**UNIVERSIDADE FEDERAL DE UBERLÂNDIA**
**FACULDADE DE ENGENHARIA MECÂNICA**
**ENGENHARIA MECATRÔNICA**
**Redes Industriais - FEMEC42082**

**Prof. José Jean-Paul Zanlucchi De Souza Tavares**

**Grupo 5**

# RELATÓRIO TRABALHO FINAL DE REDES INDUSTRIAIS:

## Acionamento de pistões utilizando I2C, ethernet e Xbee

ERICK TOMAZ OLIVEIRA 11911EMT020
ARTHUR REIS BELLO 11811EMT009

**UBERLÂNDIA**
**2023**

**Sumário**


**1. Introdução** **3**

**2. Objetivo** **3**

**3. Protocolos de Comunicação** **5**

3.1. I2c 5

3.2. Ethernet 7

3.3. ZigBee 10

**4. Sobre o sistema** **13**

**5. Resultados** **17**

**6. Conclusão** **17**

**7. Bibliografia** **18**

# 1. Introdução

As redes de comunicação industriais são utilizadas para conectar diferentes dispositivos e sistemas, permitindo a troca de informações e comandos entre eles. As principais características das redes industriais incluem alta confiabilidade, segurança, escalabilidade e capacidade de lidar com condições adversas.

Neste projeto, serão utilizados protocolos de comunicação Ethernet, I2C e Zigbee. Ethernet é uma tecnologia de rede amplamente utilizada em processos industriais devido à sua confiabilidade e capacidade de transferir grandes volumes de dados. I2C é um protocolo de comunicação serial que permite a conexão de vários dispositivos em uma única linha, o que é útil para sistemas de pequena escala. Zigbee é uma tecnologia de rede sem fio de baixa potência e baixo custo, projetada para ser usada em dispositivos com baixa potência, como sensores e atuadores.

Este projeto tem como objetivo desenvolver uma solução baseada em atuadores pneumáticos controlados por Arduinos, que sejam conectados através de protocolos de comunicação Ethernet, I2C e Zigbee. O uso destes protocolos permitirá a comunicação entre os dispositivos e a central de controle, possibilitando a automação e otimização do processo. Este projeto pretende mostrar como a combinação de tecnologias de comunicação e automação pode ser utilizada para melhorar a eficiência e a eficácia de processos industriais.

# 2. Objetivo

O sistema é composto por 1 botão, 2 cilindros de dupla ação (A e B), dois fins de curso (A0/A1B0/B1), duas válvulas simples solenoide 5x2 com retorno por mola, conforme mostra a Figura 1.

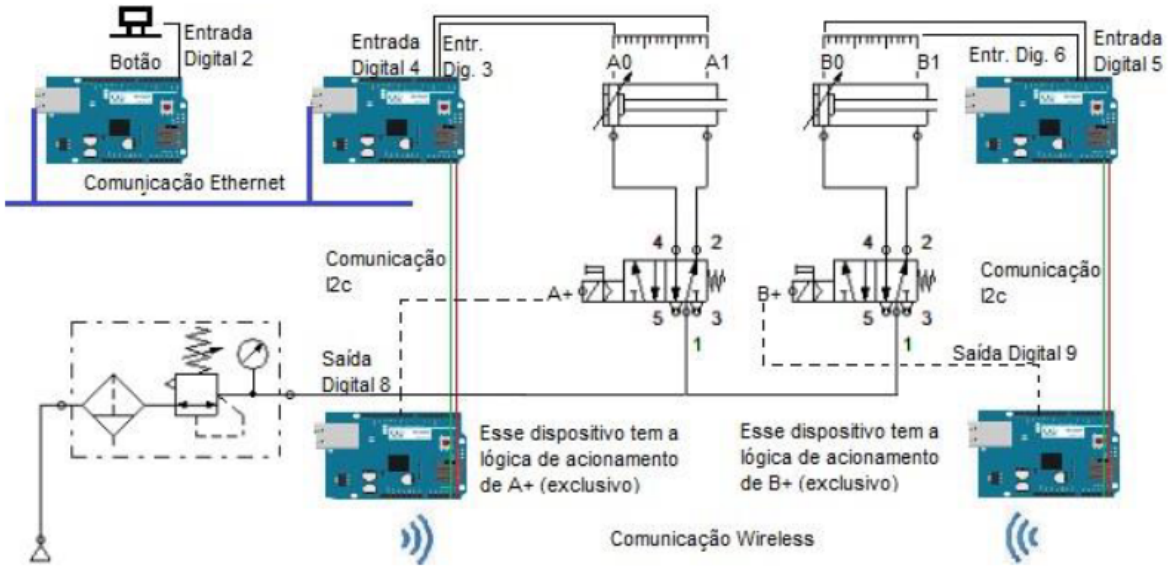


Figura 1: Esquema eletropneumático. (Fonte: Roteiro da Atividade)

O sistema é montado da seguinte forma:

- Arduino 1
  - Um Botão deverá estar na entrada digital 2.
- O Arduino 1 se comunica com o Arduino 2 por meio de rede Ethernet.
- Arduino 2 (Sensor da Válvula A)
  - Sensor do posicionamento do cilindro A
  - Os fins de curso A0 e A1 são ligados nas entradas 3 e 4
  - Conectado via i2c, pelas portas SCL e SDA, com o Arduino 3.
  - Escravo do Arduino 3.
- Arduino 3 (Controlador da Válvula A )
  - Controlador do cilindro A
  - Porta 8 responsável pela válvula que comanda o cilindro A.
  - Mestre do Arduino 2.
- Arduino 4 (Sensor da Válvula B)
  - Sensor do posicionamento do cilindro B
  - Os fins de curso B0 e B1 são ligados nas entradas 5 e 6
  - Conectado via i2c, pelas portas SCL e SDA, com o Arduino 5.
  - Escravo do Arduino 5.
- Arduino 5 (Controlador da Válvula B )
  - Controlador do cilindro B
  - Porta 9 responsável pela válvula que comanda o cilindro A.
  - Mestre do Arduino 2.
- Os Arduinos 3 e 5 (controladores), comunicam entre si utilizando o protocolo zigBee.

Esse sistema deve ser capaz de realizar uma sequência para esses pistões utilizando os protocolos designados. Para o grupo que escreve esse relatório, grupo 5, a sequência designada é A+A-B+{A+//B-}A-.

# 3. Protocolos de Comunicação

## 3.1. I2c

I2C (Inter-Integrated Circuit) é um protocolo de comunicação serial de baixa velocidade. Ele é usado para conectar dispositivos eletrônicos, como sensores, dispositivos de entrada/saída, dispositivos de memória e dispositivos de interface com o usuário. Ele permite que vários dispositivos compartilhem informações e se comuniquem com um único controlador de host.

O I2C utiliza dois fios para transmitir dados (o que o torna conhecido também como Two Wire Communication - TWI), SDA (Serial Data) e SCL (Serial Clock), e permite a conexão de até 127 dispositivos em uma única linha. O protocolo I2C é projetado para ser fácil de implementar e usar, e é compatível com vários sistemas de processamento, incluindo microcontroladores, microprocessadores e sistemas embarcados.

O protocolo I2C é baseado em um esquema mestre-escravo, onde o dispositivo mestre inicia a comunicação e controla o fluxo de dados, enquanto os dispositivos escravos respondem às solicitações do mestre. O I2C também suporta a comunicação multi-mestre, onde vários dispositivos mestres podem compartilhar a mesma linha de comunicação.

Devido à sua flexibilidade, facilidade de uso e baixo consumo de energia, o protocolo I2C é amplamente utilizado em aplicações como sistemas embarcados, dispositivos móveis, computadores e equipamentos de automação industrial.

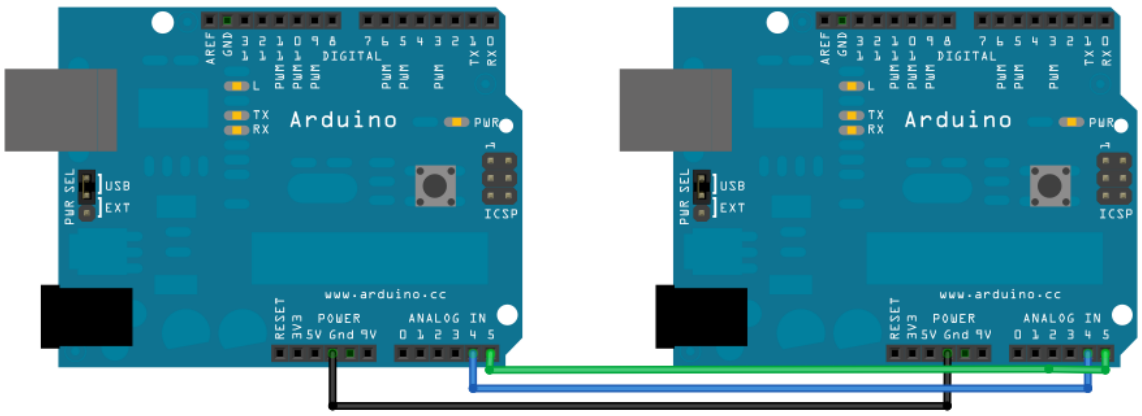


Figura 2: Arduinos Conectados para Comunicação I2C. (Fonte: Mundo da Robótica)

Os microcontroladores produzidos pela empresa Arduino usualmente possuem o protocolo I2C nativo. No Arduino UNO não é diferente. Ele possui 2 pares de portas que permitem a conexão I2C, as portas SDA e SCL e as analógicas A4 e A5. Ao conectar essas portas entre dois arduinos como apresentado na figura 2, é possível fazer o upload dos seguintes códigos (retirados do site da Arduino) para testar a biblioteca Wire da Arduino.

Mestre-Leitor

```
// Wire Controller Reader
// by Nicholas Zambetti <http://www.zambetti.com>
// Demonstrates use of the Wire library
// Reads data from an I2C/TWI peripheral device
// Refer to the "Wire Peripheral Sender" example for use with this
// Created 29 March 2006
// This example code is in the public domain
#include <Wire.h>

void setup() {
    Wire.begin();       // join i2c bus (address optional for master)
    Serial.begin(9600);  // start serial for output
}

void loop() {
    Wire.requestFrom(8, 6);    // request 6 bytes from peripheral device #8

    while (Wire.available()) { // peripheral may send less than requested
      char c = Wire.read(); // receive a byte as character
      Serial.print(c);         // print the character
    }

    delay(500);
}
```

Escravo-enviador

```
// Wire Peripheral Sender
// by Nicholas Zambetti <http://www.zambetti.com>
// Demonstrates use of the Wire library
// Sends data as an I2C/TWI peripheral device
// Refer to the "Wire Master Reader" example for use with this
// Created 29 March 2006
// This example code is in the public domain.

#include <Wire.h>

void setup() {
```

```
    Wire.begin(8);                // join i2c bus with address #8
    Wire.onRequest(requestEvent); // register event
}

void loop() {
    delay(100);
}

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() {
    Wire.write("hello "); // respond with message of 6 bytes
    // as expected by master
}
```

O código acima define um escravo com endereço 8 que envia dados para o mestre que faz requisição da frase e lê a frase e printa na serial.

### 3.2. Ethernet

A conexão Ethernet permite que um Arduino se conecte à rede e se comunique com outros dispositivos conectados à rede, como computadores, dispositivos móveis e outros Arduinos. Isso abre muitas possibilidades para aplicações de automação, monitoramento remoto e comunicação de dados.

Para se conectar a uma rede Ethernet, é necessário usar um shield Ethernet ou um módulo Ethernet que seja compatível com o Arduino. Esses dispositivos geralmente incluem um chip Ethernet que gerencia a comunicação de rede e uma conexão física para o cabo Ethernet. Alguns shields Ethernet também incluem recursos adicionais, como uma interface SD card para armazenamento de dados.

Uma vez que o shield ou módulo Ethernet esteja conectado ao Arduino, é possível usar bibliotecas de software para gerenciar a comunicação de rede. A biblioteca Ethernet é a biblioteca oficial do Arduino para gerenciamento de rede Ethernet e é compatível com a maioria dos shields e módulos Ethernet. Ela fornece funções para configurar e gerenciar conexões de rede, enviar e receber dados e trabalhar com protocolos de rede comuns, como o protocolo TCP/IP.

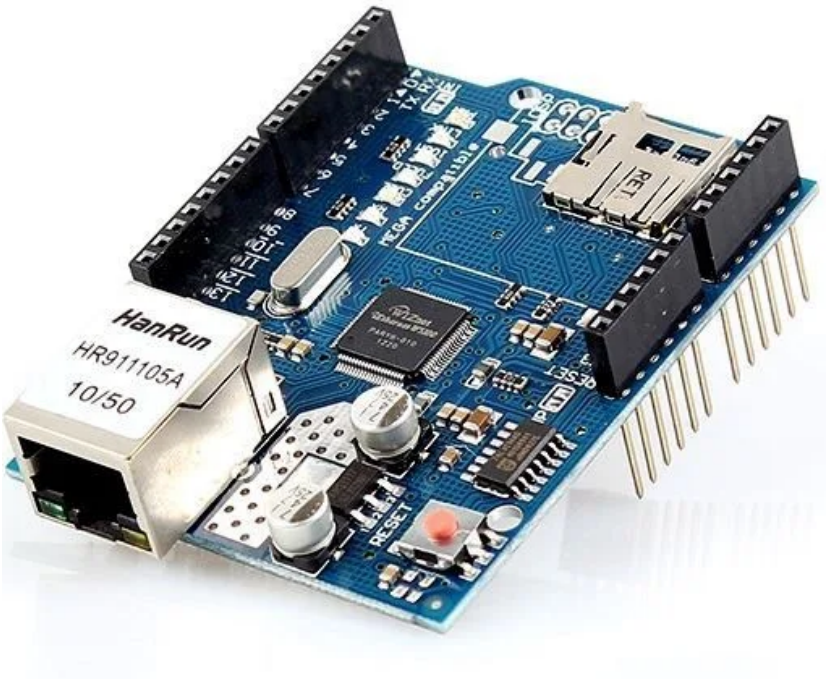


Figura 3: Arduino Ethernet Shield W5100 (Fonte: arducore)

Emissor:

```
#include <Ethernet.h>
#include <EthernetUdp.h>

EthernetUDP Udp;
byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x64, 0x44};
byte ip[] = {192, 168, 1, 131};
byte remoteip[] = {192, 168, 1, 151};

unsigned int port = 8888;

char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
int led = LED_BUILTIN;

void setup()
{
  Ethernet.begin(mac, ip);
  Udp.begin(port);
}
```

```
void loop()
{
  Udp.beginPacket(remoteip, port);
  Udp.write('1');
  Udp.endPacket();
  delay(3000);
  Udp.beginPacket(remoteip, port);
  Udp.write('2');
  Serial.println('2');
  Udp.endPacket();
  delay(3000);
}
```

Receptor:

```
#include <SPI.h>
#include "Ethernet.h"
#include "EthernetUdp.h"

EthernetUDP Udp;
byte mac[] = {0x90, 0xA2, 0xDA, 0x00, 0x64, 0x50};
byte ip[] = {192, 168, 1, 151};
unsigned int port = 8888;

char packetBuffer[UDP_TX_PACKET_MAX_SIZE];
int led = 4;

void setup()
{
  Ethernet.begin(mac, ip);
  Udp.begin(port);
  Serial.begin(38400);
  pinMode(led, OUTPUT);
}
```

```
void loop()
{
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {
    Udp.read(packetBuffer, UDP_TX_PACKET_MAX_SIZE);
    Serial.println(packetBuffer[0]);
    if (packetBuffer[0] == '1')
    {
      Serial.println("Acendendo o LED...");
      digitalWrite(led, HIGH);
    }
    if (packetBuffer[0] == '2')
    {
      Serial.println("Apagando o LED...");
      digitalWrite(led, LOW);
    }
  }
  delay(10);
}
```

O código acima foi feito em um laboratório sobre os estudos do Ethernet onde um arduino envia ‘1’ ou ‘2’ a cada 3000 ms e o segundo arduino possui uma porta aberta e recebe essa informação e acende um LED se recebe ‘1’ e apaga se recebe ‘2’. Importante destacar que o mac address depende do valor indicado no shield. O IP depende dos IPs disponíveis na rede. Neste laboratório, os arduinos foram conectados a um Switch que administra as conexões ethernet e o mapeamento dos IPs nas respectivas portas.

### 3.3. ZigBee

Zigbee é uma tecnologia sem fio de baixa potência e baixo custo que é usada para criar redes de dispositivos sem fio. Ele é baseado em uma especificação IEEE 802.15.4 e é projetado para ser usado em dispositivos com baixa potência, como sensores e atuadores.

Para usar Zigbee com Arduino, você precisará de uma placa de Zigbee compatível com o Arduino e de uma biblioteca de software para controlar a comunicação entre o Arduino e o Zigbee. Usualmente, ZigBee utiliza do Serial para

a comunicação, o que facilita e simplifica o código mas segura o controle do Serial por todo o processo de comunicação.

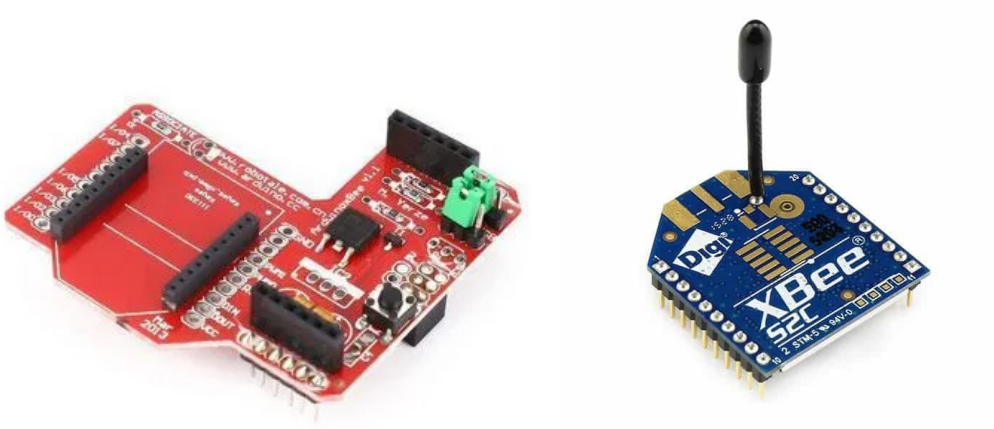


Figura 4: Shield de expansão e Módulo XBee S2C 802.15.4 Wire Antenna (Fonte: FilipeFlop)

Código emissor:

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

XBee xbee = XBee();

XBeeAddress64 add64 = XBeeAddress64(0x0013A200,
                                    0x40C1B208);
uint8_t payload[] = {'1', '2'};

int ledBuiltinState = 0;

void setup()
{
  Serial.begin(9600);
  xbee.setSerial(Serial);

  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
       ZBTxRequest    data    =    ZBTxRequest(add64,    payload,
sizeof(payload));
  xbee.send(data);
```

```
  ledBuiltinState = !ledBuiltinState;
  digitalWrite(LED_BUILTIN, ledBuiltinState);
  delay(1000);
}
```

Código receptor:

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

XBee xbee = XBee();
ZBRxResponse rx = ZBRxResponse();

int LED_PORT = 8;
int ledPortState = 0;
int ledBuiltinState = 0;

void setup()
{
  Serial.begin(9600);
  xbee.setSerial(Serial);

  pinMode(LED_PORT, OUTPUT);
  pinMode(LED_BUILTIN, OUTPUT);
}

void loop()
{
  String sample;
  xbee.readPacket();
  if (xbee.getResponse().isAvailable())
  {
    if (xbee.getResponse().getApiId() == ZB_RX_RESPONSE)
    {
      xbee.getResponse().getZBRxResponse(rx);
      for (int i = 0; i < rx.getDataLength(); i++)
      {
        sample += (char)rx.getData(i);
      }
```

```
      Serial.println(sample);
      ledPortState = !ledPortState;
    }
    ledBuiltinState = !ledBuiltinState;
  }

  digitalWrite(LED_PORT, !ledPortState);
  digitalWrite(LED_BUILTIN, ledBuiltinState);
}
```

O código acima realiza o mesmo processo que o apresentado na seção Ethernet. Entretanto, como o Serial não pode ser utilizado eficientemente, dois LEDs são usados para debugar o sucesso da comunicação (tanto para saber se os dados estão sendo recebidos quanto se os dados recebidos são corretos). Importante ressaltar que a diferença do Ethernet para o zigbee em relação à lógica é que o valor enviado não é um ciclo de 3000 ms e sim dependente da entrada de uma porta digital. Isso foi feito para que não haja dados fantasmas e o controle do dado enviado possa ser manual.

## 4. Sobre o sistema

Para otimizar a implementação do sistema, foi desenvolvida uma estrutura em que os arduinos 3 e 5 atuam como controladores principais. A lógica principal do sistema está programada neles, enquanto o arduino 1 é utilizado como sensor e os arduinos 2 e 4 são responsáveis por acionar os atuadores de acordo com as ordens dos controladores principais. Com este objetivo em mente, a programação foi elaborada com a premissa de que os códigos dos controladores principais devem ser o mais semelhantes possível.

Para facilitar a comunicação entre os dois mestres, foi estabelecido um padrão de comunicação. Esse padrão consiste em um código que define completamente o estado de cada mestre. Com isso, foi criada uma tabela da verdade que ilustra essa lógica de comunicação. As tabelas apresentam linhas vermelhas que indicam estados inválidos, desde que o sistema esteja operando de forma adequada.

| isActive | A0 | A1 | Saida I2C Arduino 2 |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 2 |
| 0 | 1 | 1 | 3 |
| 1 | 0 | 0 | 4 |
| 1 | 0 | 1 | 5 |
| 1 | 1 | 0 | 6 |
| 1 | 1 | 1 | 7 |

Figura 5: Tabela da verdade do arduino 3

| B0 | B1 | Saida I2C Arduino 4 |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 2 |

Figura 6: Tabela da verdade para o arduino 5

O Arduino 1 possui uma programação simplificada, pois sua principal função é ler os dados do botão e transmiti-los para o Arduino 2. A lógica desse processo é apresentada abaixo:

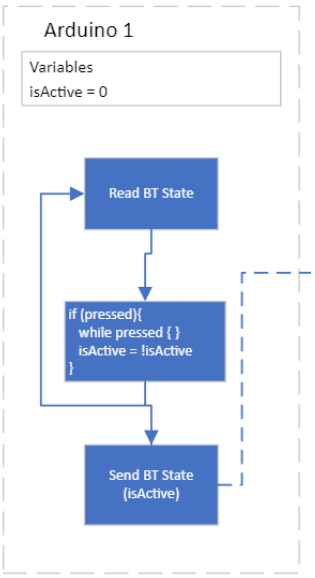


Figura 7: Lógica do arduino 1

O Arduino 2 deve, de forma periódica, monitorar as entradas dos sensores de final de curso A0 e A1, bem como o sinal recebido pelo Arduino 1. Com base nessas informações, ele então calcula o código de comunicação de acordo com a tabela apresentada na figura 5. Quando solicitado, ele então envia esse valor para o Arduino 3

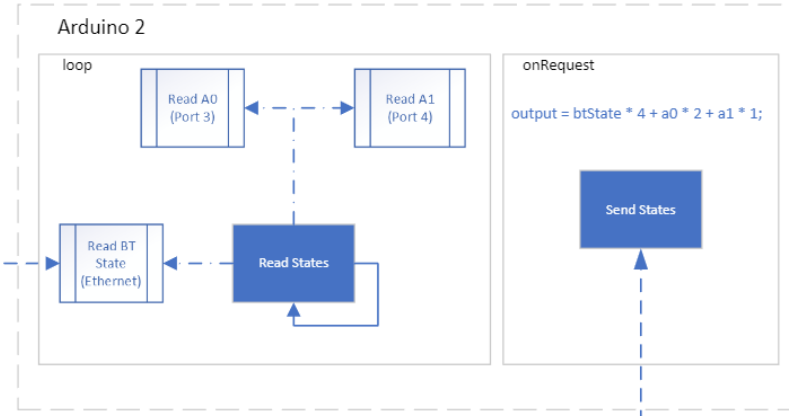


Figura 8: Lógica do arduino 2

O Arduino 3 é responsável por processar e armazenar a sequência de movimentos do pistão A. Essa sequência é armazenada em um array denominado "seq_A" e cada etapa é atribuída um valor específico: 1 para avançar, -1 para recuar e 0 para manter a posição (que é utilizado em etapas em que somente o pistão B está em ação). Ao receber o código de comunicação do Arduino 2, ele extrai o estado atual do pistão e o compara com o array "seq_A". Se a comparação for igual, o Arduino 3 avança para a próxima etapa, envia o valor da etapa para o Arduino 4 e aguarda o pistão B alcançar a mesma etapa para executar o próximo passo, assegurando a perfeição no processo de sincronização entre pistões e precisão nos movimentos. O arduino 5 funciona com a mesma lógica, uma vez que a premissa inicial era de que os mestres possuem códigos semelhantes,

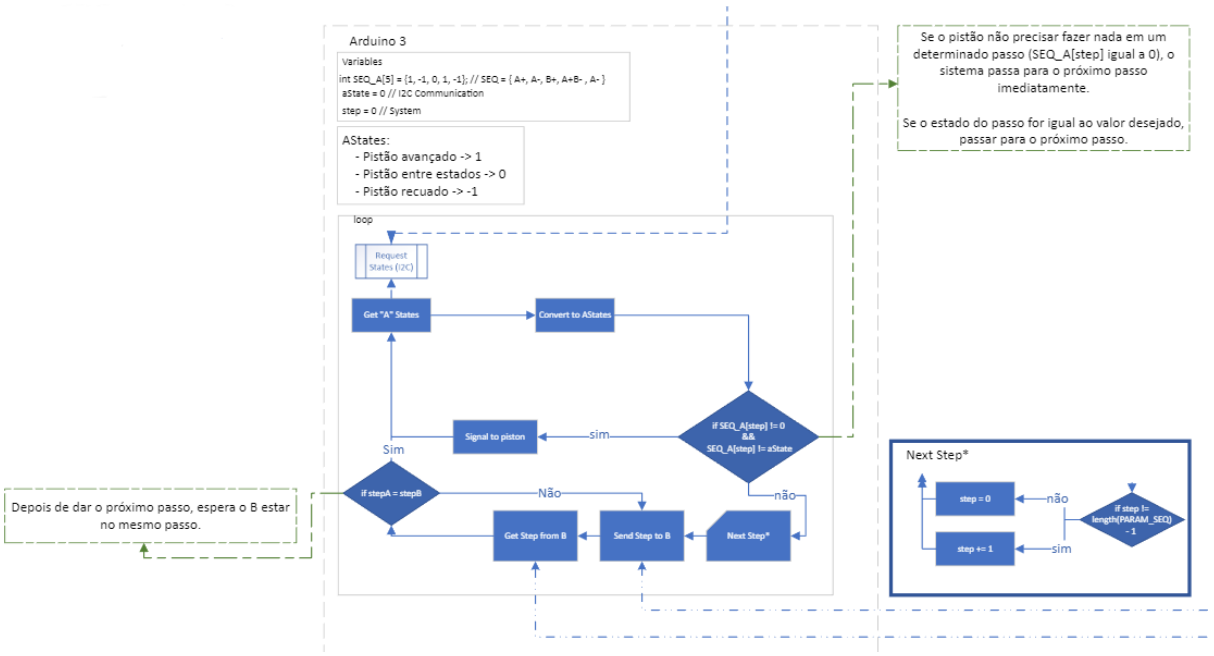


Figura 9: Lógica do arduino 3

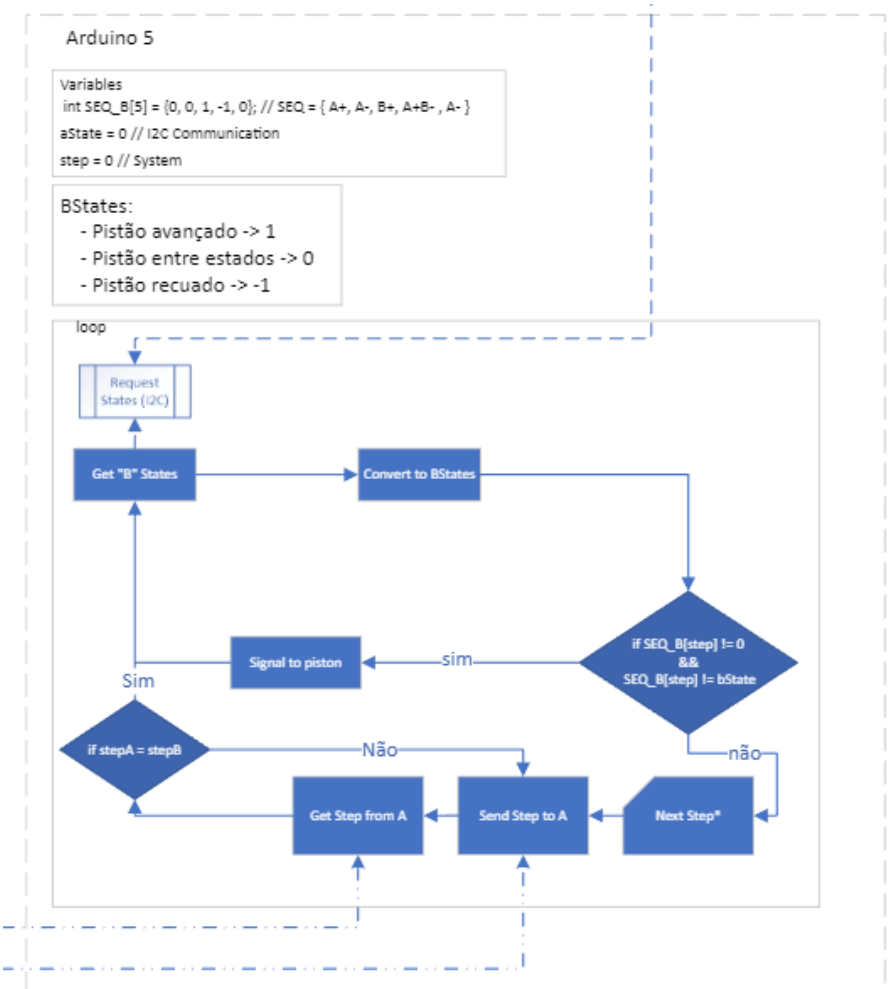


Figura 10: Lógica do arduino 5

O arduino 4 funciona de forma semelhante ao arduino 2, com exceção de se comunicar com o arduino 1. Sua lógica completa é mostrada na figura abaixo:

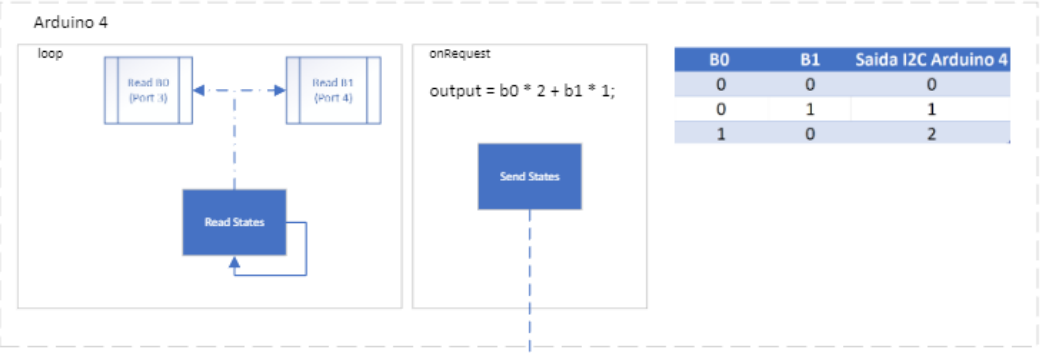


Figura 11: Lógica do arduino 4

Com esta arquitetura em mente, foi desenvolvido e implementado o código de programação para cada um dos microcontroladores. Para isso, foi utilizado o VSCode, com a extensão PlatformIO, que substitui a IDE do Arduino e possui algumas vantagens que facilitam a programação. O código completo pode ser acessado pelo link a seguir:

https://github.com/Erick-Oliveira-ET/xbee-i2c-ethernet-automatization

## 5. Resultados

Com essa estratégia, conseguimos executar a sequência de comandos com precisão e eficiência, como pode ser visto no vídeo disponível neste link (https://youtu.be/W5L_wn6ObV8 ). No entanto, enfrentamos alguns desafios no hardware, especificamente na comunicação entre os dois Zigbees inicialmente instalados na planta. Após identificarmos essa questão, optamos por trocar esses dispositivos por outros que passaram com sucesso nos testes de bancada. Com esforço e dedicação, conseguimos resolver essas dificuldades e alcançar os resultados desejados.

## 6. Conclusão

No desenvolvimento deste projeto, adotamos uma abordagem baseada em atuadores pneumáticos controlados por Arduinos, que foram conectados através de protocolos de comunicação Ethernet, I2C e Zigbee. Através de um planejamento rigoroso e reuniões frequentes, conseguimos estruturar um projeto sólido e robusto, que atendia aos requisitos do projeto sem a necessidade de ajustes lógicos do código. Isso nos permitiu concentrar nossos esforços na implementação física do projeto e alcançar resultados satisfatórios.

## 7. Bibliografia

[1] ARDUINO. **Language Reference**. Disponível em: https://www.arduino.cc/reference/en/. Acesso em: 20 dez. 2022.

[2] PLATFORMIO LABS. **Https://platformio.org/**. Disponível em: https://docs.platformio.org/en/latest/frameworks/arduino.html. Acesso em: 20 dez. 2022.

[3] RAPP, Andrew. **Xbee-arduino**. Disponível em: https://github.com/andrewrapp/xbee-arduino. Acesso em: 19 jan. 2023.

**Anexo I - Fluxograma da lógica dos Arduinos**