Rubens Rodrigues Teixeira Filho

Internet das coisas aplicada em sistemas eletropneumáticos no controle de atuadores utilizando os protocolos ZigBee e MQTT.

Uberlândia

12 de agosto de 2022

Rubens Rodrigues Teixeira Filho

Internet das coisas aplicada em sistemas eletropneumáticos no controle de atuadores utilizando os protocolos ZigBee e MQTT.

Universidade Federal de Uberlândia - UFU

Faculdade de Engenharia Mecânica

Graduação em Engenharia Mecatrônica

Orientador: José Jean-Paul Zanlucchi de Souza Tavares

Uberlândia

12 de agosto de 2022

Teixeira Filho, Rubens Rodrigues
Internet das coisas aplicada em sistemas eletropneumáticos
no controle de atuadores utilizando os protocolos ZigBee e MQTT./ Rubens Rodrigues Teixeira Filho. – Uberlândia, 12 de agosto de 2022-
72p. : il. (algumas color.) ; 30 cm.
Orientador: José Jean-Paul Zanlucchi de Souza Tavares
Monografia – Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Mecânica
Graduação em Engenharia Mecatrônica, 12 de agosto de 2022.
1. Indústria 4.0. 2. ZigBee. 2. MQTT. I. Orientador: José Jean-Paul Zanlucchi de Souza TavaresII. Universidade Federal de Uberlândia - UFU
Faculdade de Engenharia Mecânica
Graduação em Engenharia MecatrônicaIII. IV. Título

Rubens Rodrigues Teixeira Filho

Internet das coisas aplicada em sistemas eletropneumáticos no controle de atuadores utilizando os protocolos ZigBee e MQTT.

Trabalho aprovado. Uberlândia, 12 de agosto de 2022:

José Jean-Paul Zanlucchi de Souza Tavares
Orientador

Professor
José Reinaldo Silva

Professor
Roberto Mendes Finzi Neto

Uberlândia
12 de agosto de 2022

Este trabalho é dedicado à minha familia pelo apoio que me deram na minha trajetória, a minha companheira e aos meus amigos.

Agradecimentos

Ao Professor José Jean-Paul Zanlucchi de Souza Tavares pelas orientações e ao Laboratório de Ensino de Mecatrônica pelo espaço espaço cedido para o desenvolvimento deste trabalho.

Alguns homens veem as coisas como são, e dizem 'Por quê?'
Eu sonho com as coisas que nunca foram e digo 'Por que não?'
(Geroge Bernard Shaw)

Resumo

A busca por eficiência e maiores detalhes da produção tem levado inovação para a indústria. A Indústria 4.0 é um novo conceito que representa uma evolução dos sistemas produtivos atuais a partir da convergência entre novas tecnologias de automação industrial e tecnologia da informação. Este trabalho apresenta uma solução em automação para um sistema eletropneumático baseado em código aberto, utilizando os conceito do IoT(*Internet of things*). Na arquitetura dessa solução os sensores dos atuadores enviam mensagens ZigBee para as válvulas, que por sua vez enviam mensagens para um broker MQTT (Mosquitto) que é o ponto de comunicação entre os atuadores.

Palavras-chave: ZigBee. MQTT. Mosquitto. IoT. Indústria 4.0.

Abstract

The focus on efficiency and higher details in production has brought innovation to the industry. The Industry 4.0 is a new concept representing the evolution of the current production systems through the convergence of new technologies of industrial automation and information technology. This dissertation presents a solution of on automation for a electropneumatic system based in open source, using concepts of IoT. In architecture of this solution the sensors of actuators send ZigBee messages to the valves, which in turn send messages for a MQTT broker (Mosquitto) that is a point of communication between the actuators.

Keywords: ZigBee. MQTT. Mosquitto. IoT. Industry 4.0.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CLP	Controlador Lógico programável
IIoT	Industrial Internet of Things
IoT	Internet of Things
ISR	Interrupt Service Routine
MQTT	Message Queuing Telemetry Transport
MQTT-SN	MQTT for Sensor Networks
M2M	Machine to Machine
NA	Normal aberta
NF	Normal fechada
SFC	Sequential Functional Charts
QoS	Quality of service

Sumário

1 Introdução

A primeira revolução industrial foi a mecanização dos processos através da energia hidráulica e do vapor. A segunda revolução industrial deu inicio a produção em massa proporcionada pela aplicação da energia elétrica. A terceira revolução industrial foi a automação dos processos, através de CLPs e introdução de robôs ao processo.

A quarta revolução industrial, no entanto, não diz respeito apenas a sistemas e máquinas inteligentes e conectadas. Seu escopo é muito mais amplo. Ondas de novas descobertas ocorrem simultaneamente em áreas que vão desde o sequenciamento genético até a nanotecnologia, das energias renováveis à computação quântica. O que torna a quarta revolução industrial fundamentalmente diferente das anteriores é a fusão dessas tecnologias e a interação entre os domínios físicos, digitas e biológicos.(SCHWAB, 2019)

Os pontos principais dessas revoluções podem ser vistos na Figura 1

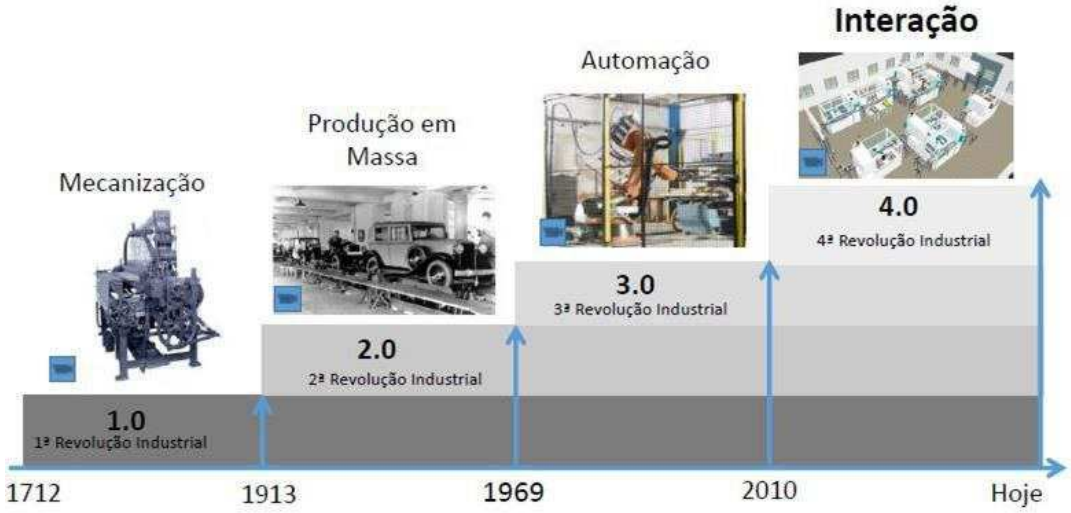


Figura 1 – Etapas da revolução industrial.

Fonte: DFKI 2011 (German Research Center for Articial Intelligence)

Um dos pilares da Indústria 4.0 é a descentralização da lógica dos processos, tendo cada elemento autonomia sobre o que ele próprio deve fazer e quando, a modularidade e a interoperabilidade são outros pilares fundamentais nessa nova indústria.

É fato que as tecnologias estão cada vez mais presentes na vida das pessoas, isso se deve a evolução dos microprocessadores que estão cada vez mais nos aparelhos consumidos pela sociedade como: Smart TVs, Smartphones, Smartwatches entre tantos outros.

Diversos conceitos ganharam os holofotes, como Inteligência Artificial, Aprendizado de Máquina, Internet das Coisas (IoT, *Internet of Things*); e muitas dessas novas tecnologias estão cada vez mais presentes na realidade de ambientes industriais.

O IoT e a Indústria 4.0 andam juntos na evolução dos computadores, e se tratando de sistemas embarcados na evolução dos protocolos. Esses protocolos são amplamente

utilizados no desenvolvimento de softwares. A facilidade de integração e a leveza na transmissão de informação desses protocolos os tornam ótimas escolhas para a IoT.

Uma das mudanças de paradigma da Indústria 4.0 é a descentralização da lógica dos processos, tendo cada elemento autonomia sobre o que ele próprio deve fazer e quando, a modularidade e a interoperabilidade são outros pilares fundamentais nessa nova indústria.

1.1 Motivação

O sistema proposto mostra um novo ponto de vista de como pode ser realizada uma automação em uma indústria, e como ela pode estar sincronizada com outros sistemas. Tendo um melhor controle sobre a produção, detalhamento do sistema, facilidade na alteração do funcionamento, e um baixo custo de implementação.

Devido a pandemia da COVID-19, as atividades presenciais foram suspensas em diversas instituições brasileiras de ensino, visando à diminuição do risco de infecção pelo coronavírus. Essas medidas distanciaram os alunos das aulas praticas de Automação Industrial e Redes Industriais, neste contexto foram avaliadas se as ferramentas da indústria 4.0 seriam capazes de auxiliar nesse novo momento do ensino.

1.2 Objetivos

1.2.1 Objetivo Geral

O projeto tem como objetivo desenvolver uma nova forma de comunicação em relação aos sistemas eletropneumáticos, criando uma rede entre sensores e válvulas que utilizem o ZigBee e o MQTT como protocolo de troca de mensagens. Com propósito de facilitar a implementação de uma nova sequencia de movimentos ou na inclusão de elementos no sistema, se comparada aos métodos usados atualmente como Ladder e Grafcet. Esse sistema apresenta um caminho possível para a modernização da indústria.

1.2.2 Objetivos Específicos

- Estudar MQTT;
- Especificar a solução lógica do MQTT e do ZigBee;
- Implementar a solução fisicamente;
- Realizar ensaios e testes;
- Comparar resultados com as soluções Ladder e Grafcet.

2 Revisão Bibliográfica

Neste capítulo serão abordados os principais elementos do projeto, o sistema possui atuadores pneumáticos controlados por válvulas pneumáticas solenoide com retorno por mola. A posição dos atuadores é obtida através de chaves de fim de curso, e seu estado é enviado através do protocolo ZigBee para o dispositivo de comando de cada atuador, que por sua vez publica uma mensagem MQTT no broker, que gerencia as publicações e subscrições dos clientes.

2.1 Sistemas eletropneumáticos

Caracterizada como uma grande evolução e refinamento da pneumática, a eletropneumática tem como base a energia elétrica e a própria energia pneumática. A eletropneumática também é uma área relevante no meio industrial, seja no hospital com os equipamentos hospitalares e clínicos, na área automotiva ou em qualquer outra que compartilhe deste sistema eletropneumático.

A eletropneumática pode ser definida como uma tecnologia ligada à geração, controle e distribuição de potência, que faz uso de fluidos comprimidos/pressurizados como combustível.Ou seja, o sistema eletropneumático é uma fusão da eletricidade e da pneumática, duas grandezas essenciais para a área industrial, permitindo o controle de movimentos e forças através da interação dessas grandezas.

> O termo pneumática é derivado do grego Pneumos ou Pneuma (respiração, sopro) e é definido como a parte da Física que se ocupa da dinâmica e dos fenômenos físicos relacionados com os gases ou vácuos. É também o estudo da conservação da energia pneumática em energia mecânica, através dos respectivos elementos de trabalho.(PARKER TRAINING, 2005, p. 4)

Vantagens na utilização de sistemas pneumáticos:

- Incremento da produção com investimento relativamente pequeno.
- Redução dos custos operacionais.
- Robustez dos componentes pneumáticos.
- Facilidade de implantação.
- Simplicidade de manipulação.

Limitações no uso:

- ar comprimido necessita de uma boa preparação para realizar o trabalho proposto como, remoção de impurezas e eliminação de umidade para evitar corrosão nos equipamentos.
- Velocidades muito baixas são difíceis de ser obtidas com o ar comprimido devido às suas propriedades físicas.
- O ar é um fluido altamente compressível, portanto, é impossível se obterem paradas intermediárias e velocidades uniformes.
- O ar comprimido é um poluidor sonoro quando são efetuadas exaustões para a atmosfera. Esta poluição pode ser evitada com o uso de silenciadores nos orifícios de escape.

2.1.1 Válvulas

As válvulas servem basicamente, para orientar fluxos de ar, impor bloqueios e controlar suas intensidades de vazão ou pressão. Uma possível classificação de acordo com o seu tipo de uso seria a seguinte:

- Válvulas de controle direcional.
- Válvulas de bloqueio.
- Válvulas de controle de fluxo.
- Válvulas de controle de pressão.

As válvulas necessitam de um agente externo ou interno para deslocar as suas partes internas de uma posição para outra, ou seja, que altere as direções do fluxo, efetue bloqueios e liberação de escapes. Os elementos responsáveis por tais alterações são os acionamentos internos, que podem ser classificados em comando direto ou indireto.

Comando Direto é quando a força de acionamento atua diretamente sobre o mecanismo que causa a inversão da válvula. Comando Indireto é quando a força de acionamento atua sobre um dispositivo intermediário, que libera o comando principal, que por sua vez inverte a válvula. Estes comandos são chamados de combinados ou servo.

Podemos definir o acionamento das válvulas em quatro tipos. Acionamento manual pode ser realizado por botão, pedal ou alavanca. Acionamento mecânico é realizado, por meio de botão, rolete, mola ou intertravamento mecânico. Acionamento pneumático é realizado por pressão, pressão diferencial ou baixa pressão. Acionamento elétrico normalmente é realizado por meio de um eletroímã. Na Figura 2 temos uma relação dos acionamentos presentes nas válvulas.

Os símbolos de acordo com ISO 1219	Descrição	Tipos de acionamentos
	Acionamento por força muscular em geral	Acionamento por força muscular
	Operação de força muscular com o botão de pressão e empurrar	
	Operação de força muscular com o botão de pressão	
	Operação de força muscular com o botão de empurrar	
	Acionamento por força muscular pela alavanca	
	Acionamento por força muscular pelo pedal	
	Pedal com duas direções de acionamento	
	Acionamento por êmbolo	Acionamento por força mecânico
	Acionamento por rolete	
	Acionamento por alavanca com roldana	
	Acionamento por mola	
	Acionamento pneumática em geral	Acionamento por pressão
	Acionamento por pressão positivo e indireta	
	Acionamento pneumática indireto e servo-pilotado	
	Acionamento combinado servo-pilotado ou manual auxiliar	
	Acionamento por solenoide	Acionamento elétrico
	Acionamento por solenoide e servo pilotada	
	Acionamento por motor elétrico	

Figura 2 – Formas de acionamento.

Fonte: https://learnchannel-tv.com/pb/pneumatics/way-valves/actuation-according-iso-1219/

2.1.1.1 Válvula Globo

Conhecida também como Válvula Globo de Pistão é uma peça com excelente vedação indicada para regulagem do fluxo de líquidos viscosos, corrosivos ou que tendam a se solidificar. Formada por um disco móvel e um anel fixo, geralmente em um corpo esférico, a válvula globo é utilizada na vazão de tubulações

Figura 3 – Válvula Globo.

Fonte: https://wmvalvulas.com.br/cinco-tipos-valvulas-controle/

2.1.1.2 Válvula de Retenção

Indicada para uma ampla faixa de temperaturas e pressão, a válvula de retenção é um mecanismo automático que age sobre seu próprio peso e pressão média. Ela bloqueia o fluxo reverso, por isso também é conhecida como válvula de retorno.

Figura 4 – Válvula de Retenção.

Fonte: https://wmvalvulas.com.br/cinco-tipos-valvulas-controle/

2.1.1.3 Válvula Esfera

Recebe esse nome pois é formada por uma esfera vazada que libera o fluxo na tubulação quando está na posição aberta. A válvula esfera é uma válvula manual, e uma das mais utilizadas em ambientes residenciais. É também muito utilizada na indústria petroleira por sua aplicabilidade em ambientes corrosivos, de alta pressão e temperatura.

Figura 5 – Válvula Esfera.

Fonte: https://wmvalvulas.com.br/cinco-tipos-valvulas-controle/

2.1.1.4 Válvula Gaveta

A válvula gaveta recebe esse nome porque abre-se levantando uma porta ou gaveta para fora da estrutura do fluxo a ser controlado. A base de vedação da gaveta é plana. Projetadas para funcionarem totalmente abertas ou totalmente fechadas, por isso são equipamentos extremamente precisos.

Figura 6 – Válvula Gaveta.

Fonte: https://wmvalvulas.com.br/cinco-tipos-valvulas-controle/

2.1.1.5 Válvula Borboleta

A válvula borboleta é um modelo muito comum e de funcionamento simples, sendo facilmente operadas manualmente através de uma alavanca. Indicada para gases e líquidos de baixa pressão, é uma das válvulas mais simples de serem operadas. A válvula borboleta recebe esse nome pelo formato, além de ser leve e de baixo custo.

Figura 7 – Válvula Borboleta.

Fonte: https://wmvalvulas.com.br/cinco-tipos-valvulas-controle/

2.1.1.6 Vávluvlas direcionais

As válvulas direcionais possuem diferentes números de posições, que é a quantidade de manobras distintas que uma válvula direcional pode executar ou permanecer sob a ação de seu acionamento. As válvulas direcionais são sempre representadas por um retângulo, que por sua vez é dividido em quadrados, onde cada um representa uma posição possível da válvula.

Número de vias é o número de conexões de trabalho que a válvula possui, são consideradas como vias a conexão de entrada de pressão, conexões de utilização e conexão de escape. As setas indicam a interligação interna das conexões, mas não necessariamente o sentido de fluxo.

Válvulas Direcionais de Cinco Vias e Duas Posições (5/2) são válvulas que possuem uma entrada de pressão, dois pontos de utilização e dois escapes. Estas válvulas também são chamadas de 4 vias com 5 orifícios, devido à norma empregada. É errado denominá-las simplesmente de válvulas de 4 vias. Um válvula de 5 vias realiza todas as funções de uma de 4 vias. Fornece ainda maiores condições de aplicação e adaptação, se comparada diretamente a válvula de 4 vias, principalmente quando a construção é do tipo distribuidor axial. Existem aplicações que uma válvula de 5 vias sozinha pode executar e que quando feitas por uma válvula de 4 vias, necessitam do auxílio de outras válvulas.

A válvula pode ter uma posição padrão, uma das formas de garantir essa posição é com o uso de molas, na Figura 8 podemos ver a simbologia de um exemplo que garante essa posição padrão e é acionada eletricamente por um solenoide.

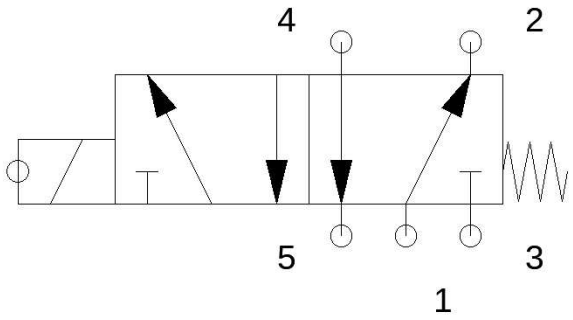


Figura 8 – Exemplo de válvula com 5 vias e 2 posições acionamento por simples solenoide e retorno por mola.

Fonte: Própria do autor

2.1.2 Atuadores

Os atuadores pneumáticos são os elementos responsáveis por transformar em trabalho a energia contida no ar comprimido, eles podem ser divididos em lineares, rotativos ou oscilantes.

Os cilindros se diferenciam entre si por detalhes construtivos, em função de suas características de funcionamento e utilização. Basicamente existem dois tipos, o de simples ação e o de dupla ação.

No cilindro de dupla ação o funcionamento pode ser descrito da seguinte maneira.

> O ar comprimido é admitido e liberado alternadamente por dois orifícios existentes nos cabeçotes, um no traseiro e outro no dianteiro que, agindo sobre o êmbolo,provoca os movimentos de avanço e retorno. Quando uma câmara está admitindo ar a outra está em comunicação com a atmosfera. Esta operação é mantida até o momento de inversão da válvula de comando; alternando a admissão ar nas câmaras, o pistão se desloca em sentido contrário.(PARKER TRAINING, 2005, p. 44)

Na Figura 9 podemos observar como se da tal funcionamento. O atuador de dupla ação é o tipo mais comum de utilização, por poder ser usado tanto no avanço quando no retorno para realização de trabalho.

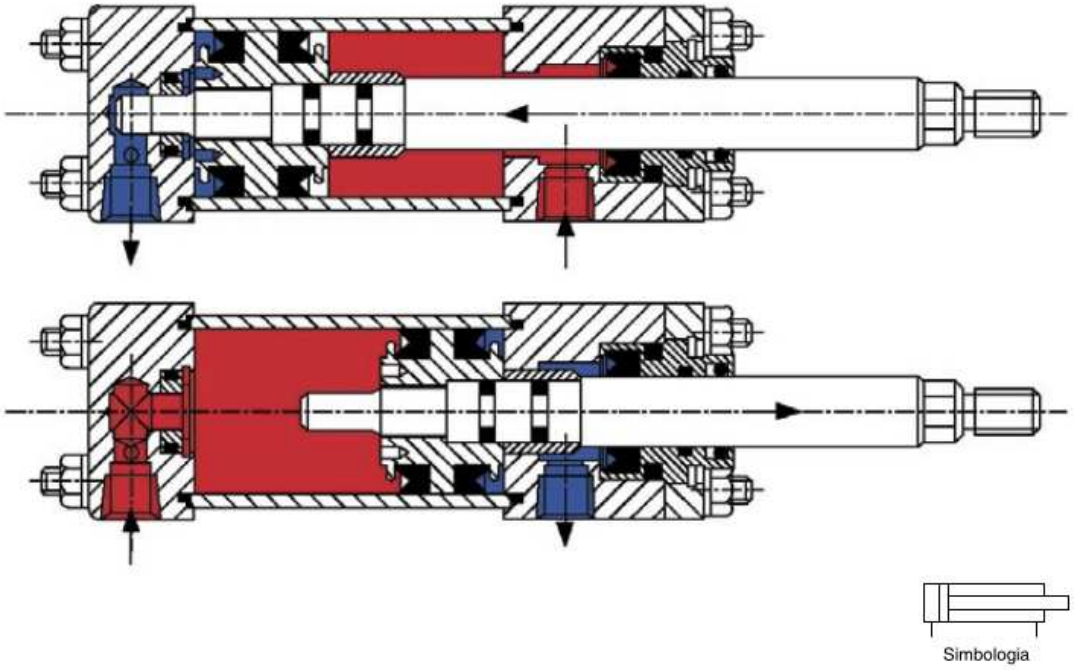


Figura 9 – Corte de atuador de dupla ação, funcionamento e simbologia

Fonte: (PARKER TRAINING, 2005)

2.1.3 Chaves fim de curso

As chaves fim de curso são comutadores elétricos de entrada de sinais, acionados mecanicamente são geralmente posicionadas no decorrer do percurso de cabeçotes móveis de máquinas e equipamentos industriais, bem como nas hastes de cilindros hidráulicos ou pneumáticos. Elas possuem 3 contatos, um borne comum, um fechado e um aberto e ao ser acionado comuta o sinal elétrico entre o contato aberto ou fechado ao contato comum.

Figura 10 – Chave fim de curso e simbologia

Fonte: (PARKER TRAINING, 2005)

2.1.4 Grafcet e Ladder

O Grafcet, também conhecido como SFC *(Sequential Functional Charts)*, é uma representação gráfica da parte de comando de um sistema automatizado, e é utilizado para desenvolver acionamentos sequenciais ou dependentes do tempo. Atualmente o Grafcet é adotado por alguns fabricantes de CLP como linguagem direta de programação. Para CLPs que não possuem essa característica, o Grafcet pode ser traduzido para Ladder tornando-se assim uma ferramenta para elaboração de comandos sequências. Os elementos de um Grafcet são as etapas, transições, arcos, receptividade, ações e regras de evolução.

Uma etapa é um estado no qual o comportamento do circuito não se altera independente da entrada ou saída, em um instante ela pode estar ativa ou inativa. A etapa inicial é a que se torna ativa após o funcionamento do sistema.

Transição ligam as etapas e são representadas por traços nos arcos orientados, duas etapas nunca podem ser conectadas diretamente entre si. Uma transição pode estar válida ou não, ela esta válida quando as etapas precedentes estiverem ativas.

Arcos orientados indicam a sequência do Grafcet pela interligação de uma etapa a uma transição e desta a outra etapa. O sentido convencionado é de cima para baixo, quando não for o caso, deve-se indicá-lo.

As ações representam os efeitos sobre os mecanismos controlados em uma determinada situação, e também representa ordens de comando Uma ação pode conter ordens de comando do tipo: contínua, condicional, memorizada, com retardo, limitada no tempo e impulsional.

Receptividade é a função lógica combinacional associada a cada transição. Quando em estado lógico verdadeiro, irá habilitar a ocorrência de uma transição válida. Uma receptividade também pode estar associada ao sentido de comutação de uma variável lógica, seja pela borda de subida, seja pela borda de descida.

Na Figura 11 podemos ver um Grafcet com as indicações de cada elemento.

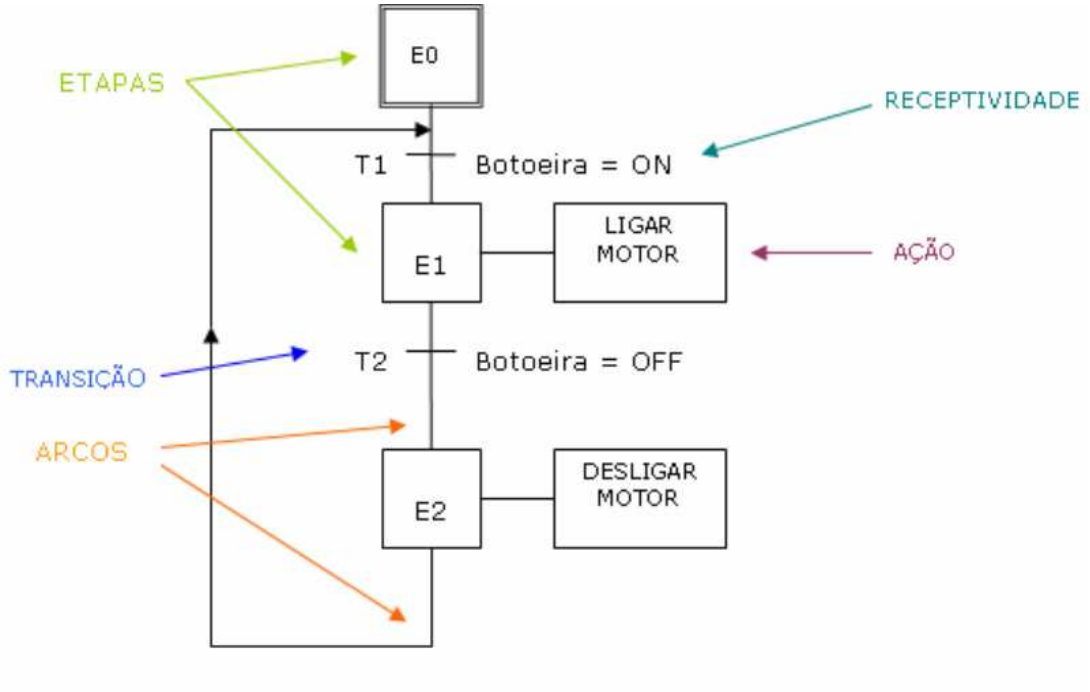


Figura 11 – Indicação dos elementos de um Grafcet.

Fonte: http://ftp.demec.ufpr.br/disciplinas/TM265/GRAFCET_utfpr_iec_848.pdf

A linguagem de programação Ladder é uma ferramenta gráfica usada para desenvolver programas ou softwares para CLPs (Controladores Lógico Programáveis).

Suas funções são de fácil compreensão por aqueles que têm noção sobre circuitos elétricos, pois são baseadas na lógica de relés e bobinas. É formada por circuitos horizontais em que a bobina fica na extremidade direita e com a alimentação feita por duas barras laterais verticais. Cada linha horizontal é uma sentença lógica, tendo os contatos como entradas.

Entradas ou contatos, que fazem a leitura do valor de uma variável booleana. Podem ser NA, que fica fechado quando a variável associada é verdadeira, do contrário, fica aberto; ou NF contato que está aberto quando a variável associada é verdadeira, ou fica fechado, caso contrário;

Saídas ou bobinas, que escrevem o valor de uma variável booleana ou ativam uma memória. Podem ser simples, set ou reset.

Blocos funcionais, que possibilitam funções avançadas, como contadores e temporizadores. Na Figura 12 podemos ver um exemplo de um circuito ladder.

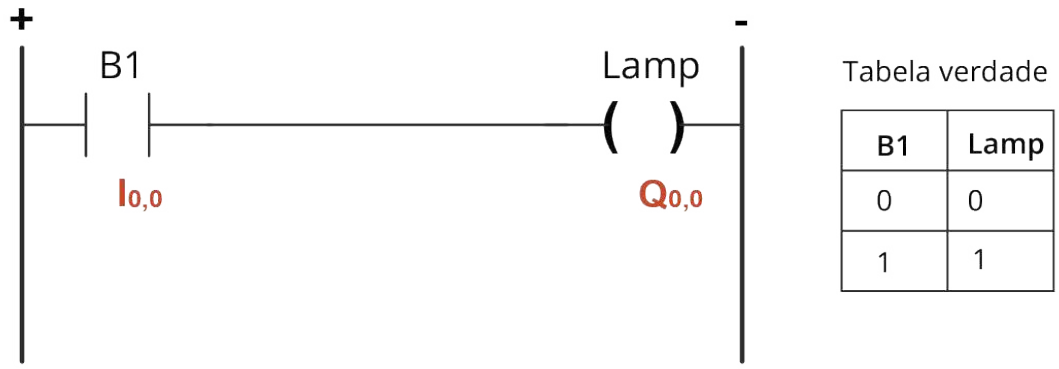


Figura 12 – Exemplo de Ladder .

Fonte: https://materialpublic.imd.ufrn.br/curso/disciplina/1/60/2/11

2.2 Internet of things (IoT)

A Internet das Coisas (do inglês Internet of Things (IoT)) emergiu dos avanços de várias áreas como sistemas embarcados, microeletrônica, comunicação e sensoriamento.

A IoT descreve a rede de objetos físicos do dia a dia a internet, com o objetivo de trocar dados com outros dispositivos e sistemas. Esses dispositivos variam de objetos comuns a ferramentas industriais sofisticadas. Embora a ideia de IoT já exista há muito tempo, como é discutida em (AZZARA et al., 2012) uma coleção de avanços recentes em diversas tecnologias diferentes tornou-a prática, como o acesso a tecnologia de sensores de baixo custo e de baixa potência, melhoria na conectividade com uma série de protocolos de rede para a internet, plataformas de computação em nuvem, entre outras.

IIoT significa Internet das Coisas Industrial, termo usado para dispositivos conectados nas áreas de manufatura, energia e em outras práticas industriais. A IIoT é importante por levar mais automação e automonitoramento a máquinas industriais, o que ajuda a aumentar a eficiência.(RED HAT, 2019)

A IIot coincide muito com os conceitos da indústria 4.0, essas ideias incluem interconectividade, transparência nas informações e tomadas de decisão descentralizadas. A IoT pode ser vista como a combinação de diversas tecnologias, as quais são complementares no sentido de viabilizar a integração dos objetos no ambiente físico.(SANTOS et al., 2016)

Com a diminuição do sensores, menor custo e mais inteligentes, abriu a possibilidade de instalação desses componentes nos mais variados produtos, como acessórios redes de transporte e energia, casas, dentre tantos outros lugares. Isso altera a maneira que gerenciamos nossas cadeias de produção, permitindo o monitoramento e e optimização de diversos setores.(SCHWAB, 2019)

2.3 ZigBee

O ZigBee é um protocolo para a criação de redes de sensores com baixo consumo energético, e necessita de uma menor largura de banda na sua operação, onde a comunicação é realizada utilizando radio frequência. Geralmente sendo usado em situações onde o gasto de energia do sistema é um fator importante. A comunicação desse protocolo é pequena se comparada com sistemas que transmitem grandes aquivos, como transmissões de vídeo. Cada dispositivo na rede transmite ou recebe uma única informação, a rede em si pode se restaurar caso haja algum problema em algum dos dispositivos sem a necessidade de intervenção. Isso faz com que seja uma rede com grande flexibilidade como é visto em(FALUDI, 2010).

Apesar dos nomes serem parecidos ZigBee e Xbee não são a mesma coisa. ZigBee é um padrão de protocolo de comunicação para dispositivos que consomem pouca energia, e utilizam de rede mesh, já Xbee é o módulo de rádio que suporta uma variedade de protocolos incluindo o ZigBee.

ZigBee é um padrão assim como o Bluetooth, e para garantir a interoperabilidade entre dispositivos de diferentes fabricantes o protocolo é construido sobre o conceito de camadas. O ZigBee é implementado sobre a norma IEEE 802.15.4 que define as camadas físicas (PHY) e de controle de acesso ao meio (MAC) para redes sem fios de baixo custo e baixa taxa de transferência. Entre as funcionalidades que o ZigBee implementa existem três que se destacam que são:

- Roteamento, que define como um rádio pode passar sua mensagem por outros dispositivos até chegar ao seu destino final.
- Criação de redes *Ad hoc*, é um processo automático que cria uma rede inteira de radios sem nenhuma intervenção humana.
- Auto recuperação é a capacidade que a rede tem de quando um dispositivo é perdido na rede, ela se reconfigura para reparar qualquer rota quebrada.

Toda rede ZigBee deve ter apenas um dispositivo coordenador. Toda rede precisa de ao menos 2 dispositivos, então pra qualquer rede ZigBee é necessário mais um outro dispositivo, que pode ser um roteador ou um dispositivo final. Muitas redes terão esses dois tipos e a maioria é maior que apenas dois dispositivos.(FALUDI, 2010)

O coordenar é responsável pela formação da rede distribuição de endereços e gerenciamento de outras funções que definem a rede, segurança, e o seu correto funcionamento.

O roteador é um nó com com diversos recursos, pode se juntar a uma rede já existente, enviar, receber e rotear informação. Ele age como entregador para outros dispositivos que estão muito distantes para trocar informações entre si.

O dispositivo final é essencialmente uma versão reduzida do roteador, ele pode se juntar a redes, enviar e receber informação. Então eles podem ter um hardware com menor potência e para economizar energia podem entrar em modo de repouso

As redes podem se conectar com *layouts* ou topologias diferentes e isso muda a estrutura da rede. A topologia define como os dispositivos vão se ligar um ao outro logicamente, os três principais são, pares, estrela ou malha.

A conexão em pares é simples, são apenas dois dispositivos conectados um ao outro onde um deve ser o coordenador. Na configuração em estrela o coordenador fica no centro da topologia rodeado de dispositivos finais, toda mensagem passa exclusivamente pelo coordenador. A topologia em arvore é pouco mais complexa, onde temos a presença de roteadores interligando os ramos da árvore, que começa no Coordenador. Há uma maior disponibilidade de caminhos e, portanto uma maior robustez, se comparado ao tipo estrela e existe apenas uma rota que liga o coordenador com cada dispositivo final. Na configuração em malha os roteadores são ligados de forma a criar diferentes rotas entre dispositivos. Os roteadores se comunicam, para realizar o roteamento das mensagens de forma a escolher a melhor rota entre dois dispositivos. Tais fatos garantem uma maior resiliência à rede, agora que há mais de um caminho entre dispositivos. Na Figura 13 podemos ver um exemplo dessas configurações.

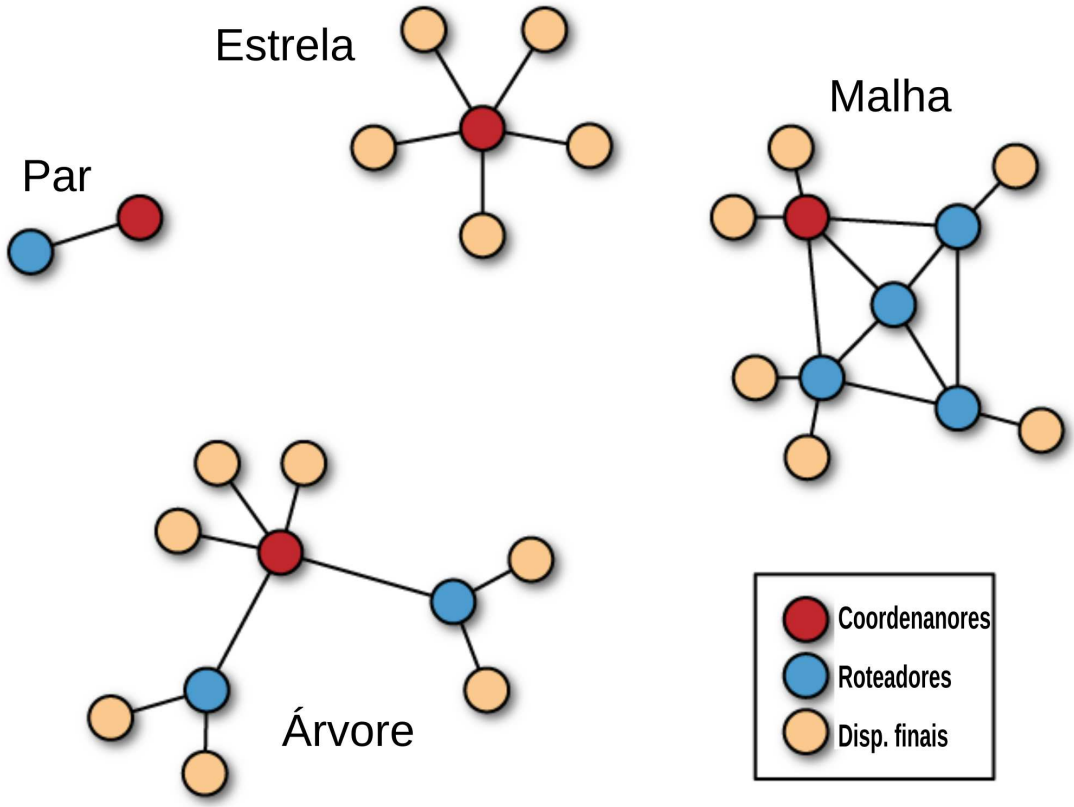


Figura 13 – Configurações de topologia para redes ZigBee

Fonte: (FALUDI, 2010)

Para enviar uma mensagem ZigBee é necessário o endereço do dispositivo de destino, todo módulo de rádio possui um endereço numérico de 64 bits, que é seu numero serial, ele é único em qualquer lugar. Quando um dispositivo ingressa em uma rede ele recebe um endereço do coordenador de 16 bits, esse endereço é único no escopo daquela rede. Cada módulo também pode receber um texto identificador de nó, mas não pode ser garantido a unicidade.

2.4 Message Queuing Telemetry Transport (MQTT)

MQTT é um protocolo de mensagens do tipo cliente servidor, publish/subscriber (publicador/subscritor) ele é leve, aberto, simples, e desenvolvido para ser facilmente implementado. Essas características o tornam ideal em ambientes restritivos, com banda ou troca de informação limitada, como M2M (machine to machine), e IoT.(STANDARD, OASIS, 2014).

Esse protocolo roda sobre a camada TCP/IP, e quando um dispositivo da rede deseja receber uma determinada informação, ele subscreve no tópico onde aquela informação é disponibilizada, fazendo uma requisição ao elemento que gerencia as publicações e subscrições, este elemento é o *broker*. Dispositivos que desejam publicar informações também o fazem através do *broker*.

As mensagens e MQTT são publicadas em tópicos, não existe uma configuração a ser feita no tópico apenas publicar nele já é o suficiente. Os tópicos tem uma hierarquia e usa a / como separador semelhante a um sistema de arquivos. Os clientes podem se subscreverem em um tópico específico, ou usar curingas. O curinga + tem a função de aceitar qualquer publicação naquele nível, enquanto o # aceita qualquer publicação abaixo daquele nível.(LIGHT, 2021)

Na Figura 14 podemos observar a arquitetura de funcionamento deste protocolo.

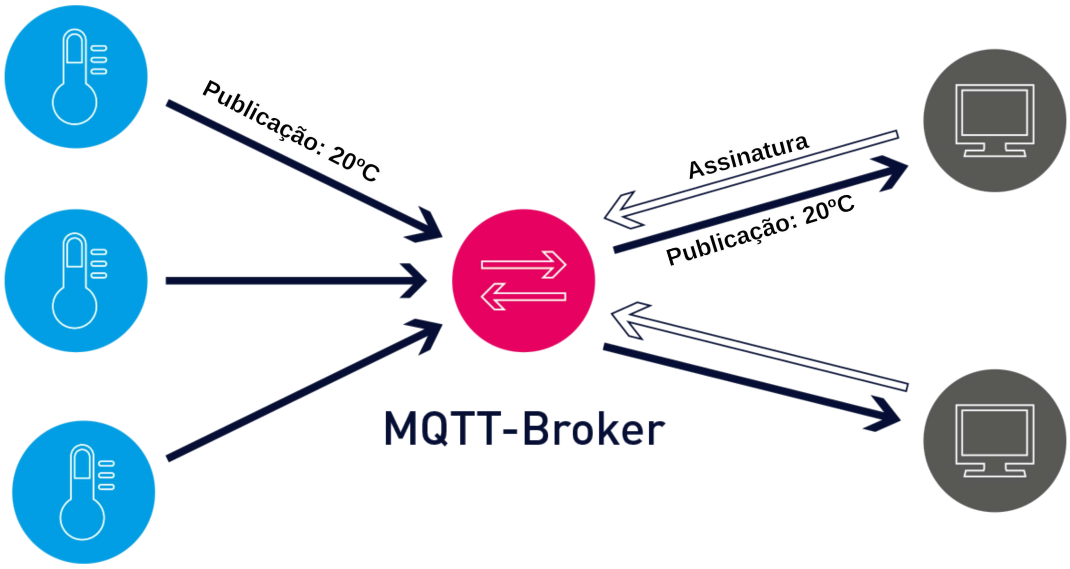


Figura 14 – Configurações de topologia para redes ZigBee

Fonte: Própria do autor

O MQTT define três níveis de QoS (*Quality of Service*). O QoS define como a mensagem será entregue. O cliente define com qual QoS a mensagem deve ser entregue a ele. Por exemplo, se uma mensagem é publicada com QoS 2 e um cliente se subscreve nesse tópico para receber a mensagem com QoS 0 a mensagem será entregue com QoS 0.

A QoS 0 é o que conhecemos como melhor esforço, e se assemelha ao protocolo de transporte UDP, a mensagem é enviada apenas uma vez, e pode ser recebida ou não. Neste tipo de entrega não há nenhuma confirmação de entrega ou recebimento. Na Tabela 1 podemos ver o fluxo do processo.

Tabela 1 – Diagrama fluxo QoS 0.

Ação do Remetente	**Fluxo do Pacote**	**Ação do destinatario**
Publish QoS 0		
	$\longrightarrow$	
		Mensagem entregue aos destinatarios adequados

A QoS 1 garante que a mensagem será entregue ao menos uma vez, no cabeçalho da mensagem tem um identificador do pacote. O remetente deve atribuir um identificador a cada nova mensagem publicada, deve enviar um pacote contendo esse identificador, deve tratar esse pacote como não reconhecido até receber uma resposta do destinatário. Esse identificador pode ser usado novamente após a confirmação de recebimento, o remetente pode enviar outros pacotes com identificadores diferentes enquanto espera uma resposta.

O destinatário deve responder ao emissor confirmando o recebimento do pacote enviado com um pacote PUBACK (*Publish acknowledgement*), após isso ele deve tratar qualquer nova publicação com mo mesmo identificador como nova. Na Tabela 2 podemos ver o fluxo do QoS 1.

Tabela 2 – Diagrama fluxo Qos 1.

Ação do Remetente	**Fluxo do Pacote**	**Ação do destinatario**
Grava Mensagem		
Publica QoS 1 (ID)	$\longrightarrow$	
		Processa a mensagem recebida
	$\longleftarrow$	Envia PUBACK (ID)
Descarta a mensagem		

A QoS 2 garante que a mensagem seja entregue exatamente uma vez, utiliza o conceito de 4 *handshake*. O remetente deve atribuir um identificador a cada pacote, deve enviar um pacote contendo esse identificador, deve tratar esse pacote como não reconhecido até receber uma resposta do destinatário, enviar um pacote PUBREL (*Publish release*) deve tratar esse pacote como não reconhecido até receber um PUBCOMP (*Publish complete*) do destinatário.

O destinatário deve responder a publicação com PUBREC (*Publish received*), até receber um PUBREL deve tratar qualquer pacote com o mesmo id enviando um PUBREC, isso evita a duplicação de mensagens, deve responder um PUBREL com um PUBCOMP, depois deve tratar qualquer publicação com o mesmo id como uma nova publicação o fluxo dessa aplicação pode ser vista na Tabela 3.

Tabela 3 – Diagrama fluxo Qos 2.

Ação do Remetente	**Fluxo do Pacote**	**Ação do destinatario**
Grava Mensagem Publica QoS 2 (ID)	$\longrightarrow$	
		Processa a mensagem recebida
	$\longleftarrow$	Envia PUBREC (ID)
Descarta a mensagem, grava PUBREC (ID) recebido PUBREL (ID)		
	$\longrightarrow$	
		descarta mensagem Envia PUBCOMP (ID)
	$\longleftarrow$	
Descarta estado gravado		

2.5 Placa Arduino

O arduino é uma plataforma de computação física *open source* (fonte aberta), que tem como base uma placa simples com diversas entradas e saídas, o arduino pode ser utilizado para desenvolver projetos interativos independentes, ou conectados a softwares(BANZI; SHILOH, 2011).

Simplificando o arduino é um computador minúsculo que você pode processar entradas e saídas entre o dispositivo e os componentes conectados a ele. Podemos nos referir a ele como um sistema embarcado, na maioria dos modelos a placa é composta por um processador Atmel AVR, um cristal oscilador, e um regulador de tensão de 5 volts. Dependendo do modelo as especificações podem ser diferentes(MCROBERTS, 2018).

Para a programação do arduino geralmente é utilizada sua própria IDE, que é um software livre e pode ser encontrada em `https://www.arduino.cc/en/software`, a linguagem usada é C/C++. A IDE permite que um código seja escrito e carregado para o arduino que após esse processo começará a executar as instruções que foram carregadas. O hardware e software são open source como mencionada anteriormente isso permite que possa ser usado livremente desde que respeite as condições de licença ao qual o projeto foi licenciado. De tal forma que nada impede que as pessoas possam produzir suas próprias placas, a única exigência feita pela equipe do arduino é não utilizar a palavra Arduino em outros projetos, pois esse nome é reservado a placa oficial.

O arduino pode ser estendido utilizando *shields* (escudos) que são placas de circuito que contém outros dispositivos como módulos de ethernet, módulos XBee entre outros. Esses *shields* podem ser conectados na parte superior do arduino, trazendo funcionalidades adicionais e estendendo os pinos até suas próprias placas para que ainda possa ser feito o acesso as entradas e saídas do arduino.

O arduino é uma ferramenta fácil para prototipagem rápida, o que permite um desempenho mais ágil na concepção de projetos até uma primeira versão que possa ser testada. Por ser *open source*, após o projeto ser validado pode-se definir apenas os recursos necessários e criar seu próprio sistema embarcado que atenda as especificações necessárias.

3 Desenvolvimento

Neste capítulo serão descritos o sistema, os equipamentos utilizados, bem como os softwares e configurações necessárias para realizar o projeto, para validação da comunicação.

3.1 Materiais e softwares utilizados

(a) Quatro placas Arduino UNO

(b) Quatro *shields* XBee

(c) Quatro módulos XBee S2

(d) Um adaptador XBee explorer USB

(e) Dois *shields* ethernet

(f) Quatro cabos USB A/B

(g) Uma bancada eletropneumática

(h) Dois atuadores de dupla ação

(i) Duas válvulas solenoides 5/2 vias retorno por mola

(j) Quatro sensores de fim de curso

(k) Dois módulos relé

(l) Um computador rodando sistema operacional Manjaro, que é uma distribuição baseada no archlinux com kernel linux na versão 5.4 ou superior.

Os softwares utilizados foram:

(a) Arduino CLI, é uma solução em linha de comando para carregar os códigos para o Arduino (Pode ser utilizado a IDE do arduino se preferível)

(b) XCTU, é uma aplicação multiplataforma que facilita a configuração e preparação dos módulos XBee.

(c) Eclipse Mosquitto, é um *broker open source* que implementa o protocolo MQTT nas versões 5.0, 3.1.1, e 3.1.

(d) MQTT Dash, é um aplicativo para sistemas android que implementa um cliente MQTT

3.2 Modelagem da arquitetura do sistema

O sistema desenvolvido se trata de uma aplicação eletropneumática, que deve realizar uma sequência de movimentos pré determinada usando conceitos de IoT. O sistema em questão é orientado a eventos discretos, pois nossos pontos de interesse se dão em troca de estados, e não precisamos manter uma coleta dos estados do sistema em função do tempo.

A sequência realizada seria A+ B+ A- B- como pode ser visto no diagrama trajeto passo na Figura 15.

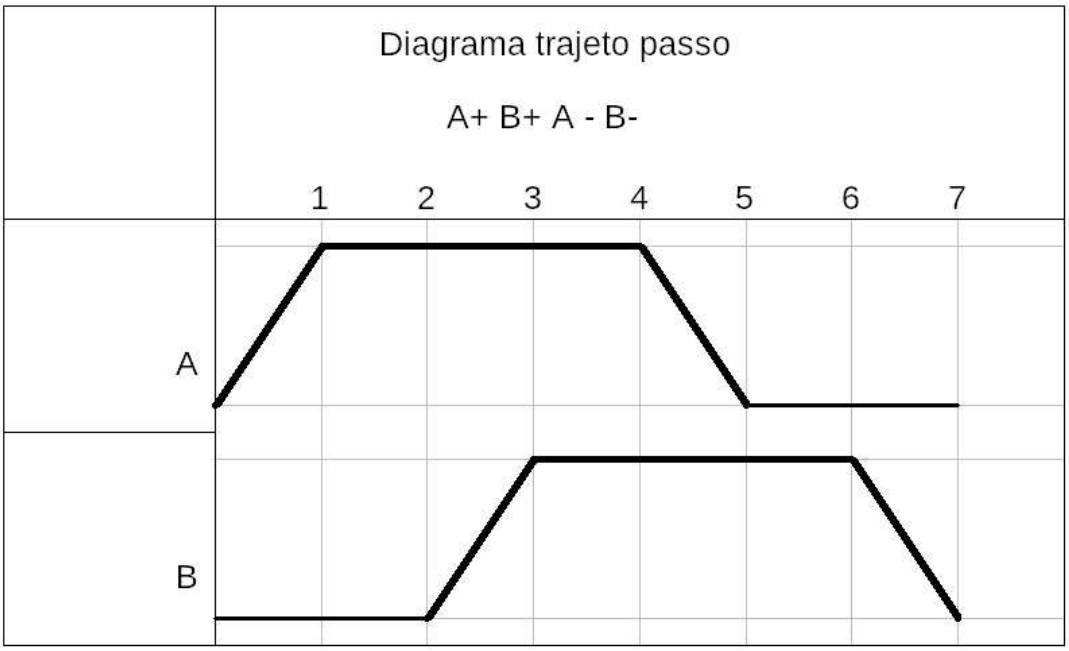


Figura 15 – Diagrama trajeto passo.

Fonte: Própria do autor

Na primeira formulação do sistema, todos os dispositivos na bancada trocariam informações apenas usando o protocolo ZigBee, e a publicação dos seus *status* no *broker* seria de responsabilidade de um *gateway* que teria como função receber essa mensagem no protocolo ZigBee e fazer a publicação utilizando MQTT-SN (*MQTT for Sensor Networks*).

O dispositivo onde estaria o *Gateway* receberia um adaptador USB para XBee, para ser possível essa comunicação. Após uma longa pesquisa e inúmeras tentativas de aplicação com essa configuração não se obteve o resultado esperado, não sendo possível empregar esse modelo. O conteúdo disponível sobre MQTT-SN se comparado ao MQTT é muito reduzido, o que foi um fator adicional a desistência desse modelo.

O segundo modelo proposto e o que de fato foi empregado nesse sistema, foi pensado como se cada elemento que estivesse associado a um determinado atuador fosse um conjunto distinto e que poderia estar em qualquer lugar, mas no caso específico se encontravam na

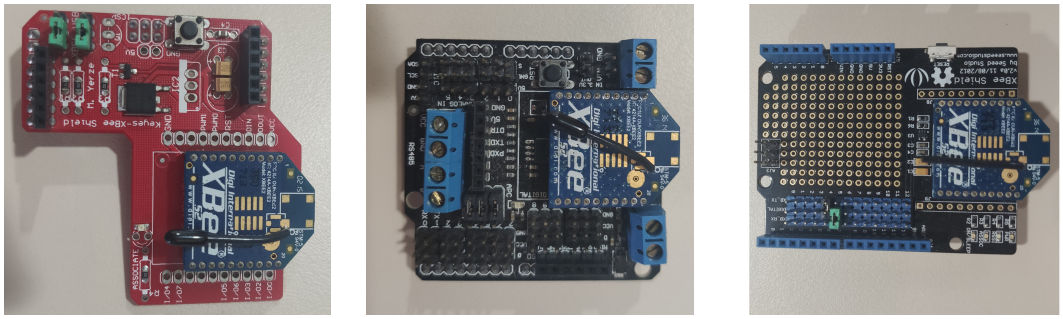
mesma bancada. Os sensores do conjunto eram gerenciados por um dispositivo e a válvula de controle do atuador era gerenciado por outro.

O dispositivo com os sensores era responsável pela aquisição dos dados e o envio utilizando ZigBee, enquanto o da válvula de controle era responsável pela leitura desses dados e da publicação deles no *broker*, então esse dispositivo precisava ser capaz de realizar comunicações MQTT e ZigBee.

Os dispositivos utilizados foram quatro Arduinos Uno, para a comunicação ZigBee foi utilizado um shield Xbee em conjunto com um modulo Xbee, enquanto a comunicação com o MQTT foi utilizado um *shield ethernet*. No Arduino responsável pela válvula então houve uma combinação com os dois *shields*.

3.2.1 ZigBee

Na Figura 16 podemos observar os *shields* Xbee que foram utilizados no sistema. Podemos observar que existem diversos modelos de *shields* XBee. Foram usados 2 *shields* da Figura 16a e um dos outros apresentados.



(a) Shield 1 Xbee. (b) Shield 2 Xbee. (c) Shield 3 Xbee.

Figura 16 – Modelos de *shields* Xbee utilizados.

Fonte: Própria do autor.

A configuração dos módulos Xbee foi feita no software XCTU, onde cada par de módulo estava em uma rede distinta, reforçando que cada atuador foi tratado como um equipamento diferente. Dentro do XCTU é possível configurar diversos aspectos da rede e do próprio modulo que está sendo configurado, como o id da rede o modo que o módulo ia operar. Para essa configuração é necessário conectar o módulo em um explorer Xbee, que é um adaptador USB para módulos Xbee, e conecta-ló a um computador com o software instalado.

Após a conexão com o software já iniciado é necessário uma varredura do sistema, onde o software procura por módulos que estejam conectados de acordo com os parâmetros definidos, como taxa de atualização e paridade. Quando o dispositivo for reconhecido basta adiciona-ló a lista de dispositivos e ele já poderá ser configurado.

Na Figura 17 podemos observar a tela de configuração dos módulos e os campos que podem ser alterados. Na página(DIGI, 2021) temos uma breve introdução do que se trata o software e em(MOUNTAINBAJA, 2018) existe um pequeno guia com as principais configurações que podem ser feitas.

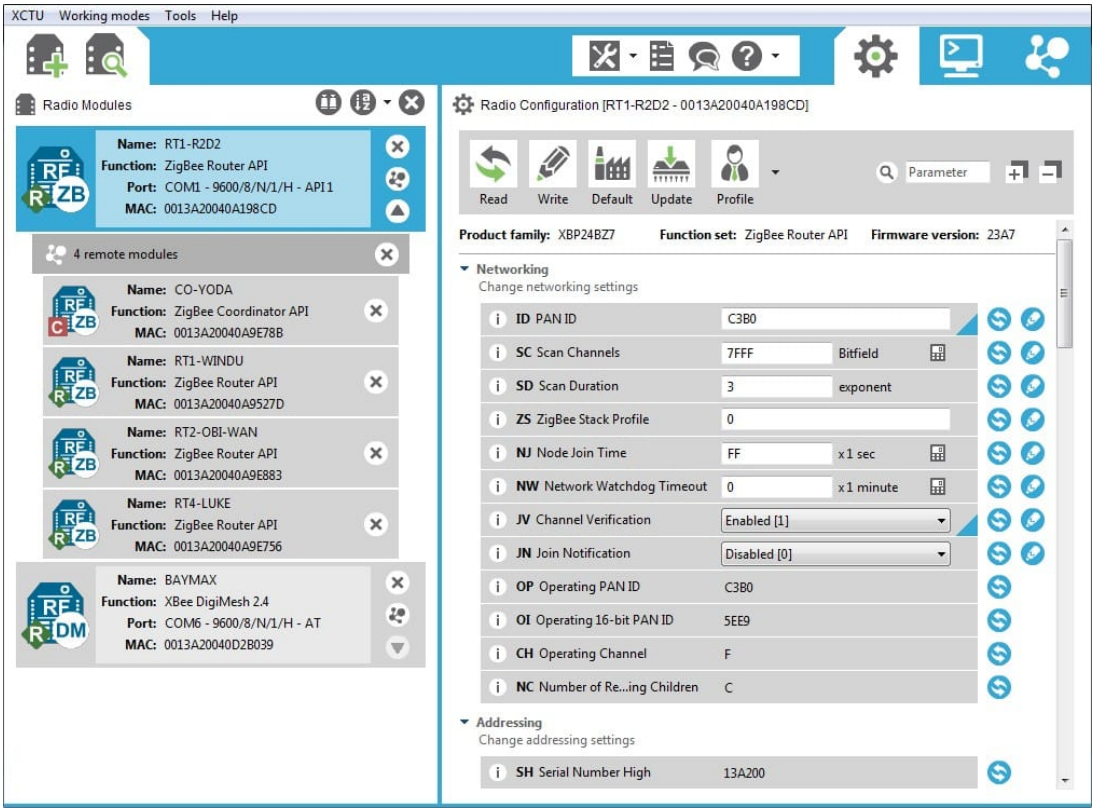


Figura 17 – Tela de configuração do XCTU.

Fonte: https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu

Entre as principais configurações que devem ser feitas vale ressaltar o campo ID PAIN ID que será o identificador da rede, lembrando que um dos módulos da rede Zigbee deve ser um coordenador. No sistema, o módulo que recebe essa atribuição esta conectado a válvula de controle.

O outro módulo na rede pode ser um *router* ou um *end point*, quando o módulo é configurado como *end point* ele pode ser ajustado para hibernar para economizar bateria dos dispositivos ao qual estão conectados. No projeto o módulo foi ajustado para ser um *router*.

A escolha dos *shields* diferentes foi feita com base nos *shields* disponíveis, e de forma que os dispositivos que fossem possuir *shields* combinados fossem compatíveis, e pudessem ser acoplados. Os módulos XBee em conjunto com os *shields* realizam uma comunicação serial com o Arduino, através de pinagens específicas, que são as mesmas para o carregamento dos códigos, então no momento de gravar um novo código no Arduino é necessário a remoção do modulo XBee do sistema, para que a operação possa ser

realizada com sucesso. No *shield* específico da Figura 16c essa pinagem pela qual é feita a comunicação serial pode ser alterada alternando a posição de *jumpers* no *shield*. Dessa forma não é necessário a remoção para o carregamento de um novo código e os pinos seriais podem ser usados para leitura e gravação de dados por um terminal do sistema, o que auxilia como ferramenta para debugar o código. Esse comportamento pode ser observado nos códigos que podem ser encontrado no Apêndice C.

3.2.2 Mosquitto

Mosquitto, é um *broker open source* que implementa o protocolo MQTT nas versões 5.0, 3.1.1, e 3.1, ele é leve e pode ser utilizado em diversas aplicações, e em sistemas com configurações de hardware mais modestas. Como dito anteriormente o *broker* é o elemento responsável pelo gerenciamento dos dispositivos na rede. Sendo suas funções, registrar tópicos criados, registrar as subscrições dos dispositivos e a entrega das mensagens.

Para instalar o mosquitto no sistema basta digitar o seguinte comando no terminal.

```
1    $ sudo pacman -S mosquitto
```

e quando solicitado informar a senha, quando o processo é concluído com êxito pode se rodar o *broker* apenas digitando mosquitto no terminal.

```
1    $ mosquitto
```

Na tela pode ser observado diversas configurações com as quais o mosquitto foi inicializado. Como a versão do *broker* que esta rodando, em qual porta o serviço esta sendo executado e outras informações.

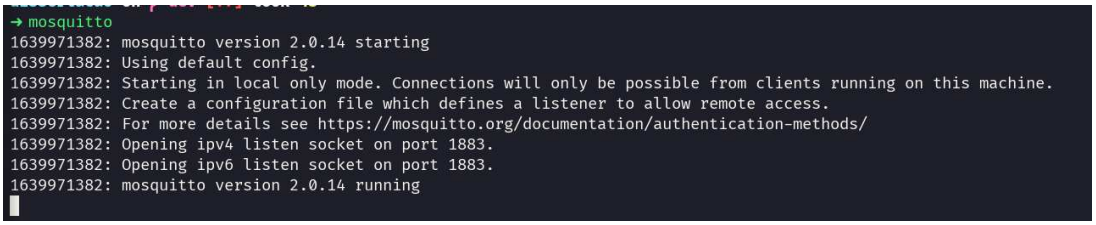


Figura 18 – Inicialização do mosquitto

No exemplo mostrado esta sendo executada a versão 2.0.14, em versões anteriores a 2.0 a opção que permitia conexões anônimas era habilitada por padrão, a partir da versão 2, esse comportamento deve ser configurado manualmente. Ao instalar o mosquitto um arquivo no caminho /etc/mosquitto/mosquitto.conf é criado onde possui toda a descrição e os parâmetros que podem ser modificados para determinar o funcionamento do *broker*. As únicas opções habilitadas foram a porta e a permissão de conexões anônimas, editando para que as linhas ficassem da seguinte forma.

```
1    listener 1883
```

```
2    allow_anonymous true
```

Para que o *broker* funcione com as alterações que foram feitas deve se informar o caminho com as configurações utilizando a *flag* -c. Dessa forma o comando para iniciar o *broker* ficou no seguinte formato.

```
1    $ mosquitto −c /etc/mosquitto/mosquitto.conf
```

Que nos mostra a seguinte saída confirmando que o arquivo de configuração foi carregado corretamente.

Figura 19 – Inicialização do mosquitto com o arquivo de configuração

Isso permite que os dispositivos possam se conectar ao *broker* sem a necessidade de nenhuma autenticação. O arquivo de configuração é extenso e podem ser habilitadas diversas funções de acordo com o sistema e com os dados que irão trafegar por essa rede, como autenticação, e mensagens criptografadas.

Juntamente com o pacote mosquitto vem implementados dois clientes, o *mosquitto_pub* e o *mosquitto_sub*, que são clientes para publicações e subscrições utilizando o protocolo MQTT, são ferramentas de grande utilidade durante o desenvolvimento, o manual dos clientes pode ser acessado da seguinte maneira.

```
1    $ man mosquitto_pub
```

```
1    $ man mosquitto_sub
```

Esses manuais vem junto com o pacote do mosquitto e neles são detalhados a forma de usar esses clientes, com a descrição de cada flag. Eles facilitam testes durante a criação dos seus próprios clientes.

3.2.3 Arduino

Como o Arduino foi criado para ser uma ferramenta de prototipação, ele engloba diferentes recursos, sendo inicialmente destinado para um uso geral. Ele possui uma pinagem com diferentes funcionalidades, existem 14 pinos digitais que podem ser usados

tanto como entrada ou saída que podem ser definidos via software, 6 pinos analógicos com 10 bits de resolução, pinagens de alimentação, e de comunicação.

Os pinos de maior interesse nesse projeto são os pinos digitais, por nosso sistema ter caracteristicas discretas. Dentre os 14 pinos digitais alguns possuem funções especializadas, os pinos 0 e 1 são usados para transmitir dados seriais o 0 é usado para receber (RX) e o 1 para enviar (TX) como os módulos XBee utilizam comunicação serial esses pinos em sua maioria no projeto são utilizados pelos *shields* e pelos módulos para a comunicação. Em *shields* mais simples esses pinos não são expostos, pois eles já ficam reservados para os módulos, como pode ser visto na Figura 16a.

Os pinos digitais podem ser configurados como *input* ou *output*, os pinos geralmente necessitam de pouca corrente para alterar seu estado, podendo ser lidos estados aleatórios no pino devido ao ruido elétrico do ambiente. Muitas vezes, é útil configurar um pino de entrada para um estado conhecido, se nenhuma entrada estiver presente. Isso pode ser feito adicionando um resistor de *pullup* (conectado ao + 5V), ou um resistor de *pulldown* (resistor ao terra) na entrada. Existem resistores de *pullup* de 20K incorporados na placa que podem ser acessados a partir do software. Esses resistores de *pullup* embutidos são acessados definindo *pinMode ()* como INPUT_PULLUP. Isso efetivamente inverte o comportamento do modo INPUT, onde HIGH significa que o sensor está desligado e LOW significa que o sensor está ligado. As entradas do nosso sistema utilizaram desse conceito para simplificar a montagem e ter entradas com leituras confiaveis. As entradas do sistema são as leituras das posições dos atuadores e do botão que inicia o processo.

Os pinos 2 e 3 podem ser configurados para tratar interrupções. Uma interrupção é um sinal enviado por um dispositivo de hardware que temporariamente interrompe a tarefa que a CPU está executando no momento, para que o dispositivo em si seja atendido. Logo após, o programa retoma seu processamento do ponto onde havia parado.

As interrupções são realizadas em nível de hardware, quando temos um evento externo sendo acionado, como por exemplo o pressionamento de um botão, e assim um sinal é enviado para o microcontrolador (ou microprocessador). Este processamento será executado imediatamente, interrompendo o que quer que a CPU esteja realizando no momento.

As boas práticas sugerem que as funções de interrupção sejam o mais breve possível, e essa abordagem foi empregada nos códigos. Ao utilizar as interrupções podemos garantir quando um sensor for acionado, uma outra alternativa seria empregar uma abordagem de sondagem onde iriamos verificar em intervalos de tempo o estado do botão, entretanto nessa abordagem o estado do pino pode sofrer alterações que não serão registradas pela sondagem e seria necessário sempre checar o estado do botão, o que não ocorre utilizando interrupções.

Quando empregamos interrupções, criamos uma (ou mais) função adicional, tecnicamente chamada de ISR *Interrupt Service Routine* (Rotina de Serviço de Interrupção). Esta função é invocada toda vez que uma interrupção é gerada no circuito e pode ser disparada dependendo do tipo de comportamento ao qual esta associada. Os comportamentos que podem ser monitorados são:

1. CHANGE Interrupção é disparada quando o pino muda de estado;
2. FALLING Interrupção é disparada quando o pino vai do estado HIGH para LOW;
3. LOW Interrupção é disparada quando o pino está no nível LOW;
4. RISING Interrupção é disparada quando o pino vai do estado LOW para HIGH.

Como nossas entradas são do tipo INPUT_PULLUP o modo de disparo utilizado foi o FALLING, que ativa o gatilho na borda de descida. Esse tipo de abordagem dispensa a constante verificação do pino, que é denominada polling e que pode não captar o acionamento dos sensores.

3.3 Funcionamento do sistema

Todos os dispositivos ao serem iniciados passam por um processo de configuração onde se conectam as redes necessárias. Como os módulos Xbee são configurados anteriormente através do XCTU, no momento da inicialização só é definida a taxa de transmissão e quais pinos serão utilizados.

Na configuração da comunicação ethernet devem ser definidos o mac e o ip que serão utilizados pelo Arduino, essa configuração será utilizada pelo cliente que ira gerenciar o protocolo MQTT.

O sistema possui uma sequência de funcionamento pré determinada, onde ao ser iniciado pelo pressionamento de um botão, ou ao receber uma mensagem MQTT com a mensagem de payload *payload* "on" que pode ser enviada de qualquer dispositivo que publique no tópico *Smartphone* o dispositivo acoplado a válvula faz com que o atuador A avance, ao acionar o sensor de fim de curso que indica que ele está avançado o Arduino responsável pelos sensores envia uma mensagem Zigbee para o dispositivo que esá conectado a válvula que controla deste atuador, que por sua vez publica essa informação no broker no tópico "Atuador_A" a mensagem "A+".

O atuador B que esta inscrito nesse tópico ao receber a mensagem dispara um timer, após 2 segundos o atuador B avança acionando o sensor de fim de curso na posição avançada enviando uma mensagem ZigBee para o dispositivo na válvula qu publica a mensagem no tópico "Atuador_B" a mensagem "B+".

O atuador A recebe a mensagem por estar escrito nesse tópico aguarda 2 segundos e recua, acionando o sensor de fim de curso na posição recuado enviando uma mensagem ZigBee do seu status e é postada no tópico "Atuador_A" a mensagem "A-".

Por fim o Atuador B recebe a mensagem aguarda os 2 segundos e recua, encerrando assim o ciclo do sistema. A forma como é feita esta troca de mensagens e acionamentos pode ser vista nos códigos fontes nos Apêndice A, Apêndice B, Apêndice C e Apêndice D

4 Resultados

Durante os experimentos iniciais não foi possível construir o sistema usando o protocolo MQTT-SN, devido a pouca documentação e de poucas soluções que implementam esse protocolo.

Após o desenvolvimento do sistema, dos testes e validações das comunicações com um *broker* externo na nuvem, a bancada foi montada fisicamente para os testes, onde se pode perceber que os dispositivos não conseguiam comunicação com um *broker* externo devido a configuração de rede da universidade, que bloqueava essa comunicação no *firewall*. Devido a esse bloqueio, a rede foi configurada localmente ligando todos os dispositivos em um *switch*.

Frente a essa limitação o setor responsável pelo gerenciamento de rede na universidade foi acionado para verificar a viabilidade do desbloqueio na rede para esse tipo de aplicação. Ao ser aprovada o desbloqueio, novas aplicações podem ser desenvolvidas com maior conectividade, tanto no laboratório quanto nas aulas.

Referente ao funcionamento do sistema, ele foi ativado 72 vezes utilizando o botão físico no dispositivo, e 46 vezes publicando a mensagem "on" no tópico *Smartphone* utilizando um celular. Em todas as vezes o sistema respondeu como o esperado, realizando toda a sequência corretamente.

Era esperado que o sistema nem sempre respondesse de forma correta, por ter sido utilizado o *QoS 0* que não possui nenhuma garantia de entrega, ou política de reenvio de mensagem. A causa mais provável desse comportamento se deve ao sistema ter sido configurado em uma rede local onde a perda de pacotes se torna mais improvável.

O sistema se mostrou robusto e escalável, ao ponto que um novo elemento pode entrar no sistema sem a necessidade de configurar fisicamente os antigos elementos. Sendo apenas necessário atualizar a lógica do sistema, e caso o novo elemento não interfira no antigo sistema, apenas ele precisará ser configurado, se inscrevendo nos tópicos de seu interesse. Se tornando mais simples neste aspecto se comparado as formas atuais de automação que utilizam CLP's , Grafcet e Ladder, além de um menor custo na implementação.

Essa abordagem facilita a forma como a telemetria de um sistema pode ser feita, apenas se inscrevendo nos tópicos que se tem interesse em monitorar. Os componentes utilizados e a montagem final do sistema podem ser vistos nas figuras a seguir.

Figura 20 – Atuador de dupla ação.

Fonte: Própria do autor.

Figura 21 – Sensores de fim de curso.

Fonte: Própria do autor.

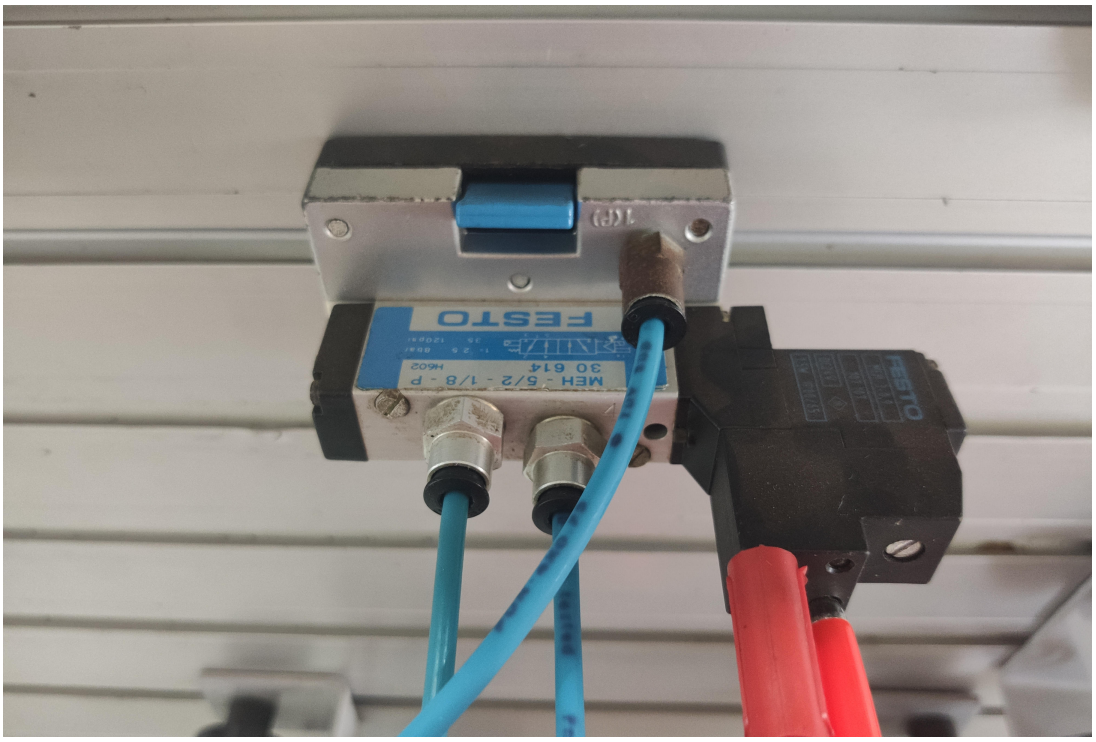


Figura 22 – Válvula 5/2 acionada eletricamente e retorno por mola.

Fonte: Própria do autor.

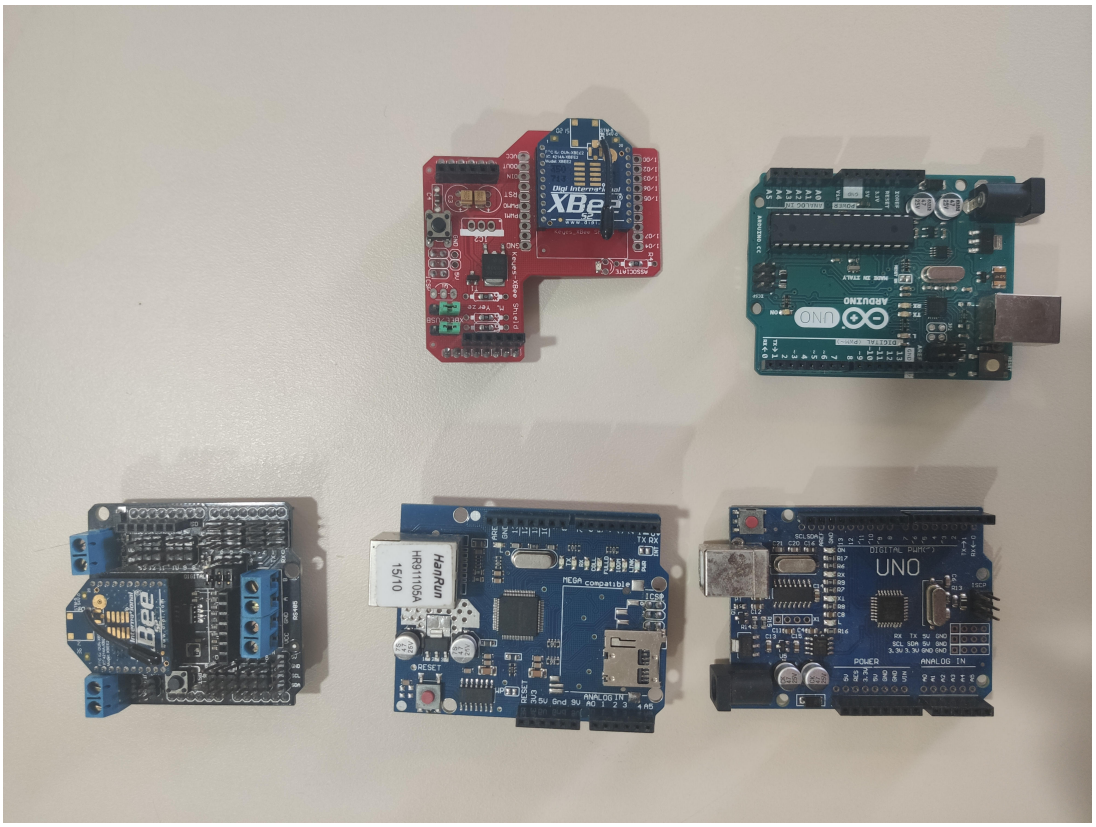


Figura 23 – Dispositivos do conjunto atuador, válvula e sensores.

Fonte: Própria do autor.

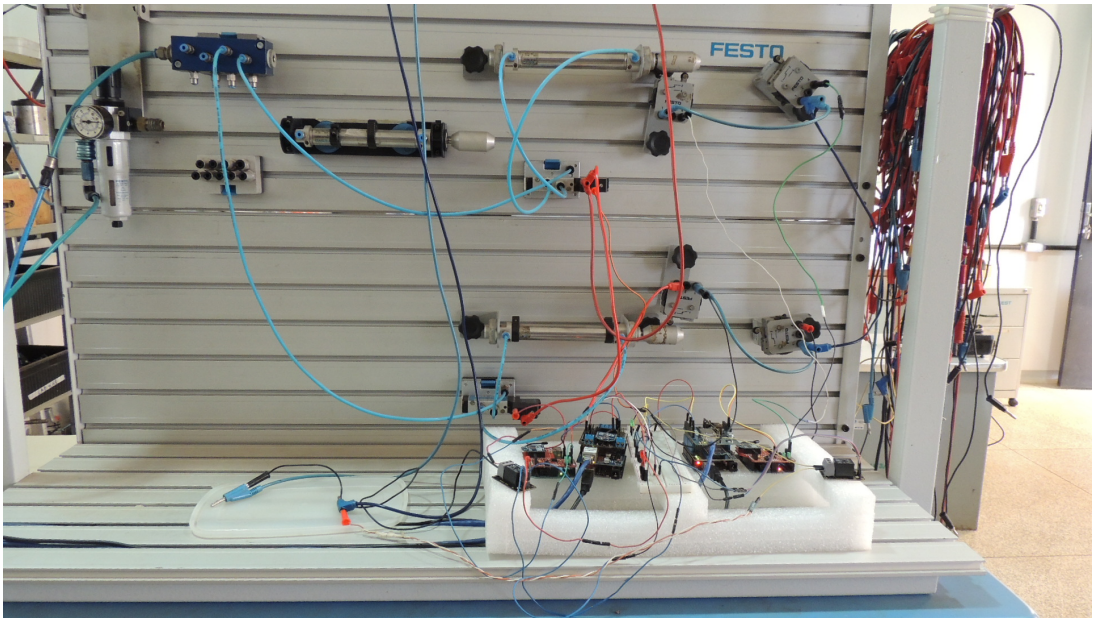


Figura 24 – Montagem completa do sistema.

Fonte: Própria do autor.

No instante inicial o sistema está em repouso, quando o sistema é energizado os dispositivos ingressam em suas redes ZigBee e assinam os tópicos de seu interesse no broke como pode ser vista na Figura 25.

Ao receber a mensagem “on ” no tópico Smartphone o sistema é iniciado, fazendo com que o atuador A avance. Ao acionar o sensor de fim de curso o dispositivo acoplado ao sensores envia uma mensagem ZigBee para a válvula, como exemplificado na Figura 26.

A válvula A publica no tópico, então a válvula do atuador B recebe essa mensagem, aguarda dois segundos e avança o atuador que aciona o sensor que troca mensagem com a válvula utilizando ZigBee como mostrado na Figura 27.

Então a válvula B publica a mensagem que é recebida pela válvula do atuador A aguarda 2 segundos recua o atuador, aciona o fim de curso do recuo e troca mensagem ZigBee, como mostrado na Figura 28.

Por fim o mesmo processo é repetido e então o atuador B recua finalizando o ciclo Figura 29.

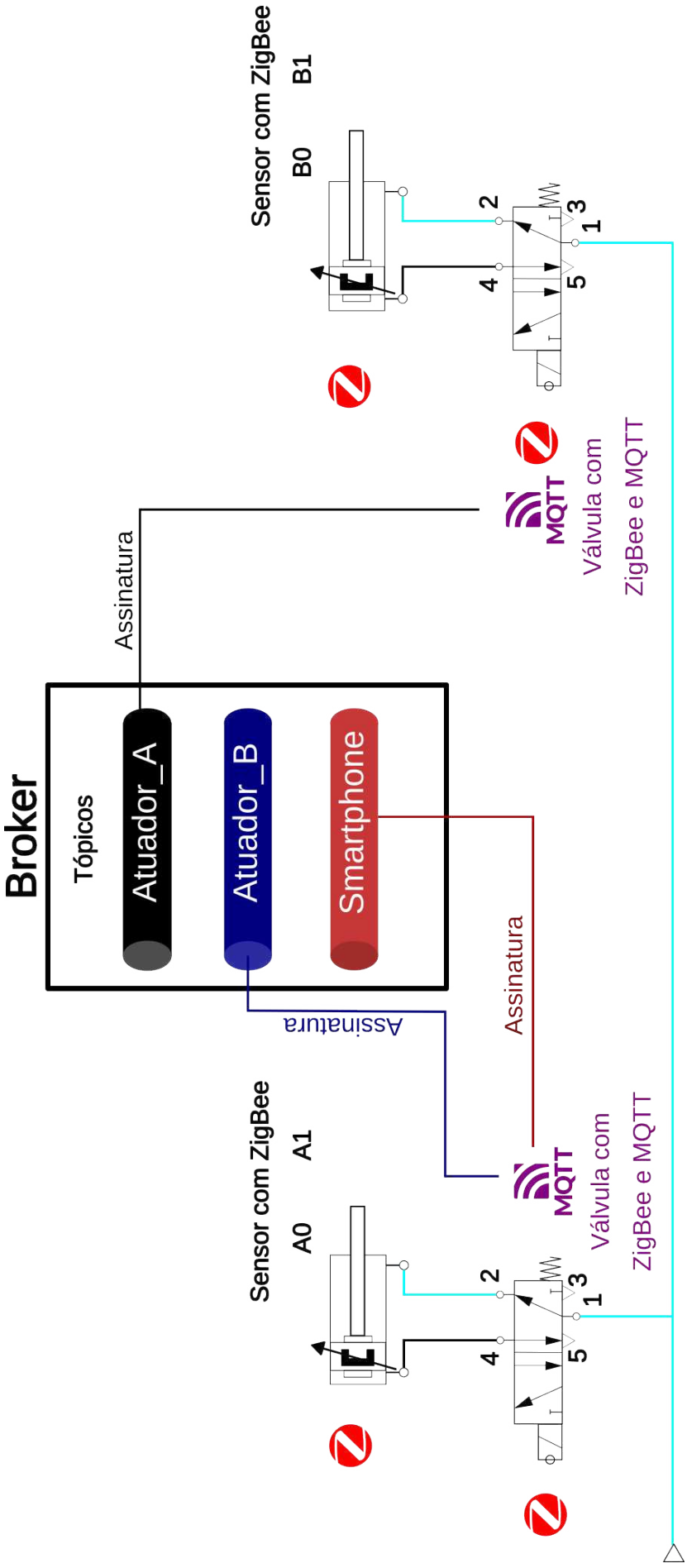


Figura 25 – Sistema inicial.

Fonte: Própria do autor.

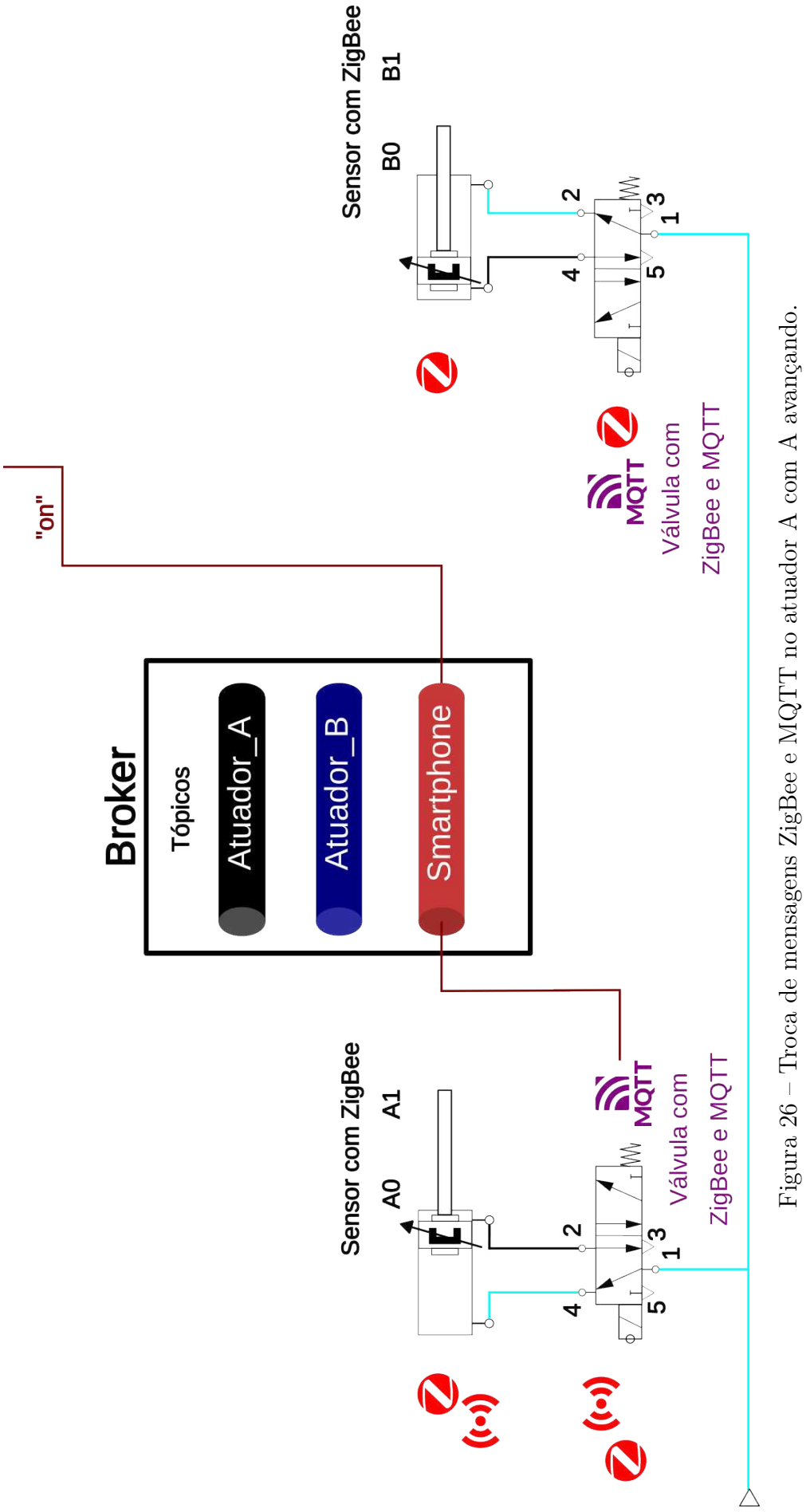


Figura 26 – Troca de mensagens ZigBee e MQTT no atuador A com A avançando.

Fonte: Própria do autor.

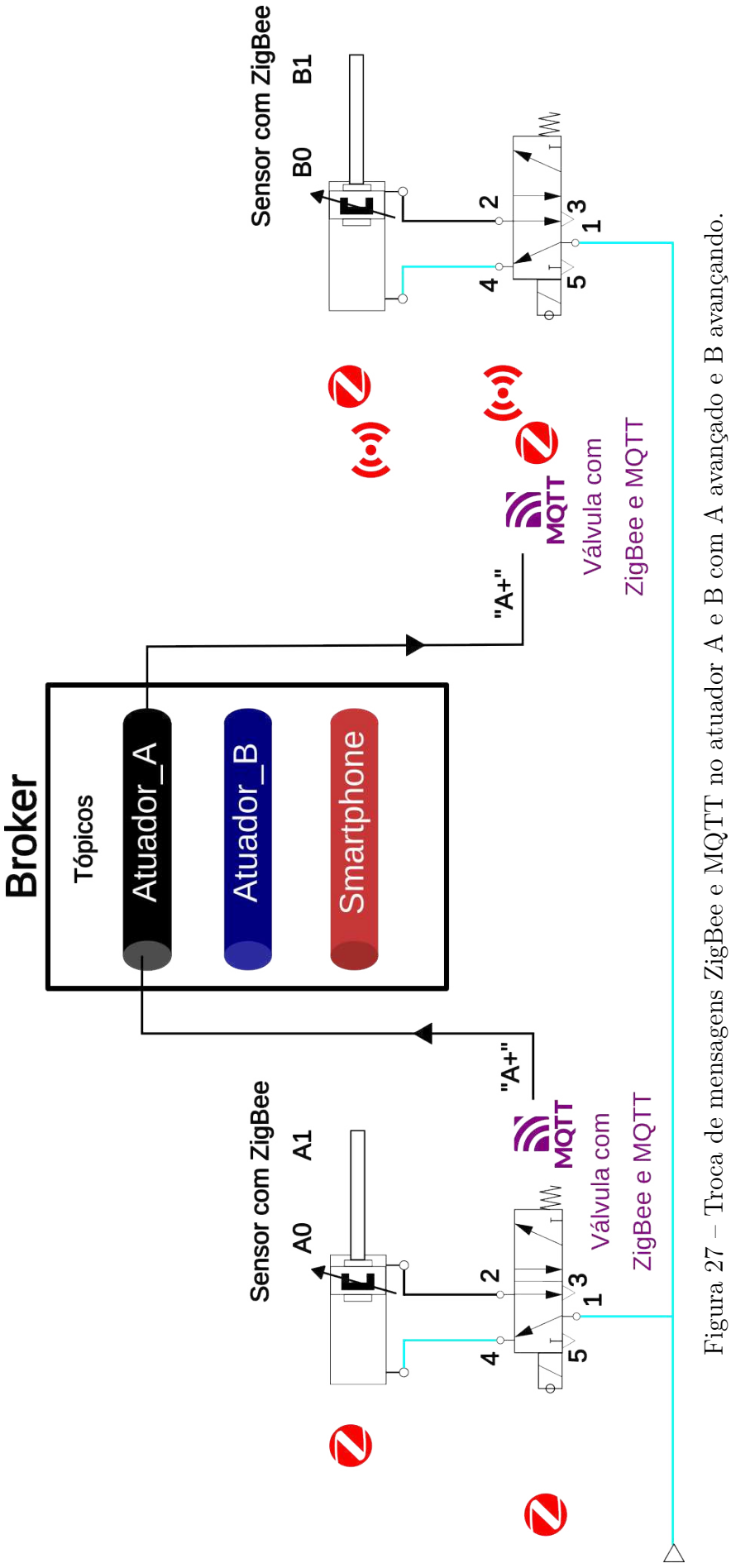


Figura 27 – Troca de mensagens ZigBee e MQTT no atuador A e B com A avançado e B avançando.

Fonte: Própria do autor.

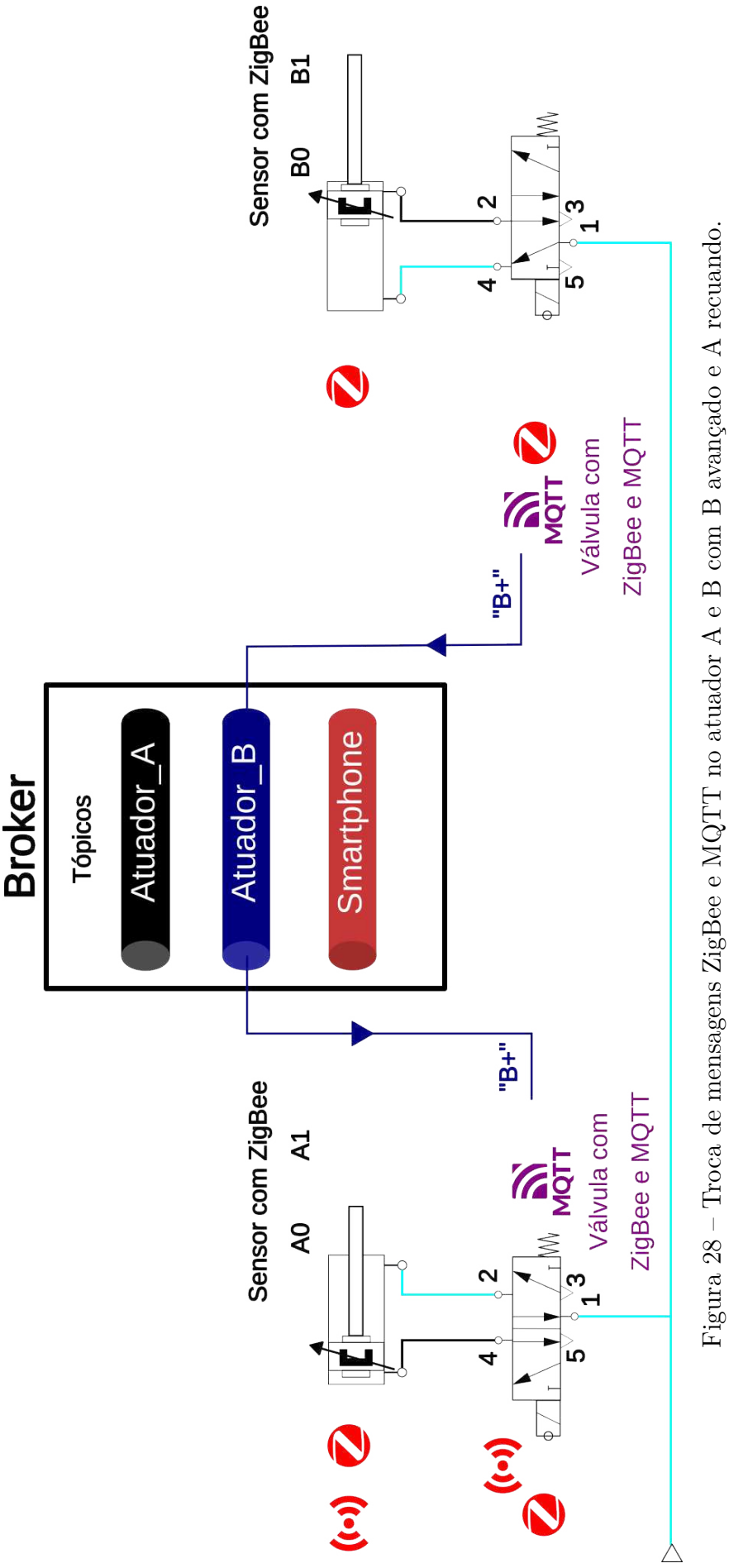


Figura 28 – Troca de mensagens ZigBee e MQTT no atuador A e B com B avançado e A recuando.

Fonte: Própria do autor.

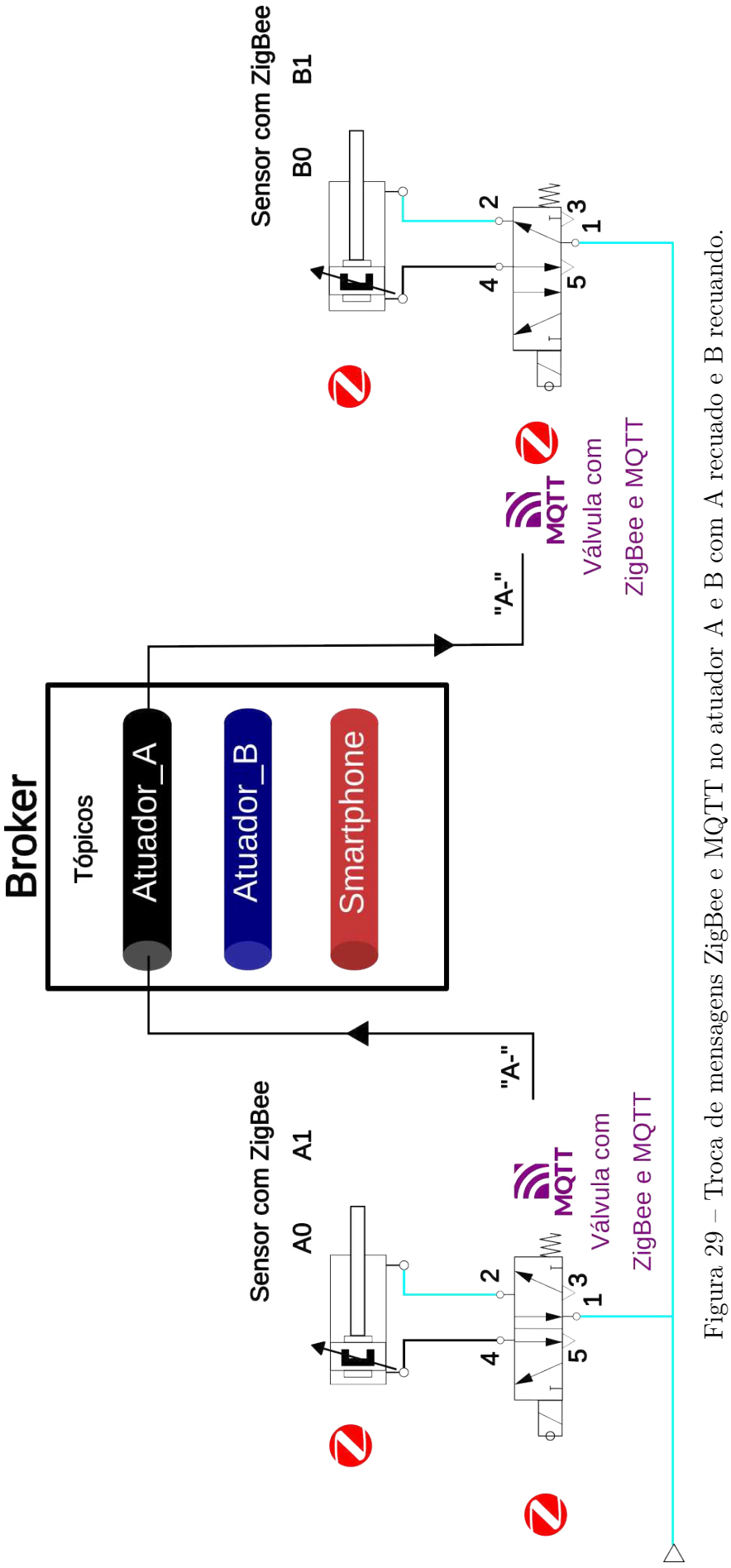


Figura 29 – Troca de mensagens ZigBee e MQTT no atuador A e B com A recuado e B recuando.

Fonte: Própria do autor.

As dificuldades encontradas durante este trabalho, foram a escassa documentação de algumas soluções como a do protocolo MQTT-SN, bibliotecas em desenvolvimento sem todas as possíveis utilidades implantadas dependendo da plataforma usada, como um exemplo a falta da implementação do QoS2 para o Arduino. A complexidade de implantação de um *broker* redundante, que é mencionado em trabalhos futuros. A montagem física de um dispositivo que pudesse receber e tratar os dois protocolos simultaneamente pois os *shields* nem sempre são compatíveis entre si, o que levou a cogitação da utilização de dispositivos que já possuíssem uma conexão com a internet como no caso do ESP32 que já possui um módulo wifi permitindo a conexão com a internet.

Apesar de se tratar de um sistema simplificado podemos comprovar que é possível que os protocolos MQTT e ZigBee trabalhem em conjunto em um mesmo sistema. Caso o sistema aumente de complexidade de forma significativa, seria necessário uma nova consideração dos hardwares especificados.

5 Conclusão

Este trabalho apresentou o desenvolvimento de uma solução para um sistema de automação baseado em IoT. O diferencial dessa solução é a integração entre a informática e as tecnologias de automação, permitindo a programação e a supervisão de processos de uma forma mais facilitada.

Os pontos fortes dessa solução são a flexibilidade, versatilidade de comunicação proporcionada pelos diferentes protocolos suportados, baixo custo de implementação, baixo consumo de banda, e ser baseada em código aberto tanto o hardware quanto o software, permitindo a personalização e expansão do sistema.

Os resultados apresentados demonstram o potencial das aplicações IoT e da indústria 4.0. O sistema foi capaz de se comunicar e gerenciar os dispositivos de forma eficiente e confiável no cenário analisado.

6 Trabalhos Futuros

Como possíveis trabalhos futuros, pode-se citar:

- Implementar redundância no broker;
- Implementar protocolo MQTT-SN;
- Expandir a comunicação para mais dispositivos;
- Avaliar o impacto do tamanho das mensagens;
- Avaliar a definição do hardware em função do local da aplicação;
- Enquadrar os dispositivos na norma ABNT NBR-IEC 60079.

Referências

AZZARA, Andrea et al. Architecture, functional requirements, and early implementation of an instrumentation grid for the IoT. In: IEEE. 2012 IEEE 14th International Conference on High Performance Computing and Communication & 2012 IEEE 9th International Conference on Embedded Software and Systems. 2012. P. 320–327. Disponível em: <https://doi.org/10.1109/HPCC.2012.50>. Citado 1 vez na página 26.

BANZI, Massimo; SHILOH, Michael. Primeiros passos com o Arduino. *São Paulo: Novatec*, p1, 2011. Citado 1 vez na página 33.

DIGI. *XCTU*. 2021. Disponível em: <https://www.digi.com/products/embedded-systems/digi-xbee/digi-xbee-tools/xctu>. Citado 1 vez na página 37.

FALUDI, Robert. *Building wireless sensor networks: with ZigBee, XBee, arduino, and processing*. "O'Reilly Media, Inc.", 2010. Citado 2 vezes nas páginas 27, 28.

LIGHT, Roger. *mqtt MQ Telemetry Transport*. 2021. Disponível em: <https://mosquitto.org/man/mqtt-7.html>. Citado 1 vez na página 30.

MCROBERTS, Michael. *Arduino básico*. Novatec Editora, 2018. Citado 1 vez na página 33.

MOUNTAINBAJA, Equipe. *Configurando o Xbee pelo software XCTU*. 2018. Disponível em: <https://portal.vidadesilicio.com.br/configurando-o-xbee-pelo-software-xctu/>. Acesso em: 6 dez. 2021. Citado 1 vez na página 37.

PARKER TRAINING. *Tecnologia Eletropneumática Industrial*. 2005. Citado 2 vezes nas páginas 16, 22, 23.

RED HAT. *O que é IIoT*. 2019. Disponível em: <https://www.redhat.com/pt-br/topics/internet-of-things/what-is-iiot?>. Acesso em: 7 set. 2021. Citado 1 vez na página 26.

SANTOS, Bruno P et al. Internet das coisas: da teoria à prática, 2016. Citado 1 vez na página 26.

SCHWAB, Klaus. *A quarta revolução industrial*. Edipro, 2019. Citado 2 vezes nas páginas 14, 26.

STANDARD, OASIS. MQTT version 3.1. 1. *http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.pdf*, v. 1, 2014. Citado 1 vez na página 30.

Apêndices

APÊNDICE A – Codigos fonte sensor atuador A.

```
#include <SoftwareSerial.h>
#include <XBee.h>

// Cria um objeto XBee e o payload que será enviado
XBee xbee = XBee();
char payload[1] = {0};

// Define o endereço do XBee recebedor
// Cria o objeto ZigBee que será enviado
// Cria um objeto para receber monitorar o status do pacote
XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x40c1b208);
ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload)
    );
ZBTxStatusResponse txStatus = ZBTxStatusResponse();

// Define os pinos que serão atribuidos aos sensores
const byte sensor_0 = 2;
const byte sensor_1 = 3;

// Define uma flag para o envio do pacote
bool send = false;

void setup() {
  // Inicia o serial e o XBee
  Serial.begin(9600);
  xbee.setSerial(Serial);

  // Define os pinos dos sensores como entradas
  pinMode(sensor_0, INPUT_PULLUP);
  pinMode(sensor_1, INPUT_PULLUP);

  // Trata os sensores como interrupções sensiveis a borda de
     descida
  attachInterrupt(digitalPinToInterrupt(sensor_0), set_0,
     FALLING);
  attachInterrupt(digitalPinToInterrupt(sensor_1), set_1,
     FALLING);
}
```

```
35
36 void loop() {
37   if (send) {
38     // Envia o pacote ZigBee
39     xbee.send(zbTx);
40     delivery_status();
41     send = false;
42   }
43 }
44
45 void delivery_status() {
46   if (xbee.readPacket(500)) {
47     // Deveria ser um pacote de status
48     if (xbee.getResponse().getApiId() == ZB_TX_STATUS_RESPONSE)
         {
49       xbee.getResponse().getZBTxStatusResponse(txStatus);
50
51       // recebemos um status de entrega
52       if (txStatus.getDeliveryStatus() == SUCCESS) {
53         Serial.println("\t␣Sucesso:␣\t␣ACK␣recebido");
54       } else {
55         // Não recebemos um ACK o outro dispositivo esta ligado
             ?
56         Serial.println("\t␣Erro:␣\t␣ACK␣não␣recebido.");
57       }
58     }
59   } else {
60     // O XBee local não criou o pacote tx a tempo -- não
        deveria acontecer.
61     Serial.println("O␣Xbee␣local␣esta␣conectado?");
62   }
63 }
64
65 void set_0() {
66   if (payload[0] != 0) {
67     payload[0] = 0;
68     send = true;
69   }
70 }
```

```
71
72 void set_1() {
73   if (payload[0] != 1) {
74     payload[0] = 1;
75     send = true;
76   }
77 }
```

APÊNDICE B – Codigos fonte sensor atuador B.

```
1 #include <XBee.h>
2 #include <SoftwareSerial.h>
3
4 // Cria um objeto XBee e o payload que será enviado
5 XBee xbee = XBee();
6 char payload [1] = {0};
7
8
9 // Define o endereço do XBee recebedor
10 // Cria o objeto ZigBee que será enviado
11 // Cria um objeto para receber monitorar o status do pacote
12 XBeeAddress64 addr64 = XBeeAddress64(0x0013a200, 0x40c1b208);
13 ZBTxRequest zbTx = ZBTxRequest(addr64, payload, sizeof(payload)
     );
14 ZBTxStatusResponse txStatus = ZBTxStatusResponse();
15
16 // Define os pinos que serão atribuidos aos sensores
17 const byte sensor_0 = 2;
18 const byte sensor_1 = 3;
19
20 // Define uma flag para o envio do pacote
21 bool send = false;
22
23 void setup() {
24     // Inicia o serial e o XBee
25     Serial.begin(9600);
26     xbee.setSerial (Serial);
27
28     // Define os pinos dos sensores como entradas
29     pinMode(sensor_0, INPUT_PULLUP);
30     pinMode(sensor_1, INPUT_PULLUP);
31
32     // Trata os sensores como interrupções sensiveis a borda de
           descida
33     attachInterrupt(digitalPinToInterrupt(sensor_0), set_0,
         FALLING);
34     attachInterrupt(digitalPinToInterrupt(sensor_1), set_1,
         FALLING);
```

```
35
36 }
37
38 void loop() {
39     if (send) {
40         // Envia o pacote ZigBee
41         xbee.send(zbTx);
42         delivery_status();
43         send = false;
44     }
45 }
46
47
48 void delivery_status() {
49     if (xbee.readPacket(500)) {
50         // Deveria ser um pacote de status
51         if (xbee.getResponse().getApiId() ==
              ZB_TX_STATUS_RESPONSE) {
52             xbee.getResponse().getZBTxStatusResponse(txStatus)
                  ;
53
54             // recebemos um status de entrega
55             if (txStatus.getDeliveryStatus() == SUCCESS) {
56                 Serial.println("\t␣Sucesso:␣\t␣ACK␣recebido");
57             } else {
58                 // Não recebemos um ACK o outro dispositivo
                      esta ligado?
59                 Serial.println("\t␣Erro:␣\t␣ACK␣não␣recebido.")
                      ;
60             }
61         }
62     } else {
63         // O XBee local não criou o pacote tx a tempo -- não
              deveria acontecer.
64         Serial.println("O␣Xbee␣local␣esta␣conectado?");
65     }
66 }
67
68 void set_0() {
```

```
69      if (payload[0] != 0) {
70          payload[0] = 0;
71          send = true;
72      }
73  }
74
75  void set_1() {
76      if (payload[0] != 1) {
77          payload[0] = 1;
78          send = true;
79      }
80  }
```

APÊNDICE C – Codigos fonte valvula de comando atuador A.

```
1 #include <XBee.h>
2 #include <SoftwareSerial.h>
3
4 #include <MQTT.h>
5 #include <Ethernet.h>
6
7 byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x64, 0x50};
8 byte ip[] = {10, 32, 83, 234};
9
10 EthernetClient net;
11 MQTTClient client;
12
13 /*
14    Tabela dos sensores:
15    Código    | Sensor
16    0         | Atuador A recuado
17    1         | Atuador A avançado
18 */
19
20 char code;
21 bool code_received = false;
22 bool flag = false;
23 unsigned long relay_timer = 0;
24
25 const byte button = 2;
26 const byte relay = 8;
27
28
29 // Estabelece os pinos 4 e 5 para comunicação serial do XBee.
30 SoftwareSerial xbee_serial(4,5);
31
32 // Instancia objetos que seram usados na comunicação XBee
33 XBee xbee = XBee();
34 XBeeResponse response = XBeeResponse();
35 ZBRxResponse rx = ZBRxResponse();
36 ModemStatusResponse msr = ModemStatusResponse();
37
38 void setup() {
```

```
39      // Define a taxa de transmissão usado pelo XBee e a o
           serial usado
40      xbee_serial.begin(9600);
41      xbee.setSerial(xbee_serial);
42
43      // Inicia a comunicação serial
44      Serial.begin(9600);
45
46      // Configurações para o uso do MQTT
47      Ethernet.begin(mac, ip);
48      //client.begin("rubens.cloud.shiftr.io", net);
49      client.begin("10.32.83.224", net);
50      client.onMessage(messageReceived);
51      connect();
52
53      pinMode(relay, OUTPUT);
54      digitalWrite(relay, LOW);
55
56      pinMode(button, INPUT_PULLUP);
57      attachInterrupt(digitalPinToInterrupt(button), start,
           FALLING);
58  }
59
60  void loop() {
61
62    client.loop();
63    if (!client.connected()) {
64        connect();
65    }
66
67     status_xbee();
68     if (code_received) {
69         // atribui a mensagem do pacote a code
70         code = rx.getData(0);
71         operation_xbee();
72         code_received = false;
73     }
74
75     if (flag) {
```

```
76          if (millis() -relay_timer > 2000) {
77              digitalWrite(relay, LOW);
78              flag = false;
79          }
80      }
81  }
82
83  void start() {
84      // Tratamento de debounce do botão
85      static unsigned long lastInterrupt = 0;
86      unsigned long interruptTime = millis();
87      if (interruptTime - lastInterrupt > 200){
88          digitalWrite(relay, HIGH);
89      }
90      lastInterrupt = interruptTime;
91  }
92
93  void connect() {
94      Serial.print("Conectando...");
95      while (!client.connect("Atuador A", "rubens", "
            pYEml6K8Q321WxGB")){
96          Serial.print(".");
97          delay("1000");
98      }
99      Serial.println("\nConectado!");
100
101     client.subscribe("/Atuador_B/#");
102     client.subscribe("/Smartphone/#");
103 }
104
105 void messageReceived(String &topic, String &payload) {
106     Serial.println("Entrada: " + topic + " - " + payload);
107
108     if (payload == "B+") {
109         flag = true;
110         relay_timer = millis();
111     } else if (payload == "on") {
112         digitalWrite(relay, HIGH);
113     }
```

```
114 }
115
116 void operation_xbee() {
117     if (code == 0) {
118         client.publish("/Atuador_A", "A-");
119     } else if (code == 1) {
120         client.publish("/Atuador_A", "A+");
121     }
122 }
123
124 void status_xbee() {
125     xbee.readPacket();
126     if (xbee.getResponse().isAvailable()) {
127         // Recebeu algo
128
129         if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
130             // Atribui o pacote ao objeto rx
131             xbee.getResponse().getZBRxResponse(rx);
132             code_received = true;
133
134             if (rx.getOption() == ZB_PACKET_ACKNOWLEDGED) {
135                 // O emissor recebeu um ACK
136                 Serial.println("Sucesso: ACK entregue");
137             } else {
138                 // Recebemo algo mas o emissor n recebeu um ACK
139                 Serial.println("Falha: ACK não foi entregue.");
140             }
141         } else if (xbee.getResponse().getApiId() ==
              MODEM_STATUS_RESPONSE) {
142             xbee.getResponse().getModemStatusResponse(msr);
143             // O XBee local envia respostas em certos eventos
144
145             if (msr.getStatus() == ASSOCIATED) {
146                 Serial.println("Status : Conectado");
147             } else if (msr.getStatus() == DISASSOCIATED) {
148                 Serial.println("Status : Desconectado");
149             } else {
150                 Serial.println("Status : Desconhecido.");
151             }
```

```
152             } else {
153                 Serial.println("Recebeu algo, mas nada do que
                       esperavamos.");
154             }
155         }
156 }
```

APÊNDICE D – Codigos fonte valvula de comando atuador B.

```
#include <XBee.h>
#include <SoftwareSerial.h>

#include <MQTT.h>
#include <Ethernet.h>

byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x64, 0x51};
byte ip[] = {10, 32, 83, 220};

EthernetClient net;
MQTTClient client;

/*
   Tabela dos sensores:
   Código    | Sensor
   0         | Atuador A recuado
   1         | Atuador A avançado
*/

char code;
bool code_received = false;
bool flag1 = false;
bool flag2 = false;

unsigned long timer1 = 0;
unsigned long timer2 = 0;
unsigned long relay_timer = 2000;


const byte relay = 8;

// Estabelece os pinos 4 e 5 para comunicação serial do XBee.
// SoftwareSerial xbee_serial(4,5);

// Instancia objetos que seram usados na comunicação XBee
XBee xbee = XBee();
XBeeResponse response = XBeeResponse();
ZBRxResponse rx = ZBRxResponse();
```

```
39 ModemStatusResponse msr = ModemStatusResponse();
40
41 void setup() {
42     // Define a taxa de transmissão usado pelo XBee e a o
          serial usado
43     // xbee_serial.begin(9600);
44     // xbee.setSerial(xbee_serial);
45
46     // Inicia a comunicação serial
47     Serial.begin(9600);
48     xbee.setSerial(Serial);
49
50     // Configurações para o uso do MQTT
51     Ethernet.begin(mac, ip);
52     //client.begin("rubens.cloud.shiftr.io", net);
53     client.begin("10.32.83.224", net);
54     client.onMessage(messageReceived);
55     connect();
56
57     pinMode(relay, OUTPUT);
58     digitalWrite(relay, LOW);
59
60 }
61
62 void loop() {
63
64    client.loop();
65    if (!client.connected()) {
66        connect();
67    }
68
69     status_xbee();
70     if (code_received) {
71         // atribui a mensagem do pacote a code
72         code = rx.getData(0);
73         operation_xbee();
74         code_received = false;
75     }
76
```

```
77      if (flag1) {
78          if ( millis() - timer1 > relay_timer) {
79              digitalWrite(relay, HIGH);
80              flag1 = false;
81          }
82      }
83
84      if (flag2) {
85          if ( millis() - timer2 > relay_timer) {
86              digitalWrite(relay, LOW);
87              flag2 = false;
88          }
89      }
90
91  }
92
93  void connect() {
94      Serial.print("Conectando...");
95      while (!client.connect("Atuador␣B", "rubens", "
            pYEml6K8Q321WxGB")){
96          Serial.print(".");
97          delay("1000");
98      }
99      Serial.println("\nConectado!");
100
101     client.subscribe("/Atuador_A/#");
102 }
103
104 void messageReceived(String &topic, String &payload) {
105     Serial.println("Entrada:␣" + topic + "␣-␣" + payload);
106
107     if (payload == "A+") {
108         flag1 = true;
109         timer1 = millis();
110     } else if (payload == "A-") {
111         flag2 = true;
112         timer2 = millis();
113     }
114
```

```
115 }
116
117 void operation_xbee() {
118     if (code == 0) {
119         client.publish("/Atuador_B", "B-");
120     } else if (code == 1) {
121         client.publish("/Atuador_B", "B+");
122     }
123 }
124
125 void status_xbee() {
126     xbee.readPacket();
127     if (xbee.getResponse().isAvailable()) {
128         // Recebeu algo
129
130         if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE) {
131             // Atribui o pacote ao objeto rx
132             xbee.getResponse().getZBRxResponse(rx);
133             code_received = true;
134
135             if (rx.getOption() == ZB_PACKET_ACKNOWLEDGED) {
136                 // O emissor recebeu um ACK
137                 Serial.println("Sucesso: ACK entregue");
138             } else {
139                 // Recebemo algo mas o emissor n recebeu um ACK
140                 Serial.println("Falha: ACK não foi entregue.");
141             }
142         } else if (xbee.getResponse().getApiId() ==
              MODEM_STATUS_RESPONSE) {
143             xbee.getResponse().getModemStatusResponse(msr);
144             // O XBee local envia respostas em certos eventos
145
146             if (msr.getStatus() == ASSOCIATED) {
147                 Serial.println("Status : Conectado");
148             } else if (msr.getStatus() == DISASSOCIATED) {
149                 Serial.println("Status : Desconectado");
150             } else {
151                 Serial.println("Status : Desconhecido.");
152             }
```

```
153             } else {
154                 Serial.println("Recebeu algo, mas nada do que
                       esperavamos.");
155             }
156         }
157 }
```