT, uma que vem se destacado bastante é a rede LoRa. Em termos de monitoramento, foi feito um estudo da viabilidade de monitoramento constante à distância dos níveis de água dos Igaparés utilizando o protocolo LoRaWAN. O autor chega na conclusão que é possível resolver essa problemática utilizando LoRaWAN [37].

Em termos de aplicações industriais utilizando o protocolo LoRaWAN, existe um trabalho desenvolvido em [38], no setor de flores, onde uma quantidade grande de carrinhos de flores precisava se comunicar com um servidor durante sua movimentação tanto fora como dentro da área do leilão. O protocolo utilizado para fazer a comunicação foi o LoRaWAN em vista da sua vantagem de ser uma especificação aberta no topo da camada física do LoRa, isso faz com que a cobertura seja grande e os dispositivos finais sejam baratos. Na aplicação específica no setor de flores, tem-se o cenário de muitos carrinhos de flores que precisam ser monitorados durante o transporte para o cliente bem como quando residem no piso de leilão. Existe a necessidade de comunicação à longas distâncias bem como dentro do piso de leilão, onde tem-se muitos troles à serem processados, nesse cenário, o protocolo LoRaWAN se torna promissor. Os testes foram realizados utilizando um único *gateway* e um único servidor de rede em um armazém de leilão de flores, o sistema de monitoramento foi capaz de cobrir toda a área industrial.

No contexto de monitoramento de gás em ambientes industriais, existe na literatura um trabalho desenvolvido [39] no qual consiste em um sistema de baixo custo baseado em UAV (*Unmanned Aerial Vehicle*) para monitoramento de CH4 em campos de petróleo, transmitindo dados em tempo real para uma estação terrestre através do protocolo LoRaWAN. A ideia desse trabalho surgiu devido à preocupação com as emissões de gases de efeito estufa decorrentes da produção de gás e petróleo e teve como objetivo acompanhar a qualidade do ar e detectar possíveis de vazamentos de gás. O sistema foi elaborado utilizando um drone (Figura 1.4) com um sensor de CO2 (MH-Z14A), um sensor TGS2600 para detectar CH4, um sensor de temperatura e umidade DHT22 e um módulo do sistema de posicionamento global (GPS). A coleta, registro, controle e comunicação dos dados foi realizada utilizando um Arduino Mega R3, e o mesmo também foi utilizado como condutor central do nó do sensor aéreo. A estação terrestre era composta de um transceptor Dragino LoRa e uma interface gráfica para que o usuário pudesse gerenciar, salvar e visualizar os dados em tempo real. Os resultados de testes deste sistema mostraram uma grande capacidade de medir concentrações de gás metano no

ambiente durante o percurso de voo do drone, transmitindo dados em tempo real para a estação terrestre via comunicação LoRa.

Figura 1.4 - Drone detector de gás metano

Fonte: [39]

A partir dos trabalhos apresentados, nota-se as possibilidades de se utilizar um sistema IoT utilizando rede LoRa para fazer um monitoramento constante no setor industrial. No caso deste trabalho, direcionado para o monitoramento de gases com potencial de trazerem riscos ao ser humano. A fim de garantir ainda mais a segurança dos trabalhadores, é proposto a utilização de um sensoriamento remoto constante com um robô móvel.

2 REFERENCIAL TEÓRICO

Neste capítulo será apresentado toda a fundamentação teórica necessária para um bom entendimento do leitor a respeito do desenvolvimento de um robô detector de gases. Será apresentado conceitos importantes sobre gases presentes na indústria e toda a teoria da parte do desenvolvimento do *hardware* e *software* do projeto.

2.1 GASES NA INDÚSTRIA

Gases industriais são extremamente importantes em diversos tipos de indústrias, já que alguns deles são utilizados como matéria prima para a produção. O gás hidrogênio, por exemplo, é muito utilizado em indústrias química e petroquímica por causa das suas propriedades reativas e protetoras, fazendo com que essas industrias se beneficiem ao melhorar a qualidade de produtos e ao otimizar o desempenho, reduzindo custos em processos [40]. O gás nitrogênio é utilizado para preservar os sabores dos alimentos embalados para evitar oxidação, sendo também muito utilizado em indústrias de óleo e gás para estimulação de poços, injeção e testes de pressão e recuperação do petróleo [41].

Além destes exemplos, outro gás bastante utilizado em uma grande variedade de tipos de indústrias é o gás carbônico. O mesmo é utilizado na indústria alimentícia, por exemplo, para refrigeração de sorvetes, carnes e de outros alimentos. Já em indústrias de fabricação de metais, ele está presente na parte de soldagem e também está presente em indústrias medicinais em misturas metabólicas [42].

Outros tipos de gases também são utilizados na indústria como a amônia, monóxido de carbono, cloro, cianeto de hidrogênio, dióxido de enxofre, sulfeto de hidrogênio, propano, butano e metano [43].

2.1.1 Vazamentos e Cuidados

Muitos dos gases utilizados em indústrias requerem cuidados no seu manuseio pois a grande maioria pode apresentar riscos ao ser humano por serem tóxicos ou inflamáveis, isso faz com que o vazamento de gás em indústrias seja uma das maiores preocupações das empresas [44].

Muitos acidentes envolvendo vazamentos de gás em indústrias aconteceram ao longo da história e o efeito em comum ocasionado por estes acidentes foi a geração de descargas de poluentes no meio ambiente resultando assim em muitas mortes. Dentre os vários acidentes temos a explosão de uma fábrica de produtos químicos em Seveso, na Itália em 1976, no qual afetou cerca de 37.000 pessoas além de contaminar o solo em uma área de aproximadamente de 18 km². Na região de Bhopal na Índia em 1984 uma repentina emissão de isocianato de Metila para atmosfera ocasionada por uma indústria da Union Carbide, resultou a morte de 2.800 pessoas além de trazer problemas respiratórios e oftalmológicos para mais de 200.000 pessoas [45]. No Brasil em 2015 um vazamento de gás amônia em uma indústria na cidade de Mineiros – GO resultou a intoxicação de 18 funcionários que estavam trabalhando no local [46].

Por meio dos acidentes mencionados e vários outros que ocorreram, vemos o quão prejudicial é o vazamento de gás em indústrias de qualquer setor que utiliza gases em seus processos e como acidentes impactam a saúde das pessoas assim como as empresas. Isso faz com que diversas medidas sejam adotadas por empresas para se evitar esse tipo de acidente, dentre essas medidas tem-se a implementação de sistemas de monitoramento e realização de manutenções preventivas [44].

2.1.2 Principais Gases da Indústria

Existe uma infinidade de tipos de gases presentes em indústrias de diversos setores, e muito desses gases podem trazer riscos de envenenamento ao ser humano devido a toxidade, os gases tóxicos mais comuns presentes nas indústrias são detalhados a seguir.

- **Monóxido de Carbono (CO):** Inodoro, e levemente inflamável o monóxido de carbono é classificado como asfixiante pois provoca deficiência de oxigênio nos tecidos orgânicos, ele é muito utilizado em indústrias químicas. Aproximadamente 260 milhões de toneladas deste gás são produzidos anualmente no mundo todo [47, 48].
- **Amônia (NH3):** O hidróxido de amônia, mais conhecido como Amônia (NH3), é produzido através de decomposição bacteriana de compostos como proteínas e ácido úrico, é um gás essencial para fabricação de fertilizantes e para a indústria de refrigeração industrial. Este gás possui um odor característico e é bastante prejudicial em altas concentrações [47, 49].

- **Cloro (CI2):** O cloro é produzido a partir do cloreto de sódio por meio da eletrólise. A principal aplicação industrial do cloro é a fabricação de PVC (cloreto de polivilina), no qual é um tipo de plástico utilizado em escala industrial desde 1930. Além do PVC o cloro é essencial para o tratamento de água. O cloro torna-se um gás liquefeito quando está sob pressão e é um gás extremamente tóxico e possui um odor irritante [47, 50].
- **Cianeto de Hidrogênio (HCN):** O cianeto de hidrogênio (HCN) é um composto químico encontrado nas formas líquida e gasosa, ele é utilizado para produção de fibras sintéticas, plásticos, corantes e pesticidas. Sais de cianeto são cáusticos e causam sensação de queimação e irritação e quando o gás cianídrico é liberado através de queima de plástico causam asfixia [47, 51].
- **Dióxido de Enxofre (SO2):** Dióxido de enxofre é um gás incolor e solúvel na água. Está presente na produção de ácido sulfúrico, e também possui utilizações secundárias como gás para refrigeração em indústrias de bebidas e gelo, fabricação de papel sulfite e na preservação de vinhos e refrigerantes. Em concentrações altas o dióxido de enxofre provoca efeitos agudos e crônicos principalmente no coração e pulmão [47, 52].
- **Sulfeto de Hidrogênio (H2S):** Sulfeto de hidrogênio é um composto corrosivo, venenoso e gasoso em seu estado natural. É um gás muito encontrado na indústria petroquímica e de refino de petróleo e também na extração de sal. O sulfeto de hidrogênio é extremamente tóxico pois afeta os sistemas nervoso e respiratório e pode ser letal em questão de minutos [47].
- **Propano(C3H8):** O propano deriva do petróleo e tem diversas utilizações no âmbito da indústria e mesmo para o lar. É um gás inodoro, incolor e tóxico. Suas características físicas e químicas exigem que sejam mantidos dentro de sistemas rigorosamente fechados [53, 54].
- **Butano (C4H10):** O butano é um gás obtido mediante ao aquecimento lento do petróleo. É um gás altamente inflamável e deve ser manuseado de maneira correta. Esse gás é muitas vezes combinado com o gás propano para formar um novo produto chamado GLP. Este é conhecido como gás de cozinha utilizado em ambiente doméstico [55].
- **Metano (CH4):** O metano é um gás incolor e inodoro e quando adicionado ao ar, pode ser altamente explosivo. A emissão desse gás é comum em indústrias de petróleo. Investidores deste tipo de indústria enfrentam crescentes riscos financeiros, de reputação e regulação oriundos de grandes emissões de metano [54, 56].

2.2 SISTEMA DE SENSORIAMENTO POR ROBÔS

Mesmo que o termo robô pareça ser algo claro, é difícil ter uma definição precisa. Segundo o *Oxford English Dictionary* um robô é uma máquina programável capaz de executar automaticamente uma série de complexas ações [57]. Por meio dessa definição podemos ver a grande aplicabilidade de robôs em diversos tipos de ambientes e isso é ainda mais notável quando vê-se o quanto empresas estão investindo nessa tecnologia. Quase 254000 robôs foram comprados por indústrias de todo o mundo apenas em 2015 segundo a Federação Internacional de Robótica [58]. Um exemplo de aplicabilidade de robôs são os esquadrões antibombas nos quais estão cada vez mais investindo na utilização desse tipo de tecnologia para inutilizar diferentes tipos de explosivos. Um dos robôs antibombas mais famosos é o Cobham tEODor (Figura 2.1) [59].

Figura 2.1- Robô antibomba, Cobham tEODor

Fonte: [59]

Essa grande aplicabilidade da robótica em diversos setores é possível graças aos componentes físicos de um robô, como atuadores que fazem o robô agir e o uso de sensores, no qual é um componente fundamental em qualquer robô. O uso de sensores em robôs permite sua interação com o ambiente que o rodeia, pois introduz nos robôs um maior nível de inteligência para lidar com seu meio. Sensores é objeto de uma pesquisa extensa no campo de robótica [60].

2.2.1 Robótica Móvel

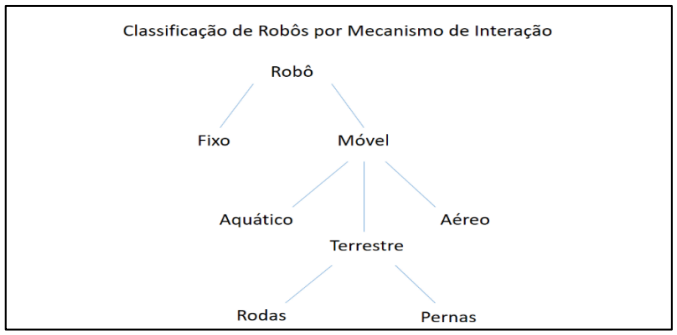
Existem dois tipos de robôs, os robôs fixos e os móveis. Robôs fixos são bastante utilizados em indústrias e executam tarefas repetitivas como soldar ou pintar peças de

automóveis (Figura 2.2) [57]. Já os robôs móveis são utilizados para executarem tarefas em ambientes grandes, mal definidos e de difícil acesso. Existem três ambientes que exigem um *design* diferentes para os robôs móveis, o ambiente aquático, ambiente terrestre e o ambiente aéreo. Os robôs móveis para estes três ambientes são divididos em subclasses. Por meio da Figura 2.3 elaborada baseada em [57], podemos visualizar melhor a divisão das classificações dos robôs.

Figura 2.2 - Robôs na linha de produção de carros

Fonte: [57]

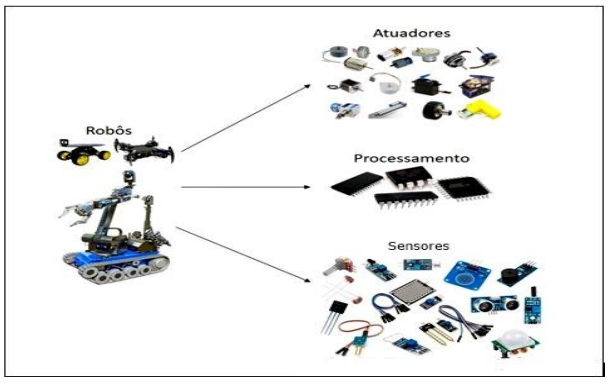
Figura 2.3 - Classificação dos Robôs



Fonte: Adaptado de [57]

Em geral os robôs são constituídos por 3 componentes: atuadores, processamento e sensores. Por meio da Figura 2.4 podemos ver uma ilustração dos componentes básicos de um robô.

Figura 2.4 - Componentes de um Robô



Fonte: Autor

Os atuadores permitem que o robô consiga operar no ambiente através do acionamento dos mecanismos de movimento como por exemplo rodas ou pernas. Esse acionamento é feito por meio da eletricidade, pneumática e hidráulica. Acionamentos elétricos geralmente são feitos utilizando motores.

Os mecanismos de interação de um robô para o mesmo se locomover, dependem do tipo de ambiente em que ele está operando, permitindo assim que o robô opere no mundo físico, por exemplo, por meio do próprio movimento das rodas de um robô terrestre. Para que esses mecanismos de interação sejam acionados, é necessário um tipo de atuador, como por exemplo, o motor de corrente contínua ou o servo motor.

Os primeiros robôs móveis foram desenvolvidos para ambientes simples, mas com a viabilidade de preços, começaram a ser desenvolvidos robôs que conseguem navegar em ambientes cheios de obstáculos de forma autônoma. Robôs autônomos tem sido muito utilizados para auxiliar profissionais em ambientes desestruturados, um exemplo disso é o robô móvel autônomo capina (Figura 2.5) [57].

Figura 2.5 - Robô móvel autônomo, capina

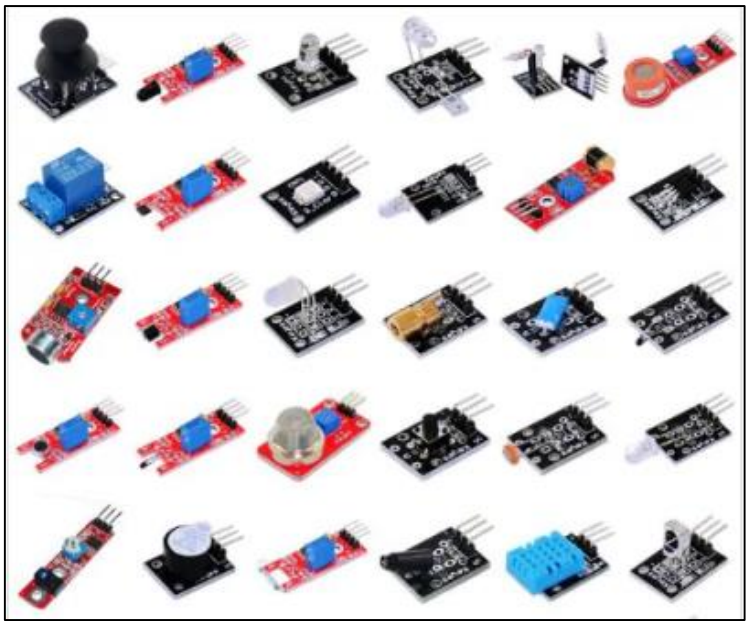
Fonte: [57]

Robôs autônomos são robôs que podem realizar objetivos desejados em ambientes desestruturados sem a ajuda humana. Isso é possível graças a utilização de sensores. Estes possibilitam que o robô perceba o ambiente físico para que o mesmo tenha informações da sua auto localização e de obstáculos que o cerca, por exemplo.

Sensores são componentes extremamente importantes em um robô, pois é por meio deles que se introduz uma maior percepção do mundo em sua volta, o que possibilita análises e

decisões mais complexas. Como há uma grande diversidade de variáveis a serem medidas, existe uma grande diversidade de tipos de sensores como mostra a Figura 2.6.

Figura 2.6 - Exemplos de sensores utilizados na robótica



Fonte: [61]

Para que o robô tome decisões com relação a informações de algum tipo de sensor ou mesmo de como se locomover no ambiente, é necessário um processamento que geralmente é feito por microcontroladores. Microcontroladores são microprocessadores que podem ser programados para muitas funções específicas, a sua estrutura interna é constituída basicamente de um processador, circuitos de memória e periféricos de entrada e saída. Os tipos mais comuns de microcontroladores utilizados na robótica são os microcontroladores Atmel AVR, PIC Microchip Technology e microcontroladores baseados em arquitetura ARM [62].

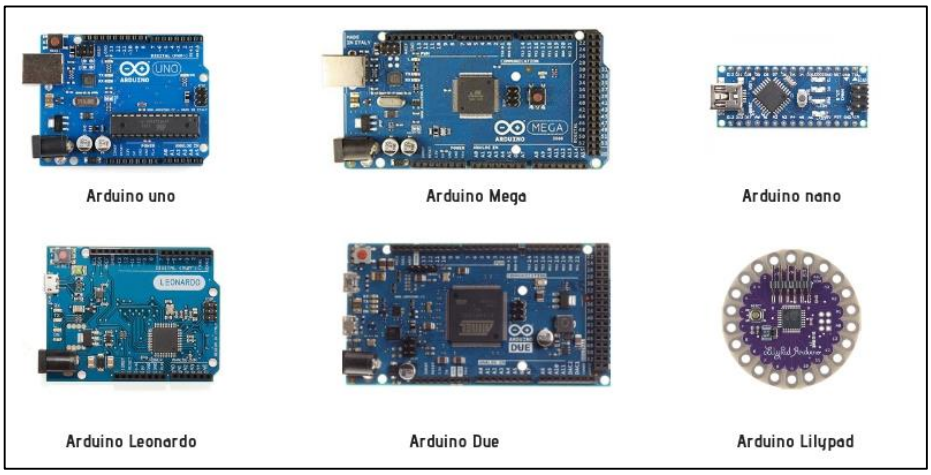
2.2.2 Arduino

O Arduino é uma placa didática que possui um microcontrolador Atmel, como o ATmega328 por exemplo, no qual é um chip que contém uma memória, processador e toda eletrônica necessária aos pinos de entrada e saída [63]. O Arduino foi criado visando o objetivo de desenvolver um dispositivo que fosse ao mesmo tempo barato, fácil de programar e bem acessível, no qual além de ter essas características ele também possui *hardware* livre, ou seja, qualquer pessoa pode modificar, montar e melhora-lo [64].

Existem vários tipos de placas Arduino (Figura 2.7), onde o que as diferencia é o modelo, tamanho e número de portas analógicas e digitais. Essas placas são facilmente conectadas ao

computador e programada via IDE (*Integrated Development Environment*) utilizando linguagem baseada em C++. As placas mais utilizadas são o Arduino UNO R3 que possui 14 portas digitais e 6 portas analógicas, o Arduino Mega que possui 54 portas digitais e o Arduino Due, no qual é baseado no processador ARM.

Figura 2.7 - Tipos de placa Arduino



Fonte: [65]

2.2.3 Comunicação Serial entre Arduinos

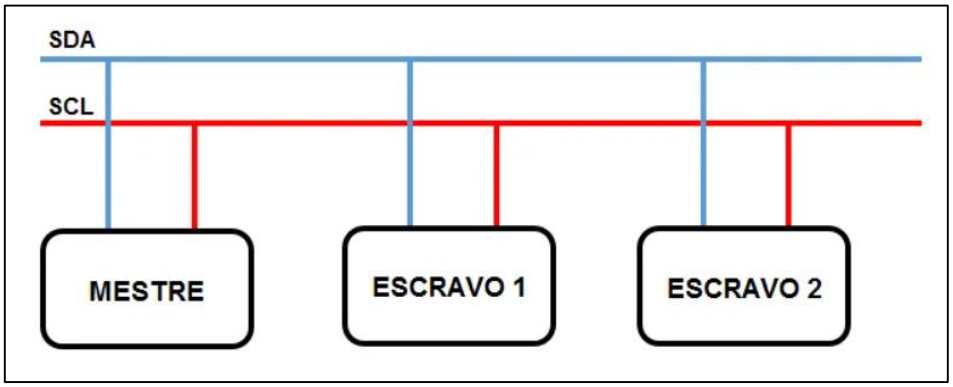
Um elemento fundamental em um sistema que possui dispositivos que interagem entre si para que um determinado resultado seja obtido, é o protocolo de comunicação. Protocolos de comunicações estabelecem regras, padrões e modelos que permitem a comunicação entre dispositivos.

No caso deste projeto o interesse maior está na comunicação entre Arduinos. Existem as possibilidades de se usar os protocolos I2C, SPI e UART, os quais são os protocolos de comunicação serial mais conhecidos.

O protocolo I2C possui um princípio de funcionamento baseado na hierarquia mestre/escravo. O mestre realiza a coordenação de toda comunicação, pois, o mesmo tem a capacidade de enviar e requisitar informações aos escravos existentes na estrutura de comunicação. Na comunicação I2C pelo menos um dispositivo deve atuar como mestre e os demais serão escravos.

A estrutura que o protocolo I2C atua é a de barramento no qual consiste em um arranjo onde todos os elementos encontram-se conectados a um ramal principal (Figura 2.8).

Figura 2.8 - Estrutura de barramento do protocolo I2C



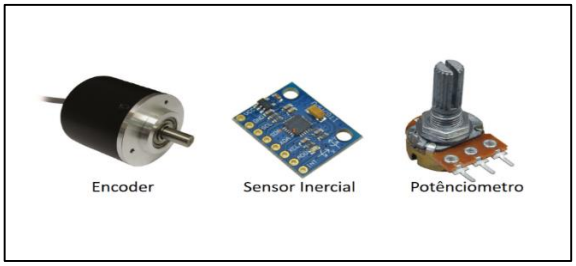
Fonte: [66]

A estrutura de barramento I2C pode ser dividida em 2 barramentos, onde um barramento é denominado de SDA (*Serial Data*) no qual é responsável pela troca de dados entre os dispositivos e o outro barramento é denominado SCL (*Serial Clock*), onde o mesmo possui a função de sincronizar os dispositivos e garantir a confiabilidade do sistema [66].

2.2.4 Sensor de Gás

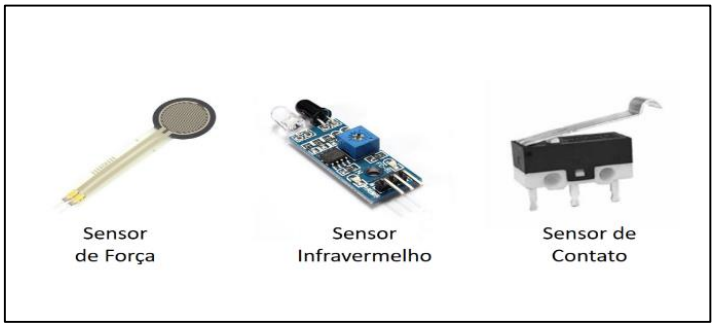
O sensor basicamente é um dispositivo que tem a função de detectar e responder com eficiência algum estímulo. E especificamente na robótica os sensores podem se dividir em duas categorias fundamentais: de estado interno e de estado externo, também encontrado na literatura sob as denominações proprioceptivos e exteroceptivos. Os sensores internos são responsáveis por fornecer informações sobre parâmetros internos do robô, como velocidade ou sentido de rotação do motor. Potenciômetros, *encoders* e sensores inerciais são exemplos de sensores internos (Figura 2.9) [60]. Os sensores externos são utilizados para que o robô lide com aspectos do mundo externo ao seu redor. Sensores de contato, de proximidade, de força, de distância, de ultrassom e sensores químicos são exemplos de sensores externos (Figura 2.10).

Figura 2.9 - Exemplos de sensores internos



Fonte: [Autor]

Figura 2.10 - Exemplos de sensores externos

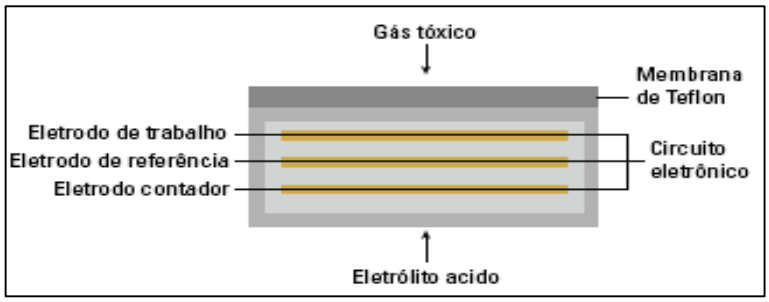


Fonte: [Autor]

Dentre os sensores de gases, existe quatro tipos principais: sensores catalíticos, sensores semicondutores, sensores infravermelhos e sensores eletroquímicos.

Os sensores eletroquímicos são formados por dois eletrodos separados por um eletrólito (Figura 2.11) e quando o gás entra em contato com o eletrodo, uma reação química acontece resultando em uma tensão elétrica entre os terminais. São bastante utilizados para detecção de gases tóxicos como CO, H2S, CI2, SO2, etc. [69,70].

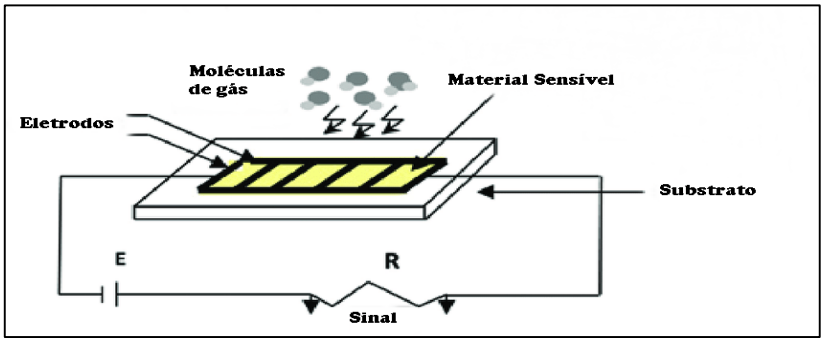
Figura 2.11 - Esquemático do sensor eletroquímico



Fonte: [71]

Os sensores semicondutores são fabricados a partir de materiais semicondutores e operam por meio da absorção de gás na superfície aquecida de um filme de óxido de metal filme (Figura 2.12), estes sensores são bastante utilizados para detectar gás sulfídrico (H2S), gás doméstico (GLP) e dióxido de carbono (CO) [69, 70].

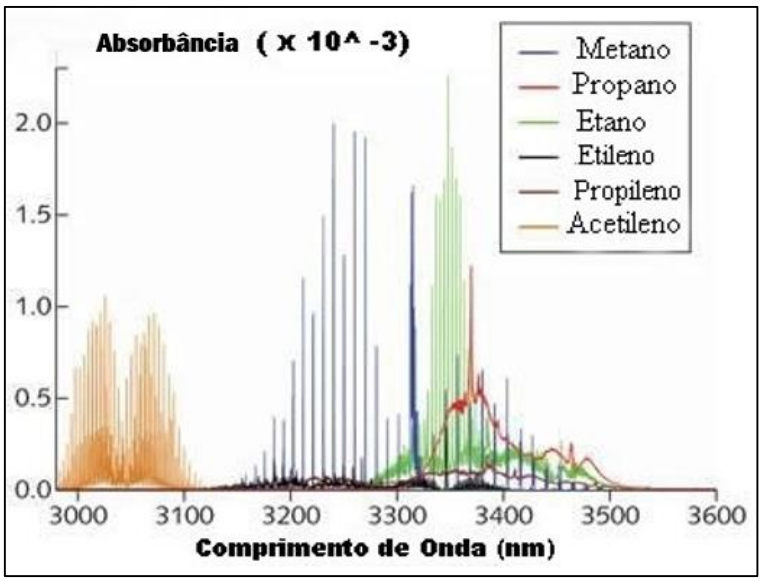
Figura 2.12 - Esquemático do sensor semicondutor



Fonte: Adaptada de [72]

Como os gases possuem um espectro de absorção de radiação infravermelha (Figura 2.13), eles também podem ser usados sensores de gás por infravermelho. O princípio de funcionamento destes sensores é a utilização de um emissor de infravermelho e um sensor de infravermelho com filtros ópticos para detectar a presença do gás a ser monitorado [69]. Estes sensores são bastante utilizados para detectar gases do tipo hidrocarbonetos como o metano (CH4) e gás carbônico (CO2) [73].

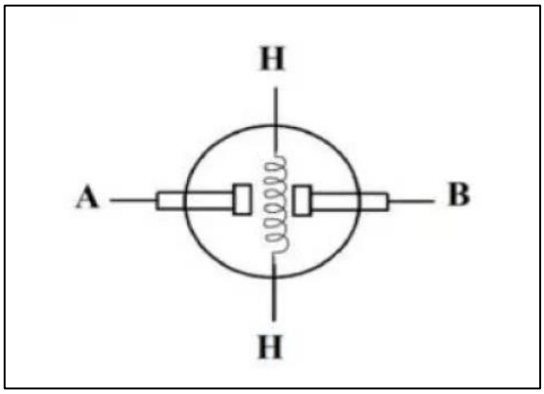
Figura 2.13 - Espectro dos gases



Fonte: Adaptado de [73]

Por fim, existem os sensores catalíticos no qual são feitos de uma bobina de fio de platina aquecido eletricamente. O elemento principal deste sensor se chama pérola e quando a mesma é aquecida através da mistura de um gás combustível com o ar, a bobina altera sua resistência fazendo com que esse sensor emita um sinal, no qual será interpretado por um dispositivo digital ou amplificado para que o conversor analógico/digital faça a leitura [69]. Este tipo de sensor é o mais utilizado por ser de baixo custo. Atualmente, a família de sensores deste tipo é conhecida como “MQ” [74]. A Figura 2.14 ilustra um como é o sensor de gás MQ por dentro. Na vertical temos uma resistência elétrica que aquece o ar, a mesma é ligada através da alimentação dos pinos H’s, um pino recebe uma tensão de 5V e o outro pino é conectado no GND (terra). De A para B tem-se o sensor de gás, em um cenário onde tem-se um gás poluidor, a resistência do sensor diminui conforme a concentração de gás aumenta. A Figura 2.15 mostra o formato de encapsulamento que é utilizado nos sensores da família MQ.

Figura 2.14 - Sensor MQ



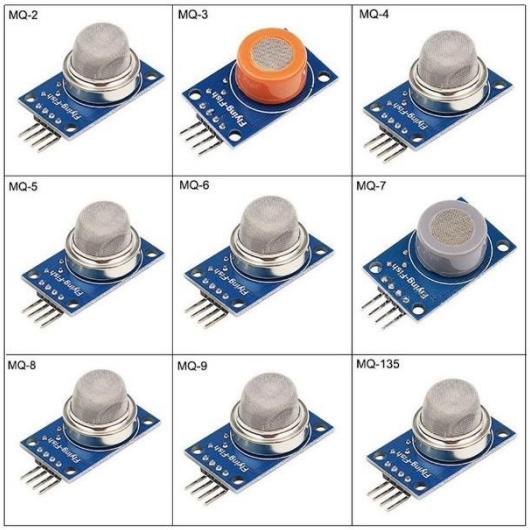
Fonte: [74]

Figura 2.15 - Encapsulamento de um sensor MQ

Fonte: [74]

Para facilitar a utilização deste sensor, é possível adquiri-lo juntamente com uma placa que disponibiliza o circuito necessário para seu funcionamento (Figura 2.16).

Figura 2.16 - Sensores MQ



Fonte: Mercado Livre

Existem vários tipos de sensores MQ's como MQ-2, MQ-3, etc. Cada um destes sensores detecta diferentes tipos de gases. Por meio da Tabela 2.1 pode-se visualizar melhor quais tipos de gases cada sensor MQ é capaz de detectar:

Tabela 2.1- Sensores MQ e os gases nos quais são sensíveis.

Nome do Sensor	Gases nos quais são sensíveis
MQ-2	Detecção de gases inflamáveis: GLP, Metano, Propano, Butano, Hidrogênio, Álcool, Gás Natural e fumaça
MQ-3	Detecção de Álcool, Etanol e fumaça
MQ-4	Detecção de Metano, Propano e Butano
MQ-5	Detecção de GLP e gás natural
MQ-6	Detecção de GLP, Propano, Isobutonato e gás natural liquefeito
MQ-7	Detecção de Monóxido de Carbono
MQ-8	Detecção de gás Hidrogênio
MQ-9	Detecção de monóxido de carbono
MQ-131	Detecção de Ozônio
MQ-135	Detecção de Gás Amônia, Óxido Nítrico, Álcool, Benzeno, Dióxido de Carbono e fumaça
MQ-136	Detecção de gás sulfídrico H2S
MQ-137	Detecção de gás Amônia
MQ-138	Detecção de n-hexano, benzeno, NH3, álcool, fumaça e CO

Fonte: Adaptada de [74]

Há um desafio em relação ao uso desses sensores, dado que é necessário fazer uma breve caracterização da operação dos mesmos em relação ao tempo para detecção, nível de partículas para detecção, repetibilidade dos resultados, dentre outras características. Além disso, muitos dos gases aqui exemplificados são tóxicos ou inflamáveis, sendo necessário um planejamento cuidadoso de segurança a fim de se evitar acidentes durante a experimentação.

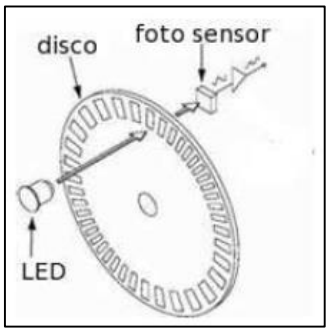
2.2.5 Sensor de Velocidade e de Distância

Enconder óptico é um sensor no qual consegue estimar a velocidade por meio de uma contagem de pulsos durante a rotação de uma roda. Isso é feito por meio de uma emissão de luz em um disco perfurado anexado no motor, à medida que esse disco rotaciona acontece inúmeras interrupções do feixe de luz.

Esse tipo de sensor de velocidade é bem sensível e está sujeito a muitos erros por causa de fatores externos como deslizamento do pneu ou imperfeiçoes no solo do ambiente.

Por meio da Figura 2.17 pode-se visualizar melhor o princípio de funcionamento e um *encoder* óptico.

Figura 2.17 - *Encoder* Óptico

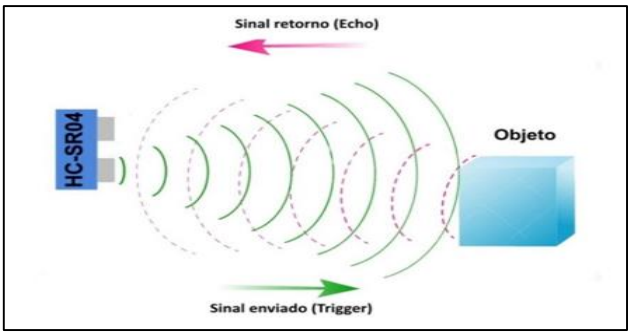


Fonte: [75]

Já o sensor ultrassônico é bastante utilizado para medir distâncias entre o ponto onde está fixado o sensor à um objeto. É muito comum a utilização destes sensores em robôs móveis autônomos para o desvio de obstáculos durante seu percurso.

Seu princípio de funcionamento consiste na emissão de uma onda de alta frequência (*trigger*) e a captação da onda (*echo*) na qual é refletida pelo objeto a ser detectado (Figura 2.18). O cálculo da distância entre o sensor e o objeto é feito baseado nas variações de propagação da onda, assim como o tempo de emissão e captação da mesma.

Figura 2.18 - Princípio de funcionamento do sensor ultrassônico



Fonte: [76]

2.3 SISTEMA DE COMUNICAÇÃO SEM FIO

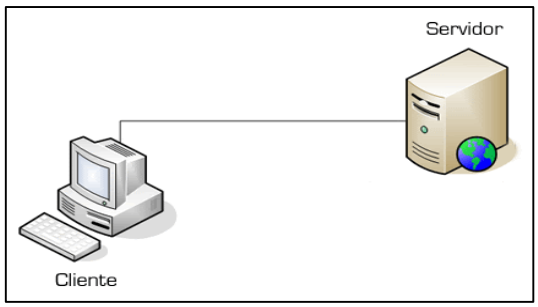
Existem diversas definições de uma rede de computadores, como por exemplo, a definição de rede de computadores segundo Meyer: “A ligação de dois ou mais computadores de forma a possibilitar a troca de dados e o compartilhamento de recursos” [77]. Segundo William

Stallings, “quando dois ou mais computadores estão interconectados via uma rede de comunicação, o conjunto das estações de computadores é chamado de rede de computadores” [78]. Além destas, existem muitas outras definições do que é uma rede de computadores na literatura, entretanto, todas possuem algumas características em comum, entre elas tem-se por exemplo a característica de ter dois ou mais computadores interligados e tem-se também o seu meio físico de comunicação (com fio, sem fio, etc).

A comunicação com fio ainda é bastante utilizada e ainda tem sido a principal escolha de empresas. Mas questões como mais mobilidade, confiabilidade, facilidade de instalação, escalabilidade e uma maior possibilidade de menor custo de instalação, tem feito a conexão sem fio ser uma tecnologia bastante reconhecida no mundo atual devido a essas vantagens [79].

Uma rede de comunicação sem fio interliga dispositivos por meio de ondas eletromagnéticas, o meio de comunicação é o ar ao invés de fios. Existem duas categorias de redes mais amplas, as redes P2P (*peer to peer*) e redes cliente-servidor. Uma rede do tipo cliente-servidor (Figura 2.19) consiste em *hosts* responsáveis por oferecer serviços de rede e os clientes nos quais são usuários que acessam os serviços de rede oferecidos pelo servidor. Um exemplo pra entender melhor essa categoria de rede é o acesso à uma página web, o cliente digita o endereço do site solicitando recursos que o servidor da página da web pode oferecer. Uma das vantagens desta categoria de rede é que o servidor é projetado para fornecer acesso a muitos arquivos ao mesmo tempo que mantém um bom desempenho e segurança ao usuário [78].

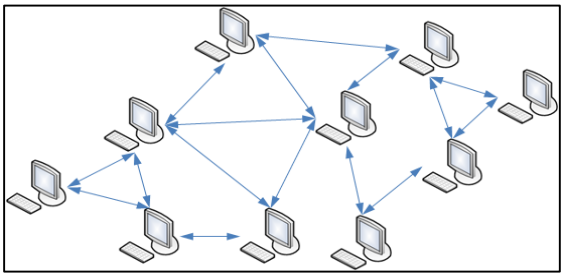
Figura 2.19 - Arquitetura de rede cliente-servidor



Fonte: [80]

Já a arquitetura de P2P (Figura 2.20) é uma arquitetura onde cada ponto ou nó da rede funciona como cliente-servidor, permitindo assim compartilhamento de dados sem necessitar de um servidor central como na arquitetura cliente-servidor. Por não haver necessidade de um servidor central ou de outros componentes para uma rede de grande necessidade, as redes par-a-par podem ser mais baratas do que as redes cliente-servidor.

Figura 2.20 - Arquitetura de rede P2P



Fonte: [81]

O envio de pacotes de dados quando tem-se a solicitação de alguma requisição para um servidor, passa por algumas etapas até chegar no destino final. A fim de criar padrões para comunicação entre computadores, surgiram dois modelos importantes ao longo da história em redes de computadores, sendo eles o modelo TCP/IP e o modelo OSI. A grande diferença entre eles é que o TCP/IP foi realmente implementado na prática sendo utilizada até os dias de hoje, porém os equipamentos de rede são classificados conforme o modelo de referência OSI. O modelo OSI por mais que não esteja funcionando na prática ele é um modelo teórico para desenvolvedores e estudiosos [82].

O modelo OSI (*Open Systems Interconection*) foi criado pela Organização Internacional para Normalização (ISO) em 1984. Esse modelo trata-se de uma arquitetura que divide a rede de computadores em 7 camadas (Figura 2.21), diferentemente do modelo TCP/IP no qual possui 4 camadas [83].

Figura 2.21 - Modelo OSI

Fonte: Adaptada de [83]

O modelo TCP/IP (*Transmission Control Protocol / Internet Protocol*) é uma coleção de protocolos para realização de comunicação entre computadores em uma rede, esse padrão foi desenvolvido inicialmente em 1969 financiada pela ARPA (*Agência de Projetos de Pesquisa Avançada*) do Departamento de Defesa dos EUA (*DoD*). O Padrão TCP/IP foi o padrão no qual a Internet se desenvolveu [84].

O modelo TCP/IP é constituído por 4 camadas, sendo elas a camada de acesso à rede, Internet, transporte e aplicação (Figura 2.22), sendo que cada camada é responsável por funcionalidades distintas.

Figura 2.22 - Modelo TCP/IP

Fonte: Adaptada de [85]

A camada de acesso a rede é responsável pelo envio de pacotes de dados IP e os envia para uma rede física específica. O Ethernet é o protocolo mais utilizado nessa camada.

A camada de Internet é responsável pela conexão de redes locais por meio do endereçamento e roteamento do pacote. Para que o pacote saiba qual caminho percorrer é adicionado um endereço IP de origem e destino ao pacote.

A camada de transporte é responsável pela garantia de que os dados cheguem sem erros na camada de aplicação. É formada por dois protocolos o TCP (*Trasmission Control Protocol*) e o UDP (*User Datagram Protocol*). A grande diferença entre estes protocolos é que o TCP preza pela confiabilidade agregando aos pacotes bits de controle de fluxo, fazendo com que assim se garanta a entrega dos pacotes, já o UDP dispensa estes bits de controle, por ele ser mais simples ele não garante a entrega dos pacotes.

Por fim a última camada é a camada de aplicação, na qual é responsável por fazer a comunicação entre os programas e os protocolos de transporte TCP/IP. Para exemplificar, quando um usuário abre uma página no navegador, ele está fazendo uma solicitação à camada de aplicação do TCP/IP que neste caso é servido pelo protocolo HTTP, por isso as páginas web geralmente iniciam-se com http://. Os protocolos importantes e conhecidos da camada de aplicação são o HTTP, FTP, DNS e DHCP.

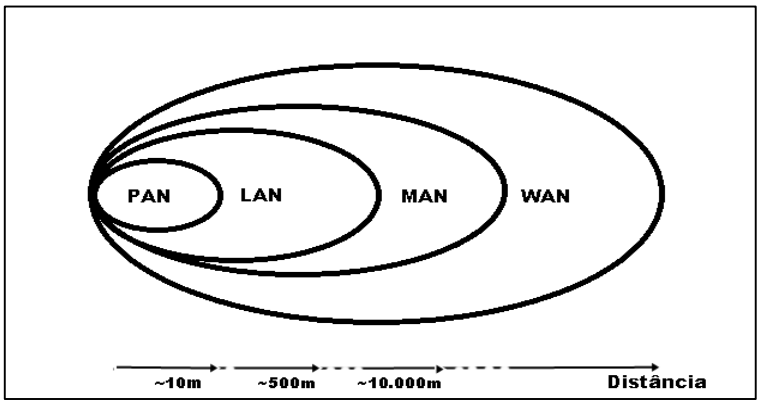
Em redes sem fio tradicionais é bastante utilizado pontos de acesso como conexão para uma rede local, eles possuem a mesma função lógica de um *hub*. Pontos de acesso são dispositivos capazes de realizar a interconexão entre todos os dispositivos móveis, são bastante utilizados em ambientes corporativos devido à necessidade de múltiplas conexões [78,86].

Em ambientes como desertos, florestas, resgate de sobreviventes em desastres terrestres e em atividades de sensoriamento em locais que colocam em risco a vida dos seres humanos, faz com que infraestruturas fixas de rede sejam inviáveis. Com isso redes do tipo Ad-Hoc vem sendo muito exploradas por ser uma rede que permite que dispositivos se comuniquem de forma direta ou por meio de múltiplos saltos, usando transmissores e receptores sem a necessidade de uma estrutura fixa de rede. Essa característica é a que mais faz diferenciar redes Ad-Hoc de redes tradicionais sem fio onde os dispositivos se comunicam através de um ponto de acesso [79].

Para tornar a tecnologia sem fio cada vez mais populares, o IEEE (*Institute of Eletrical and Electronics Engineers*) constituiu um grupo de pesquisas com a finalidade de criar protocolos abertos chamados de IEEE 802, a fim de ter métodos de acesso e controle para redes locais e metropolitanas.

Um método muito utilizado para classificação de redes leva em conta a área de cobertura. Esse critério é importante pois uma rede depende muito da extensão geográfica, as técnicas de transmissões de dados variam em função da distância. As redes são classificadas como redes pessoais PANs (*Personal Area Networks*), redes locais LANs (*Local Area Networks*), redes metropolitanas MANs (*Metropolitan Area Networks*) e redes de longa distância WANs (*Wide Area Networks*) [87]. A Figura 2.23 ilustra a classificação dos tipos de rede com relação a suas distâncias de transmissão.

Figura 2.23 - Taxonomia de redes sem fio



Fonte: Autor

O protocolo mais utilizado em redes do tipo PAN é conhecido como IEEE 802.15, e o seu maior sucesso comercial é o *Bluetooth*. Ele é bastante utilizado para interligação de dispositivos sem fio de comunicação de curto alcance como, por exemplo, fones de ouvido e equipamentos de som. Já nas redes do tipo LAN surgiram segundo os padrões IEEE 802.3, IEEE 802.4 e IEEE 802.5, com o objetivo de serem aplicadas em escritórios e automação de fábrica. Seu grande sucesso comercial foi a Ethernet na qual se tornou tecnologia dominante em redes locais. As redes do tipo MAN surgiram com o padrão IEEE 802.6 e faz parte das tecnologias de acesso à Internet ou ISP (*Internet Service Provider*). As tecnologias MAN mais utilizadas no acesso a ISP's são o ADSL e o modem a cabo (*cable-modem*). E por fim, as redes WAN são suportes de telecomunicações com cobertura nacional e internacional, onde os meios mais utilizados são as fibras ópticas [87]. Por meio da Tabela 2.2 podemos visualizar melhor os tipos de redes, suas características e principais tecnologias.

Tabela 2.2 - Tipos e principais características de redes sem fio.

Tipo de Rede	**Cobertura**	**Meios**	**Taxas Típicas**	**Padrões e implementações representativas**
PAN	Alguns metros	Canais de RF (Wireless)	2 Mbit/s	Bluetooth (IEEE 802.15)
LAN	Alguns quilômetros	Par trançado, fibra óptica e RF	10Mbit/s a 10Gbit/s	Ethernet, Token ring, Token bus IEEE: 802.3, 802.4802.5, WLAN 802.11 (Wi-Fi)
MAN	Centenas de quilômetros	Fibra óptica e canais de RF	155Mbit/s a 10Gbit/s	DQDB (IEEE: 802.6), Metro-Ethernet, NG-SDH, WMAN IEEE 802.16 (WiMAX)
WAN	Nacional e internacional	Fibra óptica	64kbit/s a Tbit/s	PDH, SDH/Sonett. Internet, MPLS, OTN ITU-T (G.709)

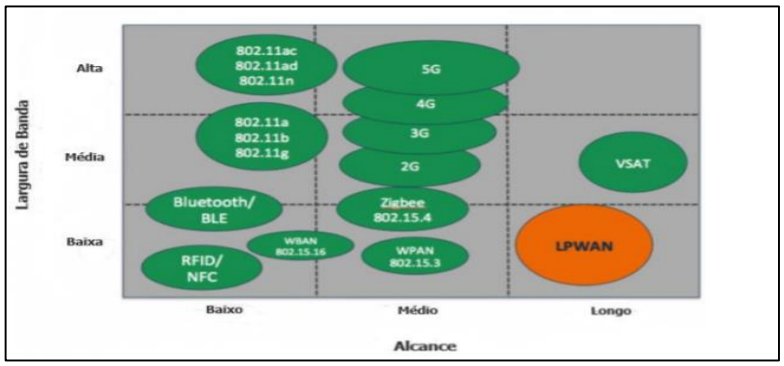
Fonte: Adaptada de [87]

Comunicações sem fio estão em constante desenvolvimento devido à demanda de avanços tecnológicos atuais como a Internet das Coisas. Redes tradicionais não foram desenvolvidas para essa aplicação, exemplo disso é o alto consumo de energia de dispositivos conectados por uma rede que utiliza tecnologia Bluetooth. Além disso, o custo de conexões tradicionais é alto quando se escala o número de dispositivos conectados [88].

Neste cenário, um tipo de rede que tem ganhado cada vez mais popularidade, principalmente nas indústrias e comunidades de pesquisa, são as redes do tipo LPWAN. LPWAN é um tipo de rede sem fio de área ampla, concebido para permitir comunicações de longo alcance e baixa taxa de dados, garantindo assim maior vida útil de baterias a serem implementadas durante os processos de comunicação e aplicação. Redes do tipo LPWAN são muito utilizadas em IoT quando há necessidade de transmitir poucos dados, em distâncias longas [89].

A Figura 2.24 mostra as principais tecnologias de comunicação sem fio, bem como as suas relações de taxa de transmissão de dados e alcance.

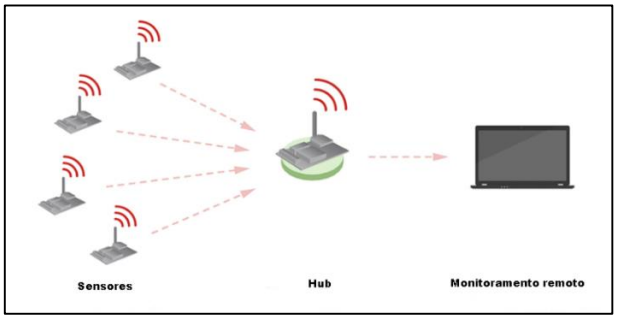
Figura 2.24 - Principais tecnologias sem fio e suas características



Fonte: [89]

Uma rede LPWAN utiliza topologia em estrela, isto é, existe um *hub*, no qual se ligam vários dispositivos de rede. A Figura 2.25 mostra um exemplo de topologia de rede estrela LPWAN.

Figura 2.25 - Topologia em estrela de rede LPWAN



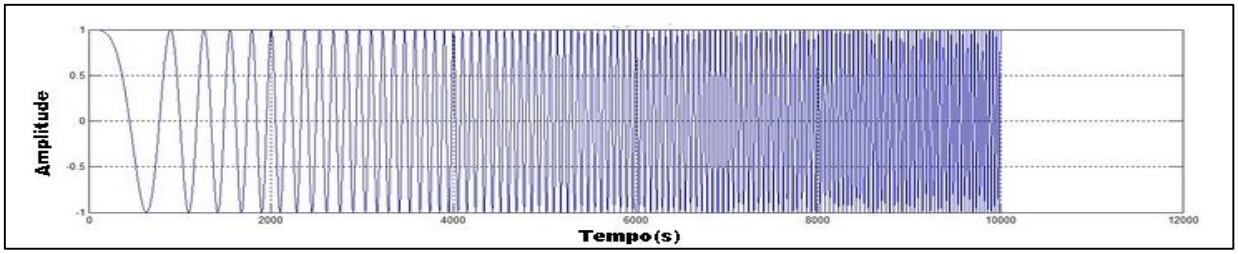
Fonte: Adaptada de [89]

Muitas tecnologias LPWAN surgiram na faixa de largura de banda de frequência licenciada e não licenciada. Entre elas temos a Sigfox, LoRa e NB-IoT nas quais tem sido líderes hoje em dia [90]. Embora exista muitas diferenças técnicas entre elas, as vantagens de utilizá-las são semelhantes como, fácil instalação, baixo custo de implementação, poucas estações base para fornecimento de cobertura e autenticação de rede dedicada.

2.3.1 LoRa e LoRaWAN

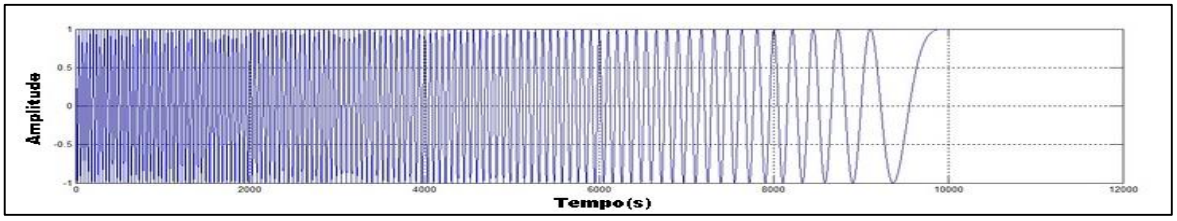
LoRa é uma tecnologia de transmissão de dados sem fio à longas distâncias desenvolvido pela Semtech. Esse nome foi dado à uma técnica de modulação chamada CSS (*Chirp Spread Spectrum*). Essa modulação opera espalhando o sinal ao longo do espectro pela variação de frequência. Os sinais do tipo Chirp (*Compressed High Intensity Radar Pulse*) possuem amplitude constante e varrem toda largura de banda variando a frequência de forma linear com o tempo. Se a frequência aumenta com tempo dizemos que é um sinal do tipo *up-chirp* (Figura 2.26) e se a frequência diminui com o tempo chamamos esse sinal de *down-chirp* (Figura 2.27) [91].

Figura 2.26 - Sinal up-chirp (Amplitude x Tempo)



Fonte: Adaptada de [92]

Figura 2.27 - Sinal Down-Chirp (Amplitude X Tempo)

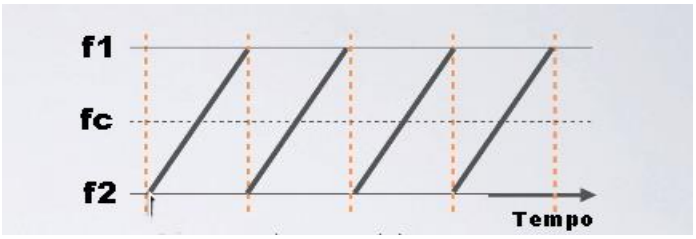


Fonte: Adaptada de [92]

Observando estes sinais no domínio da frequência, os *chirps* são sinais que variam em rampa com o tempo, onde os *up-chirps* variam da frequência mínima (f2) para frequência máxima (f1) e os *down-chirps* variam da frequência máxima (f1) para a frequência mínima (f2) [91]. A frequência central (fc) é a média entre os dois extremos da largura de banda (BW), ou seja, f1 e f2. A largura de banda é uma medida que nos mostra o quanto o sinal se espalha em frequência. As larguras de banda mais comuns utilizadas em uma comunicação que usa tecnologia LoRa são as de 125 kHz, 250 kHz e 500 kHz.

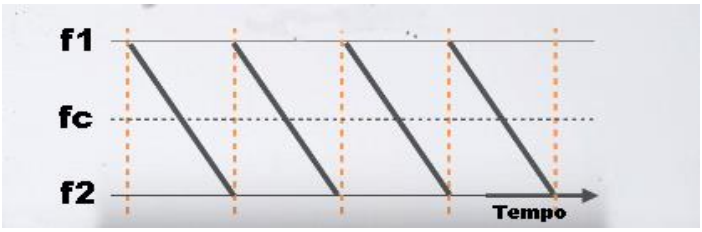
Por meio da Figura 2.28 e Figura 2.29 pode-se ver que os *chirps* são deslocados ciclicamente de forma que quando se atinge a frequência mais alta, retorna-se para a frequência mais baixa, com isso a informação que o sinal carrega é dada por saltos de frequência.

Figura 2.28 - Sinal *up-chirp* (Frequência X Tempo)



Fonte: Adaptada de [91]

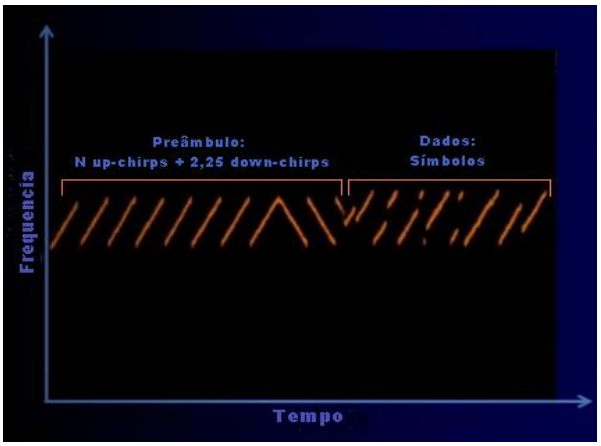
Figura 2.29 - Sinal *down-chirp* (Frequência X Tempo)



Fonte: Adaptada de [91]

Um *frame* LoRa é formado pela combinação de sinais *up-chirps* e *down-chirps*, onde tem-se o preâmbulo formado por "n" quantidade de *up-chirps* e em seguida um valor fixo de 2,25 *down-chirps*. O preâmbulo tem a função de preparar o receptor para receber os dados que vem logo em seguida, esses dados são codificados a partir de símbolos formados por recortes pré determinados de *up-chirps* [93]. Por meio da Figura 2.30 pode-se visualizar melhor como é um *frame* LoRa.

Figura 2.30 - Frame LoRa

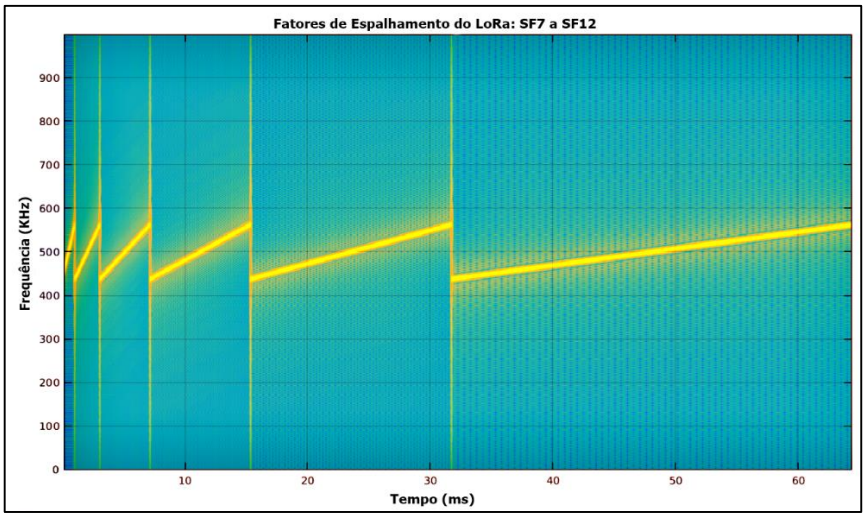


Fonte: Adaptado de [93]

Os símbolos representam um ou mais bits de dados e apresenta 2^{SF} valores possíveis (Seria equivalente a um número de SF bits). SF (*Spreading Factor*) representa o número de bits da modulação. De forma prática, o SF representa o tempo de duração de um *chirp*, ou seja, quanto menor o SF menor será o tempo de *chirp* e menos bits serão transmitidos por *chirp*. Os componentes mais usuais no mercado são SF's que variam de 6 a 12. Conforme se aumenta o

SF o tempo que o *frame* vai ser transmitido vai sendo dobrado, isso melhora a robustez do sinal perante a muitos ruídos quanto maiores distâncias e isso é uma das grandes vantagens da modulação LoRa perante as demais. A Figura 2.31 adaptada do trabalho desenvolvido em [94] ilustra o tempo necessário para transmitir um símbolo de SF7 a SF12 com largura de banda de 125KHz.

Figura 2.31 - Relação de cada SF com o tempo de transmissão



Fonte: Adaptado de [94]

Outro parâmetro importante para definição de um canal de comunicação LoRa é o CR (*Coding Rate*) no qual é um algoritmo cíclico de redundância que é adicionado aos dados uma quantidade de bits para detecção e correção de erros para melhorar a robustez do sinal.

Definindo os parâmetros como largura de banda, SF (*Spreading Factor*), CR (*Coding Rate*) e o comprimento do preâmbulo, temos um canal de comunicação LoRa mandando dados de um ponto até outro por ondas de rádio [93].

Além do significado de técnica de comunicação sem fio, o LoRa também possui um significado que surgiu ao longo do tempo com a popularização de um padrão aberto de rede que se chama oficialmente de LoRaWAN. O padrão de rede LoRaWAN foi criado por uma comunidade chamada LoRa Alliance e utiliza a modulação LoRa para operar.

Por meio do modelo OSI pode-se entender melhor as diferenças de conceitos entre LoRa e LoRaWAN (Figura 2.32). Em LoRa um sistema final possui um módulo capaz de codificar e decodificar informações em um sinal, esse sinal é transmitido e recebido por um *gateway* no qual possui o mesmo módulo que é capaz de modular e demodular o sinal [91]. Isso cobre claramente a primeira camada do modelo OSI, ou seja, a camada física. LoRaWAN é um protocolo de controle de acesso à mídia (MAC) para redes de área ampla, ele pode ser mapeado para a camada de enlace e claramente para a camada de rede.

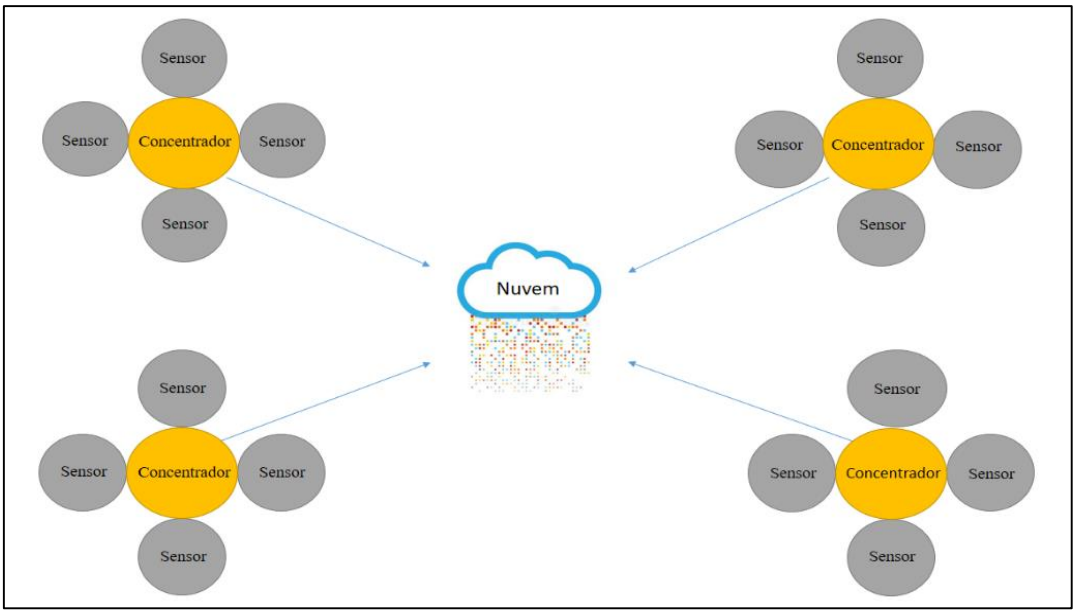
As aplicações de IoT ocupam as demais camadas e são apoiadas sobre esse conjunto básico.

Figura 2.32 - Diferenças entre LoRa e LoRaWAN no modelo OSI

Fonte: Adaptada de [83]

O LoRaWAN opera em topologia estrela, ou seja, cada nó ou sensor precisa se comunicar com um *gateway* (concentrador). Diversas unidades semelhantes podem existir simultaneamente na mesma rede em locais diferentes. Os concentradores se comunicam com a nuvem onde os dados são encaminhados (Figura 2.33).

Figura 2.33 - Topologia LoRaWAN



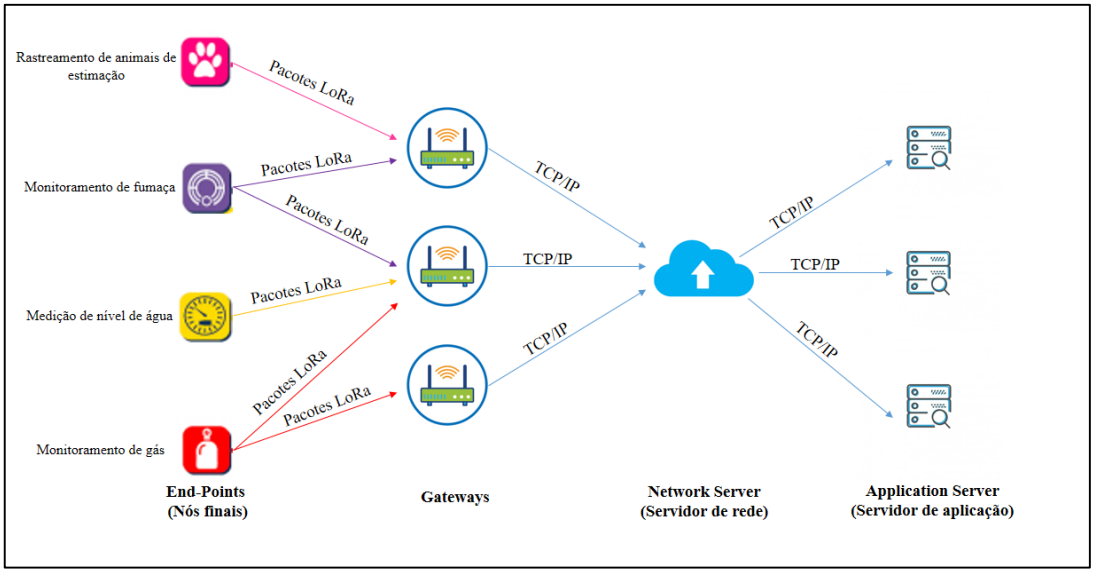
Fonte: Autor

Os principais componentes que compõem uma rede LoRaWAN são os *end-points*, *gateway*s, *network-server* e *application-server* [95]. *End-point* (dispositivo final) é um terminal de comunicação, ele é um dispositivo final de comunicação, como por exemplo um sensor. Os *gateway*s são responsáveis por interligar redes, é um dispositivo intermediário no qual recebe

os pacotes LoRa dos *end-points* e os encaminha para um servidor de rede via TCP/IP. Através do protocolo TCP/IP é possível conectar-se à Internet e enviar e receber informações. O *network-server* (servidor de rede) tem a função de receber e armazenar os dados vindos dos *gateway*s, esse tipo de serviço é conhecido como armazenamento em nuvem.

E por fim, servidores de rede encaminha as mensagens para os servidores de aplicação (*application server*), eles são responsáveis por processar as mensagens. Estes servidores podem ser genéricos nos quais podem ser contratados e configurados de acordo com a aplicação ou pode ser uma parte da estrutura de quem oferece um serviço IoT, é por meio do servidor de aplicação que os dados são mostrados de alguma forma para o usuário. Por meio da Figura 2.34 podemos visualizar melhor como é arquitetura LoRaWAN.

Figura 2.34 - Arquitetura LoRaWAN



Fonte: Autor

O processo de encaminhar mensagens dos *end-points* até um servidor de aplicação é chamado de *uplink*. Pode-se observar por meio da Figura 2.34 que mais de um *gateway* pode ser selecionado para enviar uma mensagem de um mesmo dispositivo final. Já o processo de enviar alguma mensagem do servidor de aplicação para o *end-point* é chamado de *downlink*. No *downlink* somente um *gateway* pode ser selecionado pelo servidor de rede para enviar uma mensagem para o *end-point*, a escolha do *gateway* para essa função é feita com base na potência do sinal recebido pelo *gateway* quando o mesmo recebe a mensagem anterior de *uplink*.

Um fator muito importante para se entender melhor como funciona a hierarquia de mensagens de *uplink* e *downlink* no LoRaWAN é a largura de banda. Existem diversas larguras de banda possíveis na modulação LoRa nas quais operam em frequências licenciadas ou não licenciadas. Para os EUA e américas, a faixa de frequência de aplicação vai de 902 MHz a 928

MHz [96]. Dispositivos que operam em 915MHz possuem três possibilidades de largura de banda, 125 kHz, 250 kHz e 500 kHz. As mensagens de *uplink* são geradas em 125 kHz e em algumas possibilidades à 500 kHz, e toda mensagem de *downlink* é gerada em 500 kHz.

2.4 MODELO CINEMÁTICO E ODOMETRIA

A odometria é uma técnica bastante utilizada para estimar posição de robôs móveis e consiste na integração do movimento incremental de sensores, no caso deste projeto, os pulsos do *encoder*. Por meio do *encoder* pode-se medir a rotação da roda e a partir de um modelo cinemático é realizado o cálculo de posição e orientação do robô.

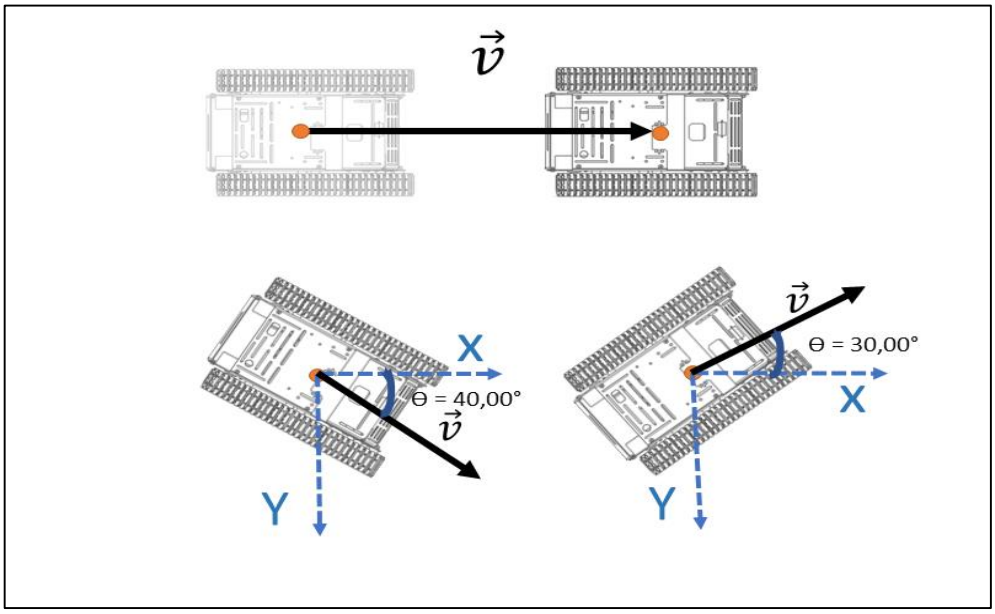
O modelo cinemático e o código de cálculo do deslocamento do robô foram feitos baseado em um projeto desenvolvido em [106].

O deslocamento do robô pode ser representado por um vetor no qual é formado quando o robô se desloca de um ponto inicial A até um ponto final B, com isso tem-se um vetor de deslocamento $\vec{v}$.

Além do módulo do vetor, no qual corresponde ao comprimento do segmento entre as posições inicial e final, tem-se o sentido do vetor no qual o pode-se representar em um sistema de coordenadas X e Y. Pensando no deslocamento do robô, o mesmo pode ser representado pelo vetor deslocamento cujo sentido pode ser da traseira até a frente a ou da frente até a traseira, ou seja, sentido oposto.

A direção do robô pode ser determinada quando o mesmo gira em torno do próprio eixo para a direita ou esquerda, de forma que quando o mesmo gira, o vetor de deslocamento $\vec{v}$ forma um ângulo com o semieixo X do sistema de coordenadas, de forma que o ângulo cresce no sentido horário e decresce no sentido anti-horário. A Figura 2.35 ilustra a representação do sentido e orientação do vetor de deslocamento.

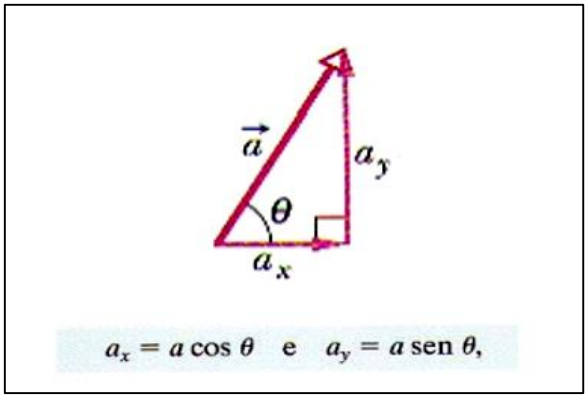
Figura 2.35 - Sentido e orientação do vetor deslocamento



Fonte: Autor

X e Y são as componentes horizontal e vertical do vetor, e conhecendo o ângulo Θ podemos definir essas componentes multiplicando o módulo do vetor pelo cosseno do ângulo formado entre o vetor e o semieixo X e para definir a componente Y temos que multiplicar o módulo do vetor pelo seno do ângulo (Figura 2.36) [107].

Figura 2.36 - Projeção de um vetor.



Fonte: Halliday (2012)

A componentes X e Y são a projeção do vetor $\vec{v}$ como mostra a Figura 2.36, além disso assumindo que o módulo do vetor $\vec{v}$ é unitário tem-se que as nossas componentes $\vec{v}_x$ e $\vec{v}_y$ serão:

$$\vec{v}_x = ||v|| \cos\theta = \cos\theta$$
$$\vec{v}_y = ||v|| \sin\theta = \sin\theta \qquad (2.1)$$

A quantidade de vetores $\vec{v}$ formados é proporcional à quantidade de pulsos gerados pelo *encoder*, com isso deslocamento do robô é a soma vetorial de todos vetores $\vec{v}$ formados e a sua posição é a soma das componentes X e Y do resultado desta soma vetorial.

Assumindo as componentes do vetor como $\vec{v}_x = x$ e $\vec{v}_y = y$ tem-se que $\vec{v}_n = x_n\hat{\imath} + y_n\hat{\jmath}$. Assumindo que $\vec{S}_n$ é o vetor da soma resultante dos vetores $\vec{v}$ e que "n" são os pulsos gerados pelo *encoder*, as componentes do eixo X e Y será:

$$S_{x_n} = \sum_1^n x_n$$
$$S_{y_n} = \sum_1^n y_n$$
$$\vec{S}_n = S_{x_n}\hat{\imath} + S_{y_n}\hat{\jmath} \qquad (2.2)$$

No código foram criadas variáveis para armazenar as posições X e Y, toda vez que tem um novo pulso gerado pelo *encoder*, é acrescentada uma nova componente x_n e y_n no somatório, fazendo com que o resultado seja a soma do resultado do somatório anterior com o valor da nova componente:

$$X = S_{x_{n-1}} + x_n$$
$$Y = S_{y_{n-1}} + y_n \qquad (2.3)$$

Assumindo que $x_n = \cos(\theta_n)$, $y_n = \text{sen}(\theta_n)$, e $\theta_n = \theta$, o resultado é:

$$X = S_{x_{n-1}} + \cos(\theta)$$
$$Y = S_{y_{n-1}} + sen(\theta) \qquad (2.4)$$

Vale ressaltar que a angulação do novo pulso só muda quando o robô gira em torno do próprio eixo, caso contrário não muda. Os valores de X e Y são atualizados a cada vez que um pulso é gerado pelo *encoder*, com isso pode-se reescrever a equação 2.4 de maneira simplificada, de forma que: $X_{atual} = X_{anterior} + \cos(\theta)$ e $Y_{atual} = Y_{anterior} + \text{sen}(\theta)$.

Para ilustrar o modelo, assuma-se que o robô se desloque pra frente de um ponto P1 cuja coordenada é B(0,0) para um ponto P2 cuja coordenada é B(2,0), tem-se que foi gerado dois pulsos e por consequência dois vetores $\vec{v}$ com $\theta = 0°$, considerando o primeiro vetor $\vec{v}$ gerado temos que P1: $x_1 = \cos(0) = 1$, $y_1 = sen(0) = 0$.

$$X_{anterior} = 0$$
$$Y_{anterior} = 0$$
$$X_{atual} = X_{anterior} + \cos(0°) = 0 + 1 = 1$$
$$Y_{atual} = Y_{anterior} + \text{sen}(0°) = 0 + 0 = 0$$
$$B(1\,,0\,) \qquad (2.5)$$

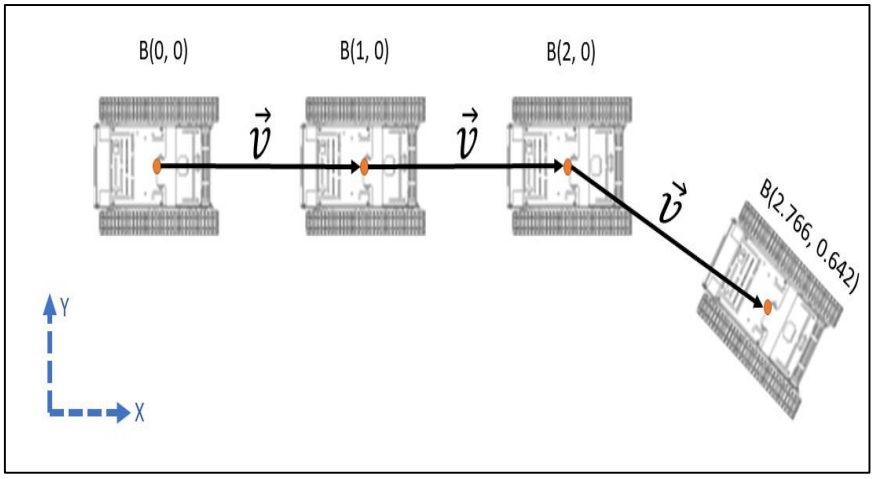
Como o robô continuou se deslocando na mesma direção ($\theta = 0°$) tem-se que o valor de deslocamento no eixo de X é de 1 e no eixo Y é de 0 com isso temos que P2: ($x_1 = \cos(0) = 1$, $y_1 = sen(0)$). A soma geométrica destes vetores nos dá o resultado de uma posição X=2 e Y=0.

$$X_{anterior} = 1$$
$$Y_{anterior} = 0$$
$$X_{atual} = X_{anterior} + \cos(0°) = 1 + 1 = 2$$
$$Y_{atual} = Y_{anterior} + \text{sen}(0°) = 0 + 0 = 0$$
$$B(2\ ,0\ ) \qquad (2.6)$$

Assumindo que o robô se desloca do ponto P2 girando 40° em torno do próprio eixo para direita, o ângulo se torna $\theta = 0° + 40° = 40°$, com isso temos que o ponto P3 é $x_3 = \cos 40° = 0{,}766$ e $y_3 = \sin 40° = 0{,}642$. Com isso, o resultado da nova soma das componentes vetoriais atuais com as anteriores dará o resultado de X= 2,766 e Y= 0,642 (Figura 2.37).

$$X_{anterior} = 2$$
$$Y_{anterior} = 0$$
$$X_{atual} = X_{anterior} + \cos(40°) = 2 + 0.766 = 2.766$$
$$Y_{atual} = Y_{anterior} + \text{sen}(40°) = 0 + 0.642 = 0.642$$
$$B(2.766\ ,0.642\ ) \qquad (2.7)$$

Figura 2.37 - Deslocamento do robô



Fonte: Autor

3 MATERIAIS E MÉTODOS

Este capítulo trata dos materiais e toda metodologia necessária para realização do projeto.

3.1 FERRAMENTAS UTILIZADAS

3.1.1 Robô do tipo esteira

Um dos grandes desafios de um robô móvel é a superação de obstáculos. Nesse cenário uma classe de robôs que tem sido muito utilizado devido a simplicidade de movimento e capacidade de locomoção em vários tipos de terrenos são os robôs com esteiras [99].

O chassi que foi utilizado no projeto possui 340 mm de comprimento, 240 mm de largura e 122 mm de altura onde o corpo e rodas de condução do chassi são feitas de liga de alumínio e o rolamento da roda é de plástico (Figura 3.1). O chassi possui o peso total de 1.16 kg e a capacidade máxima de carga quando o mesmo está em funcionamento é de 5 kg e a tensão dos motores para seu funcionamento podem variar de 7 V a 12 V.

Figura 3.1 - Chassi que será utilizado no projeto

Fonte: AliExpress

3.1.2 Módulo ponte H TB6612FNG

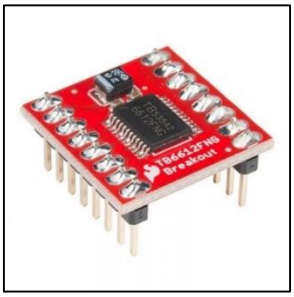
A ponte H é um circuito que serve para variar o sentido da corrente em uma determinada carga. Tem como principal função o controle de velocidade e sentido de motores DC.

O módulo ponte H que foi utilizado no projeto é o TB6612FNG (Figura 3.2), este módulo pode controlar até dois motores à uma corrente constante de 1,2A (pico de 3,2 A). Dois sinais

de entrada IN1 IN2 podem ser utilizados para o controlar um motor em um dos quatro modos de funcionamento, sendo eles CW, CCW, freio rápido e parada.

Além dos pinos de controle de sentido dos motores, tem os pinos de PWM utilizados para controlar a velocidade dos motores, a frequência do sinal PWM tem que ser de até 100 kHz. PWM (*Pulse Width Modulation*) ou modulação por largura de pulso permite o controle de potência ou velocidade por meio da largura de pulso de uma onda quadrada. O fornecimento de tensão lógica (VCC) deve estar na faixa de 2,7 V a 5,5 V, enquanto a tensão de fornecimento de alimentação para os motores (VM) é limitada a 15 V.

Figura 3.2 - Módulo ponte H TB6612FNG

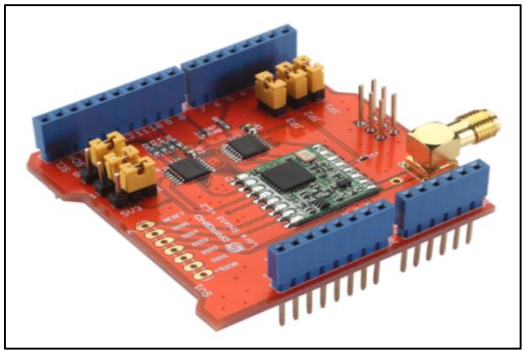


Fonte: Mercado Livre

3.1.3 Modulo LoRa *Shield* IoT (Dragino)

O modulo LoRa *shield* para o Arduino é um transceptor de longo alcance desenvolvido pela Dragino (Figura 3.3). A *Dragino Technology Co., Limited* é uma empresa localizada em Shenzhen na China, a mesma atua no fornecimento e fabricação de produtos eletrônicos concentrada no setor de Internet das coisas [100]. O Módulo *shield* LoRa usa técnica de modulação LoRa e é baseado no chip Semtech SX1276/SX1278. Tem como alvo aplicações profissionais de rede de sensores sem fio permitindo ao usuário enviar dados atingindo longas distâncias com baixa taxa de dados, com um consumo mínimo de corrente [101].

Figura 3.3 - Módulo *shield* LoRa



Fonte: [101]

3.1.4 LG01-P LoRa Gateway (Dragino)

LG01-P (Figura 3.4) é um LoRa *gateway* de canal único no qual permite conectar uma rede LoRa a uma rede IP via Wi-Fi, Ethernet ou celular, fornecendo assim métodos flexíveis aos usuários para conexão de uma rede de sensores à Internet. O LG01-P suporta o protocolo LoRaWAN de forma limitada em frequência única. Ele fornece comunicação de espectro espalhado de alcance longo e alta imunidade a interferências [102].

Figura 3.4 - LoRa *gateway* LG01-P

Fonte: [102]

O Quadro 3.1 mostra maiores especificações técnicas do *gateway* LG01-P e o módulo LoRa *shield*.

Quadro 3.1 - Especificações técnicas do módulo LoRa *shield* e o *gateway* LG01-P.

Módulo LoRa *Shield*	**Gateway LG01-P**
Opera nas bandas de Frequência de 915 MHz / 868 MHz / 433 MHz	Banda LoRa Disponível de 433 MHz / 868 MHz / 915 MHz / 920 MHz
Compatível com Arduino Leonardo, Uno, Mega e DUE	Compatível com Arduino IDE e simples de programar
Além da modulação LoRa também trabalha com modulação FSK, GFSK, MSK, GMSK e OOK	Possui uma faixa máxima de distância de comunicação LoRa de 5 km ~ 10 km
Baixo consumo de energia	Conexão a Internet via LAN, WiFi ou 3G / 4G via módulo LTE opcional
Possui sensor de temperatura embutido e indicador de bateria fraca	Servidor da Web incorporado
Possui detecção de preâmbulo	Software atualizável via rede
Taxa de bits programável de até 300 kbps	Possui suporte WiFi AP, cliente ou modo Ad-Hoc
Alta sensibilidade de até -148 dBm	Possui microcontrolador ATMega328P e o chip LoRa SX2176/78
Excelente imunidade de bloqueio	O design à prova de falhas fornece um sistema robusto

Fonte: Adaptada de [101, 102]

3.1.5 *ThingSpeak*

O *ThingSpeak* é um serviço de plataforma de análise de IoT no qual permite agregar, visualizar e analisar fluxos de dados ao vivo na nuvem usando uma API Rest ou MQTT. O *ThingSpeak* foi lançado pela ioBridge em 2010 como um serviço e suporte a aplicativos IoT [*ThingSpeak*].

Além de oferecer uma infraestrutura de web e um protocolo de comunicação baseado em HTTP para envio e recebimento de dados, o *ThingSpeak* possui funcionalidades interessantes como a possibilidade de gerar alarmes e direcionar tais alarmes como mensagens em redes sociais como o Twitter. A Figura 3.5 mostra a logo do *ThingSpeak*.

Figura 3.5 - Logo do *ThingSpeak*

Fonte: Adaptada de [*ThingSpeak*]

Os dados são inseridos em canais onde cada canal permite o armazenamento de até 8 campos de dados, usando até 255 caracteres alfanuméricos cada. Os dados recebidos possuem data e hora e recebem uma identificação sequencial [103].

É possível visualizar e entender melhor o armazenamento de dados em canais no *ThingSpeak* por meio de um artigo desenvolvido em [104], no qual consiste em uma construção de uma mini estação meteorológica utilizando o Arduino. Essa estação é capaz de realizar medição de luminosidade, temperatura, umidade, índice UV, pressão atmosférica e material particulado. Esses dados são comunicados com a Internet por meio de um módulo WiFi ESP8266 ESP-01 utilizando o *ThingSpeak*. A Figura 3.6 mostra a visualização dos dados de um canal onde existem 4 campos de dados vindos de sensores, sendo eles temperatura, humidade, pressão e luminosidade no *ThingSpeak*.

Os dados são publicados em um canal no *ThingSpeak* mediante ao acesso a API *ThingSpeak* com uma chave de escrita '*write key'*, na qual consiste em uma cadeia alfanumérica única que é criada de forma aleatória utilizada para autenticação. E para que os dados sejam lidos é criada uma chave de leitura '*read key*' se o canal for privado. Os canais também podem ser públicos caso não haja nenhuma chave de leitura.

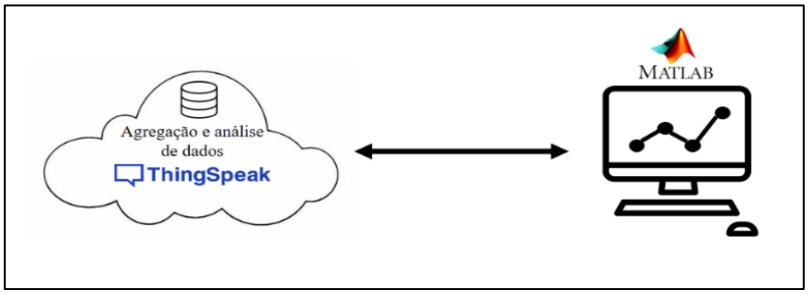
Figura 3.6 - Visualização de dados no *ThingSpeak*

Fonte: [104]

3.1.6 *ThingSpeak* e Matlab

O MATLAB é um *software* interativo de alta performance voltado para cálculo numérico. Ele possui diversas funcionalidades sendo uma delas ajudar o usuário projetar, criar protótipos e implantar aplicativos IoT. O *ThingSpeak* fornece acesso ao MATLAB para fazer análise e processamento de dados (Figura 3.7). O MATLAB fornece várias ferramentas (toolbox) de apoio ao *ThingSpeak* como, por exemplo, análises estatísticas, processamento de sinal, mapeamento e *Machine Learning*.

Figura 3.7 - Interação entre *ThingSpeak* e Matlab

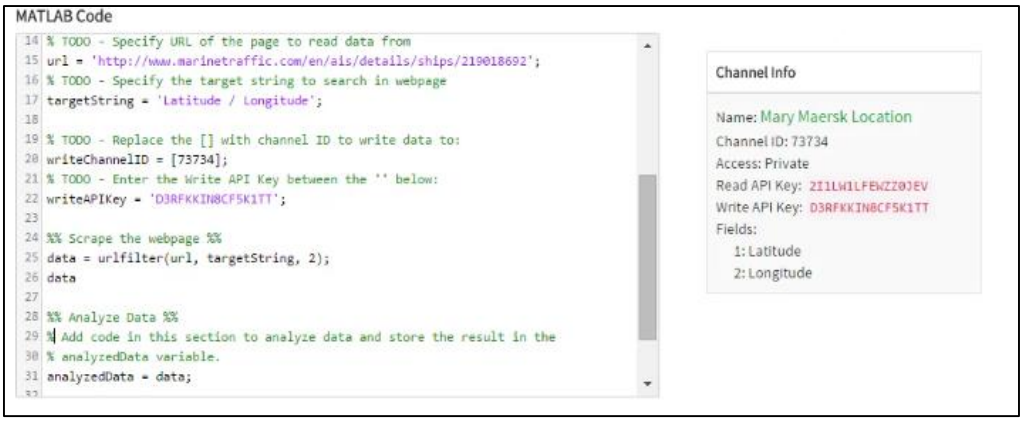


Fonte: Autor

Para se obter os dados do *ThingSpeak* com sucesso no MATLAB são usados um ID de canal e uma chave API criado pelo usuário no *ThingSpeak* para coletar e ler esses dados no MATLAB. Para melhor compreendimento tem-se um exemplo na própria comunidade do

ThingSpeak de como executar código Matlab no *ThingSpeak*. Por meio da Figura 3.8 podemos ver o ID do canal e API utilizada para se fazer a visualização dos dados no MATLAB. .

Figura 3.8 - Id e API do canal no *ThingSpeak*



Fonte: Adaptada de [*ThingSpeak Community*]

Depois de executar o código MATLAB, o mesmo começa a ler o ID do canal e a chave API e os dados começam a ser portados no MATLAB.

3.2 MÉTODOS

- **Pesquisa Bibliográfica:** Para a realização do trabalho foi feita primeiramente uma revisão bibliográfica por meio de leitura de livros, *sites* e artigos da área a fim de fazer uma análise de possíveis dificuldades, vantagens e desvantagens das tecnologias utilizadas no trabalho.
- **Construção do Protótipo:** Após a revisão bibliográfica o próximo passo foi construir o protótipo robótico do tipo esteira. Foi utilizado placas baseadas em Arduino para programar o protótipo. O protótipo contou com sensor capaz de medir concentração de gás em um ambiente controlado e também contou com um módulo ponte H para acionamento dos motores, sensor ultrassônico como sensor de distância e *encoder* para estimação da trajetória do robô e um transceptor LoRa para comunicação. O sistema de energia escolhido para o protótipo foi uma bateria LiPo de 7,4 V.
- **Comunicação e Interface de Usuário:** A comunicação utilizada foi o LoRa, onde as informações eram transmitidas do protótipo robótico para um *gateway* LoRa. As informações ficavam disponíveis ao usuário no *software ThingSpeak* após o estabelecimento da comunicação LoRa e a conexão do *gateway* à Internet.
- **Finalização do Desenvolvimento do Protótipo:** Após a certificação de que todos os periféricos utilizados no protótipo assim como o sistema de comunicação estavam

funcionando adequadamente, a programação dos microcontroladores foram feitas de modo a interligar todos os componentes do projeto.

- **Teste Final:** O teste final do protótipo robótico foi feito em um ambiente controlado a fim de realizar um teste prático de medição de gás, estimação da trajetória do robô e a comunicação dessas informações por rede LoRa enquanto o robô explorava o ambiente de forma autônoma.

3.3 CUSTOS

Já foi mencionado em sessões anteriores os materiais e *softwares* necessários que foram utilizados no projeto e por meio da Tabela 3.1 pode-se visualizar melhor de forma simplificada os materiais tanto de *hardware* como de *software* que foram utilizados assim como seus custos.

Tabela 3.1 - Materiais e custos, valores de 2020.

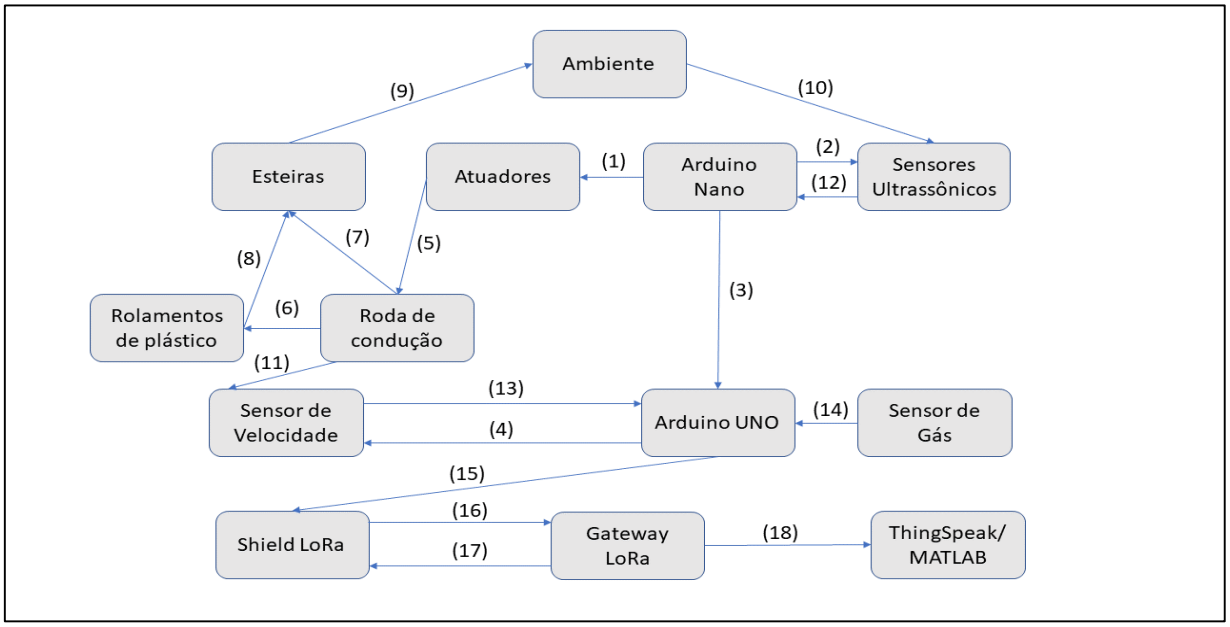
MATERIAIS	**PREÇOS**
Arduino UNO	R$ 49,90
Robô Esteira + Motores + Ponte H + Bateria LiPo	R$ 569,90
Kit Dragino (Lora*Shield* + *Gateway* LoRa LG01-P + Arduino UNO)	R$ 590,00
Sensor Ultrassónico	R$ 32,97
Sensores de gás	R$ 29,90
Encoder Óptico	R$ 25,00
Arduino IDE	Gratuito
ThingSpeak	Gratuito

Fonte: Mercado Livre, American Express

4 DESENVOLVIMENTO

O desenvolvimento do projeto foi feito conforme descrito na metodologia e é ilustrado por meio de um fluxograma mostrado na Figura 4.1.

Figura 4.1 - Fluxograma do projeto



Fonte: Autor

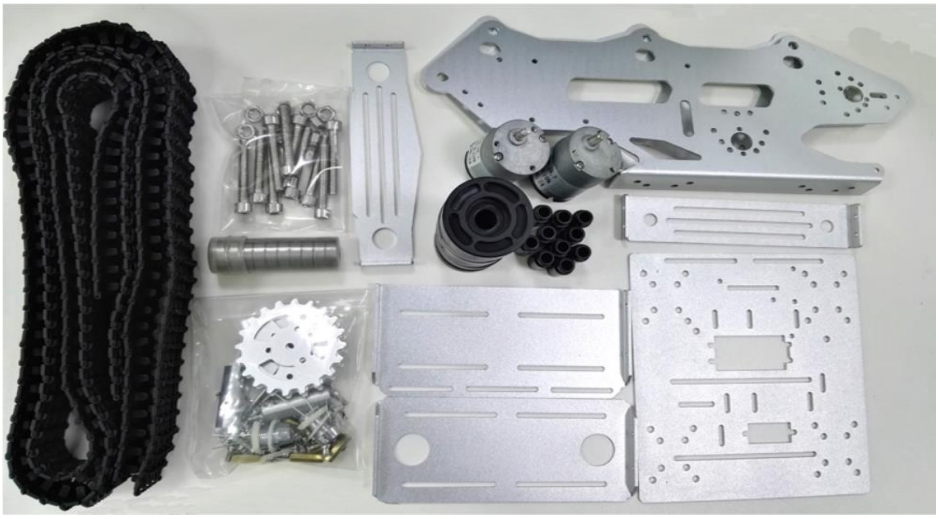
Os itens (1), (2), (4) estão ligados a programação dos atuadores (motores DC) e sensores, em (3) tem-se a comunicação entre os dois Arduinos e em (5) a atuação da roda de condução, que está acoplado o *encoder*. Nos itens (6), (7), (8), tem-se a interação dos movimentos da roda de condução e rolamentos de plástico. Assim, as esteiras acopladas a esses rolamentos e à roda de condução se movem, fazendo com que o robô se movimente no ambiente, item (9). Os itens (10) e (11) são informações obtidas por meio do deslocamento do robô no ambiente. Os itens (12) e (13) se referem ao envio dos dados dos sensores relacionados ao movimento do robô para os Arduinos, e o item (14) se refere ao envio de dados do sensor de gás para o Arduino UNO. Vale ressaltar que o trabalho que o Arduino UNO faz com os dados obtidos do sensor de velocidade, depende dos dados recebidos do Arduino nano. O item (15) descreve uma *shield* com tecnologia LoRa acoplada no Arduino UNO. O item (16) descreve o envio de informações via LoRa ao *gateway* LoRa, e o item (17) o envio de respostas do *gateway* ao Arduino UNO. Por fim o item (18) descreve o envio de informações ao *ThingSpeak*.

4.1 CONSTRUÇÃO MECÂNICA

Foi feita aquisição da estrutura mecânica do robô por meio de uma compra comercial. A estrutura é do tipo tanque chamado YP-100 e é fabricada por uma empresa chamada Doit.

O corpo do carro é feito de liga de alumínio, possui rolamentos de roda de plástico e rodas de condução de liga de alumínio. Os motores cuja tensão é de 12V vem incluído na compra. A estrutura já vem com furações e parafusos, motores, rodas, esteiras e apoios (Figura 4.2).

Figura 4.2 - Chassi do Robô desmontado



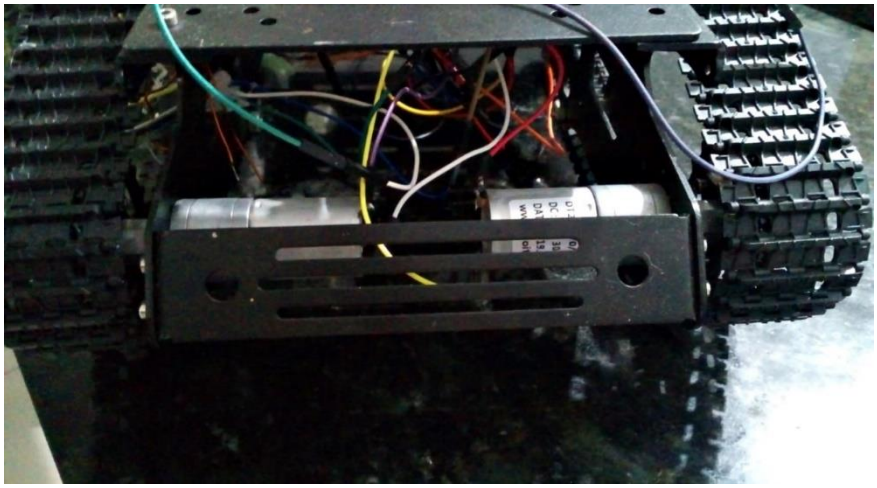
Fonte: [105]

A montagem da estrutura necessita encaixar as peças e posicionar os parafusos. A Figura 4.3 mostra a estrutura montada. Os motores ficam fixados às rodas de condução (Figura 4.4) e quando as mesmas giram, faz com que as esteiras se movam fazendo os rolamentos girarem também.

Figura 4.3 - Chassi Montado

Fonte: Autor

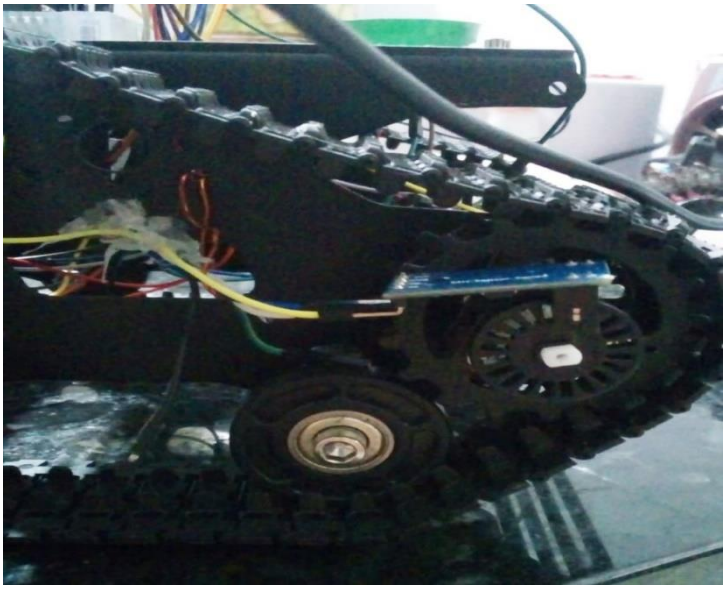
Figura 4.4 - Rodas de condução fixadas



Fonte: Autor

Foi necessário fazer uma adaptação para a colocar o *encoder* na estrutura, pois as rodas de condução não possuem adaptabilidade para a utilização deste sensor. O disco assim como o *encoder*, foi colocado em uma das rodas de condução. Vale ressaltar que todas as adaptabilidades que envolvem fixação na estrutura, foram feitas com auxílio de cola quente (Figura 4.5).

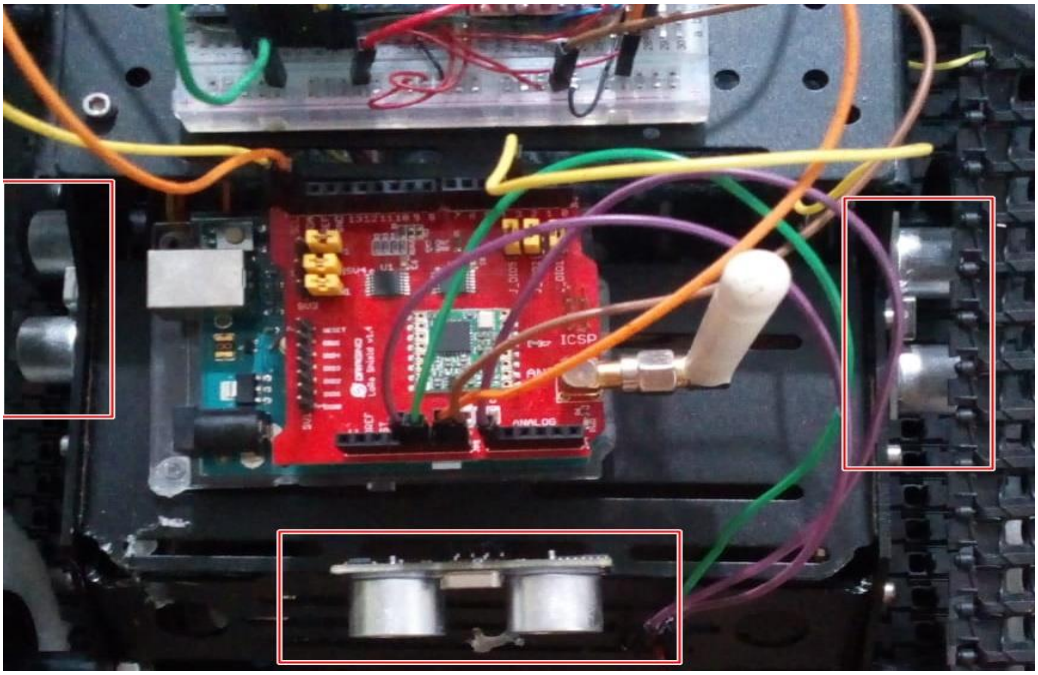
Figura 4.5 - *Encoder* fixado na roda de condução



Fonte: Autor

Neste projeto foram utilizados três sensores ultrassônicos, um deles foi fixado na parte frontal da estrutura e os outros dois foram fixados nas laterais (Figura 4.6). Foi utilizado três sensores ultrssônicos neste projeto para aumentar a precisão do robô em desviar dos obstáculos.

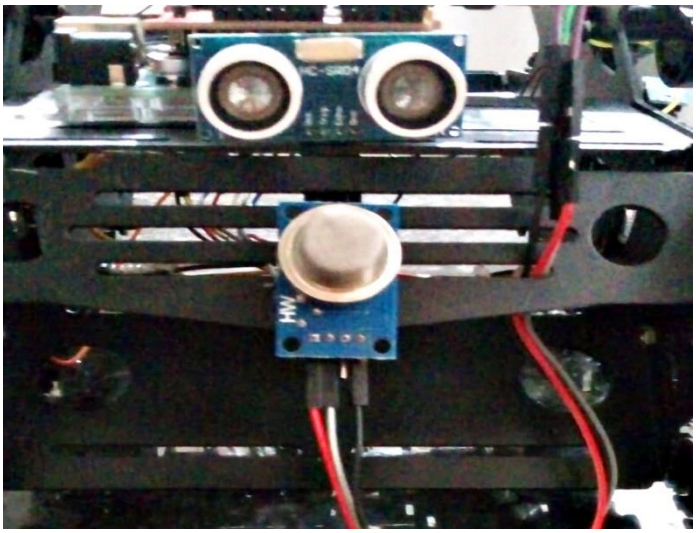
Figura 4.6 - Sensores ultrassônicos fixados



Fonte: Autor

O sensor de gás também foi fixado na parte frontal da estrutura, próximo ao sensor ultrassônico, não tem um motivo específico para ter o colocado nesta posição (Figura 4.7).

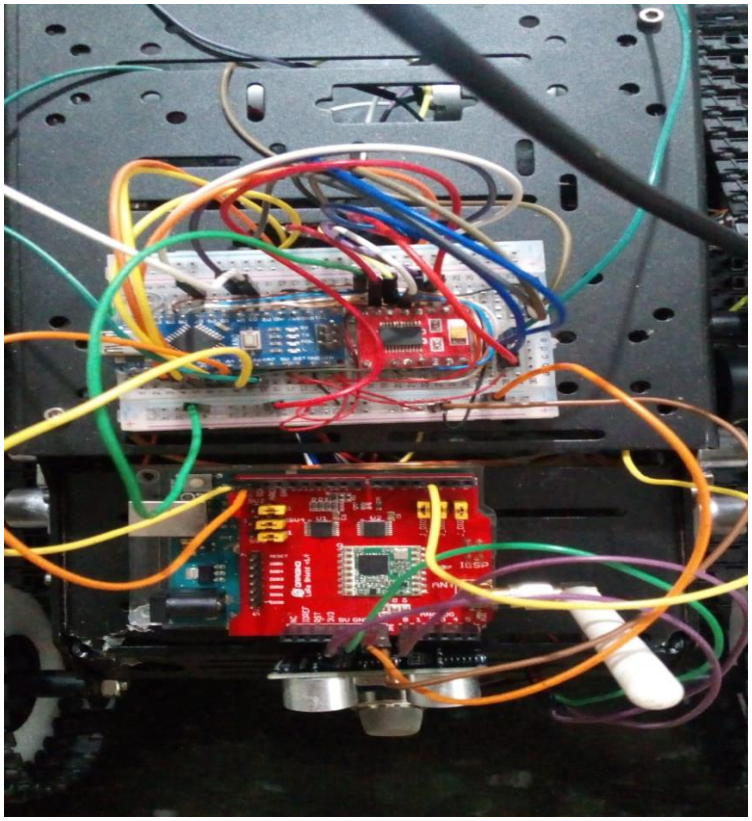
Figura 4.7 - Sensor de gás fixado



Fonte: Autor

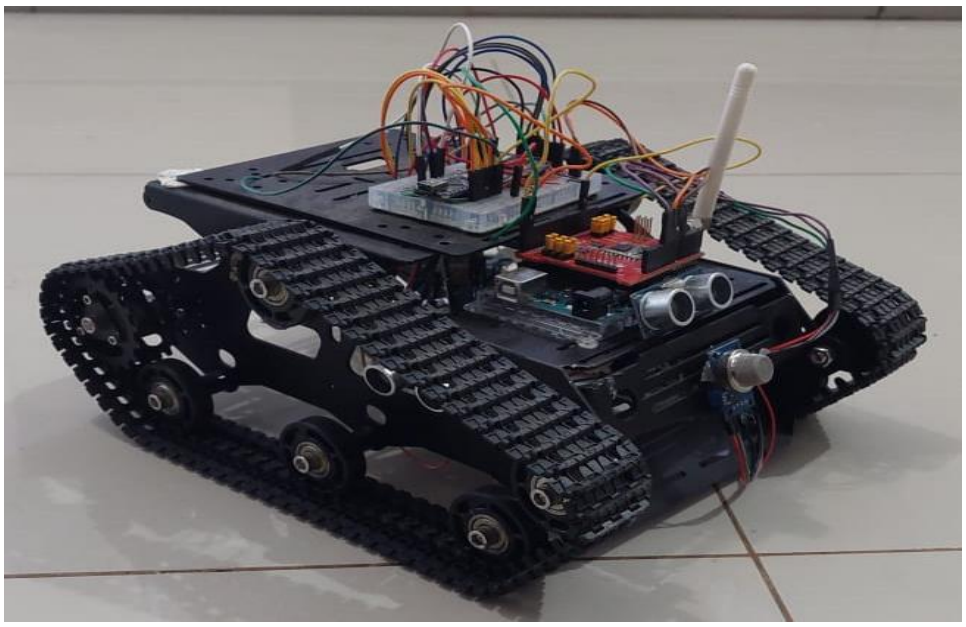
Na parte de cima da estrutura de alumínio foi fixado o Arduino UNO, e um pequeno *protoboard* onde no mesmo está conectado o Arduino nano e a ponte H (Figura 4.8). A Figura 4.9 mostra como ficou a estrutura completa finalizada.

Figura 4.8 - Protoboard com Arduino e ponte H conectados



Fonte: Autor

Figura 4.9 - Estrutura completamente montada



Fonte: Autor

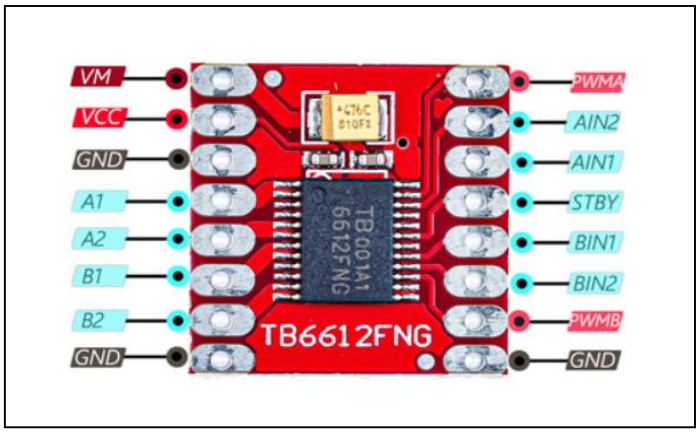
4.2 PROGRAMAÇÃO DOS COMPONENTES ELETRÔNICOS NOS MICROCONTROLADORES

Todos componentes eletrônicos utilizados no projeto assim com suas programações serão detalhados neste item.

4.2.1 Motores e Locomoção do Robô

Para o controle dos motores foi necessário o uso de uma ponte H. A ponte H serve para controlar o sentido da corrente elétrica de entrada no motor, fazendo assim o controle do sentido de rotação dos motores. A ponte H utilizada no projeto foi o módulo duplo driver motor TB6612FNG, esse módulo possui uma base em MOSFET e pode ser aplicado para controlar dois motores DC de forma independente. Além disso ele é compatível com Arduino, AVR, PIC, ARM, Raspberry PI, etc. O módulo possui entrada de 4,5 V a 13,5 V com capacidade de geração de uma corrente contínua de 1A em cada canal, com pico de 3A (Figura 4.10).

Figura 4.10 - Pinagens da ponte H TB6612FNG

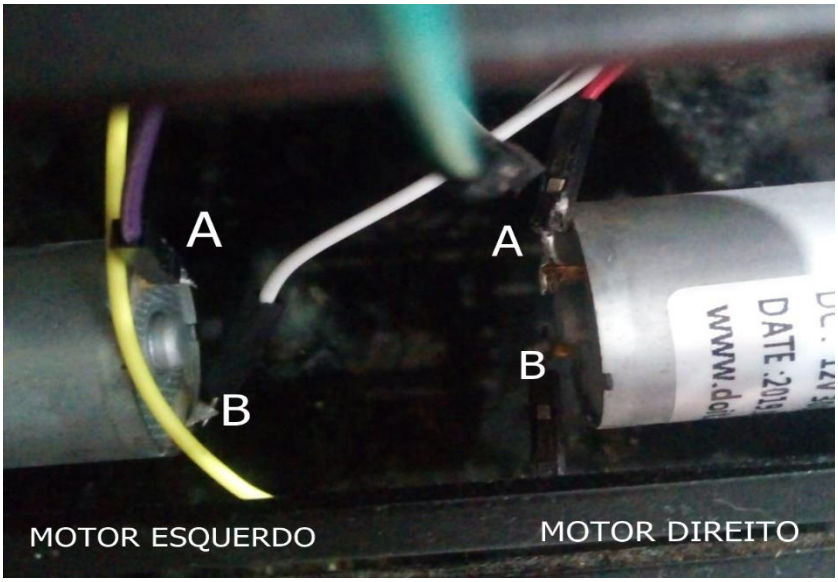


Fonte:[electropeak]

Foi soldado conectores no módulo para colocá-lo na *protoboard* juntamente com o Arduino nano. Os pinos de tensão do motor “VM” e “GND” do módulo foram conectados na saída 5V e no “GND” do Arduino nano respectivamente.

Cada motor do protótipo possui conexões denominadas de A e B e foram soldados *jumpers* nestas conexões (Figura 4.11) para conectá-los no driver.

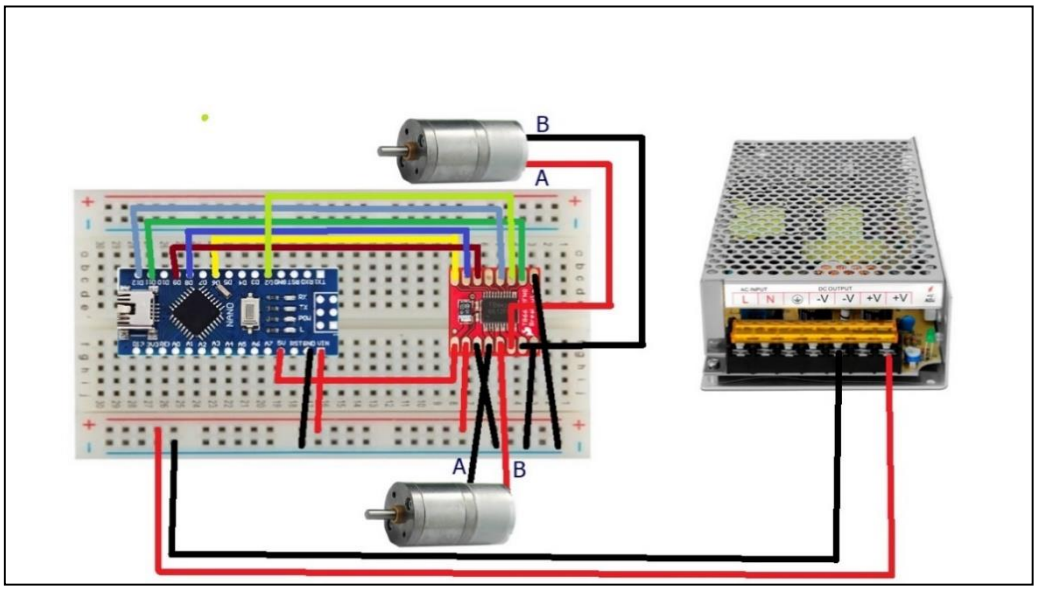
Figura 4.11 - *Jumpers* soldados nas conexões dos motores



Fonte: Autor

O motor esquerdo foi conectado nos pinos “A1” e “A2” e o motor direito foi conectado nos pinos “B1” e “B2”. Os pinos “PWMA” e “PWMB” nos quais são pinos de PWM para controlar a velocidade dos motores, foram conectados nos pinos “D6” e “D11” do Arduino nano. Os pinos referentes aos sinas de controle de direção dos motores, “AIN1”, “AIN2”, “BIN1” e “BIN2” foram conectados nos pinos “D9”, “D8”, “D12” e “D2”. Para fins de testes foi utilizado uma fonte de 12 V para alimentar o módulo e o Arduino nano, vale ressaltar que os “GND’s” do módulo, Arduino nano e da fonte foram conectados entre si. A Figura 4.12 a seguir mostra o esquemático do circuito:

Figura 4.12 - Esquemático do circuito dos motores



Fonte: Adaptada pelo Autor

Foram usadas variáveis auxiliares pra definir os motores e direção de giro. A variável “MotorEsq” serve para identificar que é o motor esquerdo enquanto a variável “MotorDir” serve pra identificar que é o motor direito, as variáveis “Frente” e “Tras” são utilizadas para definir a direção de giro do motor. A variáveis “PWMA”, “AIN1” E “AIN2” referentes ao motor esquerdo foram definidas para os pinos “6”, “9” e “8” do Arduino nano, respectivamente. Já as variáveis “PWMB”, “BIN1” E “BIN2” referentes ao motor direito foram definidas para os pinos “11”, “12” e “2” do Arduino nano respectivamente (Quadro 4.1).

Quadro 4.1 - Referência das pinagens da ponte H no código

```
int Frente = 0;
int Tras = 1;
int MotorEsq = 1;
int MotorDir = 2;
int PWMA = 6;
int AIN1 = 9;
int AIN2 = 8;
int PWMB = 11;
int BIN1 = 12;
int BIN2 = 2;
```

Fonte: Autor

Cada motor possui sua direção controlada por dois pinos do módulo, sendo “AIN1”, “AIN2” do motor direito e os pinos, “BIN1” e “BIN2” motor esquerdo. Escrevendo nível alto ou baixo nesses pinos define-se a direção de giro do motor sendo horário ou anti-horário. Já a velocidade é escrita nos pinos de PWM do driver onde pode variar de 0 a 255, vale lembrar que cada motor tem seu pino de configuração de PWM. O Quadro 4.2 mostra a função “move()” criada, função na qual foi utilizada para definir o motor, velocidade e direção de giro do motor:

Quadro 4.2 - Código de controle dos motores

```
void move(int motor, int speed, int direction) {
  boolean inPin1 = LOW;
  boolean inPin2 = HIGH;
  if (direction == 1) {
   inPin1 = HIGH;
   inPin2 = LOW;
  }
  if (motor == 1) {
   digitalWrite(AIN1, inPin1);
   digitalWrite(AIN2, inPin2);
   analogWrite(PWMA, speed);
  } else if (motor == 2) {
   digitalWrite(BIN1, inPin1);
   digitalWrite(BIN2, inPin2);
   analogWrite(PWMB, speed);
  }
 }
```

Fonte: Autor

As variáveis “inPin1” e “inPin2” já foram declaradas nesse padrão “LOW” e “RIGHT” respectivamente, pois escrevendo esses valores nos respectivos pinos de controle de direção, o

motor irá girar no sentido horário, e caso seja “RIGHT” e “LOW” o motor irá girar no sentido anti-horário. Por fim escreve-se o valor de velocidade desejado nos pinos de PWM sendo “PWMA” e “PWMB”, vale ressaltar que o valor a ser escrito tem que ser entre 0 e 255.

Para fins de teste, para ver se realmente os motores iriam girar no sentido esperado, foi definido para os mesmos girarem no sentido horário (“frente”) com uma velocidade de 50. Para girar no sentido anti-horário teria que passar a variável auxiliar (“trás”) como parâmetro de direção na função (Quadro 4.3).

Quadro 4.3 - Código de teste de funcionamento dos motores

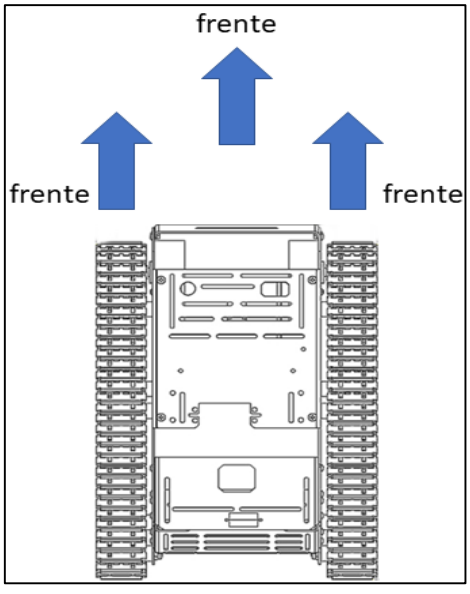
```
void loop() {
  move(MotorEsq, 50, Frente);
  move(MotorDir, 50, Frente);
}
```

Fonte: Autor

Caso os motores girassem ao contrário do esperado, teria que inverter os fios A e B dos motores onde estão conectados os pinos “AIN1”, “AIN2” ou “BIN1” e “BIN2”, mas não foi o caso pois os mesmos giraram no sentido esperado.

Para o robô se locomover foram criados quatro tipos de movimento, sendo eles o movimento de ir para frente, girar a direita em torno do próprio eixo, girar a esquerda em torno do próprio eixo e o movimento de girar para trás. Estes movimentos foram criados em funções diferentes a fim de serem usadas em situações específicas. A função “Forward()” (Quadro 4.4) combina as direções das rodas de condução para frente fazendo com que o robô se mova para frente (Figura 4.13).

Figura 4.13 - Movimento do robô para frente



Fonte: Autor

Quadro 4.4 -Código de locomoção para frente

```
void Forward () {
 move(MotorEsq, 145, Frente);
 move(MotorDir, 150, Frente);
}
```

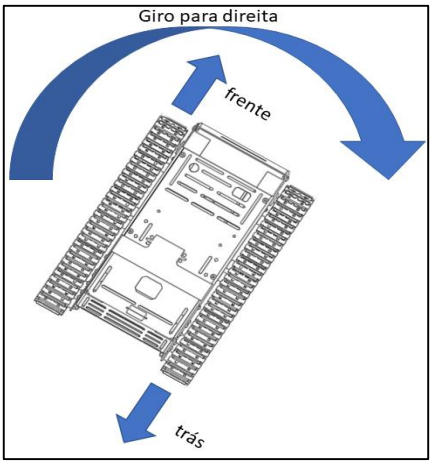
Fonte: Autor

Usando a mesma velocidade para os dois motores foi notado durantes os testes que o robô dava uma leve inclinada para a direita, para eliminar este efeito aumentou-se um pouco a velocidade do motor direito. Foi observado que o motor direito é mais potente que o motor esquerdo.

Para que o robô se movimentasse de forma mais lenta e suave foi adotada uma velocidade de “145” e “150” para os motores. Este valor foi o suficiente para manter o robô movendo para frente.

A função “TurnRight()” (Quadro 4.5) combina as direções do motor esquerdo girando no sentido horário e o motor direito girando no sentido anti-horário fazendo com que o robô gire a direita em torno do próprio eixo (Figura 4.14).

Figura 4.14 - Movimento do robô para direita



Fonte: Autor

Quadro 4.5 - Código de locomoção para direita

```
void TurnRight () {
move(MotorEsq, 150, Frente);
move(MotorDir, 145, Tras);
}
```

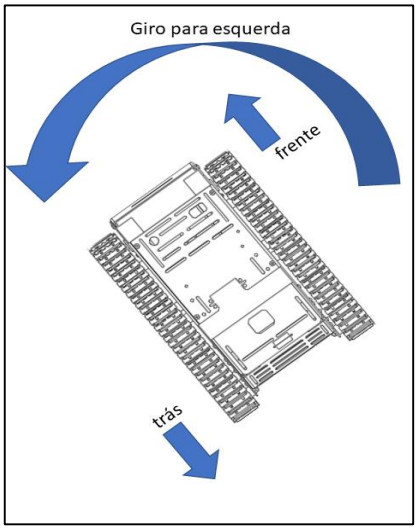
Fonte: Autor

Para o giro a direita foi utilizado os valores de “150” e “145” de velocidade, com estes valores teve um resultado de giro em torno do próprio eixo de maneira satisfatória.

Para o giro a esquerda foi criada a função “TurnLeft()” (Quadro 4.6), esta função combina as direções de giro do motor esquerdo girando no sentido anti-horário e o motor direito girando

no sentido horário, fazendo com que o robô gire a esquerda em torno do próprio eixo (Figura 4.15).

Figura 4.15 - Movimento do robô para a esquerda



Fonte: Autor

Quadro 4.6 - Código de locomoção para esquerda

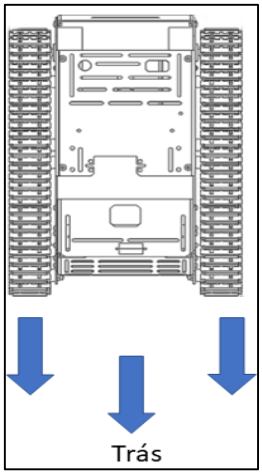
```
void TurnLeft () {
move(MotorEsq, 145, Frente);
move(MotorDir, 150, Tras);
}
```

Fonte: Autor

Como o motor direito é mais potente, a diferença entre as velocidades dos motores foi um pouco maior, os valores de “145” e “150” foram satisfatórios para uma curva suave à esquerda em torno do próprio eixo.

Por fim o último movimento é o de ré, para este movimento foi criada uma função chamada “Backward()”, neste movimento o robô gira para trás (Figura 4.16 e Quadro 4.7).

Figura 4.16 - Movimento do robô para trás



Fonte: Autor

Quadro 4.7 - Código de locomoção para trás

```
void Backward () {
move(MotorEsq, 145, Tras);
move(MotorDir, 150, Tras);
}
```

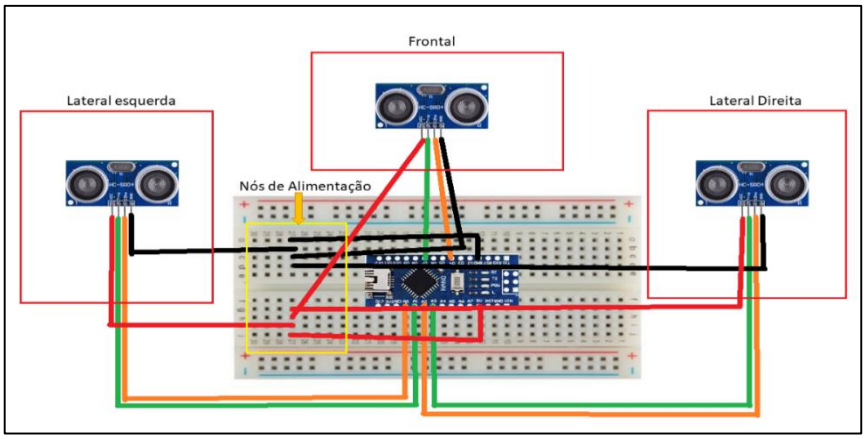
Fonte: Autor

Foi utilizada as mesmas velocidades do movimento para frente.

4.2.2 Sensor ultrassônico e lógica de autonomia do robô

Neste projeto foram utilizados três sensores ultrassônicos do tipo HC-SR04. Os pinos "trig" e "echo" do sensor ultrassônico que fica localizado na parte frontal do robô foram conectados nos pinos "D7" e "D4" do Arduino nano. Já o sensor ultrassônico que fica localizado na parte lateral esquerda do robô teve seus pinos "trig" e "echo" conectados nos pinos "A1" e "A0" do Arduino nano. E por fim, o sensor ultrassônico que fica na parte lateral direita do robô teve seus pinos "trig" e "echo" conectados nos pinos "A3" e "A2" do Arduino nano. Os pinos de alimentação dos sensores, "VCC" e "GND" foram conectados na *protoboard* em nós comuns e os pinos de saída de 5V e GND do Arduino nano foram conectados a esses nós (Figura 4.17).

Figura 4.17 - Esquemático do circuito dos sensores ultrassônicos



Fonte: Autor

Por meio do sensor ultrassônico pode-se detectar objetos que podem estar a distâncias de 2cm a 4m. O princípio de funcionamento de detecção de objetos por meio deste sensor, consiste em enviar um pulso por meio do Arduino ao pino "*trigger*" do sensor, quando o sensor recebe esse sinal o mesmo emite pulsos e detecta se há algum sinal de retorno. Se for identificado um sinal de retorno, o sensor gera um sinal de nível alto no pino de saída "*echo*". A duração desse sinal é igual a variação do tempo do envio e retorno do sinal ultrassônico.

Sabendo que a velocidade do som é de 340 m/s e que se o sensor estiver a uma distância "d" do objeto, o sinal percorrerá uma distância 2d, já que o sinal é emitido e depois retorna ao sensor. Convertendo a velocidade do som para centímetros por microssegundos (por causa da unidade de armazenamento do Arduino) tem-se um valor de 0,034 cm/µs.

$$340\,\frac{\mathrm{m}}{\mathrm{s}} = \frac{34000\ \mathrm{cm}}{10^6 \mu\mathrm{s}} = 0{,}034\,\frac{\mathrm{cm}}{\mu\mathrm{s}} \qquad (4.1)$$

E para saber quanto tempo leva para o som percorrer um centímetro tem-se que:

$$\frac{1\mu s}{0{,}034\ cm} = 29{,}4117647059\,\frac{\mu s}{cm} \qquad (4.2)$$

A distância em centímetros do sensor ao objeto pode ser calculada dividindo o tempo de duração do sinal por 29,4 e depois dividir por dois por causa da distância percorrida pelo sinal:

$$d(cm) = (\frac{duracao\ do\ sinal}{29{,}4})/2 \qquad (4.3)$$

A configuração das variáveis e pinagens no código foram feitas. As pinagens do "*trigger*" e "*echo*" do sensor frontal foram definidos como "7" e "4" respectivamente, de maneira análoga as pinagens do sensor na lateral direita foram definidas como "A3" e "A2" e por fim as pinagens do sensor da lateral esquerda foram definidos como "A1" e "A2". Foram criadas variáveis referentes ao tempo de duração do sinal de saída sendo essas variáveis, "Duration1", "Duration2" e "Duration3". Também foram criadas variáveis de retorno das funções de medição das distâncias, nas quais são "distancia_cm1", "distancia_cm2" e "distancia_cm3". Foram criadas variáveis que armazenam os valores retornados dessas funções, nas quais são "RightDistance", "LeftDistance" e "MiddletDistance" (Quadro 4.8).

Quadro 4.8 - Variáveis necessárias para programar os sensores ultrassônicos

```
#define EchoPIN1 4
#define TrigPIN1  7
#define EchoPIN2  A2
#define TrigPIN2 A3
#define EchoPIN3  A0
#define TrigPIN3  A1
long Duration1;
long Duration2;
long Duration3;
long distancia_cm1 = 0;
long distancia_cm2 = 0;
long distancia_cm3 = 0;
float RightDistance = 0;
float LeftDistance = 0;
```

```
float MiddletDistance = 0;
```

Fonte: Autor

Foram criadas três funções de medição sendo uma função pra cada sensor. A função responsável pela medição no sensor da parte frontal do robô foi denominada de “Distance_Middle()”, do sensor que fica na parte lateral direita do robô foi denominada de “Distance_Right()” e do sensor da parte lateral esquerda do robô foi denominada de “Distance_Left()” (Quadro 4.9).

Quadro 4.9 - Código das funções de leitura dos sensores ultrassônicos

```
float Distance_Right (void) {
  digitalWrite (TrigPIN2, LOW);
  delayMicroseconds (2);
  digitalWrite (TrigPIN2, HIGH);
  delayMicroseconds (20);
  digitalWrite (TrigPIN2, LOW);
  Duration2 = pulseIn (EchoPIN2, HIGH);
  distancia_cm2 = (Duration2/29/2);
  return distancia_cm2;
}
float Distance_Left (void) {
  digitalWrite (TrigPIN3, LOW);
  delayMicroseconds (2);
  digitalWrite (TrigPIN3, HIGH);
  delayMicroseconds (20);
  digitalWrite (TrigPIN3, LOW);
  Duration3 = pulseIn (EchoPIN3, HIGH);
  distancia_cm3 = (Duration3/29/2);
  return distancia_cm3;
}
float Distance_Middle (void) {
  digitalWrite (TrigPIN1, LOW);
  delayMicroseconds (2);
  digitalWrite (TrigPIN1, HIGH);
  delayMicroseconds (20);
  digitalWrite (TrigPIN1, LOW);
  Duration1 = pulseIn (EchoPIN1, HIGH);
  distancia_cm1 = (Duration1/29/2);
  return distancia_cm1;
}
```

Fonte: Autor

Essas funções retornam o valor da distância em centímetros do sensor ao objeto detectado. A lógica dessas funções é emitir pulsos ultrassônicos do pino "*trigger*" por 20μs, e por meio da função "pulseIn" tem-se a duração do pulso de retorno no pino "*echo*". A função "pulseIn" lê um pulso sendo ele alto ou baixo (neste caso alto) e retorna a duração do pulso em microssegundos ou retorna 0 se nenhum pulso completo foi recebido (Arduino 2022). Por fim, calcula-se a distância em cm do sensor ao objeto utilizando a equação 4.3.

Durante a execução, as distâncias eram retornadas constantemente e mostradas no "Monitor Serial". Para conferir se a distância do sensor ao objeto estava realmente coerente foi utilizado uma régua para comparar com as distâncias retornadas do sensor, no teste em questão o objeto estava à uma distância de 10 cm. Ambas as medições estavam corretas, ou seja, o sensor funcionou de maneira satisfatória.

A lógica de autonomia foi desenvolvida combinando os movimentos do robô descritos na sessão 4.2.2 com as medições das distâncias dos sensores ao obstáculo detectado. As funções de medição das distâncias são chamadas dentro da função "loop()" e as variáveis "MiddletDistance", "RightDistance" e "LeftDistance" armazenam os valores lidos (Quadro 4.10).

Quadro 4.10 - Armazenamento dos valores lidos pelos sensores ultrassônicos

```
void loop(){
  LeftDistance = Distance_Left();
  RightDistance = Distance_Right();
  MiddletDistance = Distance_Middle();
}
```

Fonte: Autor

Ainda dentro da função "loop()" foram criadas condições para que o robô se desvie do obstáculo mudando de direção. Caso a distância lida do obstáculo ao sensor da parte frontal do robô ("MiddletDistance") for menor ou igual a 10 cm, verifica-se qual é a maior distância lida dos sensores das laterais do robô ("RightDistance" e "LeftDistance"), o sensor que apresentar maior distância lida, será a direção na qual o robô irá girar em torno do próprio eixo, sendo pra direita utilizando a função "TurnRight()" , ou esquerda usando a função "TurnLeft()". Caso as condições de comparação das distâncias dos sensores laterais não forem atendidas para girar para esquerda ou direita o robô gira para trás por meio da função "Backward()". Agora caso a distância do sensor frontal ("MiddletDistance") ao obstáculo for superior a 10 cm, mas a distância lida do sensor da parte lateral direita do robô ("RightDistance") for menor ou igual a 5cm o robô gira em torno do próprio eixo para a esquerda por meio da função "TurnLeft()", e

caso a distância lida do sensor da parte lateral esquerda do robô (“LeftDistance”) for menor ou igual a 5cm o robô gira em torno do próprio eixo para a direita por meio da função “TurnRight()”. Por fim se a distância do sensor frontal (“MiddletDistance”) ao obstáculo for superior a 10 cm e as distancias lidas dos sensores laterais, “RightDistance” e “LeftDistance” forem maior que 5 cm o robô se move pra frente utilizando a função “Forward()” (Quadro 4.11).

Quadro 4.11 - Código referente a autonomia do robô.

```
void loop() {
  if(MiddletDistance<=10){
    if(RightDistance > LeftDistance ){
        TurnRight();
    }else if(RightDistance < LeftDistance){
        TurnLeft();
    }else{
        Backward();
    }
  }else if(RightDistance<=5){
      TurnLeft();
  }else if(LeftDistance<=5){
      TurnRight();
  }else{
       Forward();
  }
}
```

Fonte: Autor

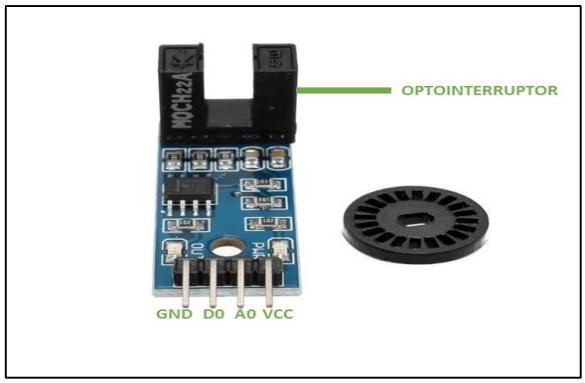
4.2.3 *Encoder* Óptico e Implementação da Interrupção

O *encoder* óptico é um tipo de sensor rotativo que é usado pra identificar mudança de posição à medida que a luz passa através do disco furado, por isso, esse sensor foi utilizado para se obter dados de deslocamento do robô. O *encoder* e disco furado é fixado na roda de condução, e quando a mesma gira, gera pulsos quando a luz é capturada no disco furado e estes pulsos são traduzidos pelo Arduino.

O *encoder* utilizado neste projeto foi o “LM293” (Figura 4.18), este sensor possui 4 pinos sendo dois de alimentação (3 a 5V e GND), uma saída digital (DO) e uma analógica (A0), o sensor vem com um disco perfurado com 20 furos. O sensor possui um opto interruptor no qual

tem de um lado um led infravermelho e do outro lado um fototransistor. Quando o feixe de luz infravermelha é interrompido, a saída D0 do sensor envia sinal 1, caso contrário envia 0.

Figura 4.18 - *Encoder* óptico LM293

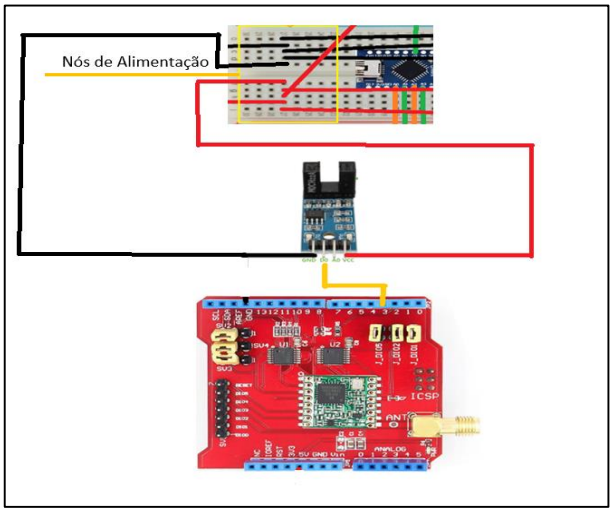


Fonte: Adaptada pelo Autor

Foi utilizado um único *encoder* na roda de condução que fica na parte esquerda da estrutura. Como mencionado na sessão 4.1 o chassi utilizado no projeto não possui adaptabilidade para este sensor, com isso foi necessário fixar o disco furado na roda de condução e colocar o sensor de forma suspensa no disco. Com auxílio dos *jumpers* utilizados para conexão e cola quente foi possível fixar o sensor de forma que o mesmo ficou firme na estrutura.

Para programar o sensor foi utilizado o Arduino UNO no qual possui um *Shield* LoRa acoplado, os fios de "5V" e "GND" do sensor foram conectados na *protoboard* nos mesmos nós onde é a alimentação dos sensores ultrassônicos, alimentação na qual é feita pelo Arduino nano, já o pino "D0" do sensor foi conectado ao pino "D3" do Arduino UNO (Figura 4.19). Este pino foi escolhido devido ao fato de o mesmo poder ser usado para sistemas nos quais envolvem interrupção (*datasheet*).

Figura 4.19 - Esquemático do circuito do *encoder* óptico



Fonte: Adaptada pelo Autor

Interrupções são funções que executam tarefas especificas paralelamente ao programa principal que no nosso caso é tudo que está dentro da função “loop()” do Arduino. Interrupções são bastante úteis em tarefas nas quais precisa de coletar dados em espaços curtos de tempo, que é o caso deste projeto.

Se a função de contagem pulsos gerados pelo *encoder* estivesse na função “loop()” do Arduino não seria possível contabilizar todos pulsos gerados pelo *encoder* a tempo, gerando um erro de contabilização, fazendo com que prejudicasse a precisão do sensor.

A função responsável por habilitar a interrupção no Arduino é a “attachInterrupt()”, essa função recebe três parâmetros sendo eles o pino digital específico da interrupção que no caso deste projeto está sendo utilizado o pino “D3”, pino no qual é utilizado como entrada de dados dos pulsos recebidos, o segundo parâmetro é a função na qual necessita ser executada e o terceiro parâmetro é modo de como a interrupção deve ser ativada. O modo utilizado para ativar a interrupção foi o modo “RISING”, nesse modo a interrupção é ativada toda vez que o pino vai do estado “LOW” para “HIGH”.

No código, o primeiro parâmetro passado para a função “attachInterrupt()” foi o “1” , o programa entende que “0” é o pino “D2” e “1” é o pino “D3” do Arduino. O segundo parâmetro é a função criada chamada “ContadorAnguloPosicao()” e o terceiro parâmetro é o modo “RISING”(Quadro 4.12).

Quadro 4.12 - Código da função de interrupção

```
pinMode(3, INPUT);
attachInterrupt(1, ContadorAnguloPosicao, RISING);
```

Fonte: Autor.

A função “attachInterrupt()” foi chamada dentro da função “setup()” do Arduino, essa função é chamada toda vez que o código se inicializa com suas devidas configurações de pinagem e tarefas.

O sensor possui um led de dados que acende toda vez que a luz infravermelha é interrompida pelo disco. Com isso fazendo testes com *encoder*, uma única vez que o led de dados do mesmo acendia, o programa acusava 5 contagens de pulsos na função “ContadorAnguloPosicao()”. Para eliminar esse erro foi criado um limitador de tempo com a função “millis()” do Arduino, essa função retorna o número de milissegundos passados desde que o Arduino começou a executar o programa. Por meio desta função criou-se uma lógica para que mesmo se ocorressem várias capturas de pulsos, iria ser contabilizado somente um pulso por execução. A lógica consiste em se a diferença entre o tempo atual e o tempo do programa

em execução for maior que 1 millisegundo, o pulso é contabilizado. A variável criada pra armazenar o tempo atual foi chamada de “delayCont”, (Quadro 4.13).

Quadro 4.13 - Lógica de contagem de pulsos

```
void ContadorAnguloPosicao(){
  static unsigned long delayCont;
  if (millis()-delayCont>1){
contador= contador+1;
}
delayCont=millis();
}
```

Fonte: Autor

Com a execução dessa lógica o número de pulsos gerados em uma volta completa foi 40.

4.2.4 Implementação da Lógica de Deslocamento

O modelo cinemático de deslocamento do robô descrito no referencial teórico foi implementado no Arduino UNO, onde os valores de posição descritos pelas componentes X e Y são frequentemente atualizados, vale ressaltar que a implementação foi feita baseado em [106].

A atualização dos valores de X e Y dependem da geração dos pulsos pelo *encoder* enquanto o robô movimenta. Vale ressaltar que o *encoder* não distingue o sentido de rotação da roda, com isso foi necessário a criação de uma variável auxiliar chamada de “status_sentido” variável na qual é utilizada para indicar que tipo de movimento o robô está fazendo. Para cada função de movimento criada no Arduino nano sendo elas, “Forward()”, “Backward()”, “TurnRight()” e “TurnLeft()”o valor de “status_sentido” muda, e esse valor é constantemente enviado para o Arduino UNO por meio da função “enviarDadosSerial(status_sentido)” (Quadro 4.14). A comunicação entre os Arduinos será detalhada em outra sessão.

Quadro 4.14 - Lógica para se obter informação de sentido do robô

```
void Forward () {
 status_sentido = 4;
 enviarDadosSerial(status_sentido);
 move(MotorEsq, 145, Frente);
 move(MotorDir, 150, Frente);
}
void TurnRight () {
 status_sentido = 1;
 enviarDadosSerial(status_sentido);
move(MotorEsq, 150, Frente);
```

```
move(MotorDir, 145, Tras);
}
void TurnLeft () {
 status_sentido = 2;
 enviarDadosSerial(status_sentido);
 move(MotorEsq, 145, Tras);
 move(MotorDir, 150, Frente);
}
void Backward () {
status_sentido = 3
enviarDadosSerial(status_sentido);
 move(MotorEsq, 145, Tras);
 move(MotorDir, 150, Tras);
}
```

Fonte: Autor

Uma variável auxiliar chamada de “sentido” armazena o valor de “status_sentido” vinda do Arduino nano. Essa variável é usada para realizar condições dentro da função “ContadorAnguloPosicao()”, função na qual é acessada toda vez que uma interrupção acontece por meio da geração de pulsos do *encoder*.

Por exemplo se a condição acessada é “sentido = 4” (robô se movendo para frente) , então é armazenado o valor do contador de pulsos nas variáveis “contadorX” e “contadorY”. A implementação da lógica de deslocamento foi feita baseada na equação: $X_{atual} = X_{anterior} + \cos(\theta)$ e $Y_{atual} = Y_{anterior} + \text{sen}(\theta)$. A váriável “contadorX” e “contadorY” e “ângulo” é inicializada com um valor, valores nos quais representam o ponto de partida do robô (Quadro 4.15), toda vez que um novo pulso é gerado essas variáveis são atualizadas. Sempre é calculado um novo valor para “contadorX”, somando ou subtraindo cosseno do ângulo, e sempre é calculado o novo valor “contadorY” somando ou subtraindo com o seno do ângulo (Quadro 4.16 e Quadro 4.17).

Quadro 4.15 - Valores de partida do robô

```
angulo=0;
contadorX=50;
 contadorY=50;
```

Fonte: [106]

Quadro 4.16 - Código de acréscimo do contador

```
contadorX= contadorX+1*cos(angulo);
contadorY= contadorY+1*sin(angulo);
```

Fonte: [106]

Quadro 4.17 - Código de decréscimo do contador

```
contadorX= contadorX-1*cos(angulo);
contadorY= contadorY-1*sin(angulo);
```

Fonte: [106]

O valor da variável "ângulo" depende do valor de uma variável chamada "contadorr", variável na qual serve para armazenar uma quantidade de pulsos durante o giro, de forma que se o giro é pra esquerda, ou seja, "sentido = 2" é subtraído uma unidade a cada novo pulso gerado. Já se o robô gira para direita, ou seja, "sentido = 1", é somado uma unidade no "contadorr". Para converter a variável "contadorr" em ângulo em radianos, foi necessário contabilizar quantos pulsos são gerados quando o robô dá uma volta completa (360°) em torno do próprio eixo.

Foi feito um teste para ver quantos pulsos eram gerados quando o robô dava uma volta completa em torno do próprio eixo, foi coletado esses dados com o robô fazendo esse movimento para direita e para esquerda para depois fazer uma média simples destes valores. Essa média foi feita devido ao fato de o robô girar com velocidades diferentes para direita e esquerda. Foi gerado 225 pulsos com o robô dando uma volta completa girando à direita, e foi gerado 219 pulsos com o robô girando à esquerda. Fazendo uma média simples temos que:

$$Médi\,ade\,pul\,sosger\,ados = \frac{225+219}{2} = 222\,pul\,sos \qquad (4.11)$$

Se para uma volta completa, ou seja, 2π rad, são gerados 222 pulsos então significa que para π rad são gerados 111 pulsos. A proporção entre o ângulo de giro θ para π rad é igual à proporção do número de pulsos gerados para 111 pulsos.

$$\frac{\theta}{\pi} = \frac{número\,de\,pul\,sosger\,ados}{111}$$

$$\theta = número\,de\,pul\,sosger\,ados \times \frac{\pi}{111}$$

$$\theta = número\,de\,pul\,sosger\,ados \times \frac{3{,}1415}{111} \qquad (4.12)$$

A equação mostrada acima é usada para converter a variável "contadorr" em "ângulo" (Quadro 4.18).

Quadro 4.18 - Código de conversão de contador para ângulo

```
angulo=contadorr*3.1415/111;
```

Fonte: Adaptada pelo Autor.

Para que o valor de "contadorr" não cresca de forma infinita fazendo com que atrapalhe o processamento do Arduino e comece a gerar erros, o valor dessa variável foi limitada de forma

que se o “ângulo” for maior ou igual a 6.283, ou seja, 2π, a variável “contadorr” é zerada (Quadro 4.19).

Quadro 4.19 - Código para zerar o ângulo

```
if(angulo>=6.283)
contadorr=0;
```

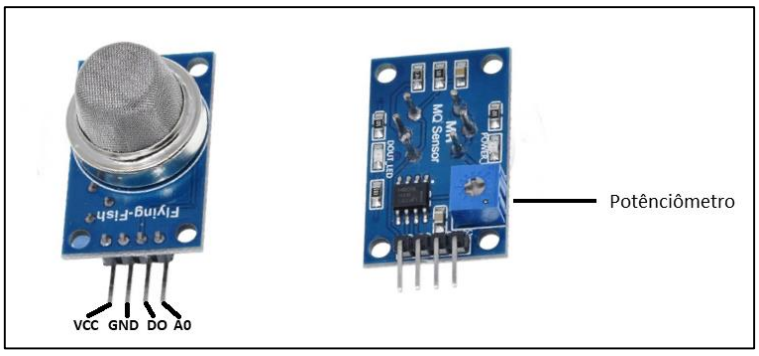
Fonte: Adaptada pelo Autor.

4.2.5 Sensor de Gás MQ-4 e MQ-6

Para fins de testes iniciais de detecção de gás, foi utilizado o sensor MQ-6 (Figura 4.20), ele possui alta sensibilidade para gás de cozinha, isobutano e propano. Ele é compatível com microcontroladores como Arduino ou placas como Raspberry Pi. Esse sensor foi usado para teste inicial devido a facilidade e baixo risco de testá-lo, foi utilizado um isqueiro para testá-lo. Uma das composições do fluído do isqueiro é o Butano, composto no qual o sensor MQ-6 é sensível à sua concentação.

O princípio de funcionamento de um sensor MQ é que quando a concentração dos gases nos quais o sensor MQ é sensível fica acima do nível ajustado pelo potênciometro do sensor, a saída digital “DO” fica em estado alto, se abaixo do nível, fica em estado baixo. Além disso é possível também medir o nível de concentração de gás por meio da sua saída analógica “A0”.

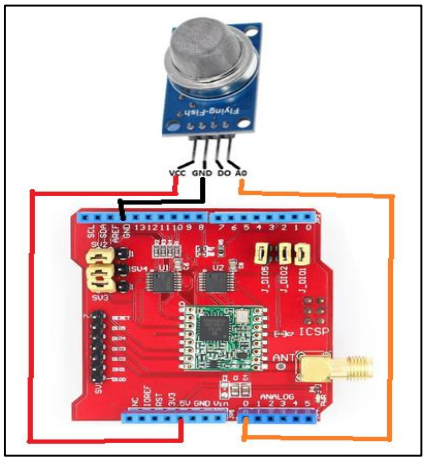
Figura 4.20 - Sensor MQ-6



Fonte: Autor

O sensor MQ-6 foi conectado à *shield* LoRa que fica acoplada no Arduino UNO (Figura 4.21), sendo que o pino de “VCC” e “GND” do sensor foi conectado no pino de saída de “5V” e “GND” do Arduino UNO respectivamente, o pino de dados “A0” do sensor foi conectado no pino “A0” do Arduino UNO, esse pino foi configurado como entrada de dados.

Figura 4.21 - Esquemático de conexão do sensor na *shield*



Fonte: Autor

Foi criado uma variável chamada “leitura”, a mesma é responsável por receber os dados da leitura vinda do sensor, para ler os valores do sensor foi utilizado a função “analogRead(pino)” do Arduino. Por meio dessa função é feita a leitura de um pino analógico específico (Arduino 2022), neste caso, o pino “A0” (Quadro 4.20).

Quadro 4.20 - Armazenamento da leitura do sensor de gás

```
leitura = analogRead(A0);
```

Fonte: Autor

Como já mencionado, os testes inicias foram feitos emitindo o gás do isqueiro diretamente no sensor (Figura 4.22) para ver os dados coletados no monitor serial do Arduino.

Figura 4.22 - Emitindo gás no sensor diretamente

Fonte: Autor

Emitindo o gás diretamente no sensor foi mostrado valores de leitura no monitor que variavam entre 500 e 800, sem emitir gás no sensor, mostrava valores entre 25 e 75 no monitor.

O sensor de gás MQ-4 no qual foi utilizado para teste final (Figura 4.23), possui características análogas ao sensor MQ-6 e a sua conexão com o Arduino UNO assim como sua programação foi a mesma também. O gás que o sensor MQ-4 é sensível é o gás metano, gás no qual foi objeto de estudo no teste final.

Figura 4.23 - Sensor MQ-6

Fonte: Autor

4.2.6 Comunicação entre Arduinos

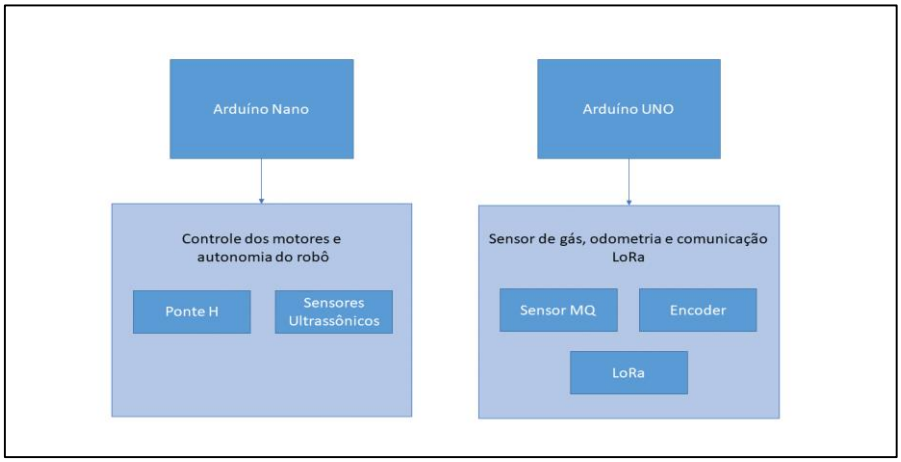
Esse trabalho envolve uma quantidade considerável de periféricos eutilização do protocolo LoRa, com isso, o risco de se esgotar a memória do MCU é alta, além disso, com muitas operações sendo feitas ao mesmo tempo, a probabilidade de travar o processamento do código é considerável. Por essas razões foram utilizados dois Arduinos para o trabalho.

É importante ressaltar que foi utilizado um Arduino nano e um Arduino UNO para realizar o projeto, sendo que o Arduino nano foi utilizado para fazer o controle dos motores e autonomia do robô. Já o Arduino UNO foi utilizado para receber as informações do sensor de gás, estimação da posição do robô e envio dessas informações para o *gateway* via LoRa (Figura 4.24).

O protocolo de comunicação utilizado para a comunicação entre os Arduinos foi o I2C. O protocolo I2C utiliza dois pinos, “SDA” e “SCL”, onde o pino “SDA” é o pino de dados e o “SCL” é o pino de clock. Por meio desse protocolo é possível transmitir e receber informações, mas não ao mesmo tempo. Além disso tem-se a formação de um barramento endereçavel onde cada componente na rede possui um endereço para ser identificado para que a informação enviada possa ser desiginada para o destino de forma correta.

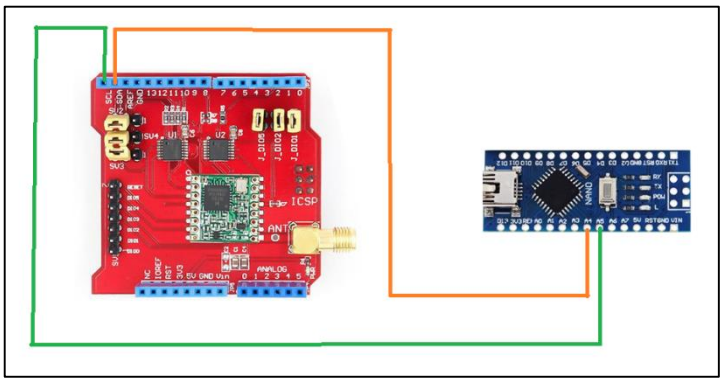
Os pinos “SDA” e “SCL” do Arduino UNO no qual possui a *shield* LoRa acopadado, foram conectados nos pinos “A4” e “A5” do Arduino nano respectivamente. Esses pinos analógicos também funcionam como pino de dados e de clock respectivamente (Figura 4.25).

Figura 4.24 - Diagrama de blocos das funcionalidades dos Arduinos



Fonte: Autor

Figura 4.25 - Esquemático de ligação dos pinos dos Arduinos



Fonte: Autor

A variável “status_sentido” criada no Arduino nano varia o valor de acordo com o movimento que o robô está fazendo. Esse valor é importante para o cálculo de deslocamento do robô, e para enviar essa informação do Arduino nano para o Arduino UNO via I2C primeiro foi importado a biblioteca “Wire.h” no Arduino nano, essa biblioteca possui funções necessárias para gerenciar a comunicação I2C entre dispositivos (Quadro 4.21).

Quadro 4.21 - Inclusão da biblioteca para comunicação I2C

```
#include "Wire.h"
```

Fonte: Autor

Com a biblioteca importada, foi utilizada a sentença “Wire.begin()” dentro da função “setup()” para que a comunicação via I2C fosse iniciada (Quadro 4.22).

Quadro 4.22 - Inicialização da comunicação I2C

```
Wire.begin();
```

Fonte: Autor

Foi criada uma função chamada “enviarDadosSerial(int sentido)”, essa função recebe como parâmetro justamente a varíável “status_sentido”, variável na qual se deseja enviar. Dentro dessa função foi usada a setença “Wire.beginTransmission()” tendo como parâmetro, o endereço do dispositivo no qual queremos enviar que no caso é o Arduino UNO, esse endereço é de 7 bits e foi definido como 0x08. Depois disso foi utilizado a sentença “Wire.Write()”, essa função recebe o parâmetro no qual se quer que envie, neste caso “status_sentido”. Após o envio da informação, é finalizado a transmissão utilizando a sentença “Wire.endTransmission()”.

Foi definido que essa informação fosse enviada em um intervalo de tempo de 500 ms, para isso foi utilizada a função “millis()” e uma variável chamada “previusMillis” na qual guarda o tempo atual , se a diferença entre esse e o tempo que o código foi executado fornecido por “millis()” for maior que 500 ms a informação é enviada (Quadro 4.23).

Quadro 4.23 - Função para enviar dados serial

```
void enviarDadosSerial(int sentido){
  if(millis() - previusMillis > 500){
    Wire.beginTransmission(slaveAdress);
    Wire.write(sentido);
    Wire.endTransmission();
   }
 }
```

Fonte: Autor

Essa função é chamada em todas funções responsáveis pelo movimento do robô.

Para receber a informação no Arduino UNO também foi importado a biblioteca “Wire.h” e dentro da função “setup()” foi utilizada a sentença “Wire.begin()” para inicializar a comunicação I2C, no entanto foi utlizado o valor 0x08 como parâmetro, isso quer dizer que o o Arduino UNO está presente no barramento e possui o endereço 0x08 (Quadro 4.24).

Quadro 4.24 - Endereço no barramento I2C

```
Wire.begin(0x08);
```

Fonte: Autor

Ainda dentro do “setup()” utilizou-se a sentença “Wire.onRecieve()” para determinar qual função seria executada quando chegasse a informação de sentido do robô proveniente do Arduino nano, a função a ser executada foi definida como “receiveEvent()” (Quadro 4.25).

Quadro 4.25 - Função a ser executada quando chegar dados via I2C

```
Wire.onReceive(receiveEvent);
```

Fonte: Autor

Dentro da função “receiveEvent()” foi utilizado a sentença “Wire.read()” para obter o valor que estava sendo enviado pelo Arduino Nano e esse valor é armazenado em uma variável chamada “sentido” (Quadro 4.26).

Quadro 4.26 - Armazenamento dos dados recebidos via I2C

```
void receiveEvent(int leitura) {
  sentido = Wire.read();
}
```

Fonte: Autor

Foi utilizado o monitor serial do Arduino UNO pra monitorar o valor que estava sendo enviado pelo Arduino nano, os valores chegaram corretamente de acordo com cada movimento sendo “4” quando o robô faz o movimento pra frente, “2” quando robô gira para esquerda, “1” quando o robô gira pra direita e “3” quando o robô gira para trás.

4.2.7 *Shield* LoRa e Envio de Dados

Foi utilizado o LoRa *Shield* fabricado pela Dragino para a comunicação LoRa, essa *shield* possui um transceptor de longo alcance baseado no chip SX1276/SX1278. Esse chip adota técnica de modulação LoRa e é patenteado pela Semtech, ele possui uma alta sensibilidade de -148 dBm e essa sensibilidade combinada com um amplificador de 20 dBm faz com que esse chip se torne ideal para qualquer aplicação que exija alcance e robustez (Dragino 2022).

Essa *shield* contém o mesmo número de portas comparada com a placa Arduino UNO, portanto, além de proporcionar a utilização da comunicação LoRa pode-se conectar periféricos na *shield*, os periféricos utilizados foram o *encoder* e o sensor de gás.

Para programar a parte de envio de dados via LoRa na IDE do Arduino, foi necessário fazer o *download* do driver RH_RF95. Com esse driver é possível enviar e receber dados por meio de um transceptor LoRa. O *download* do *driver* está disponível em (Airspayce 2022).

Depois de feito o *download* foi importado o *driver* no código e criado um objeto desse driver chamado de “rf95”, com isso, foi possível utilizar as funções necessárias para a comunicação LoRa (Quadro 4.27).

Quadro 4.27 - Importação do driver RH_RF95

```
#include <RH_RF95.h>
```

```
RH_RF95 rf95;
```

Fonte: Autor

Algumas configurações iniciais são necessárias, como por exemplo a frequência de operação do LoRa, no Brasil é adotado o padrão australiano de 915 MHz. Uma variável chamada de “frequency” foi criada e atrbuida a ela o valor 915, e na função “setup()” foi configurada essa frequência na comunicação por meio da instância “rf95.setFrequency()”, onde essa função recebe como parâmetro o valor da frequência de operação, nesse caso a variável “frequency”(Quadro 4.28).

Quadro 4.28 - Configuração da frequência de operação do LoRa

```
float frequency = 915.0
rf95.setFrequency(frequency);
```

Fonte: Autor

Além dessa configuração também foi configurada a potência do transmissor no transceptor utilizando a instância rf95.setTxPower(). Essa função recebe como parâmetro um valor que pode variar entre 2 e 20 dBm, mas tem como valor padrao 13 dBm segundo a documentação do driver (Quadro 4.29).

Quadro 4.29 - Configuração da potência do transmissor

```
rf95.setTxPower(13);
```

Fonte: Autor

Caso tenha alguma falha na inicialização do *driver*, é emitido uma mensagem no monitor serial do Arduino. Para isso foi usado a instância “rf95.init()”, essa função retorna um valor booleano como padrão *true*, e caso dê falha retorna *false*. Foi criada uma condição para que se a função retornasse um valor diferente do padrão imprimia uma mensagem no monitor serial (Quadro 4.30).

Quadro 4.30 - Verificação de inicialização do driver

```
if (!rf95.init()){
    Serial.println("falha na inicialização do driver");
 }
```

Fonte: Autor

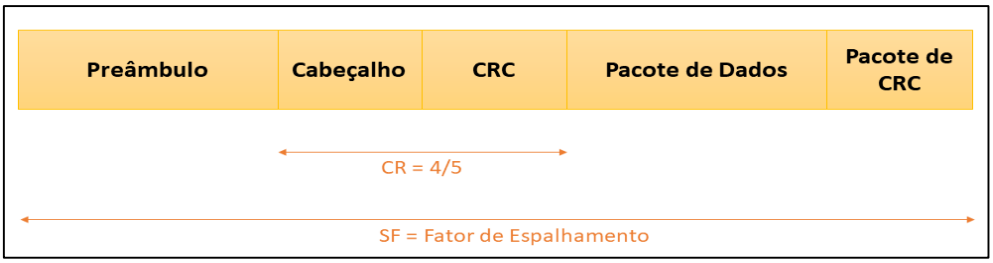
A estrutura do pacote construido para ser enviado para o *gateway* é formado por cinco partes (Figura 4.26), sendo elas o preâmbulo, cabeçalho, CRC, os dados a serem enviados e bytes de CRC, e o que determina a velocidade do *chirp*, é o fator de espalhamento (SF). Foi utilizado um fator de espalhamento de 7 (Quadro 4.31).

Quadro 4.31 - Configuração do spreanding factor

```
rf95.setSpreadingFactor(7);
```

Fonte: Autor

Figura 4.26 - Estrutura do pacote LoRa



Fonte: Autor

O preâmbulo é utilizado para sincronizar o transmissor e o receptor, e segundo a documentação do driver RF95 utilizado, todas mensagens enviadas e recebidas estão em conformidade de pacote com preâmbulo de 8 símbolos (Airspayce 2022).

Segundo a documentação do driver a mensagem também já possui um cabeçalho de modo explícito no qual possui quatro bytes com cabeçalho CRC já tratados internamente pelo rádio como padrão. Esse CRC já existente no cabeçalho traz a possibilidade de descartar cabeçalhos inválidos, o conceito de CRC será detalhado posteriormente. O cabeçalho no modo explícito sempre será transmitido com a maior taxa de codificação e possui um padrão de 4/5.

Os pacotes de dados são formados por bytes nos quais contém as informações a serem enviadas. Byte é um conjunto de 8 bits e como um byte representa uma combinação de 1s e 0s, tem-se a possibilidade de 2^8 conjuntos possíveis ou seja 256.

Um dos dados a serem enviados é o ID do dispositivo, no caso o robô que é o nó LoRa, esse ID é importante para saber identificar que nó está enviando dados. No código esse identificador é um vetor do tipo char, ou seja, tem tamanho de 1 byte. O mesmo possui três posições e foi declarado com o nome “node_id”, cada valor dentro do vetor é um byte a ser enviado, ou seja, o identificador possui três bytes onde cada byte tem valor 1 (Quadro 4.32).

Quadro 4.32 - Identificador do nó LoRa

```
char node_id[3] = {1,1,1};
```

Fonte: Autor

Além do identificador tem-se a informação do sensor de gás MQ. O sensor MQ possui um limite de medição de 10000 ppm, com isso, não dá para armazenar valores nessa faixa em um único byte, com isso, a informação do gás foi armazenada em 2 bytes, ou seja, com 16 bits

tem-se a possibilidade de 2^{16} conjuntos, ou seja, 65535, valor no qual satisfaz o armazenamento da faixa possível de leitura do sensor.

A variável que recebe o valor lido do sensor é “leitura” ela é do tipo uint_16t, ou seja, possui 16 bits (2 bytes), e para armazenar esses bytes foi criado um vetor do tipo char com 2 posições chamado de “sensLoc_dat”, a primeira posição do vetor recebe o valor de “leitura” deslocado de 8 bits pra direita por meio do operador “>>” e a segunda posição recebe o valor de leitura em byte (Quadro 4.33).

Quadro 4.33 - Armazenamento dos bytes da informação do gás

```
sensLoc_dat[0] = (uint8_t)((leitura>>8)&0xFF);
sensLoc_dat[1] = (uint8_t)leitura;
```

Fonte: Autor

Por exemplo, assumindo que o valor de leitura foi 355, representando esse valor em binário tem-se um valor de 0000 0001 0110 0011, deslocando 8 bits à direita tem-se

$$0000000101100011 \gg 8 = \begin{bmatrix} 0000000010110001 \\ 0000000001011000 \\ 0000000000101100 \\ 0000000000010110 \\ 0000000000001011 \\ 0000000000000101 \\ 0000000000000010 \\ 0000000000000001 \end{bmatrix} = 0000000000000001$$

Depois é feito a operação lógica “AND” com 0xFF, ou seja, com 11111111 para garantir que apenas os 8 bits menos significativos podem ser diferentes de zero. Vale ressaltar que na operação “AND” o resultado é 1 somente se os bits operandos forem 1, caso contrário é zero.

$$\& \frac{0000000000000001}{0000000011111111} = 00000000000000001$$

Com isso tem-se que a primeira posição de “sensLoc_data” é:

sensLoc_dat [0] = 00000001

A segunda posição recebe o valor de “leitura” em byte ou seja:

sensLoc_dat [1] = 01100011

Além da informação do ID {1,1,1} e da informação do sensor de gás, tem também a informação do deslocamento em X armazenada na variável “contadorX” e a informação de deslocamento em Y armazenada na variável “contadorY”. Vale ressaltar que essas variáveis são do tipo float, tipagem na qual tem tamanho de 4 bytes. Para armazenar essas variáveis no formato de bytes foram criadas duas funções denominadas “bytesLocX()” e “bytesLocY()”,

ambas funções recebem como parâmetro variáveis do tipo float, sendo “contadorX” e “contadorY” (Quadro 4.34).

Quadro 4.34 - Funções para obter os bytes de deslocamento

```
void bytesLocX(float locX){
  union{
     float valX;
     unsigned char bvalX[4];
  }floatAsBytes;
  floatAsBytes.valX = locX;
  BytesX[0] = floatAsBytes.bvalX[0];
  BytesX[1] = floatAsBytes.bvalX[1];
  BytesX[2] = floatAsBytes.bvalX[2];
  BytesX[3] = floatAsBytes.bvalX[3];
 }
void bytesLocY(float locY){
  union{
     float valY;
     unsigned char bvalY[4];
  }floatAsBytes;
  floatAsBytes.valY = locY;
  BytesY[0] = floatAsBytes.bvalY[0];
  BytesY[1] = floatAsBytes.bvalY[1];
  BytesY[2] = floatAsBytes.bvalY[2];
  BytesY[3] = floatAsBytes.bvalY[3];
 }
```

Fonte: Autor

Dentro destas funções criadas é utilizado uma função chamada “union”. Por meio desta função é possível agrupar dados de tipos diferentes, no caso, tipo float que possui tamanho de 4 bytes e do tipo char que possui tamanho de 1 byte. A union foi chamada de “floatAsBytes” e contém dois membros sendo a variável “valX” na função “bytesLocX()” e “valY” na função “bytesLocY()” para um valor float e a variável “bvalX” na função “bytesLocX()” e “bvalY” na função “bytesLocY()” para um valor char que são vetores de tamanho 4, cujos valores aramazenados são de 1 byte para cada posição do vetor. Foi declarado duas variáveis globais que são vetores do tipo char de tamanho 4 chamadas de “BytesX” e “BytesY” , elas foram utilizadas para armazenar os valores do deslocamento em X e em Y respectivamente no formato de bytes para serem enviados. Essas funções “bytesLocX()” e “bytesLocY()” foram chamadas dentro da função “ContadorAnguloPosicao()”, função na qual é utilizada para estimar o deslocamento do robô toda vez que uma interrupção é gerada pelo *encoder* (Quadro 4.35).

Quadro 4.35 - Função para obter dados de deslocamento do robô

```
void ContadorAnguloPosicao(){ // Função para valor de Angulo e Posição
  static unsigned long delayCont; // delay falso para retornar um estado sem "tremida"
  switch(sentido){
    case 1: //direita
     if(millis()-delayCont>1){
      contadorr= contadorr+1;
      angulo=contadorr*3.1415/111;
```

```
      if(angulo>=6.283){
        contadorr=0;
       }
      }
       delayCont=millis();
       break;
     case 2 : //esquerda
      if(millis()-delayCont>1){
       contadorr= contadorr-1;
       angulo=contadorr*3.1415/200;
       if(angulo>=6.283){
        contadorr=0;
       }
      }
      delayCont=millis();
     break;
     case 3: //re
       if(millis()-delayCont>1){
         contadorX= contadorX-1*cos(angulo);
         contadorY= contadorY-1*sin(angulo);
       }
       delayCont=millis();
     break;
     case 4: //frente
        if(millis()-delayCont>1){
          contadorX= contadorX+1*cos(angulo);
          contadorY= contadorY+1*sin(angulo);
        }
        delayCont=millis();
     break;
   }
   bytesLocX(contadorX);
   bytesLocY(contadorY);
}
```

Fonte: Autor

Por fim, a estrutura do pacote também possui um CRC (*Cyclic Redundancy Check)*. O CRC é um código de 8, 16 ou 32 bits que é anexado ao pacote a ser transmitido com a finalidade de verificação de erros, pois a camada física nem sempre é confiável por causa de erros de transmissão. Então cabe a camada de enlace inserir esses bits adicionais nos dados enviados. Esse código possui um valor baseado no resto da divisão polinomial do conteúdo do pacote da mensagem, por meio desse valor é possível detectar erros na cadeia de dados transmitidos.

Foi utilizado um CRC de 16 bits e a função utilizada para o cálculo do CRC assim como a lógica de envio de dados foi baseada em um exemplo disponível no site da Dragino no qual consiste no envio de dados de um sensor de umidade e temperatura via LoRa para o *gateway* LoRa conectado ao *ThingSpeak*. A função de cálculo do CRC é chamada de "CRC16", ela recebe como parâmetros, o *buffer* dos dados e o tamanho desse *buffer*, com esses dados a função calcula o CRC e retorna o valor de CRC (Quadro 4.36).

Quadro 4.36 - Funções para obter CRC

```
uint16_t calcByte(uint16_t crc, uint8_t b){
```

```
  uint32_t i;
  crc = crc ^ (uint32_t)b << 8;
    for ( i = 0; i < 8; i++){
      if ((crc & 0x8000) == 0x8000){
          crc = crc << 1 ^ 0x1021;
      }else{
          crc = crc << 1;
      }
    }
 return crc & 0xffff;
}

uint16_t CRC16(uint8_t *pBuffer,uint32_t length){
   uint16_t wCRC16=0;
   uint32_t i;
    if (( pBuffer==0 )||( length==0 )){
      return 0;
    }
    for ( i = 0; i < length; i++){
      wCRC16 = calcByte(wCRC16, pBuffer[i]);
    }
    return wCRC16;
}
```

Fonte: [Dragino 2022]

O *buffer* de dados é um vetor chamado “data” essa variável é do tipo char e tem como valor padrão 0 com tamanho de 50 (Quadro 4.37).

Quadro 4.37 - Vetor de dados para envio via LoRa

```
char data[50] = {0} ;
```

Fonte: Adaptado pelo Autor

Cada posição desse vetor corresponde a um byte das informações que são enviadas, as posições 0, 1 e 2 desse vetor correspondem aos bytes do identificador do dispositivo, as posições 3 e 4 correspondem aos bytes da informação do sensor de gás, as posições 5, 6, 7 e 8 correspondem aos bytes do deslocamento do robô em X e por fim as posições 9, 10, 11 e 12 correspondem aos bytes do deslocamento do robô em Y (Quadro 4.38).

Quadro 4.38 - Armazenamento dos dados a serem enviados

```
data[0] = node_id[0] ;
data[1] = node_id[1] ;
data[2] = node_id[2] ;
data[3] = sensLoc_dat[0];
data[4] = sensLoc_dat[1];
data[5] = BytesX[0];
data[6] = BytesX[1];
data[7] = BytesX[2];
data[8] = BytesX[3];
data[9] = BytesY[0];
data[10] = BytesY[1];
data[11] = BytesY[2];
data[12] = BytesY[3];
```

Fonte: Autor

O tamanho do *buffer* de dados é 13, esse valor foi armazenado em uma variável chamada “dataLenght” (Quadro 4.39).

Quadro 4.39 - Tamanho do vetor de dados

```
int dataLength = 13;
```

Fonte: Autor

Uma variável chamada “crcData” é utilizada para armazenar o valor dos bytes de CRC calculados na função “CRC16()” passando “data” e “dataLength” como parâmetros na mesma (Quadro 4.40).

Quadro 4.40 - Armazenamento de CRC

```
uint16_t crcData = CRC16((unsigned char*)data, dataLength);
```

Fonte: Adaptado pelo Autor

Detalhe que o primeiro parâmetro da função “CRC16()” espera um dado do tipo “uint_8” e como “data” é do tipo char é utilizado um recurso chamado *casting,* esse recurso é utilizado para converter tipos de dados, nesse caso o ponteiro interpreta o dado como char e não uint_8 por isso é passado (unsigned char*) juntamente com o parâmetro.

Para fins de teste era retornado constantemente os dados no formato hexadecimal a serem enviados sem CRC no monitor serial (Quadro 4.41).

Quadro 4.41 - Logica para mostrar os dados enviados com CRC no monitor serial

```
Serial.print("Dados a serem enviados(sem CRC): ");
   int i;
   for(i = 0;i < dataLength; i++){
      Serial.print(data[i],HEX);
      Serial.print(" ");
   }
   Serial.println();
```

Fonte: Adaptado pelo Autor

O vetor “sendBuf” é utilizado para armazenar os valores de “data” (Quadro 4.42).

Quadro 4.42 - Armazenamento de "data" no vetor de envio de dados via LoRa

```
  unsigned char sendBuf[50]={0};
   for(i = 0;i < dataLength;i++){
      sendBuf[i] = data[i] ;
   }
```

Fonte: Adaptado pelo Autor

Além dos dados também foi adicionado os bytes de CRC nesse vetor, a lógica para obter os bytes do CRC é semelhante a lógica usada para obter os bytes dos dados do sensor de gás, lógica na qual usa deslocamento de bits para obter os bytes (Quadro 4.43).

Quadro 4.43 - Lógica para obter os bytes de CRC

```
sendBuf[dataLength] = (unsigned char)crcData;
sendBuf[dataLength+1] = (unsigned char)(crcData>>8);
```

Fonte: Adaptado pelo Autor

Também era retornado constantemente os dados no formato hexadecimal a serem enviados com CRC no monitor serial (Quadro 4.44).

Quadro 4.44 - Logica para mostrar os dados enviados sem CRC no monitor serial

```
Serial.print("Dados a serem enviados com CRC    ");
for(i = 0;i < (dataLength +2); i++){
   Serial.print(sendBuf[i],HEX);
   Serial.print(" ");
}
Serial.println();
```

Fonte: Adaptado pelo Autor

Para enviar os dados via LoRa foi utilizado a função “send()” nativa do driver RH_RF95, essa função recebe dois argumentos, sendo o *buffer* de dados e o tamanho do *buffer*, no caso foi passado a variável “sendBuf” e “dataLength + 2”, o tamanho do pacote foi somado com dois, pois além dos dados tem mais dois bytes referentes ao CRC (Quadro 4.45).

Quadro 4.45 - Função de envio de dados via LoRa

```
rf95.send(sendBuf, dataLength+2);
```

Fonte: Adaptado pelo Autor

Para a parte de recepção de mensagens no nó LoRa, é iniciado o receptor por meio da função “waitAvailableTimeout()”. Essa função recebe como parâmetro o tempo máximo de espera de resposta de uma mensagem recebida, segundo a documentação do driver esse tempo tem como padrão 3 segundos. Se uma mensagem de resposta do gateway chegar nesse intervalo tempo, é utilizado o método “recv()” método no qual recebe dois parâmetros, o primeiro parâmetro é uma variável para copiar uma mensagem válida disponível de recepção, essa variável foi chamada de “buf”. Se uma mensagem for copiada o outro parâmetro dessa função é definido para o tamanho de “buf” essa variável foi chamada de “len”. Caso essa função retorne uma mensagem válida, primeiro é verificado se a mensagem de resposta tem o id do nó cujo valor é {1,1,1}, caso tenha, escreve-se nivel alto no pino “D4” fazendo com que acenda um led no *shield* LoRa. Isso foi feito só pra indicar que o módulo recebeu uma mensagem de resposta do *gateway*, além do led também é mostrado a resposta no monitor serial do Arduino. Caso a mensagem de recepção for invalida é enviado uma mensagem de falha no monitor serial e caso não tenha resposta do *gateway* nos 3 segundos de espera, os dados são reenviados novamente (Quadro 4.46).

Quadro 4.46 - Lógica de envio e recebimento de mensagens via LoRa

```
if (rf95.waitAvailableTimeout(3000)){
   //Deve ser uma mensagem de resposta para nós agora
     if (rf95.recv(buf, &len)){
        if(buf[0] == node_id[0] && buf[1] == node_id[1 ] && buf[2] == node_id[2] ){
           pinMode(4, OUTPUT);
           digitalWrite(4, HIGH);
           Serial.print("Resposta do Gateway: ");//imprime a resposta
           Serial.println((char*)buf);
           delay(400);
           digitalWrite(4, LOW);
        }
     }else{
       Serial.println("Falha na Recepção");
       rf95.send(sendBuf, strlen((char*)sendBuf));
     }
   }else{
     Serial.println("Sem resposta, o gateway LoRa está em execução ?");
     rf95.send(sendBuf, strlen((char*)sendBuf));
   }
  delay(3000);
  Serial.println("");
}
```

Fonte: [Dragino 2022]

4.2.8 Gateway LoRa e Recepção de Dados

O *gateway* utilizado para recepção de dados é o LG01, esse *gateway* assim como a *shield* LoRa é fabricado pela Dragino. Esse *gateway* permite conectar uma rede sem fio LoRa a uma base de rede IP em WiFi, Ethernet, celular 3G ou 4G (Dragino 2022).

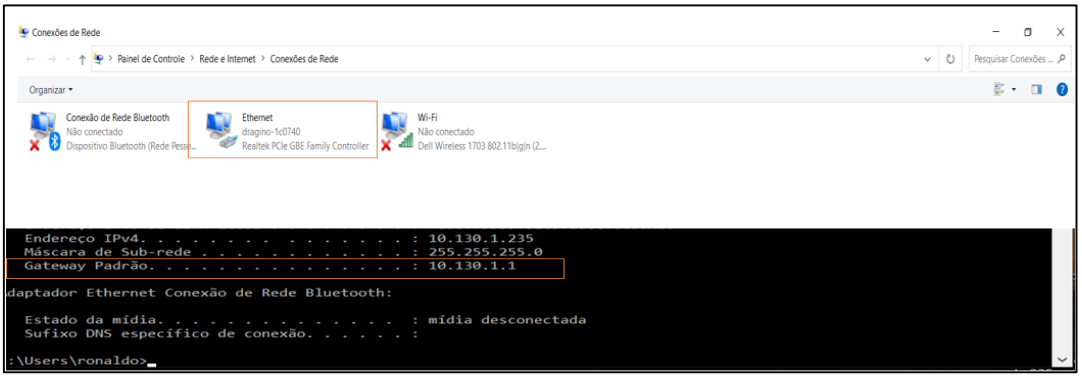
No caso deste projeto a rede LoRa foi conectada por meio da Ethernet, mas antes de conectar o *gateway* em si à Internet, foi necessário fazer algumas configurações iniciais, para isso foi conectado um cabo rj45 na porta LAN do *gateway* e no computador (Figura 4.27).

Figura 4.27 - Conexão do cabo rj45

Fonte: Autor

Após conectar o computador a rede local gerada pelo *gateway*, foi acessado às configurações do mesmo por meio do endereço IP padrão do LG01 cujo valor é 10.130.1.1 (Figura 4.28).

Figura 4.28 - IP padrão do *gateway*



Fonte: Autor

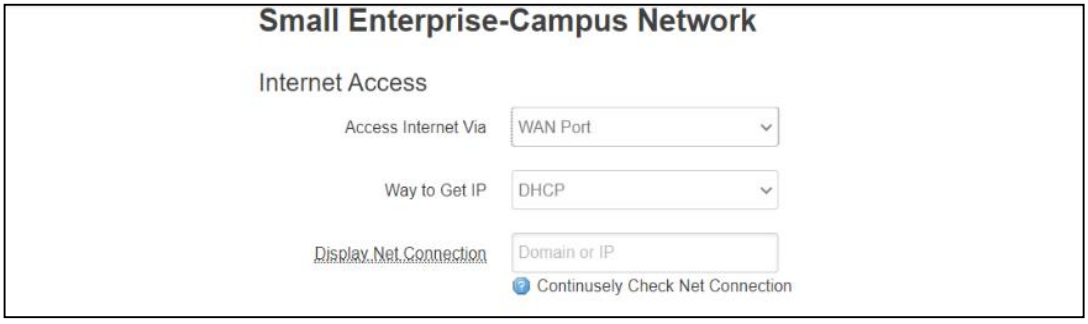
Para acessar a página de configurações do *gateway* foi preciso fornecer o usuário e senha disponibilizado no manual de uso do *gateway* (Figura 4.29).

Figura 4.29 - Pagina inicial de configuração do *gateway*

Fonte: Autor

Foi configurado para o *gateway* se conectar à Internet pela porta WAN e obter endereço IP via DHCP (Figura 4.30).

Figura 4.30 - Configuração de Internet no *gateway*



Fonte: Autor

A frequência de operação do LoRa foi definida como 915 MHz, o fator de espalhamento (SF) foi configurado com mesmo valor do módulo, ou seja, 7. Além disso o preâmbulo também foi configurado com 8 símbolos, que é o padrão do driver utilizado no módulo. A taxa de codificação foi mantido o padrão de 4/5 (Figura 4.31).

Com essas configurações salvas, outro cabo rj45 foi conectado na porta WAN do *gateway* e em uma porta LAN de um roteador com Internet, fazendo o *gateway* conectar-se à Internet.

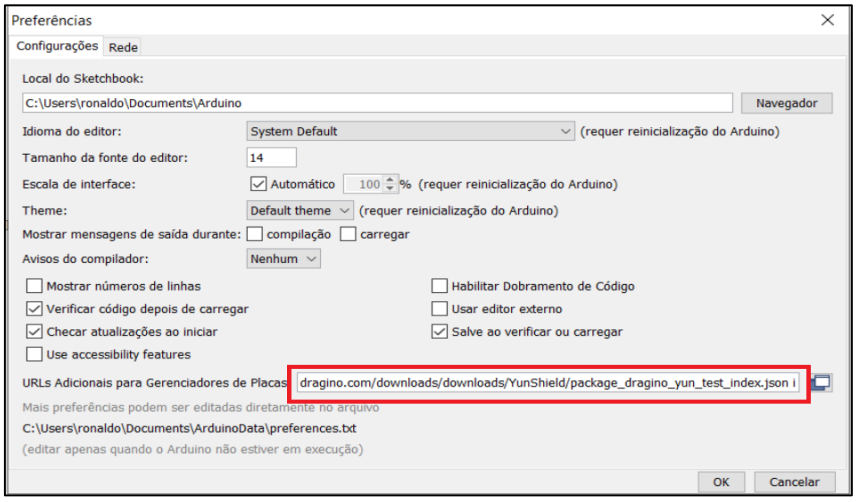
O *gateway* possui um MCU M38P interno que é usado para se comunicar com a parte LoRa e o módulo Dragino Linux. A linguagem para programar o MCU é baseada em C e a ferramenta utilizada para programar é o Arduino IDE.

Para programar o *gateway* foi necessário primeiramente adicionar uma URL de *download* da placa do *gateway* utilizado nas preferências do Arduino (Figura 4.32).

Figura 4.31 - Configurações do LoRa no *gateway*

Fonte: Autor

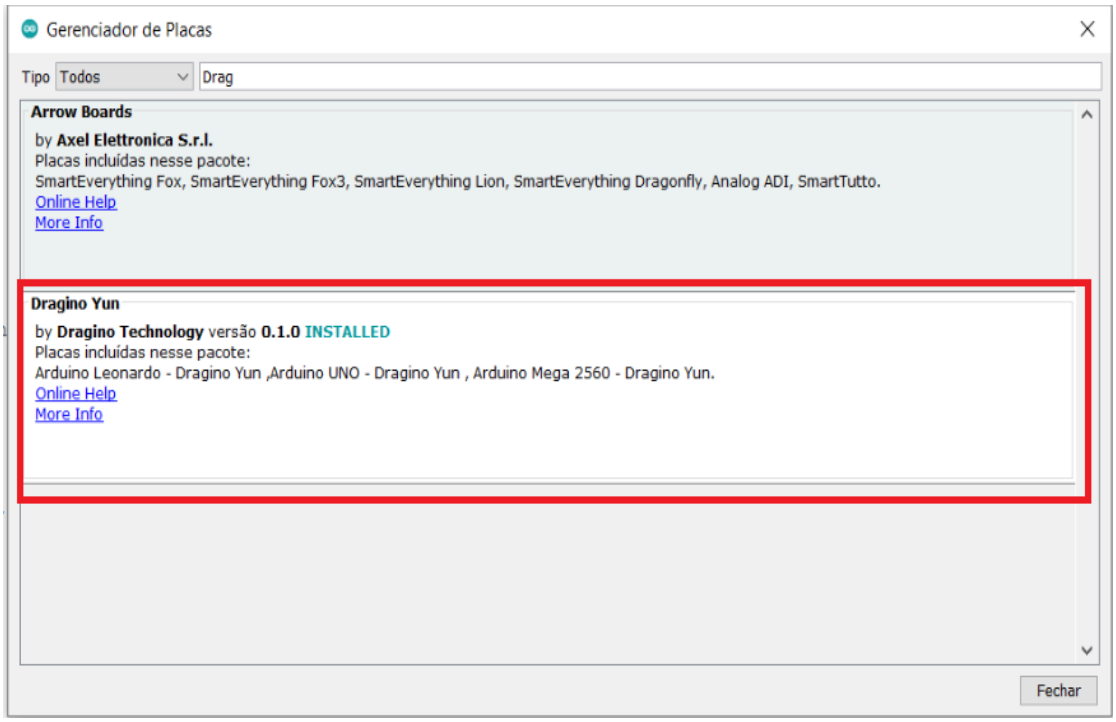
Figura 4.32 - Download da placa dragino do *gateway*



Fonte: Autor

Após fazer esse procedimento, as placas Dragino ficam disponíveis para *download* no gerenciador de placas (Figura 4.33).

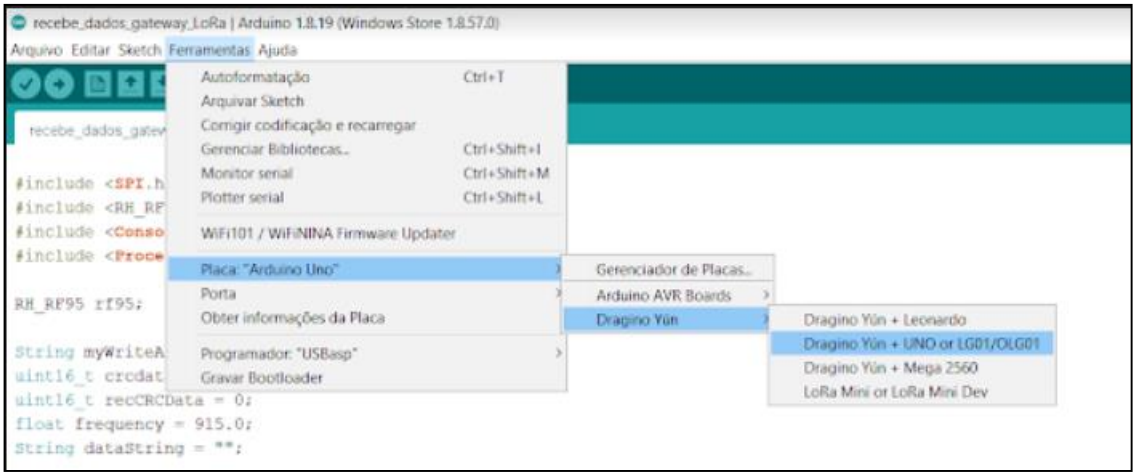
Figura 4.33 - Instalação da placa na IDE Arduino



Fonte: Autor

Após o *download*, as informações das placas ficam disponíveis na IDE, Para o LG01, deve-se escolher a placa “Dragino Yun-UNO or LG01/OLG01”(Figura 4.34).

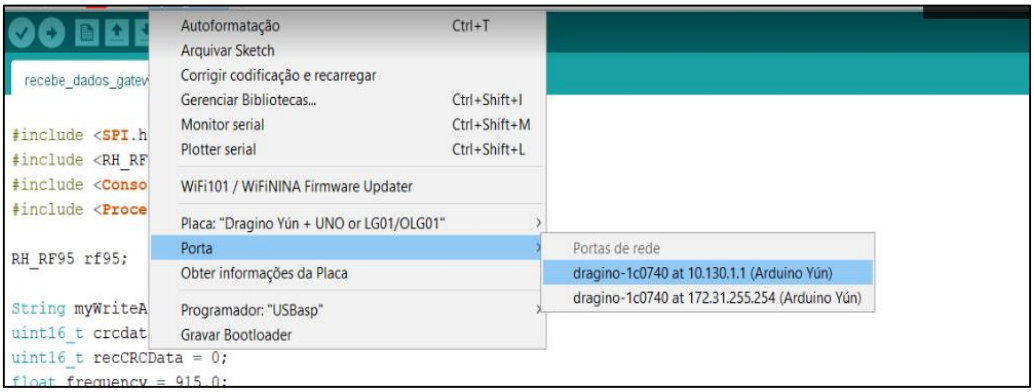
Figura 4.34 - Seleção da placa LG01 para programar



Fonte: Autor

Para carregar um esboço de código no *gateway* é necessário que o computador esteja na mesma rede do LG01, com isso é só selecionar a porta correta como mostra a Figura 4.35.

Figura 4.35 - Carregamento de um código no MCU do *gateway*



Fonte: Autor

O driver RH_RF95 também foi importado no código de programção do MCU do *gateway*, e o objeto dessa biblioteca foi criado de maneira análoga ao código do Arduino UNO. A inicialização do driver assim como as configrações de frequência de operação LoRa e de potência do transmissor, também foram feitas de maneira análoga à programação no Arduino UNO (Quadro 4.47).

Quadro 4.47 - Importação do driver RH_RF95 e configurações iniciais do LoRa no *gateway*

```
#include <RH_RF95.h>
RH_RF95 rf95;
float frequency = 915.0;
void setup() {
  if (!rf95.init()){
   Console.println("Falha na inicialização");
 }
   rf95.setFrequency(frequency);
   rf95.setTxPower(13);
}
```

Fonte: Adaptado pelo Autor

Além da inicialização do driver também foi habilitado a comunicação entre o processador Linux e o MCU, para isso foi chamado a função “Bridge.begin()” dentro da função “setup()” do Arduino, a função “Bridge.begin()” recebe como parâmetro a *Baud Rate* que possui um valor padrão de 115200 (Quadro 4.48).

Quadro 4.48 - Habilitação da comunicação entre Linux e MCU

```
Bridge.begin(115200);
```

Fonte: Autor

Para ler os dados dessa comunicação do processador e MCU e mostrar no monitor serial do Arduino, foi utilizado uma biblioteca chamada “Console” e a mesma foi inicializada na função “setup()” também (Quadro 4.49).

Quadro 4.49 - Habilitação da biblioteca serial para mostrar dados de comunicação entre Linux e MCU

```
#include <Console.h>

Console.begin();
```

Fonte: Autor

Vale ressaltar que as configurações assim como o código de recepção das mensagem LoRa foram feitos baseado em um exemplo de envio de dados de um sensor de umidade e de temperatura via LoRa á um *gateway* LoRa LG01 conectado ao servidor *ThingSpeak* disponibilizado pela Dragino em [105].

Foi utilizado a mesma lógica de programção de recepção de mensagens no nó LoRa. O receptor do *gateway* é inicado por meio da função “waitAvailableTimeout()”. Essa função recebe como parâmetro o tempo máximo de espera de resposta que também é de três segundos. Para verificar se há dados recebidos do nó LoRa é utilizado o método “recv()”. Dentro dessa função foi passado para o primeiro parâmetro uma variável chamada “buf” que é um vetor cujo tamanho é o tamanho da mensagem recebida, essa variável é utilizada para copiar a mensagem recebida. Foi passado como segundo parâmetro uma variável chamada “len” para receber o tamanho do pacote de dados recebido.

Enquanto houver mensagens recebidas do nó LoRa é enviado constantemente uma mensagem no monitor Serial contendo o pacote de dados recebido (Quadro 4.50).

Quadro 4.50 - Lógica de recebimento e envio de dados via LoRa no *gateway*

```
if (rf95.waitAvailableTimeout(3000)){
      uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
      uint8_t len = sizeof(buf);
       if (rf95.recv(buf, &len)){
          recdata( buf, len);
          Console.print("Obtenha o pacote LoRa: ");
          for (int i = 0; i < len; i++){
             Console.print(buf[i],HEX);
             Console.print(" ");
          }
          Console.println();
```

Fonte: Adaptado pelo Autor

Para calcular o CRC recebido foi criado uma função chamada “recdata()”. Foi usada a mesma função de cálculo do CRC utilizada no transmissor. A função “recdata()” recebe como parâmetros o vetor de dados recebidos (“recbuf”) e o tamanho desse vetor (“Length”). Uma variável chamada “crcdata” foi utilizada para armazenar o retorno da função de cálculo do CRC.

Vale ressaltar que nesse cálculo de CRC foi desprezado os bytes de CRC transmitidos pelo nó LoRa. (Quadro 4.51).

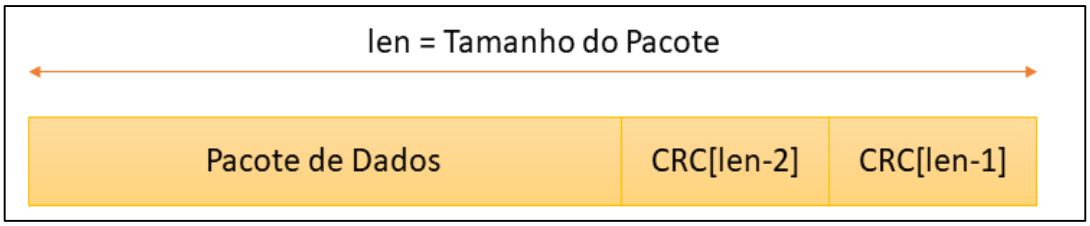
Quadro 4.51 -Armazenamento dos dados de CRC recebidos do nó LoRa

```
crcdata = CRC16(recbuf, Length - 2);
```

Fonte: Adaptado pelo Autor

Ainda dentro da função “recdata()” foi utilizada uma variável chamada “recCRCData” para atribuir o valor do CRC que o nó LoRa enviou. A Figura 4.36 ilustra a estrutura do pacote de dados recebido.

Figura 4.36 - Estrutura do pacote de dados recebido



Fonte: Autor

Para exemplificar a lógica de obter o valor de 16 bits do CRC que foi enviado pelo nó LoRa, assumimos que “CRC[len-1]” possui valor 10110101 e “CRC[len-2] possui valor 01101101, primeiro atribui-se “recCRCData” igual a CRC[len-1]. Como “recCRCData” é do tipo uint16_t ele possui 16 bits, então adiciona-se 8 bits com valor 0 ao valor de CRC[len-1] .

recCRCData = CRC[len-1] = 0000000010110101

Depois atribui-se a “recCRCData” o próprio valor de “recCRCData” deslocado de de 8 bits à esquerda

recCRCData = CRC[len-1] = 1011010100000000

Depois disso é usado o operador OR bit a bit (“ | ”) do valor de “recCRCData” com o valor de “CRC[len-2]” , esse operador compara cada bit do primeiro operando com o bit de seu segundo operando. Cada bit resultante é zero somente quando os dois bits operandos são zero, em qualquer outra operação o resultado é um (Quadro 4.52).

Quadro 4.52 - Logica para obter os 16 bits de CRC recebidos

```
recCRCData |= recbuf[Length - 2];
```

Fonte : Adaptado pelo Autor

$$recCRCData = | \frac{recCRCData}{CRC\,[l\,en - 2]} = | \frac{1011010100000000}{0000000001101101} = 1011010101101101$$

É comparado se os valores de CRC calculado (“crcdata”) e recebido (“recCRCData”) são iguais, se for, é verificado se os valores de “buf[0]”, “buf[1] e “buf[2] correspondem ao valor do ID do nó LoRa cujo valor é {1,1,1}, se essa condição for verdadeira é enviado uma mensagem ao nó LoRA de confirmação de recebimento de dados por meio da função nativa do driver chamada “send()”, a mensagem enviada contém o ID do nó e a mensagem “Servidor ACK”. Além disso também é armazenada as informações do sensor de gás e do deslocamento enviadas pelo nó LoRa em um vetor chamado de “newData” (Quadro 4.53).

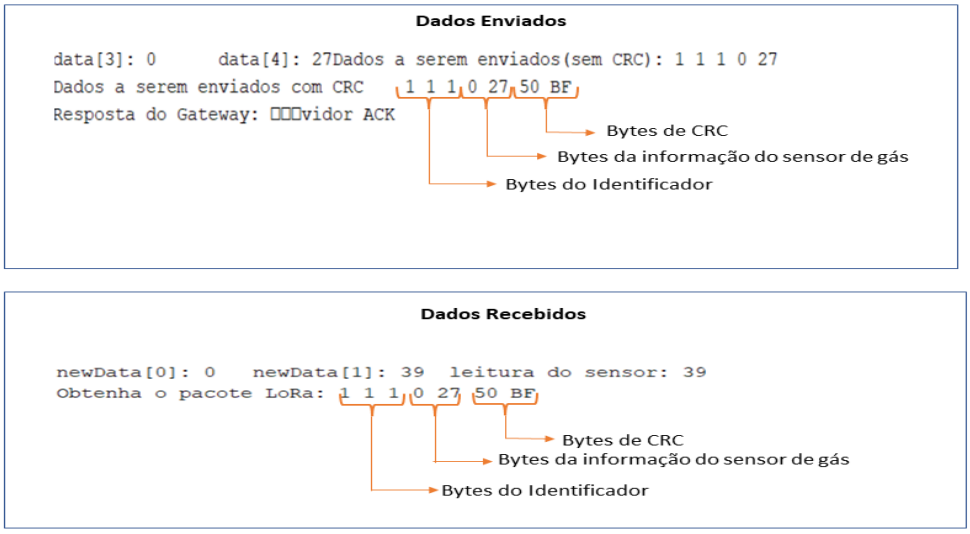
Quadro 4.53 - Lógica de verificação de mensagens válidas recebidas

```
if(crcdata == recCRCData){
        if(buf[0] == 1 && buf[1] == 1 && buf[2] ==1){
            uint8_t data[] = "Servidor ACK";
            data[0] = buf[0];
            data[1] = buf[1];
            data[2] = buf[2];
            rf95.send(data, sizeof(data));
            rf95.waitPacketSent();
           int newData[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0};
            for (int i = 0; i < 10; i++){
              newData[i] = buf[i + 3];
            }
}
}
```

Fonte: Adaptado pelo Autor

Foram feito testes iniciais de envio de dados do sensor de gás do nó LoRa para o *gateway*, a Figura 4.37 mostra no monitor serial do Arduino, as mensagem enviadas do nó LoRa e mostra outro monitor serial com os dados recebidos pelo *gateway*, vale ressaltar que os valores mostrados estão no formato hexadecimal.

Figura 4.37 - Mensagens enviadas pelo nó LoRa e recebidas no *gateway*



Fonte: Autor

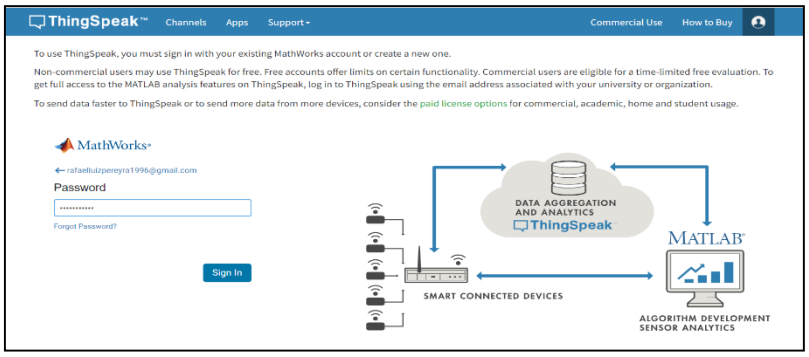
4.2.9 Configuração do *ThingSpeak* e Matlab

O *ThingSpeak* é um *software* de código aberto que permite aos usuários se comunicarem com dispositivos habilitados pela Internet, neste *software* é possível acessar e recuperar dados forncendo uma API para os dispositivos.

O primeiro passo para utilizar o *ThingSpeak* é criar uma conta na Mathworks que é uma empresa de *software* de computação matemática e possui como seus principais *softwares* o MATLAB e Simulink. Com a conta criada é só acessar o *ThingSpeak* com o email e senha (Figura 4.38).

O elemento básico do *ThingSpeak* é o canal. Um canal é onde são armazenados os dados que são enviados pelos dispostivos remotos. O canal precisa ter um nome e mesmo pode ter até 8 campos, campo é um elemento do canal utilizado para armazenar dados de qualquer tipo, além disso o canal pode ser público ou privado, um canal público fica vísivel para toda comunidade e ser for privado fica visível só ao autor. O nome do canal criado foi “gas_detector_robot” e foi criado três campos, sendo eles, “Concentração de gás” (Field1), “posição x”(Field5) e “posição y”(Field6) (Figura 4.39).

Figura 4.38- Página de login no *ThingSpeak*



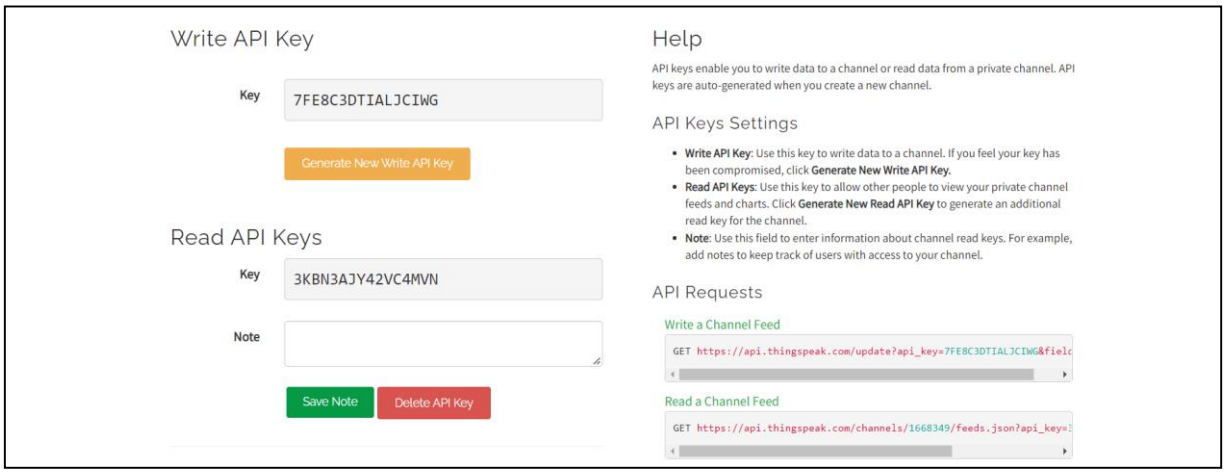
Fonte: Autor

Figura 4.39 - Configuração do canal no *ThingSpeak*

Fonte: Autor

Quando o canal é criado são geradas chaves de API que são utilizadas para ler e gravar em um canal no *ThingSpeak* utilizando requisiões HTTP. As chaves possuem um código de 16 bits, a chave de escrita do canal criada possui valor “7FE8C3DTIALJCIWG” e a chave de leitura possui valor “3KBN3AJY42VC4MVN” (Figura 4.40).

Figura 4.40 - Chaves de escrita e leitura no *ThingSpeak*



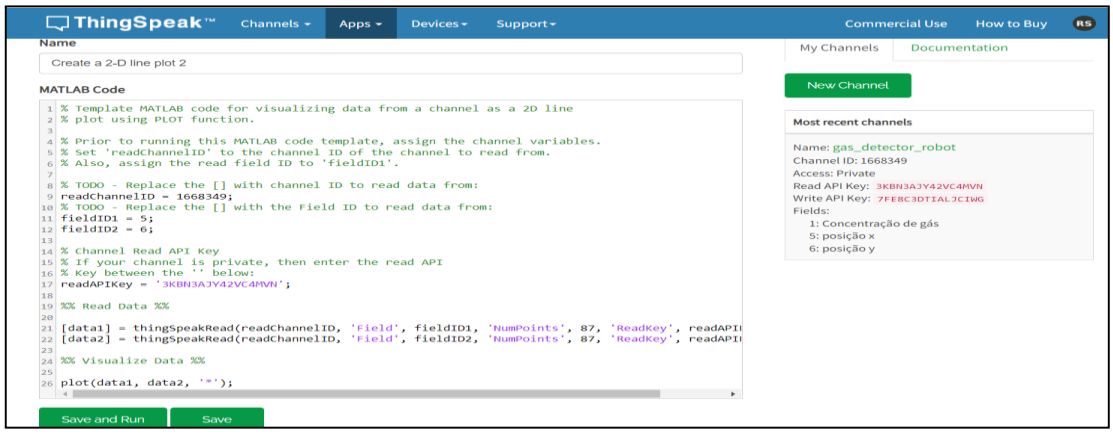
Fonte: Autor

O *ThingSpeak* possui a funcionalidade de executar código MATLAB, trazendo a possibilidade de realizar análises on-line e processar dados à medida que eles chegam.

Foi utilizado o aplicativo MATLAB Visualizations para visualizar os dados de deslocamento do robô no canal do *ThingSpeak*. Por meio desse aplicativo é possível visualizar e explorar dados usando visualizações interativas, como gráfico de área, gráfico de linha ou gráfico de dispersão do MATLAB (Mathworks 2022).

Nesse projeto foi criada uma visualização Matlab com um gráfico de linha. Existe um espaço onde pode escrever código MATLAB nessa visualização como mostra a Figura 4.41.

Figura 4.41 - Código MATLAB no *ThingSpeak*



Fonte: Autor

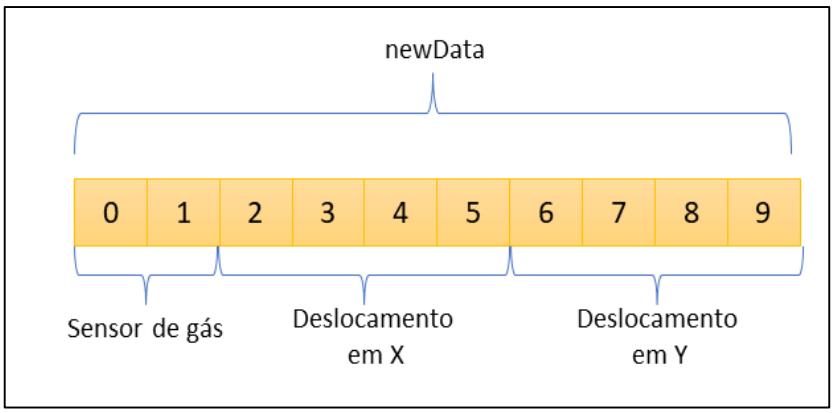
Para programar a visualização precisa das informações do ID do canal, chave de leitura e quais campos desejados para montar a visualização. Essas informações já ficam disponíveis como mostra a Figura 4.41.

Tem uma váriável chamada de “readChannelID” onde a mesma recebe o valor do ID do canal, os ID’s dos campos nos quais se deseja fazer a visualização, que no caso são os campos “posição x” cujo ID é 5 e “posição y” cujo ID é 6. Esses ID’s foram armazenados nas variáveis “fieldID1” e “fieldID2” respectivamente. A chave de leitura do canal foi armazenada em uma variável chamada “readAPIKey”, e para criar o grafico de linhas do deslocamento em X com relação ao deslocamento em Y é utilizada a função “*ThingSpeakRead*”, essa função recebe como parâmetros o ID do canal, o campo a ser lido, o numero de pontos do gráfico e a chave de leitura. Foi utilizada duas variáveis “data1” e “data2” para armazenar os valores lidos dos dois campos do canal *ThingSpeak* utilizados para armazenar os dados de deslocamento para que por fim, fosse utilizado a função “plot” para plotar o gráfico de deslocamento do robô. A aparência da linha plotada possui formato de asterisco “ * ”.

4.2.10 Envio de Dados para o *ThingSpeak*

A variável “newData” armazena os valores recebidos do nó LoRa, a posição 0 e 1 de “newData” que corresponde aos dados do sensor de gás são armazenadas em variáveis chamadas “b1” e “b2”. Um vetor chamado de “BytesX” armazena os valores das posições 2, 3, 4 e 5 de “newData”, valores nos quais correspondem ao deslocamento em X e outro vetor chamado “BytesY” armazena os valores das posições de 6, 7, 8 e 9 de “newData” valores nos quais correspondem o deslocamento em Y (Figura 4.42 e Quadro 4.54).

Figura 4.42 - Ilustração do vetor de dados recebidos no *gateway*



Fonte: Autor

Quadro 4.54 - Lógica de armazenamento dos dados recebidos do nó LoRa

```
uint8_t b1 = newData[0];
uint8_t b2 = newData[1];
BytesX[0] = newData[2];
BytesX[1] = newData[3];
BytesX[2] = newData[4];
BytesX[3] = newData[5];
BytesY[0] = newData[6];
BytesY[1] = newData[7];
BytesY[2] = newData[8];
BytesY[3] = newData[9];
```

Fonte: Autor

Uma variável chamada “leiSen” armazena o valor de 16 bits do sensor de gás, essa variável recebe o valor de “b1” deslocado de 8 bits à esquerda fazendo a operação lógicar “OR” com “b2” (Quadro 4.55). Para exemplificar assume-se que *o gateway* receba os dois bytes de um valor de concentação de gás cujo valor é 355. A variável “newData” na posição 0 recebe o valor igual a 1 e “newData” na posição 1 recebe o valor 01100011.

$$b1 = newData[0] = 00000001 = 1$$

$$b2 = newData[1] = 01100011$$

Deslocando 8 bits de b1 à esquerda tem-se que:

$$0000000000000001 \ll 8 = \begin{bmatrix} 0000000000000010 \\ 0000000000000100 \\ 0000000000001000 \\ 0000000000010000 \\ 0000000000100000 \\ 0000000001000000 \\ 0000000010000000 \\ 0000000100000000 \end{bmatrix} = 0000000100000000$$

b1 = 100000000 e fazendo a operação “OR” com b2 tem-se que:

$$lei\ Sen = |\frac{b1}{b2} = |\frac{100000000}{001100011} = 101100011 = 355$$

Quadro 4.55 - Lógica para obter os 16 bits da informação do sensor de gás

```
uint16_t leiSen = b1 << 8 | b2;
```

Fonte: Autor

Duas funções chamadas de “floatLocX()” e “floatLocY()” retornam os valores de 32 bits do deslocamento em X e em Y, essas funções recebem como parâmetro o vetor de bytes de deslocamento em X ou Y. A lógica dessas funções é semelhante as funções “bytesLocX()” e “bytesLocX()” utilizadas no envio das informações de deslocamento para o *gateway*, só que ao

invés de receber um float e retornar um vetor de bytes, as funções “floatLocX()” e “floatLocY()” recebem o vetor de bytes e retorna o valor do tipo float, esses vetores são “BytesX” e “BytesY” ,vetores nos quais armazenam os valores de deslocamento recebidos do nó LoRa. Os valores retornados destas funções são armazenados nas variáveis “locX” e “locY” (Quadro 4.56).

Quadro 4.56 - Funções para obter os valores de 32 bits do deslocamento em X e em Y

```
float floatLocX(char teste[4]){
  union{
    float valX;
    unsigned char bvalX[4];
 }floatAsBytes;
 floatAsBytes.bvalX[0] =teste[0];
 floatAsBytes.bvalX[1] = teste[1];
 floatAsBytes.bvalX[2] = teste[2];
 floatAsBytes.bvalX[3] = teste[3];

 return floatAsBytes.valX;
 }

float floatLocY(char teste[4]){
  union{
    float valY;
    unsigned char bvalY[4];
 }floatAsBytes;
 floatAsBytes.bvalY[0] =teste[0];
 floatAsBytes.bvalY[1] = teste[1];
 floatAsBytes.bvalY[2] = teste[2];
 floatAsBytes.bvalY[3] = teste[3];

 return floatAsBytes.valY;
 }
locX = floatLocX(BytesX);
locY = floatLocY(BytesY);
```

Fonte: Autor

Para enviar os dados armazenados nas variáveis, LeiSen, locX e locY para o *ThingSpeak*, primeiro foi declarado uma variável chamada “myWriteAPIString”, variável na qual recebe o valor da chave de escrita do canal criado no *ThingSpeak* sendo ela “7FE8C3DTIALJCIWG” (Quadro 4.57).

Quadro 4.57 - Armazenamento da chave de escrita do *ThingSpeak*

```
String myWriteAPIString = "7FE8C3DTIALJCIWG";
```

Fonte: Autor

Também foi criado uma variável chamada “dataString” essa variável recebe os nomes dos campos criados no *ThingSpeak* com suas respectivas variáveis sendo elas, leiSen, locX e locY como mostra a Quadro 4.58. Essa variável é passada para a URL a ser executada e transferir os dados por meio do protocolo HTTP para o *ThingSpeak*.

Quadro 4.58 - Logica de concatenação dos campos para construir a URL

```
dataString ="field1=";
dataString += leiSen;
dataString +="&field5=";
dataString += locX;
dataString +="&field6=";
dataString += locY;
```

Fonte: Autor

Para transferir os dados primeiro foi necessário importar uma biblioteca chamada “Process.h”, essa biblioteca é utilizada para executar comandos do Linux.

Uma função chamada “uploadData” é responsável por transferir os dados para o *ThingSpeak*. Dentro dessa função possui uma variável chamada “upload_url” que recebe parte da URL já com o cabeçalho http para transferir os dados (Quadro 4.59). Vale ressaltar que a letra “s” junto ao http indica que o site em questão está protegido pelo Certificado Digital SSL, ou seja, durante a troca de informações entre o servidor do *ThingSpeak* e o computador, os dados estão protegidos contra interceptação (Certisign 2022).

Quadro 4.59 - String URL com cabeçalho https

```
String upload_url = "https://api.ThingSpeak.com/update?api_key=";
```

Fonte: Autor

A “api_key” dentro da URL espera o valor da chave de escrita do canal *ThingSpeak*, então concatena-se o valor de “upload_url” com o valor da variável “myWriteAPIString” (Quadro 4.60).

Quadro 4.60 - Concatenação da URL com a chave de escrita

```
upload_url += myWriteAPIString;
```

Fonte: Autor

Concatena-se novamente essa variável com “&” para depois concatenar com a variável “dataString”(Quadro 4.61).

Quadro 4.61 - Concatenação da URL com dataString

```
upload_url += "&";
upload_url += dataString;
```

Fonte: Autor

Com isso a URL a ser executada está pronta. Depois foi criado um objeto da biblioteca Process denominado de “p”, o processo é iniciado com o comando “p.begin(“curl”)”. Curl é um comando disponível na grande maioria dos sistemas Linux utilizada para transferência de dados, esse comando suporta o protocolo HTTP no qual foi utilizado.

Após criar o processo adiciona-se a URL a ser executada (upload_url) no processo, para isso, é usado uma função chamada “addParameter”(Quadro 4.62).

Quadro 4.62 - Adicionamento da URL como parâmetro para ser executada

```
p.addParameter(upload_url);
```

Fonte: Autor

E para executar o processo em questão é utilzado a função “run”(Quadro 4.63).

Quadro 4.63 - Execução do processo com a URL

```
p.run();
```

Fonte: Autor

Era emitido mensagens constantemente no monitor serial do Arduino enquanto tinha saída de dados do Linux (Quadro 4.64).

Quadro 4.64 - Lógica para mostrar mensagens no monitor serial enquanto houver saída do Linux

```
while (p.available()>0){
  char c = p.read();
  Console.write(c);
}
```

Fonte: Adaptada pelo Autor

5 RESULTADOS

Os testes finais de locomoção do robô e medição de emissão de gás foram realizados em uma granja comercial na Empresa Agrícola Folhados S.A na qual possui escritório principal em Patrocínio, a mesma opera no setor de suinocultura. Os testes foram autorizados pelo gerente de produção da empresa Luiz Carlos Crestani.

Esse local foi escolhido devido ao fato da produção de suínos incidir em emissões de gases como o metano, principalmente nas fases de crescimento e terminação, fazendo com que esse ambiente se torne um local ideal para teste de medição de emissão de gás. O gás metano foi medido nas regiões da granja onde possui barracões de suínos em fases de terminação.

5.1 LOCOMOÇÃO E MEDIÇÃO DE GÁS EM BARRACÕES DE SUÍNOS DE TERMINAÇÃO

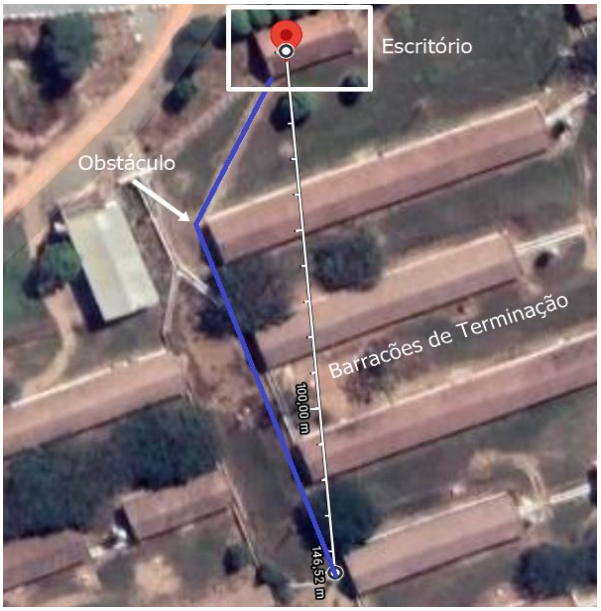
Para iniciar o teste, foi colocado o *gateway* LoRa no escritório da granja. O gerente de produção Luiz Carlos Crestani autorizou a utilização do roteador do escritório para conectar o *gateway* à Internet, a Figura 5.1 mostra o *gateway* colocado no escritório.

Figura 5.1 - *Gateway* LoRa no escritório da granja

Fonte: Autor

O teste foi realizado com o robô se locomovendo e medindo emissão de gás metano nos barracões de suínos de terminação, barracões nos quais ficam localizados em um terreno plano com relação ao escritório onde estava o *gateway* LoRa. A linha em azul na Figura 5.2 representa o percurso esperado que o robô fizesse até o quarto barracão, que dá o total de 146,52 metros de distância do escritório. Foi utilizado uma bateria LiPo de 7,4 V e 6500 mAh como fonte de alimentação para o robô.

Figura 5.2 - Percurso esperado do robô



Fonte: Autor

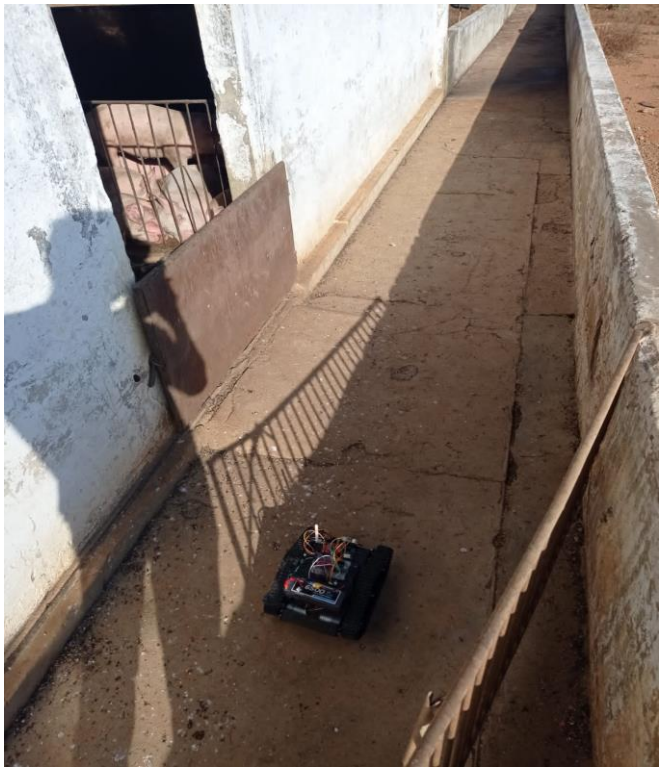
A Figura 5.3 mostra um obstáculo que é um muro no qual foi detectado pelo sensor ultrassônico fixado na lateral direita da estrutura do robô, com isso o robô conseguiu desviar do obstáculo girando à esquerda.

Figura 5.3 - Robô desviando do obstáculo

Fonte: Autor

A Figura 5.4 mostra o quarto barracão onde foi o local final de locomoção do robô no teste.

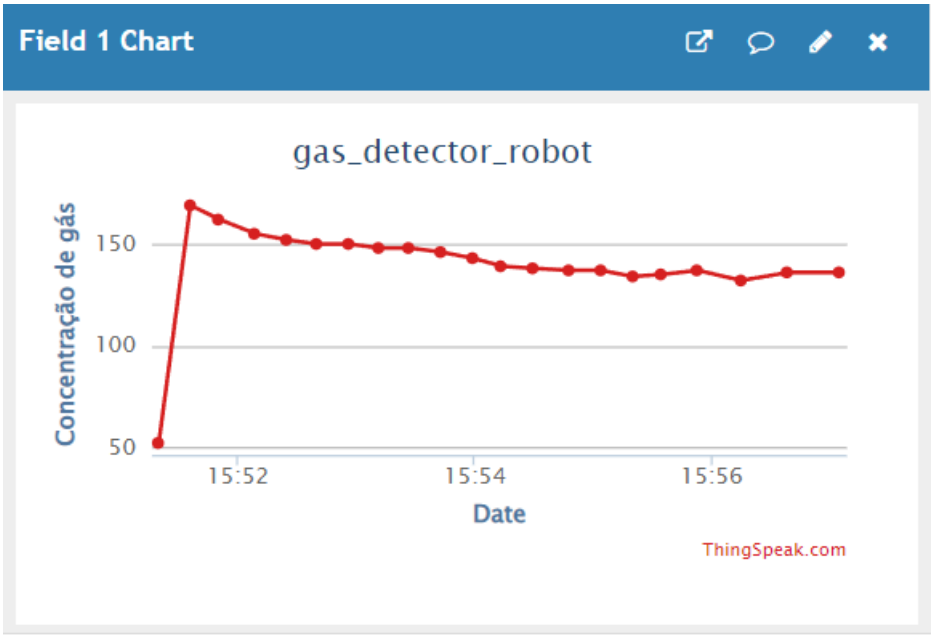
Figura 5.4 - Local final de locomoção do robô



Fonte: Autor

Os resultados da medição de gás metano com relação ao tempo no *ThingSpeak* durante este percurso pode ser visto na Figura 5.5.

Figura 5.5 - Gráfico de concentração de gás com relação ao tempo

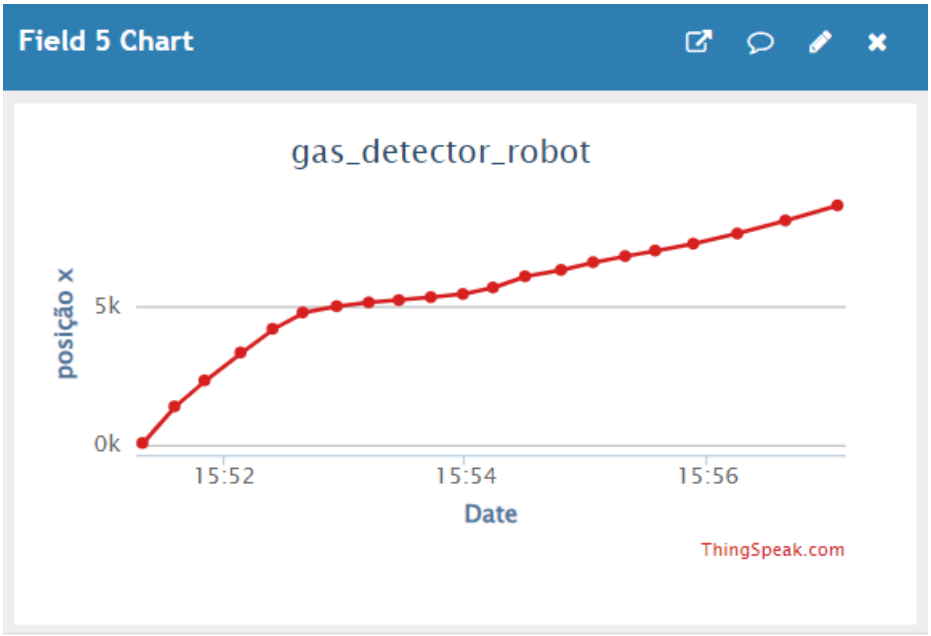


Fonte: Autor

Quando o robô começou o percurso, a leitura analógica do sensor MQ-4 saltou do valor 50 para um valor um pouco acima de 150 e depois foi caindo até permanecer em um valor entre 100 e 150, e assim permaneceu durante todo o percurso do robô.

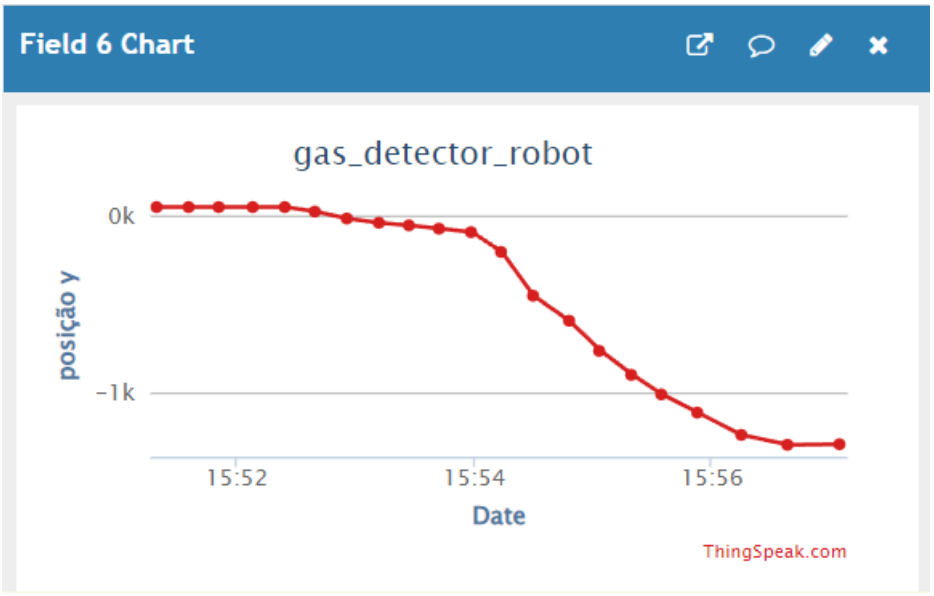
A Figura 5.6 mostra o deslocamento do robô em X com relação ao tempo enquanto a Figura 5.7 mostra o deslocamento do robô em Y com relação ao tempo.

Figura 5.6 - Gráfico de deslocamento em X com relação ao tempo



Fonte: Autor

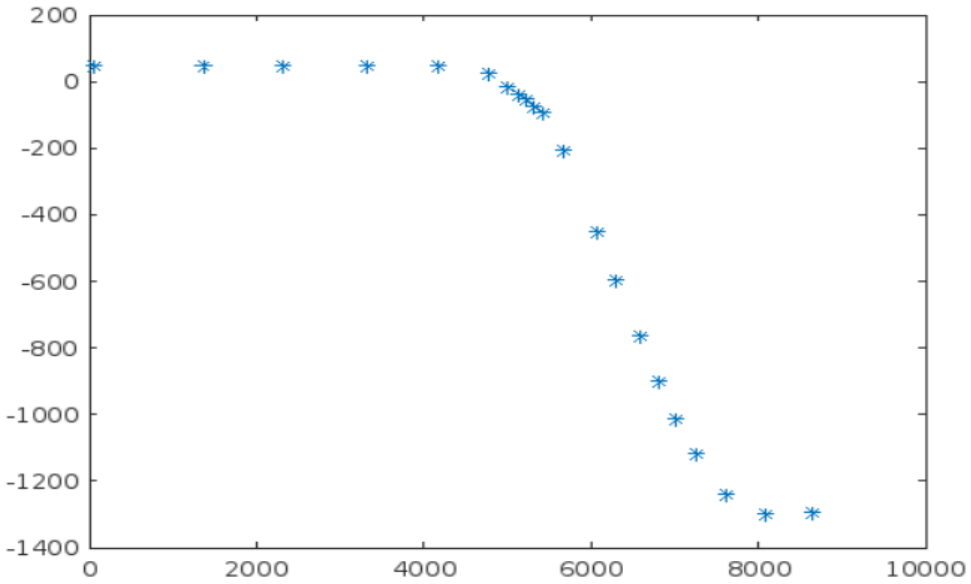
Figura 5.7 - Gráfico de deslocamento em Y com relação ao tempo



Fonte: Autor

Utilizando a visualização MATLAB no *ThingSpeak* pode-se ver a trajetória que o robô fez durante seu percurso plotando o gráfico de X com relação a Y, como mostra a Figura 5.8.

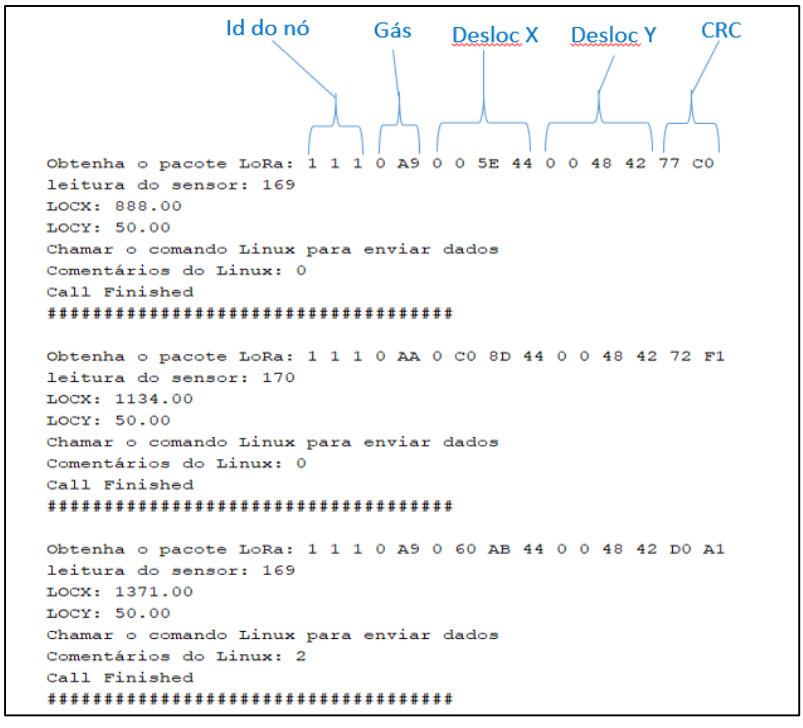
Figura 5.8 - Trajetória do robô



Fonte: Autor

Durante todo o percurso não ocorreu erros de CRC e nem falha de resposta do transmissor, isso pôde ser monitorado por meio do serial monitor do Arduino IDE. A Figura 5.9 mostra parte do *debug* no serial monitor, o *debug* mostra o pacote de bytes recebidos em formato hexadecimal, e os valores de leitura do sensor de gás (leitura do sensor), deslocamento em X (LOCX) e deslocamento em Y (LOCY) em formato decimal.

Figura 5.9 - Debug de pacotes LoRa recebidos



Fonte: Autor

A posição e sentido inicial do robô são pré-determinados de acordo com a inicialização do robô. O robô começou a se locomover de uma posição inicial X=50 e Y=50 com angulação inicial de 0° em linha reta, depois ele fez a curva a esquerda para desviar do obstáculo fazendo com que a angulação variasse de forma decrescente, por fim, o robô se manteve em linha reta até o final do percurso, como o esperado (Figura 5.8).

Pode-se observar nos gráficos no *ThingSpeak* que no intervalo de 8 minutos foram coletadas 21 informações relacionadas a deslocamento e emissão de gás, com isso tem-se que cada informação demorou em média 22,8 segundos para fazer *upload* no servidor do *ThingSpeak*.

5.2 LIMITAÇÕES ENCONTRADAS

- **Diferença dos motores DC:** Mesmo fazendo a calibração dos motores por causa da diferença de potência dos mesmos, durante a movimentação reta depois de um tempo prolongado o robô tendeu um pouco para uma direção. Este problema poderia ser evitado se os motores fossem de melhor qualidade.
- **Contagem de interrupções:** Como já mencionado, o chassi utilizado não possui adaptabilidade para utilização do *encoder,* o mesmo foi fixado no chassi com cola quente*,* com isso, houve momentos que enquanto a roda de condução girava às vezes tinha ausência de contagem dos pulsos. Este problema seria resolvido com um chassi que possua adaptabilidade para o *encoder.* Além disso, foi necessário criar um "falso delay" para limitar a quantidade de pulsos captados devido a contagem de interrupções variar em um único pulso.
- **Sensor de gás:** O sensor MQ não foi desenvolvido para aplicações profissionais, ele é destinado apenas a propósitos experimentais. Para uma aplicação profissional o sensor MQ deveria ser substituído por um sensor de gás profissional.
- **Distância máxima do gateway ao transmissor:** Os testes foram autorizados pelo gerente nos barracões de terminação cuja distância máxima do escritório onde ficou o *gateway* é de 148 metros, distância que é bem menor que a capacidade máxima de comunicação à longas distâncias que a tecnologia LoRa possui.
- **Informação em tempo real:** Como já mencionado**,** a média de tempo de chegada de informação no *ThingSpeak* foi de 22,8 segundos. Um fator que influenciou nesse tempo é a limitação do *ThingSpeak* de, no mínimo, 15 segundos de intervalo entre dois envios seguidos.

- **Conexão com a Internet:** As informações ficam dispostas ao usuário por meio da Internet, ou seja, se não tiver conexão com a Internet, o usuário não poderá ver as informações dispostas no servidor *ThingSpeak*. Poderia ser utilizado um banco de dados que no qual seria utilizado para gravar as informações de deslocamento e sensoriamento de gás, e desenvolver uma aplicação web que disponibiliza estes dados ao usuário. Assim o sistema não dependeria da Internet para mostrar os dados ao usuário.

6 CONCLUSÃO, CONTRIBUIÇÕES E TRABALHOS FUTUROS

A robótica móvel e a Internet das Coisas são campos de estudo que estão em crescente evolução devido às possibilidades que esses tipos de tecnologia podem trazer, como por exemplo, solucionar problemas que necessitam a substituição ou cooperação com os humanos. Este trabalho exemplifica a utilidade de um robô móvel com tecnologia LoRa para monitoramento de gás.

Foram estudados muitos trabalhos desenvolvidos que envolvem o uso de robótica móvel, IoT e sensoriamento de gás, trabalhos nos quais possui finalidades parecidas com este trabalho, trabalho no qual é proposto um protótipo robótico móvel com tecnologia LoRa para monitoramento de gás. A construção do protótipo foi apresentada de forma detalhada e de forma didática a fim de facilitar a sua reprodução e o entendimento dos resultados obtidos comprovando o correto funcionamento.

Os resultados do protótipo construído foram satisfatórios considerando os objetivos do trabalho. Foi possível criar um protótipo robótico funcional autônomo conectado à uma rede LoRa, onde o mesmo é capaz de reconhecer gás por meio de um sensor, e disponibilizar as informações de sensoriamento para o usuário na Internet.

Como já mencionado, a robótica móvel, IoT e sensoriamento de gás são campos de estudos que estão em constante evolução e demandam conhecimentos multidisciplinares, mas que pôde-se abordar inicialmente de forma satisfatória apesar das limitações apresentadas.

Há uma diversidade de sugestões para continuidade deste trabalho. É possível utilizar outros tipos de robôs para sensoriamento de gás móvel, como por exemplo drones. Outra sugestão é utilizar algoritmos mais avançados para a autonomia do robô como por exemplo a utilização de algoritmos genéticos. Além do protocolo LoRa utilizado tem a possibilidade de utilizar outros protocolos de comunicação aplicados à Internet das coisas, como por exemplo o protocolo ZigBee.

Outra sugestão é que ao invés de os dados serem carregados diretamente a um servidor conectado à Internet, os mesmos seriam inseridos em um banco de dados onde teria um processamento interno desses dados para mostrá-los por meio de uma interface gráfica para o usuário, com isso o sistema deixaria de depender da conexão à Internet.

A construção deste protótipo foi uma forma de enriquecer os conhecimentos multidisciplinares contribuindo para a formação acadêmica do aluno, pois consiste em estudar diversos conceitos como planejar sistema de comunicação, eletrônica, mecânica e programação de microcontroladores e colocar todos estes conhecimentos para trabalharem em conjunto.

REFERÊNCIAS

[1] ZHOU, K. et al (2015) – **Industry 4.0: Towards future industrial opportunities and challenges**, Disponível em: < https://ieeexplore.ieee.org/document/7382284 >, Acesso em: 07 de março de 2020

[2] LORENZ, M. et al (2015) **- Industry 4.0: The Future of Productivity and Growth in Manufacturing Industries**, Disponível em : <https://www.bcg.com/pt-br/publications/2015/engineered_products_project_business_industry_4_future_productivity_growth_manufacturing_industries.aspx >, Acesso em: 07 de março de 2020

[3] HERMANN, M. et al (2015) - **Design Principles for Industrie 4.0 Scenarios: A Literature Review**, Disponível em: < https://www.researchgate.net/publication/307864150_Design_Principles_for_Industrie_40_Scenarios_A_Literature_Review/link/57cfd2fb08aed6789701cbeb/download>, Acesso em : 10 de março de 2020

[4] PEDERNEIRAS, G. (2019) – **Robótica Colaborativa: o que é e quais seus benefícios?**, Disponível em: < https://www.industria40.ind.br/artigo/18270-robotica-colaborativa-o-que-e-e-quais-seus-beneficios >, Acesso em: 10 de março de 2020

[5] CIO (2019) **– 4 recomendações para acompanhar a 4° Revolução Industrial** Disponível em: < https://cio.com.br/4-recomendacoes-para-acompanhar-a-4a-revolucao-industrial/ >, Acesso em: 11 de março de 2020

[6] FILHO, T. (2015) **- Aplicação de Robôs nas Industrias**, Disponível em: < https://www.researchgate.net/profile/Teodiano_Freire/publication/266069182_Aplicacao_de_Robos_nas_Industrias/links/55f7439f08aeba1d9ef613ad.pdf >, Acesso em: 10 de março de 2020

[7] GUIMARÃES, C. (2019) – **Conheça Colossus, o robô bombeiro que ajudou a apagar o incêndio na Notre-Dame**, Disponível em: < https://olhardigital.com.br/noticia/conheca-o-colossus-robo-bombeiro-que-ajudou-a-apagar-o-incendio-na-notre-dame/84846 >, Acesso em: 20 de março de 2020

[8] MANDAVILLI, A. (2018) – **The World's Worst Industrial Disaster Is Still Unfolding**, Disponível em: < https://www.theatlantic.com/science/archive/2018/07/the-worlds-worst-industrial-disaster-is-still-unfolding/560726/ >, Acesso em 15 de março de 2020

[9] NEWS.BBC (2010) – **Bhopal trial: Eight convicted over India gas disaster**, Disponível em: < http://news.bbc.co.uk/2/hi/south_asia/8725140.stm >, Acesso em 15 de março de 2020

[10] GUIA TRABALHISTA (2018) – **Segurança e Saúde no Trabalho com Inflamáveis e Combustíveis**, Disponível em: < http://www.guiatrabalhista.com.br/legislacao/nr/nr20.htm#20.9_Inspe%C3%A7%C3%A3o_em_Seguran%C3%A7a_e_Sa%C3%BAde_no_Ambiente_de_Trabalho >, Acesso em 21 de março de 2020

[11] PTRIUNFO (2019) – **Quais os riscos de um vazamento de gás em uma indústria**, Disponível em: < http://ptriunfo.com.br/2019/09/20/quais-os-riscos-de-um-vazamento-de-gas-em-uma-industria/ >, Acesso em 21 de março de 2020

[12] ALIGIER (2019) – **A importância da comunicação sem fio e sua relação com IoT**, Disponível em: < https://www.aliger.com.br/blog/importancia-da-comunicacao-sem-fio-e-sua-relacao-com-iot >, Acesso em 22 de março de 2020

[13] VEDOIS (2018) **– Importância do monitoramento de produção na Indústria 4.0**, Disponível em: < http://vedois.com.br/site/importancia-do-monitoramento-de-producao-na-industria-4-0/ >, Acesso em 22 de março de 2020

[14] TELECO (2020) - **LAN / MAN Wireless I: Introdução**, Disponível em: < https://www.teleco.com.br/tutoriais/tutorialrwlanman1/pagina_1.asp >, Acesso em 22 de março de 2020

[15] CRUZ, O. (2020) – **Estudo de protocolo de comunicação LoRaWAN aplicado a robótica**, Disponível em: < https://github.com/OtaviodaCruz/IC_LoRa >, Acesso em 23 de março de 2020

[16] ANURADHA, J., TRIPATHY, B.K. (2018) - **Internet of the kings (IoT) Technologies**, Applictions, challenges, and solutions, Editora CRC Press

[17] M&S INDUSTRIAL (2020) – Monitoramento Remoto – Internet of Things (IoT), Disponível em: < http://mesindustrial.com.br/monitoramento-remoto_Internet-of-things-iot/ >, Acesso em 24 de março de 2020

[18] BERTELLI, G. (2017) – **Controle sobre Redes Industriais sem Fio: Uma Avaliação de Desempenho dos Padrões WirelessHART e ISA100.11a**, Disponível em: < http://bdtd.ibict.br/vufind/Record/UFRN_0b56eeb0f3491d95e5717d9c6cc70a41 >, Acesso em 24 de março de 2020

[19] EVANIR, A. (2014) – **ISA100 e WirelessHART: Aplicações na Instrumentação e Controle de Processos Industriais**, Disponível em: < https://www.automacaoindustrial.info/redes-industriais/wirelesshart/>, Acesso em 24 de março de 2020

[20] WILLIG, A. et al (2005) – **Wireless Technology in Industrial Networks**, Disponível em: < https://ieeexplore.ieee.org/abstract/document/1435743 >, Acesso em 24 de março d 2020

[21] PÉREZ, D., RISCO, R. (2019) – **Implementation of lora and LoRaWAN as a future scenario of industry 4.0 in peruvian agro-industry sector**, Disponível em: < https://www.usmp.edu.pe/campus/pdf/articulos/articulo_20.pdf >, Acesso em 24 de março de 2020

[22] NOVIDÁ (2019) – **Rede LoRa: o que é e quais são as aplicações**, Disponível em: < https://www.novida.com.br/blog/rede-lora/ >, Acesso em 25 de março de 2019

[23] CONTECHIND (2017) – **É preciso que as indústrias sejam cuidadosas na detecção de gases tóxicos**, Disponível em : < http://contechind.com.br/blog/2017/04/10/e-preciso-que-as-industrias-sejam-cuidadosas-na-deteccao-de-gases-toxicos/ >, Acesso em 02 de abril de 2020

[24] CONTECHIND (2017) – **Detector de gás para área não classificada**, Disponível em: < https://www.contechind.com.br/incendio/detectores-enderecaveis-de-sistema-de-gas/ >, Acessado em 02 de abril de 2020

[25] SILVA, J., ROCHA, T., FERREIRA, A. (2010) – **A importância de detecção de gases para prevenção de danos à segurança, meio ambiente e saúde: fontes de interferência em sinais 4 a 20 Ma**, Disponível em: < https://shtservicos.com.br/Boletim%20T%C3%A9cnico/A%20import%C3%A2ncia%20de%20detec%C3%A7%C3%A3o%20de%20gases%20para%20preven%C3%A7%C3%A3o%20de%20danos.pdf >, Acesso em 02 de abril de 2020

[26] ANPEI (2016) – **Internet das Coisas vai transformar a indústria de petróleo e gás**, Disponível em: < http://anpei.org.br/Internet-das-coisas-vai-transformar-a-industria-de-petroleo-e-gas/ >, Acesso em 03 de abril de 2020

[27] LEE, H., KE, K. (2018) – **Monitoring of Large-Area IoT Sensors Using a LoRa Wireless Mesh Network System: Design and Evaluation**, Disponível em: < https://ieeexplore.ieee.org/document/8326735 >, Acesso em 03 de abril de 2020

[28] PORAZZA, R. (2018) – **Robôs móveis: qual sua importância na indústria ?**, Disponível em: < https://www.pollux.com.br/blog/robos-moveis-qual-sua-importancia-na-industria/ >, Acesso em: 03 de abril de 2020

[29] NAVARRO, A. et al (2010) – **Os acidentes industriais e suas consequências**, Disponível em: < https://www.researchgate.net/publication/228452373_Os_acidentes_industriais_e_suas_consequencias >, Acesso em 15 de abril de 2020

[30] SOUZA, R. (2019) – **7 Desastres industriais mais terríveis de todos os tempos**, Disponível em < https://www.megacurioso.com.br/acontecimentos-historicos/36612-7-desastres-industriais-mais-terriveis-de-todos-os-tempos.htm >, Acesso em 15 de abril de 2020

[31] SEVERO, J. (2018) – **Protótipo para detecção de vazamento de gás glp**, Disponível em: < http://repositorio.upf.br/handle/riupf/1697 >, Acesso em 15 de abril de 2020

[32] CARDOSO, M. – **Metano**, Disponível em: < https://www.infoescola.com/compostos-quimicos/metano/ >, Acesso em 15 de abril de 2020

[33] SOUZA, T. et al (2014) – **Botch4: um robô remotamente controlado para detecção de gás metano em aterros sanitários**, Disponível em: < http://copec.eu/congresses/shewc2014/proc/works/31.pdf >, Acesso em 16 de abril de 2020

[34] VISVANATHAN, R. et al (2015) – **Implementation of behaviour based robot with sense of smell and sight**, Disponível em: < https://www.sciencedirect.com/science/article/pii/S1877050915038016 >, Acesso em 16 de abril de 2020

[35] MARTÍNEZ, D. et al (2014) – **A Mobile Robot Agent for Gas Leak Source Detection**, Disponível em: < https://repositori.udl.cat/handle/10459.1/62842 >, Acesso em 16 de abril de 2020

[36] SILVA, H. (2017) – **Protótipo de Monitoramento de Amônia para uma Granja Universitária**, disponível em: < http://200.201.11.152/handle/123456789/230 >, Acesso em 17 de abril de 2020

[37] ARAÚJO, F. (2019) – **Desenvolvimento de um sistema para medição do nível de agua a distância utilizando o protocolo LoRaWAN para estudo da viabilidade de monitoramentos dos níveis dos Igaparés**, Disponível em: < http://repositorioinstitucional.uea.edu.br/bitstream/riuea/2539/1/Desenvolvimento%20de%20um%20sistema%20para%20medi%C3%A7%C3%A3o%20do%20n%C3%ADvel%20de%20%C3%A1gua%20a%20dist%C3%A2ncia%20utilizando%20o%20protocolo%20LoRawan%20para%20estudo%20da%20viabilidade%20de%20monitoramentos%20dos%20n%C3%ADveis%20dos%20igarap%C3%A9.pdf > Acesso em 17 de abril de 2020

[38] HAXHIBEQIRI, J. et al (2017) – **LoRa Indoor Coverage and Perfomance in na Industrial Environment: Case Study**, Disponível em: < https://ieeexplore.ieee.org/document/8247601/authors#authors >, Acesso em 18 de abril de 2020

[39] LIU, S. et al (2020) – **Development of a low-cost UAV-based system for CH_4 monitoring over oil fields**, Disponível em: < https://www.tandfonline.com/doi/full/10.1080/09593330.2020.1724199 >, Acesso em 18 de abril de 2020

[40] AIRPRODUCTS – **Hidrogênio**, disponível em: < http://www.airproducts.com.br/Products/Gases/Hydrogen.aspx >, Acesso em: 18 de abril de 2020

[41] MUNIZ, L. (2019) – **Nitrogênio industrial: conheça suas principais aplicações**, Disponível em: < https://eficienciaenergetica.atlascopco.com.br/nitrogenio-industrial-principais-aplicacoes/ >, Acesso em 18 de abril de 2020

[42] AIRPRODUCTS – **Dióxido de carbono**, Disponível em: < http://www.airproducts.com.br/Products/Gases/Carbon-Dioxide.aspx >, Acesso em 18 de abril de 2020

[43] CONSELHO REGIONAL DE QUÍMICA – IV REGIÃO – **Gases Industriais**, Disponível em: < https://www.crq4.org.br/quimica_viva__gases_industriais >, Acesso em 18 de abril de 2020

[44] DPUNION (2017) – **Vazamento de gás na indústria: Quais são os riscos e como evitar**, Disponível em: < https://dpunion.com.br/vazamento-de-gas-na-industria-quais-sao-os-riscos-e-como-evitar/ >, Acesso em 18 de abril de 2020

[45] MAINIER, F. (2001) – **Os acidentes químicos: Um alerta as disciplinas de processos industriais**, Disponível em: < http://www.abenge.org.br/cobenge/arquivos/18/trabalhos/EMA003.pdf >, Acesso em 18 de abril de 2020

[46] G1 (2015) – **Vazamento de amônia em indústria intoxica trabalhadores em Mineiros**, Disponível em: < http://g1.globo.com/goias/noticia/2015/02/vazamento-de-amonia-em-industria-intoxica-trabalhadores-em-mineiros.html >, Acesso em 18 de abril de 2020

[47] CONTECHIND (2018) – **Prevenção contra envenenamento por gás industrial**, Disponível em: < http://contechind.com.br/blog/2018/08/06/prevencao-contra-o-envenenamento-por-gas-industrial/?rdst_srcid=1395333>, Acesso em 19 de abril de 2020

[48] EVANS, J. (1999) – **Monóxido de carbono mais do que somente um gás letal !**, Disponível em: < http://qnesc.sbq.org.br/online/qnesc09/atual.pdf >, Acesso em 19 de abril de 2020

[49] CONFOR (2018) – **Gás amônia em ambientes industriais**, Disponível em: < https://www.confor.com.br/blog/gas-amonia-em-ambientes-industriais-nh3 >, Acesso em 19 de abril de 2020

[50] HIDROGERON (2016) – **Por que as Estações de Tratamento estão fugindo do Cloro Gás**, Disponível em: < https://hidrogeron.com/2016/10/25/por-que-as-estacoes-de-tratamento-estao-fugindo-do-cloro-gas/ >, Acesso em 19 de abril de 2020

[51] INDUSTRIAL SCIENTIFIC – **Detectores de gás cianeto de hidrogênio**, Disponível em: < https://www.indsci.br.com/detectores-de-gas/por-gas/cianeto-de-hidrogenio-hcn/ >, Acesso em 19 de abril de 2020

[52] FERREIRA, V. – **Dióxido de enxofre**, Disponível em: < https://www.manualdaquimica.com/quimica-ambiental/dioxido-enxofre.htm >, Acesso em 19 de abril de 2020

[53] CONCEITO.DE – **Conceito de propano**, Disponível em: < https://conceito.de/propano >, Acesso em 19 de abril de 2020

[54] OZÓRIO, M. (2014) – **Desenvolvimento de robô detector de gases**, Disponível em: < https://www.unifacvest.edu.br/assets/uploads/files/arquivos/7a455-ozorio,-m.m.-desenvolvimento-de-robo-detector-de-gases.-unifacvest,-2014..pdf >, Acesso em 19 de abril de 2020

[55] MANUTENÇÃO E SUPRIMENTOS (2020) – **Aplicações do gás butano na indústria**, Disponível em < https://www.manutencaoesuprimentos.com.br/aplicacoes-do-gas-butano-na-industria/#gsc.tab=0 >, Acesso em 19 de abril de 2020

[56] SILVA, R. (2016) – **Emissões de metano da indústria de petróleo e gás geram riscos aos investidores**, Disponível em: < https://cetesb.sp.gov.br/proclima/2016/10/31/emissoes-de-metano-da-industria-de-petroleo-e-gas-geram-riscos-ao-investidores/ >, Acesso em 19 de abril de 2020

[57] MORDECHAI, B. FRANCESCO, M. (2018) – **Elements of Robotics**, Editora Springer International Publishing

[58] AGÊNCIA DE NOTÍCIAS (2017) – **A era dos robôs: tecnologia amplia produtividade, transforma educação e salva vidas**, Disponível em: < https://noticias.portaldaindustria.com.br/noticias/educacao/a-era-dos-robos-tecnologia-amplia-produtividade-transforma-educacao-e-salva-vidas/ >, Acesso em 25 de abril de 2020

[59] CASTELLI, I. (2013) – **Como funcionam os robôs antibombas**, Disponível em: < https://www.tecmundo.com.br/robotica/38103-como-funcionam-os-robos-antibombas-.htm >, Acesso em 25 de abril de 2020

[60] OLIVEIRA,B. et al (2017) – **Tipos e aplicações de sensores na robótica**, Disponível em: < https://periodicos.set.edu.br/index.php/fitsexatas/article/view/4403 >, Acesso em 25 de abril de 2020

[61] GEARBEST (2019) **Tipos de sensores ,** Disponível em: < https://br.gearbest.com/kits/pp_226897.html > , Acesso em 25 de abril de 2020

[62] ROBOLIV (2013) **Microcontroladores**, Disponível em: < http://www.roboliv.re/conteudo/microcontroladores >, Acesso em 26 de abril de 2020

[63] MONK, S. (2017) – **Programação com** Arduino**: Começando com Sketches**, Editora Bookman

[64] THOMSEN, A. (2014) – **O que é** Arduino, Disponível em: < https://www.filipeflop.com/blog/o-que-e-Arduino/ >, Acesso em 26 de abril de 2020

[65] SILVA, T. (2019) – **O que é** Arduino **? E para que serve ? – O guia definitivo**, Disponível em: < https://blog.silvatronics.com.br/Arduino-o-guia-definitivo/ >, Acesso em 26 de abril de 2020

[66] MADEIRA, D. (2017) – **Protocolo I2C – Comunicação entre** Arduinos, Disponível em: < https://portal.vidadesilicio.com.br/i2c-comunicacao-entre-Arduinos/>, Acesso em 27 de abril de 2020

[67] SACCO, F. (2014) – **Comunicação SPI – Parte 1**, Disponível em: < https://www.embarcados.com.br/spi-parte-1/ >, Acesso em 27 de abril de 2020

[68] REZENDE, R. (2020) – **Interface gráfica em C# para comunicação serial (UART)**, Disponível em: < https://www.embarcados.com.br/interface-grafica-para-uart/#:~:text=O%20protocolo%20de%20comunica%C3%A7%C3%A3o%20UART,transmitido%20com%20o%20sinal%20recebido.>, Acesso em 27 de abril de 2020

[69] CONTECHIND (2017) – **Saiba como funcionam os principais sensores para detecção de gás**, Disponível em: < http://contechind.com.br/blog/2017/06/23/saiba-como-funcionam-os-principais-sensores-para-deteccao-de-gas/ >, Acesso em 28 de abril de 2020

[70] PROTEÇÃO RESPIRATÓRIA – **Sensores de gás – Princípios e Tecnologias**, Disponível em: < https://protecaorespiratoria.com/sensores-de-gas-principios-e-tecnologias/ >, Acesso em 28 de abril de 2020

[71] INDUSTRIAL SCIENTIFIC – **Sensores Eletroquímicos**, Disponível em: < https://www.indsci.br.com/treinamento/gas--educa%C3%A7%C3%A3o-geral/electrochemical-sensors/ >, Acesso em 28 de abril de 2020

[72] LARRAMENDY, M. SOLONESKI, S. (2016) – **Green** Nano**technology – Overview and Further Prospects**, Disponível em: <researchgate.net/publication/303856611_gree_Nanotechnology_-_Overiview_and_Further_Prospects >, Acesso em 28 de abril de 2020

[73] PERCON – **Detecção de gases e os tipos de sensores**, Disponível em: < https://acessopercon.com.br/percon/deteccao-de-gases-e-os-tipos-de-sensores/

>, Acesso em 28 de abril de 2020

[74] CANDIDO, G. (2017) – **Sensor de gás MQ-135 e a família de sensores MQ**, Disponível em: < https://portal.vidadesilicio.com.br/sensor-de-gas-mq-135/>, Acesso em 28 de abril de 2020

[75] JÚNIO, C. (2019) – **Sistema de monitoramento e mapeamento por robô móvel**, Disponível em: < https://repositorio.ufu.br/bitstream/123456789/27914/1/SistemaMonitoramentoMapeamento.pdf >, Acesso em 29 de abril de 2020

[76] THOMSEN, A. (2011) - **Como conectar o Sensor Ultrassônico HC-SR04 ao** Arduino, Disponível em: <robots.stanford.edu/papers/thrun.mapping-tr.pdf>, Acesso em: 29 de abril de 2020

[77] MEYER, M. (2000) – **Nosso futuro e o computador**, Editora Bookman

[78] ELIAS, G., LOBATO, L. (2013) – **Arquitetura e Protocolos de Rede TCP-IP**, Editora RNP/ESR

[79] CORRÊA, U. et al (2006) – **Redes Locais Sem Fio: Conceito e Aplicações**, Disponível em: <researchgate.net/publication/233869181_Redes_Locais_Sem_Fio_Conceitos_e_Aplicacoes>, Acesso em 06 de maio de 2020

[80] SANTOS, A. (2016) – **Módulo 6 – Programação de servidores**, Disponível em: < https://sites.google.com/site/redesdecomunicacaonivel3/modulo-6 >, Acesso em 06 de maio de 2020

[81] KIKUCHI, R. (2009) – **Arquiteturas de Distribuição**, Disponível em: < https://www.gta.ufrj.br/ensino/eel879/trabalhos_vf_2009_2/kikuchi/arquiteturas.html >, Acesso em 06 de maio de 2020

[82] NASCIMENTO, M. (2019) – **Qual Diferença entre Modelo OSI e TCP/IP**, Disponível em: < http://www.dltec.com.br/blog/redes/diferenca-entre-modelo-osi-e-tcp-ip/ >, Acesso em 06 de maio de 2020

[83] PINTO, P. (2010) – **Redes – Sabe o que é modelo OSI ?**, Disponível em: < https://pplware.sapo.pt/tutoriais/networking/redes-sabe-o-que-e-o-modelo-osi/ >, Acesso em 06 de maio de 2020

[84] REIS, R. (2017) – **Modelo TCP/IP – Definição, camadas e funcionamento**, Disponível em: < http://infotecnews.com.br/modelo-tcpip/ >, Acesso em 06 de maio de 2020

[85] AUZEIR, J. (2018) – **Modelo de Camadas OSI e TCP/IP**, Disponível em: < https://redesafins.blogspot.com/2018/10/modelo-de-camadas-osi-e-tcpip.html >, Acesso em 06 de maio de 2020

[86] MATHEUS, Y. (2018) – **O modelo OSI e suas camadas**, Disponível em: < https://www.alura.com.br/artigos/conhecendo-o-modelo-osi >, Acesso em 06 de maio de 2020

[87] CARISSIMI, A., ROCHOL, J., GRANVILLE, L. (2009) – **redes de computadores**, Editora bookman

[88] SANTIN, R. et al (2017) – **LPWAN Low power wide área network**, Disponível em: < https://www.ime.usp.br/~diogojp/computacao-movel-2017/seminar/rene_santin_LPWAN.pdf >, Acesso em 07 de maio de 2020

[89] MELO, P. (2020) – **Introdução ao LPWAN (Low Power Wide Area Network**), Disponível em: < https://www.embarcados.com.br/introducao-ao-lpwan/>, Acesso em 07 de maio de 2020

[90] MEYER, F. (2017) – **A comparative study of LPWAN Technologies for large-scale IoT deployment**, Disponível em: < https://www.sciencedirect.com/science/article/pii/S2405959517302953 >, Acesso em 07 de maio de 2020

[91] MARQUES, J., BOCHIE, K. (2018) – **LoRa**, Disponível em: < https://www.gta.ufrj.br/ensino/eel878/redes1-2018-1/trabalhos-vf/lora/ >, Acesso em 14 de maio de 2020

[92] BOUNCEUR, A. et al (2017) – **A study of LoRa low power and wide área network technology**, Disponível em: < https://www.researchgate.net/figure/Up-chirp-and-down-chirp-waveform_fig2_320649650 >, Acesso em 14 de maio de 2020

[93] VERMONT REP. (2019) – **Camada Física do LoRa**, Disponível em: < https://www.youtube.com/watch?v=lkgynwPjymo >, Acesso em 14 de maio de 2020

[94] WANG, Y. et al (2019) – **LoRa-Hybrid: A LoRaWAN Based multishop solution for regional microgrid**, Disponível em: < https://www.researchgate.net/figure/LoRa-modulation-simulation-with-different-SF-Fig-2-illustrates-the-time-that-takes-to_fig2_331319905 >, Acesso em 14 de maio de 2020

[95] VERMONT REP. (2019) – **Organização e Topologia de LoRaWAN**, Disponível em: < https://www.youtube.com/watch?v=HtqjnTl2XWE >, Acesso em 14 de maio de 2020

[96] JUNIOR, V. (2020) – **Conheça a tecnologia LoRa e o protocolo LoRaWAN**, Disponível em: < https://www.embarcados.com.br/conheca-tecnologia-lora-e-o-protocolo-lorawan/ >, Acesso em 14 de maio de 2020

[97] VERMONT REP. (2020) – **Organização Lógica em LoRaWAN**, Disponível em: < https://www.youtube.com/watch?v=3QfpWIjJNyQ >, Acesso em 15 de maio de 2020

[98] NEWIE VENTURES (2018) – **LoRaWAN: OTAA or ABP**, Disponível em: <https://www.newieventures.com.au/blogtext/2018/2/26/lorawan-otaa-or-abp >, Acesso em 15 de maio de 2020

[99] AZEVEDO, T. (2017) – **Locomoção de um robô móvel com esteiras em escadas**, Disponível em: < http://monografias.poli.ufrj.br/monografias/monopoli10020024.pdf >, Acesso em 22 de maio de 2020

[100] DRAGINO – **About Dragino**, Disponível em: < https://www.dragino.com/about/about.html >, Acesso em 22 de maio de 2020

[101] DRAGINO – **LoRa *Shield* for** Arduino, Disponível em: < https://www.dragino.com/products/lora/item/102-lora-*shield*.html >, Acesso em 22 de maio de 2020

[102] DRAGINO – **LG01-P Gateway featuring LoRa technology**, Disponível em: < https://www.dragino.com/products/lora/item/117-lg01-p.html >, Acesso em 22 de maio de 2020

[103] MAUREIRA, M. et al (2014) – ***ThingSpeak* – na API and Web Service for the Internet of Things**, Disponível em: < https://staas.home.xs4all.nl/t/swtr/documents/wt2014_*ThingSpeak*.pdf > Acesso em 22 de maio de 2020

[104] ALBUQUERQUE, I. (2020) – **Estação meteorológica com** Arduino, Disponível em: < https://www.embarcados.com.br/estacao-meteorologica-com-Arduino/ >, Acesso em 22 de maio de 2020

[105] DRAGINO (2022) - **Downstream to *ThingSpeak* Iot Server,** Disponível em: **<** https://wiki.dragino.com/index.php/Lora_*Shield*#Example6__Upstream.2FDownstream_to_*ThingSpeak*_IoT_Server > Acesso em 22 de fevereiro de 2021

[106] CESAR (2019) – **Sistema de Monitoramento e Mapeamento por Robô Móvel**, Disponível em: < https://repositorio.ufu.br/bitstream/123456789/27914/1/SistemaMonitoramentoMapeamento.pdf>, Acesso e, 23 de março de 2021

[107] Halliday, K. S. Krane (2003) – Física, vol. 1 Mecânica, 5° ed., LTC

APÊNDICE I

Código do Arduino IDE programado no Arduino nano.

```
/// Referencias: http://tixplicando.blogspot.com/2015/10/criando-um-robo-para-derrubar-obstaculos.html
  //              https://github.com/JacksonDuvan/Machine-Learning-/blob/master/laberinto1.ino
#include "Wire.h"
#define slaveAdress 0x08
// Variaves para controle do motor
int URTRIG = 5;
int Frente = 0;
int Tras  = 1;
int MotorEsq = 1;
int MotorDir = 2;
int URPWM = 3;
uint8_t EnPwmCmd[4] = {0x44, 0x22, 0xbb, 0x01};
int STBY = 10;
 //Motor A
int PWMA = 6;
int AIN1 = 9;
int AIN2 = 8;
 //Motor B
int PWMB = 11;
int BIN1 = 12;
int BIN2 = 2;
//vetor de envio da localização
char locVet [2];
char loc[2];
unsigned long previusMillis = 0;
// Variaves para controle do sensor ultrassônico
const int EchoPIN1 = 4;
const int TrigPIN1 = 7;
#define EchoPIN2  A2
#define TrigPIN2 A3
#define EchoPIN3  A0
#define TrigPIN3  A1
long distancia_cm1 = 0;
long distancia_cm2 = 0;
long distancia_cm3 = 0;
long Duration1;
long Duration2;
long Duration3;
int status_sentido;
float RightDistance = 0;
float LeftDistance = 0;
float MiddletDistance = 0;
int MaxX; //Armazena a distância X maxima no mapa em centimetros
int MaxY; //Armazena a distância Y maxima no mapa em centimetros

//Ditancia em cm do sensor direito
int Distance_Right (void) {
  digitalWrite (TrigPIN2, LOW);
  delayMicroseconds (2);
  digitalWrite (TrigPIN2, HIGH);
  delayMicroseconds (20);
  digitalWrite (TrigPIN2, LOW);
  Duration2 = pulseIn (EchoPIN2, HIGH);
  distancia_cm2 = (Duration2/29/2);
  return distancia_cm2;
}
```

```
//Ditancia em cm do sensor esquerdo
int Distance_Left (void) {
  digitalWrite (TrigPIN3, LOW);
  delayMicroseconds (2);
  digitalWrite (TrigPIN3, HIGH);
  delayMicroseconds (20);
  digitalWrite (TrigPIN3, LOW);
  Duration3 = pulseIn (EchoPIN3, HIGH);
  distancia_cm3 = (Duration3/29/2);
  return distancia_cm3;
}

//Ditancia em cm do sensor do meio
int Distance_Middle (void) {
  digitalWrite (TrigPIN1, LOW);
  delayMicroseconds (2);
  digitalWrite (TrigPIN1, HIGH);
  delayMicroseconds (20);
  digitalWrite (TrigPIN1, LOW);
  Duration1 = pulseIn (EchoPIN1, HIGH);
  distancia_cm1 = (Duration1/29/2);
  return distancia_cm1;
}

void setup() {
   pinMode(EchoPIN1, INPUT);
   pinMode(TrigPIN1, OUTPUT);
   pinMode(EchoPIN2, INPUT);
   pinMode(TrigPIN2, OUTPUT);
   pinMode(EchoPIN3, INPUT);
   pinMode(TrigPIN3, OUTPUT);
   Serial.begin(9600);
   Wire.begin();
   pinMode(STBY, OUTPUT);
   pinMode(PWMA, OUTPUT);
   pinMode(AIN1, OUTPUT);
   pinMode(AIN2, OUTPUT);
   pinMode(PWMB, OUTPUT);
   pinMode(BIN1, OUTPUT);
   pinMode(BIN2, OUTPUT);
   pinMode(A5 ,OUTPUT);
}
void loop() {
  LeftDistance = Distance_Left();
  delay(10);
  RightDistance = Distance_Right();
  delay(10);
  MiddletDistance = Distance_Middle();
  delay(10);

  if(MiddletDistance<=10){
    if(RightDistance > LeftDistance ){
        TurnRight();
    }else if(RightDistance < LeftDistance){
        TurnLeft();
    }else{
        Backward();
    }
   }else if(RightDistance<=5){
```

```
      TurnLeft();
   }else if(LeftDistance<=5){
      TurnRight();
   }else{
       Forward();
   }
  }

void Forward () {
 status_sentido = 4;
 enviarDadosSerial(status_sentido);
 move(MotorEsq, 145, Frente);
 move(MotorDir, 150, Frente);
}
void Backward () {
 status_sentido = 3;
 enviarDadosSerial(status_sentido);
move(MotorEsq, 145, Tras);
move(MotorDir, 150, Tras);
}

void TurnRight () {
 status_sentido = 1;
 enviarDadosSerial(status_sentido);
move(MotorEsq, 150, Frente);
move(MotorDir, 145, Tras);
}
void TurnLeft () {
 status_sentido = 2;
 enviarDadosSerial(status_sentido);
 move(MotorEsq, 145, Tras);
 move(MotorDir, 150, Frente);
}
void move(int motor, int speed, int direction) {
  digitalWrite(STBY, HIGH);
  boolean inPin1 = LOW;
  boolean inPin2 = HIGH;
  if (direction == 1) {
   inPin1 = HIGH;
   inPin2 = LOW;
  }
  if (motor == 1) {
   digitalWrite(AIN1, inPin1);
   digitalWrite(AIN2, inPin2);
   analogWrite(PWMA, speed);
  } else if (motor == 2) {
   digitalWrite(BIN1, inPin1);
   digitalWrite(BIN2, inPin2);
   analogWrite(PWMB, speed);
  }
 }
 void enviarDadosSerial(int sentido){
  if(millis() - previusMillis > 500){
    // incia a transmissao para o endereco 0x08 (slaveAdress)
    Wire.beginTransmission(slaveAdress);
    Wire.write(sentido); // envia um byte contendo o estado do LED
    Wire.endTransmission(); // encerra a transmissao
   }
 }
```

APÊNDICE II

Código do Arduino IDE programado no Arduino uno.

```
#include <SPI.h>
#include "Wire.h"
#include <RH_RF95.h>

RH_RF95 rf95; // driver utilizado para envio e recebimento de dados LoRa

byte bGlobalErr;//variavel para controle de erros no envio e recebimento de dados
char sensLoc_dat[2]; //Vetor para Armazenar dados do sensor de gás
char BytesX[4];
char BytesY[4];
char node_id[3] = {1,1,1}; //ID do nó final LoRa usado para sabermos identificar que nó está enviando dados
float frequency = 915.0; //frequencia de operação do LoRa, no Brasil usamos padrão australiano (915 MHz)
unsigned int count = 1;
//char locSerial[4];
char vetorDados [2];
int sentido = 0;
uint16_t leitura;
//Variaveis Odometria
float angulo;    //Armazena angulo em radiandos do giro do robô
float contadorX; //Contador usado para calculo de distância X
float contadorY; //Contador usado para calculo de distância Y
int contadorr;   //Contador usado para calculo da angulação em radianos
int status_sentido; //Marca o sentido do robô

void setup() {
  Wire.begin(0x08);
  //Registra um evento para ser chamado quando chegar algum dado via I2C
  Wire.onReceive(receiveEvent);
  pinMode(A0, INPUT);
  pinMode(3, INPUT); // Primeiro pino de interrupção
  attachInterrupt(1, ContadorAnguloPosicao, RISING); //
  Serial.begin(9600);
     if (!rf95.init()){
      Serial.println("falha na inicialização do driver");
    }
    //Configurando a frequencia de operação
    rf95.setFrequency(frequency);

    // Potência de configuração, dBm
    rf95.setTxPower(13);
    rf95.setSpreadingFactor(7);
    Serial.println("Nó Final LoRa");
    Serial.println("Sensor de gás MQ-6");
    Serial.print("ID do nó final LoRa:: ");
    for(int i = 0;i < 3; i++){
        Serial.print(node_id[i],HEX);
    }
    Serial.println();
  angulo=0;      //Angulo inicial
  contadorX=50;  //Começa em uma posição X determinada no mapa
  contadorY=50;  //Começa em uma posição Y determinada no mapa
}

void ReadSensorMQ(){
bGlobalErr=0;
leitura = analogRead(A0);
```

```
byte i;
 // for(i =0; i<2; i++){
    sensLoc_dat[0] = (uint8_t)((leitura>>8)&0xFF); //byte mais significativo do sinal
    sensLoc_dat[1] = (uint8_t)leitura; //byte menos significativo do sinal
 // }
}
uint16_t calcByte(uint16_t crc, uint8_t b){
  uint32_t i;
  crc = crc ^ (uint32_t)b << 8;
    for ( i = 0; i < 8; i++){
      if ((crc & 0x8000) == 0x8000){
          crc = crc << 1 ^ 0x1021;
      }else{
          crc = crc << 1;
      }
    }
 return crc & 0xffff;
}
uint16_t CRC16(uint8_t *pBuffer,uint32_t length){
   uint16_t wCRC16=0;
   uint32_t i;
    if (( pBuffer==0 )||( length==0 )){
      return 0;
    }
    for ( i = 0; i < length; i++){
      wCRC16 = calcByte(wCRC16, pBuffer[i]);
    }
    return wCRC16;
}
void loop() {
   Serial.print("###########    ");
    Serial.print("COUNT=");
    Serial.print(count);
    Serial.println("    ###########");
    count++;

    ReadSensorMQ();
    char data[50] = {0} ;
    int dataLength = 13; // tamanho do pacote

    // Use data[0], data[1],data[2] como id do nó
    data[0] = node_id[0] ;
    data[1] = node_id[1] ;
    data[2] = node_id[2] ;
    data[3] = sensLoc_dat[0]; //byte mais significativo da leitura do sensor
    data[4] = sensLoc_dat[1]; //byte menos significaivo da leitura do sensor
    //Bytes posição X
    data[5] = BytesX[0];
    data[6] = BytesX[1];
    data[7] = BytesX[2];
    data[8] = BytesX[3];
    //Bytes posição Y
    data[9] = BytesY[0];
    data[10] = BytesY[1];
    data[11] = BytesY[2];
    data[12] = BytesY[3];

    uint16_t crcData = CRC16((unsigned char*)data,dataLength);//Obter CRC
      Serial.print("Dados a serem enviados(sem CRC): ");
    int i;
```

```
    for(i = 0;i < dataLength; i++){
       Serial.print(data[i],HEX);
       Serial.print(" ");
    }
    Serial.println();

    unsigned char sendBuf[50]={0};
    for(i = 0;i < dataLength;i++){
       sendBuf[i] = data[i] ;
    }
    // Adicionar CRC aos dados LoRa
    sendBuf[dataLength] = (unsigned char)crcData;
    sendBuf[dataLength+1] = (unsigned char)(crcData>>8);

    Serial.print("Dados a serem enviados com CRC    ");
    for(i = 0;i < (dataLength +2); i++){
       Serial.print(sendBuf[i],HEX);
       Serial.print(" ");
    }
    Serial.println();

    rf95.send(sendBuf, dataLength+2);//envio de dos dados LoRa
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];//Matriz de dados de resposta
    uint8_t len = sizeof(buf);//comprimento dos dados de resposta

    if (rf95.waitAvailableTimeout(3000)){ //Verifique se há resposta em 3 segundos
    //Deve ser uma mensagem de resposta para nós agora
      if (rf95.recv(buf, &len)){ //verifique se a mensagem de resposta está correta
         if(buf[0] == node_id[0] && buf[1] == node_id[2] && buf[2] == node_id[2] ){ // Verifique se a
mensagem de resposte tem o ID do nó
            pinMode(4, OUTPUT);
            digitalWrite(4, HIGH);
            Serial.print("Resposta do Gateway: ");//imprime a resposta
            Serial.println((char*)buf);
            delay(400);
            digitalWrite(4, LOW);
         }
      }else{
         Serial.println("Falha na Recepção");//
         rf95.send(sendBuf, strlen((char*)sendBuf));//reenviar se não houver resposta
      }
    }else{
      Serial.println("Sem resposta, o gateway LoRa está em execução ?");//Sem resposta do sinal
      rf95.send(sendBuf, strlen((char*)sendBuf));//reenviar dados
   }
   delay(3000); // Envie dados do sensor a cada 3 segundos
   Serial.println("");

}

void receiveEvent(int leitura)
{
  sentido = Wire.read();
}
 void ContadorAnguloPosicao(){ // Função para valor de Angulo e Posição
  static unsigned long delayCont; // delay falso para retornar um estado sem "tremida"
  switch(sentido){
    case 1: //direita
     if(millis()-delayCont>1){
      contadorr= contadorr+1;
```

```
      angulo=contadorr*3.1415/111;
      if(angulo>=6.283){
        contadorr=0;
      }
     }
      delayCont=millis();
      break;
    case 2 : //esquerda
     if(millis()-delayCont>1){
      contadorr= contadorr-1;
      angulo=contadorr*3.1415/200;
      if(angulo>=6.283){
        contadorr=0;
      }
     }
     delayCont=millis();
    break;
    case 3: //re
      if(millis()-delayCont>1){
        contadorX= contadorX-1*cos(angulo);
        contadorY= contadorY-1*sin(angulo);
      }
      delayCont=millis();
    break;
    case 4: //frente
       if(millis()-delayCont>1){
         contadorX= contadorX+1*cos(angulo);
         contadorY= contadorY+1*sin(angulo);
       }
       delayCont=millis();
    break;
  }
  bytesLocX(contadorX);
  bytesLocY(contadorY);
}

void bytesLocX(float locX){
  union{
     float valX;
     unsigned char bvalX[4];
  }floatAsBytes;
  floatAsBytes.valX = locX;
  BytesX[0] = floatAsBytes.bvalX[0];
  BytesX[1] = floatAsBytes.bvalX[1];
  BytesX[2] = floatAsBytes.bvalX[2];
  BytesX[3] = floatAsBytes.bvalX[3];
 }
 void bytesLocY(float locY){
 union{
     float valY;
     unsigned char bvalY[4];
 }floatAsBytes;
 floatAsBytes.valY = locY;
 BytesY[0] = floatAsBytes.bvalY[0];
 BytesY[1] = floatAsBytes.bvalY[1];
 BytesY[2] = floatAsBytes.bvalY[2];
 BytesY[3] = floatAsBytes.bvalY[3];
 }
```

APÊNDICE III

Código do Arduino IDE programado no *gateway* LoRa

```
#include <SPI.h>
#include <RH_RF95.h>
#include <Console.h>
#include <Process.h>

RH_RF95 rf95;

String myWriteAPIString = "7FE8C3DTIALJCIWG";
uint16_t crcdata = 0;
uint16_t recCRCData = 0;
float frequency = 915.0;
String dataString = "";
char BytesX[4];
char BytesY[4];
float locX;
float locY;

void uploadData(); // Carregar dados para o ThingSpeak.

void setup() {
   Bridge.begin(115200);
   Console.begin();
   if (!rf95.init()){
    Console.println("Falha na inicialização");
  }
  // Configurar frequência ISM
    rf95.setFrequency(frequency);
  // Potência de instalação, dBm
    rf95.setTxPower(13);
   // rf95.setSyncWord(0x34);
}

uint16_t calcByte(uint16_t crc, uint8_t b){
   uint32_t i;
  crc = crc ^ (uint32_t)b << 8;
  for ( i = 0; i < 8; i++) {
   if ((crc & 0x8000) == 0x8000)
   crc = crc << 1 ^ 0x1021;
   else
   crc = crc << 1;
 }
  return crc & 0xffff;
}

uint16_t CRC16(uint8_t *pBuffer, uint32_t length){
 uint16_t wCRC16 = 0;
 uint32_t i;

if (( pBuffer == 0 ) || ( length == 0 )) {
  return 0;
}
for ( i = 0; i < length; i++){
   wCRC16 = calcByte(wCRC16, pBuffer[i]);
 }
return wCRC16;
}
```

```
uint16_t recdata( unsigned char* recbuf, int Length){
   crcdata = CRC16(recbuf, Length - 2); //Obter código CRC
   recCRCData = recbuf[Length - 1]; //Calcular dados CRC
   recCRCData = recCRCData << 8; //
   recCRCData |= recbuf[Length - 2];
}

void loop(){
  if (rf95.waitAvailableTimeout(3000)){ //Ouvir dados do nó LoRa
      uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];//receber buffer de dados
      uint8_t len = sizeof(buf);//comprimento do buffer de dados
       if (rf95.recv(buf, &len)){ //Verifique se há dados recebidos
          recdata( buf, len);
          Console.print("Obtenha o pacote LoRa: ");
          for (int i = 0; i < len; i++){
             Console.print(buf[i],HEX);
             Console.print(" ");
          }
          Console.println();
          if(crcdata == recCRCData){ //Verifique se o CRC está correto
           if(buf[0] == 1 && buf[1] == 1 && buf[2] ==1){ //Verifique se o ID corresponde ao LoRa Node ID
                uint8_t data[] = "Servidor ACK";//Resposta (confirmação de recebimento de dados)
                data[0] = buf[0];
                data[1] = buf[1];
                data[2] = buf[2];
                rf95.send(data, sizeof(data));//Enviar resposta para o nó LoRa
                rf95.waitPacketSent();
               int newData[10] = {0, 0, 0, 0, 0, 0, 0, 0, 0, 0}; //Armazenar dados do sensor aqui
                for (int i = 0; i < 10; i++){
                  newData[i] = buf[i + 3];
                }
                 uint8_t b1 = newData[0];
                 uint8_t b2 = newData[1];
                 BytesX[0] = newData[2];
                 BytesX[1] = newData[3];
                 BytesX[2] = newData[4];
                 BytesX[3] = newData[5];
                 BytesY[0] = newData[6];
                 BytesY[1] = newData[7];
                 BytesY[2] = newData[8];
                 BytesY[3] = newData[9];
                 uint16_t leiSen = b1 << 8 | b2;
                 locX = floatLocX(BytesX);
                 locY = floatLocY(BytesY);
                 Console.print("leitura do sensor: ");
                 Console.println(leiSen);
                 Console.print("LOCX: ");
                 Console.println(locX);
                 Console.print("LOCY: ");
                 Console.println(locY);
                 dataString ="field1=";
                 dataString += leiSen;
                 dataString +="&field5=";
                 dataString += locX;
                 dataString +="&field6=";
                 dataString += locY;
                 uploadData(); //
                 dataString="";
   }
```

```
  }else{
    Console.println(" CRC Fail");
  }
}else{
    Console.println("recv failed");
}
}
}

void uploadData() {//Carregar dados para o ThingSpeak
   // forma a sequência do parâmetro do cabeçalho da API:
  // forme a sequência do parâmetro URL, tenha cuidado com os "
  String upload_url = "https://api.thingspeak.com/update?api_key=";
  upload_url += myWriteAPIString;
  upload_url += "&";
  upload_url += dataString;
  Console.println("Chamar o comando Linux para enviar dados");
  Process p;    //Crie um processo e chame-o de "p", este processo executará um comando curl do Linux
  p.begin("curl");
  p.addParameter("-k");
  p.addParameter(upload_url);
  p.run();    // Execute o processo e aguarde seu término
  Console.print("Comentários do Linux: ");
  //Se houver saída do Linux,
 //envie para o Console:
  while (p.available()>0){
    char c = p.read();
    Console.write(c);
  }
 Console.println("");
 Console.println("Call Finished");
 Console.println("#####################################");
 Console.println("");
}

float floatLocX(char teste[4]){
   union{
     float valX;
     unsigned char bvalX[4];
  }floatAsBytes;
  floatAsBytes.bvalX[0] =teste[0];
  floatAsBytes.bvalX[1] = teste[1];
  floatAsBytes.bvalX[2] = teste[2];
  floatAsBytes.bvalX[3] = teste[3];

  return floatAsBytes.valX;
  }

 float floatLocY(char teste[4]){
   union{
     float valY;
     unsigned char bvalY[4];
  }floatAsBytes;
  floatAsBytes.bvalY[0] =teste[0];
  floatAsBytes.bvalY[1] = teste[1];
  floatAsBytes.bvalY[2] = teste[2];
  floatAsBytes.bvalY[3] = teste[3];

  return floatAsBytes.valY;
  }
```

APÊNDICE IV

Código do MATLAB no *ThingSpeak*

```
% Template MATLAB code for visualizing data from a channel as a 2D line
% plot using PLOT function.


% Prior to running this MATLAB code template, assign the channel variables.
% Set 'readChannelID' to the channel ID of the channel to read from.
% Also, assign the read field ID to 'fieldID1'.


% TODO - Replace the [] with channel ID to read data from:
readChannelID = 1668349;
% TODO - Replace the [] with the Field ID to read data from:
fieldID1 = 5;
fieldID2 = 6;


% Channel Read API Key
% If your channel is private, then enter the read API
% Key between the '' below:
readAPIKey = '3KBN3AJY42VC4MVN';


%% Read Data %%


[data1] = thingSpeakRead(readChannelID, 'Field', fieldID1, 'NumPoints', 30,
'ReadKey', readAPIKey);
[data2] = thingSpeakRead(readChannelID, 'Field', fieldID2, 'NumPoints', 30,
'ReadKey', readAPIKey);


%% Visualize Data %%


plot(data1, data2, '*');
```