

**UNIVERSIDADE FEDERAL DE SANTA MARIA  
CENTRO DE TECNOLOGIA  
CURSO DE ENGENHARIA DE CONTROLE E AUTOMAÇÃO**

**USO DE UM SISTEMA EMBARCADO E SOCKETS  
UDP PARA SIMULAR SUPERVISÃO E CONTROLE  
REMOTOS DE UM CONJUNTO SENSOR  
CONTROLADOR ATUADOR VIA REDE**

**RELATÓRIO DA DISCIPLINA DE REDES INDUSTRAIS  
Prof. Frederico Schaf**

**Andrei Schopf  
Davi Calil  
William Rocha**

**Santa Maria, RS, Brasil**

**2018**

## SUMÁRIO

<b>INTRODUÇÃO .....</b>	<b>2</b>
<b>CAPÍTULO 1 DESCRIÇÃO DO PROJETO .....</b>	<b>3</b>
1.1 Objetivo .....	3
1.2 Estrutura física .....	3
1.3 Configuração da Rede.....	5
<b>CAPÍTULO 2 PRÁTICA.....</b>	<b>8</b>
2.1 Fluxogramas .....	8
2.2 Resultados obtidos .....	10
<b>CONCLUSÃO.....</b>	<b>12</b>
<b>ANEXO A.....</b>	<b>13</b>

## **INTRODUÇÃO**

O projeto trata-se de um trabalho semestral que tem por objetivo a implementação na prática dos conhecimentos obtidos durante o semestre na disciplina de Redes Industriais. Para cumprir esse objetivo foi utilizado um sistema embarcado e sockets UDP para simular supervisão e controle de um sistema composto por uma bolinha em um tubo, com a finalidade de manter a bolinha em determinada altura, dada pela referência estabelecida via código,

## CAPÍTULO 1 DESCRIÇÃO DO PROJETO

### 1.1 Objetivo

O objetivo desse projeto é construção de um protótipo em malha fechada através do emprego de redes e protocolos UDP. Conforme a Figura 1 abaixo.

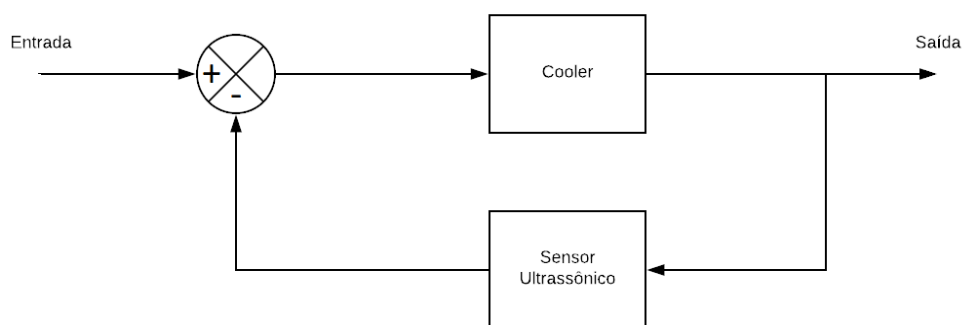
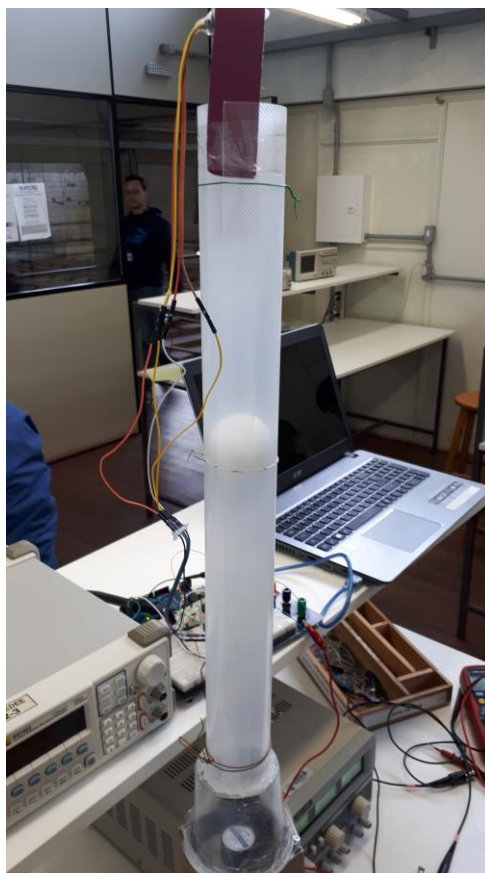


Figura 1 – Digrama de blocos em malha fechada

### 1.2 Estrutura física

A estrutura física é constituída de um tubo transparente no qual é acoplado em sua base um cooler responsável por criar fluxo de ar no interior do tubo, a fim de manter a bolinha que está no interior do tubo suspensa no ar.

Na Figura 2 podemos ver a construção física do sistema de controle.



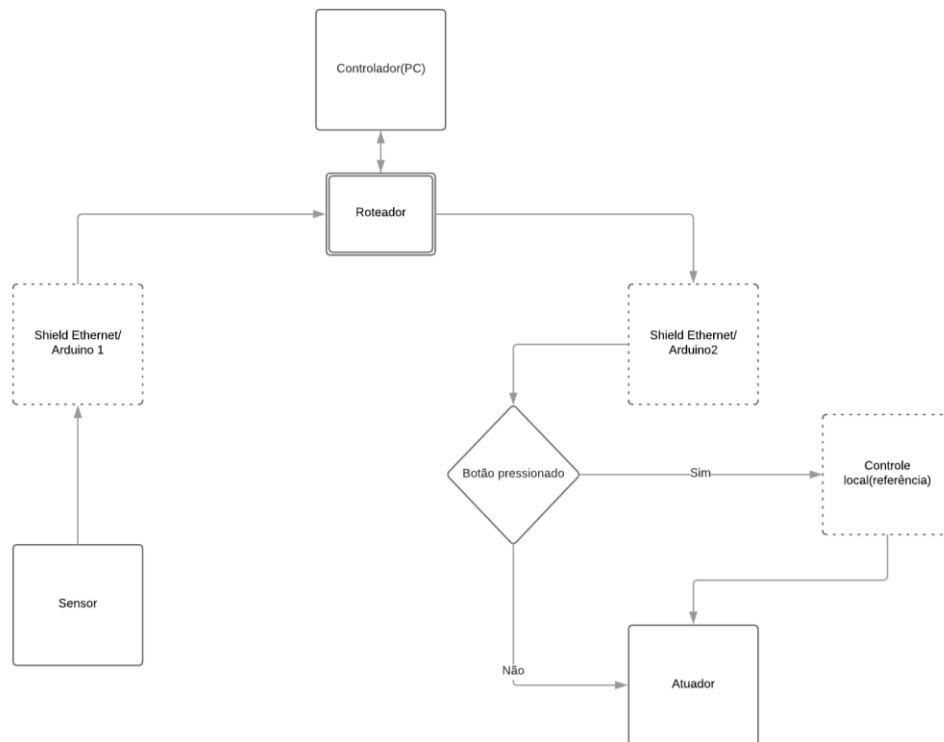
**Figura 2 – Modelo físico do projeto**

A parte destinada ao sensor de distância é constituída de um sensor ultrassônico responsável pela leitura da altura em que a bolinha se encontra no tubo, e posteriormente utilizando *shield ethernet*, envia periodicamente a distância ao servidor UDP.

A variação da altura é realizada pelo cooler, que possui duas opções para receber a referência, através do servidor UDP (controle remoto) ou através da leitura de um potenciômetro (controle local), a escolha do tipo de controle a ser realizado é através de um botão, caso o botão esteja pressionado o controle local será utilizado, senão o controle remoto atuará enviando a referência ao Arduino, que deverá atuar controlando o *pwm* aplicado ao Arduino.

Para a comunicação do cooler com o Arduino foi utilizada uma *shield* Ponte H L293D, que permite ligar motores DC e uma fonte de tensão de até 16V. Assim, é acoplado ao Arduino, a *shield Ethernet* e a nelas são acopladas a *shield* Ponte H.

O diagrama de funcionamento do sistema pode ser visto de uma forma mais simples na Figura 3.



**Figura 3 – Diagrama do funcionamento do sistema.**

### 1.3 Configuração da Rede

Para o funcionamento correto do *Shield Ethernet* para arduino é preciso configurar um IP fixo de acordo com o seu *MAC Address* configurado no código de configuração no arduino.

Nesse projeto, utilizamos um roteador D-Link para fazer a comunicação de rede entre os periféricos e o computador. A disposição dos dispositivos será explicada posteriormente com mais detalhes.

Essa configuração pode variar de acordo com a fabricante do roteador, no projeto será usado o roteador da D-Link DIR-900L, na Figura 4.



**Figura 4 – Roteador D-Link DIR-900L**

Com o computador conectado à rede, podemos saber o IP do roteador, *Gateway padrão* através do *prompt* de comando do Windows com o comando “*ipconfig*”. Como pode ser visualizado na Figura 5 abaixo.

```

C:\WINDOWS\system32\cmd.exe

C:\Users\WILLIAM>ipconfig

Configuração de IP do Windows

Adaptador Ethernet Ethernet:

    Estado da mídia. . . . . : mídia desconectada
    Sufixo DNS específico de conexão. . . . . :

Adaptador de Rede sem Fio Conexão Local* 3:

    Estado da mídia. . . . . : mídia desconectada
    Sufixo DNS específico de conexão. . . . . :

Adaptador de Rede sem Fio Conexão Local* 4:

    Estado da mídia. . . . . : mídia desconectada
    Sufixo DNS específico de conexão. . . . . :

Adaptador de Rede sem Fio Wi-Fi:

    Sufixo DNS específico de conexão. . . . . :
    Endereço IPv6 de link local . . . . . : fe80::119c:8a11:fac5:d512%10
    Endereço IPv4. . . . . : 192.168.0.103
    Máscara de Sub-rede . . . . . : 255.255.255.0
    Gateway Padrão. . . . . : 192.168.0.1

C:\Users\WILLIAM>

```

**Figura 5 – Prompt de comandos do Windows mostrando configurações de IP.**

Após entrarmos nas configurações do roteador utilizado o *IP Gateway Padrão*, devemos configurar os dispositivos como IP fixo para o correto funcionamento. Como mostra a Figura 6 abaixo nas configurações de LAN do roteador.



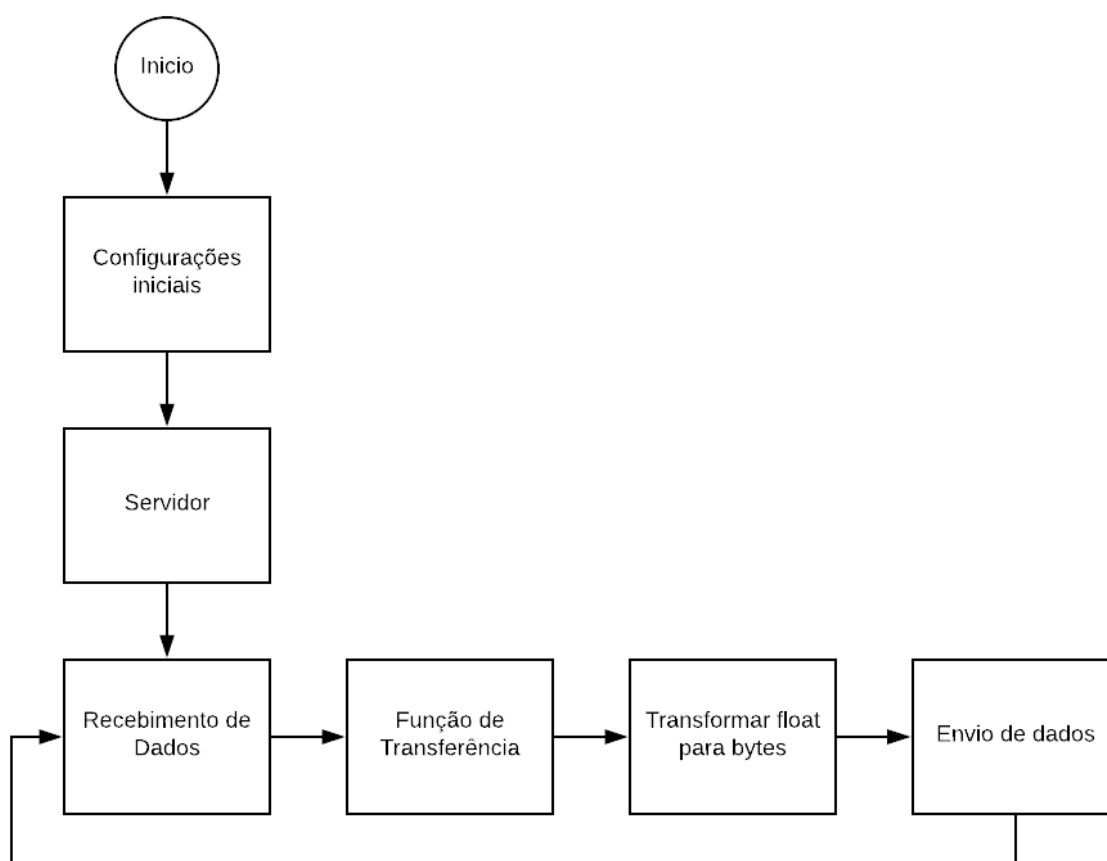
**Figura 6 – Tela de configuração do roteador utilizado para configurar o IP fixo dos dispositivos**



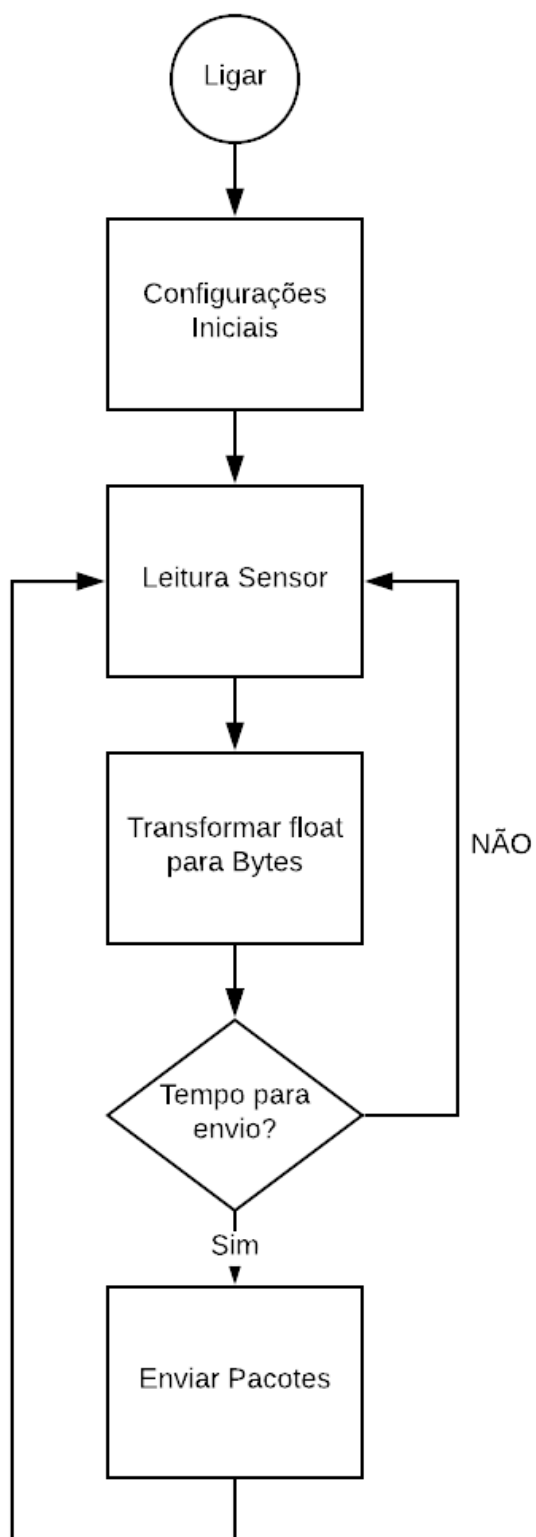
## CAPÍTULO 2 PRÁTICA

### 2.1 Fluxogramas

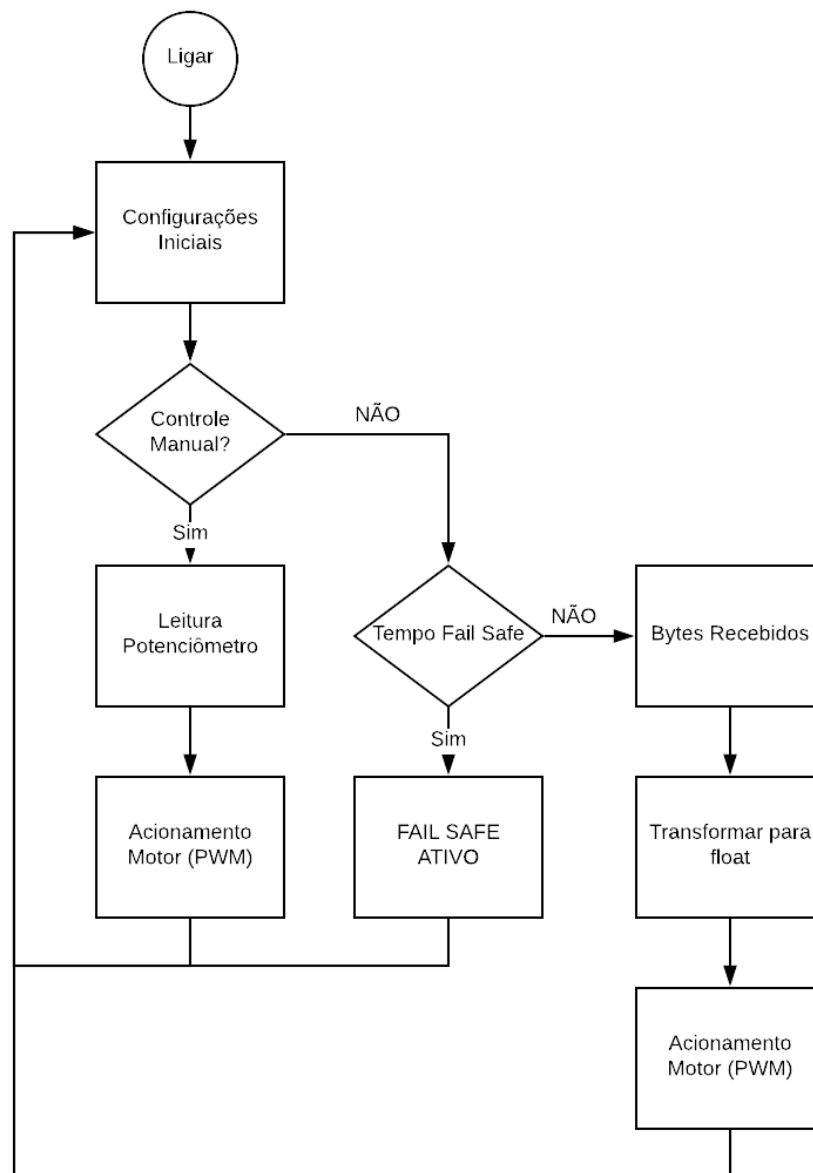
A seguir são apresentados os fluxogramas relativos aos algoritmos implementados para o sensor, o atuador e o controlador. Eles mostram como se deu a lógica do processo em forma de uma representação gráfica de fácil entendimento.



**Figura 7 – Fluxograma representando código do controlador (PC).**



**Figura 8 – Fluxograma representando o código do sensor (arduino 1).**



**Figura 9 – Fluxograma representando o código do atuador (arduino 2).**

## 2.2 Resultados obtidos

Na Figura 10, podemos visualizar os dados brutos em float 32 bits no PC onde se encontram a malha de controle do sistema. Nela estão presentes os valores recebidos de distância em centímetros do arduino 1 e os valores enviados em PWM para o controle do cooler no arduino 2.

```

C:\WINDOWS\system32\cmd.exe - python controlador.py
Microsoft Windows [versão 10.0.17134.112]
(c) 2018 Microsoft Corporation. Todos os direitos reservados.

C:\Users\WILLIAM>cd Desktop
C:\Users\WILLIAM\Desktop>cd python_udp
C:\Users\WILLIAM\Desktop\python_udp>python controlador.py
Server is open on port 9876

Distancia do Sensor: 49.4329910278
Potencia do Cooler: 100 % PWM: 255

Distancia do Sensor: 43.2324905396
Potencia do Cooler: 100 % PWM: 255

Distancia do Sensor: 33.5463905334
Potencia do Cooler: 95.8468300956 % PWM: 244.409416744

Distancia do Sensor: 27.4564800262
Potencia do Cooler: 78.4470857893 % PWM: 200.040068763

Distancia do Sensor: 29.4344501495
Potencia do Cooler: 84.0984289987 % PWM: 214.450993947

```

Figura 10 – Prompt de Comando do PC.

Através do *software Wireshark*, que se trata de um analisador de protocolos que nos permite capturar e navegar interativamente no tráfego de uma rede de computadores em tempo de execução, podemos visualizar a troca de mensagens, via protocolo UDP, entre os 3 dispositivos conectados na rede, mostrados através do seu respectivo IP. Sendo eles o PC, o arduino 1, responsável pelo sensor, e o arduino 2 responsável pelo atuador. Como podemos ver na Figura 11, o fluxo de dados segue o diagrama de blocos da Figura 3, ou seja, arduino 1 (IP 192.168.0.101) que recebe os dados do sensor envia para o PC (IP 192.168.0.103), onde está a malha de controle. O PC, a partir dos dados recebidos, envia dados de informação PWM, para o arduino 2 (IP 192.168.0.102) que aciona o cooler, e assim sucessivamente.

No.	Time	Source	Destination	Protocol	Length	Info
5	0.72585400	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
6	0.72956900	192.168.0.103	192.168.0.102	UDP	46	Source port: 54875 Destination port: 9876
21	5.72975100	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
22	5.73141900	192.168.0.103	192.168.0.102	UDP	46	Source port: 49751 Destination port: 9876
41	10.72989200	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
42	10.73257900	192.168.0.103	192.168.0.102	UDP	46	Source port: 54756 Destination port: 9876
60	15.73809100	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
61	15.74086500	192.168.0.103	192.168.0.102	UDP	46	Source port: 57490 Destination port: 9876
102	20.74341900	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
103	20.74660400	192.168.0.103	192.168.0.102	UDP	46	Source port: 53048 Destination port: 9876
118	25.92568700	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
119	25.92867300	192.168.0.103	192.168.0.102	UDP	46	Source port: 62878 Destination port: 9876
129	30.81799100	192.168.0.101	192.168.0.103	UDP	60	Source port: 9876 Destination port: 9876
130	30.82149000	192.168.0.103	192.168.0.102	UDP	46	Source port: 62879 Destination port: 9876

Frame 5: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0	
Ethernet II, Src: Gheosa_0e:b6:07 (90:a2:da:0e:b6:07), Dst: 80:a5:89:e7:72:73 (80:a5:89:e7:72:73)	
Internet Protocol Version 4, Src: 192.168.0.101 (192.168.0.101), Dst: 192.168.0.103 (192.168.0.103)	
User Datagram Protocol, Src Port: 9876 (9876), Dst Port: 9876 (9876)	
Data (4 bytes)	
Data: 1a61d742	
[Length: 4]	

Offset	Hex	ASCII
0000	80 a5 89 e7 72 73 90 a2 da 0e b6 07 08 00 45 00	.....E.
0010	00 20 00 1f 40 00 80 11 78 91 c0 a8 00 05 c0 a8	...8...X...E.
0020	00 67 26 94 26 94 00 0c 3e ad 1a 61 d7 42 00 00	.9&6...>..a.B..
0030	00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	.....

Figura 11 – Fluxo de dados através do Wireshark.

## CONCLUSÃO

Este projeto nos permitiu compreender conceitos estudados em aula teórica aplicados na prática. A rede foi projetada empregando sockets que garantam a intercomunicação bidirecional entre os processos. Mas especificamente os sockets com protocolo UDP foram utilizados, especificando a camada de transporte. Além desse parâmetro a arquitetura Ethernet, responsável pela interconexão para redes locais que se baseia no envio de pacotes, foi o modelo da camada física escolhida. Esses fatores nos permitiram cumprirmos com o objetivo de fazermos uma rede, usando sistemas embarcados e os sockets UDP responsável pela supervisão e controle remotos da bola no tubo.

O fluxo de dados na rede foi capturado pelo *software* Wireshark, e os dados brutos que atravessaram a rede foram mostrados no *prompt* de comando do controlador, mostrando que o propósito deste projeto foi alcançado. Além disso, o controle da bola de modo remoto, através da rede, e localmente, através do botão, foi realizado com sucesso. Assim dentro das muitas dificuldades encontradas, o objetivo principal do projeto foi alcançado.

## ANEXO A

Como explicado anteriormente no relatório, a linguagem de programação utilizada foi o Python para o servidor de controle do sistema. Na Figura 12 abaixo podemos visualizar o algoritmo.

```

1  """ Server Malha de Controle UDP"""
2
3  import socket #biblioteca socket para a utilização protocolo UDP
4  import struct #biblioteca para a conversão de bytes para float
5
6  UDP_IP = "192.168.0.103" #ip do servidor de controle
7  UDP_PORT = 9876 #porta de comunicação do servidor
8
9  UDP_IP_ARD = "192.168.0.102" #ip arduino atuação do sistema
10 UDP_PORT_ARD = 9876 #porta de comunicação do arduino de atuação
11 server_address_ARD = (UDP_IP_ARD, UDP_PORT_ARD) #configuração para a comunicação com o arduino
12
13 sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #criação do socket para UDP
14 server_address = (UDP_IP, UDP_PORT) #configuração do servidor de controle
15 sock.bind(server_address) #inicialização do servidor
16
17 print "Server is open on port", UDP_PORT
18
19 while True: #servidor roda continuamente
20     data, addr = sock.recvfrom(4) #dados recebidos e endereço de recebimento
21     value = struct.unpack('f', data) #conversão de bytes para float
22
23     pwm = value[0]
24     pwm = pwm*255/35 #algoritmo de controle a partir dos dados recebidos de distância do sensor
25
26     if pwm > 255: #item de segurança para não enviar dados fora de escala
27         pwm = 255
28     elif pwm < 0:
29         pwm = 0
30
31     print "value message:", value[0] #visualização de valores recebidos
32     print "pwm:", pwm #visualização dos valores pwm que será enviados ao atuador
33
34     pwmpack = struct.pack('f', pwm) #empacotamento de dados float em bytes para envio
35
36     sock1 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #criação do socket para envio para o arduino de atuação
37     sock1.sendto(pwmpack, server_address_ARD) #envio dos dados para o arduino de atuação

```

**Figura 12 – Algoritmo em Python do Controlador.**

Abaixo pode-se visualizar o algoritmo nos arduinos, um para o sensor e outro para a atuação do sistema, que foi utilizado para a comunicação e controle do sistema. Na Figura 13 temos o algoritmo do sensor.

```

1  #include <SPI.h>
2  #include <Ethernet.h>
3  #include <EthernetUdp.h>
4  #include <Ultrasonic.h>
5  // Configuração da Rede, MAC, IP e Porta
6  byte mac[] = {
7      0x90, 0xA2, 0xDA, 0x0E, 0xB6, 0x07
8  };
9  IPAddress ip(192, 168, 0, 101);
10 IPAddress ip_pc(192,168,0,103);
11 unsigned int localPort = 9876;
12 char ReplyBuffer[5];
13 int outputTiming = 1000;
14 Ultrasonic sensor_frente(39, 37);
15 EthernetUDP Udp;
16 typedef union //transformar float em array de bytes
17 {
18     float number;
19     uint8_t bytes[4];
20 } FLOATUNION_t;
21 void setup() {
22     Ethernet.begin(mac, ip);
23     Udp.begin(localPort);
24     Serial.begin(9600);
25     pinMode(41,OUTPUT);
26     digitalWrite(41,HIGH);
27     pinMode(35,OUTPUT);
28     digitalWrite(35,LOW);
29 }
30 void loop() {
31     static unsigned long loopTime = 0; //Variáveis de tempo
32     static unsigned long time1 = 0;
33     loopTime = millis();
34     float dist = sensor_frente.timing(); //leitura do sensor e saída em centímetros
35     dist = dist / 58;
36     FLOATUNION_t myFloat;
37     myFloat.number = dist;
38     for(int i=0; i<4; i++)
39     {
40         ReplyBuffer[i] = myFloat.bytes[i]; //variavel em bytes para o envio na rede
41     }
42
43     if((loopTime - time1) >= outputTiming) //envio dos dados uma vez a cada segundo
44     {
45         time1 = loopTime;
46         Udp.beginPacket(ip_pc, localPort); //criação do pacote
47         Udp.write(ReplyBuffer); //escrita no pacote
48         Udp.endPacket(); //finalização e envio do pacote
49     }
50 }

```

**Figura 13 – Algoritmo em C para a leitura do sensor.**

Nas Figuras 14 temos o algoritmo do arduino para a atuação do cooler, em malha fechada (controle remoto) e em malha aberta através do potenciômetro (controle local), no sistema de controle.

```

1  #include <AFMotor.h>
2  #include <SPI.h>
3  #include <Ethernet.h>
4  #include <EthernetUdp.h>
5  // Configuração da Rede, MAC, IP e Porta
6  byte mac[] = {0x90,0xA2,0xDA,0x0E,0xB6,0x06};
7  };
8  IPAddress ip(192,168,0,102);
9  int localPort = 9876;
10 EthernetUDP Udp;
11 AF_DCMotor motor(4); //Seleciona o motor 1
12 //Variaveis Globais
13 float anterior = 2;
14 char packetBuffer[4]; //buffer para o envio de informacoes pela rede via UDP
15 int pinBotao = 40; //Declara pino do botao
16 bool fail = false; //caso erro de comunicacao ir para estado seguro
17 int outputTiming = 10000; //fail safe
18 void setup() { //start
19     Serial.begin(9600);
20     pinMode(A15, INPUT); //potenciometro
21     pinMode(pinBotao, INPUT); //botao digital
22     motor.setSpeed(255);
23     motor.run(RELEASE);
24     Ethernet.begin(mac, ip);
25     Udp.begin(localPort);
26     Serial.println("Start");
27 }
28 void loop() {
29     //Variaveis de tempo
30     static unsigned long loopTime = 0;
31     static unsigned long timel = 0;
32     loopTime = millis();
33     float potenciometro = analogRead(A15); // valor potenciometro para controle
34     int estado_botao = digitalRead(pinBotao); //estado botao
35     int packetSize = Udp.parsePacket(); //tamanho pacote recebido
36     Udp.read(packetBuffer, 4); // leitura do pacote
37     float pwm;
38     pwm = *(float*)packetBuffer; //transformacao de bytes para float
39     if((loopTime - timel) >= outputTiming)
40     {
41         timel = loopTime;
42         if ((anterior - pwm) == 0)
43         {
44             if ((pwm != 0) || (pwm!=255))
45             {
46                 fail = true; //FAIL SAFE
47             }
48         }
49         anterior = pwm;
50     }
51     if (estado_botao == HIGH)
52     {
53         fail = false;
54         anterior = 0;
55         //CONTROLE MANUAL
56         pwm = map(potenciometro, 0, 1023, 0, 255);
57         if (pwm > 255) //segurança para nao entrar fora de escala
58         {
59             pwm = 255;
60         }
61         else if (pwm < 0)
62         {
63             pwm = 0;
64         }
65         motor.setSpeed(pwm); //acionamento motores
66         motor.run(FORWARD);
67     }
68     else
69     {
70         //CONTROLE PELA REDE
71         if (fail == false)
72         {
73             motor.setSpeed(pwm);
74             motor.run(FORWARD);
75         }
76     }
77     if (fail == true) // FAIL SAFE
78     {
79         Serial.println("FAIL");
80         motor.setSpeed(0);
81         motor.run(RELEASE);
82     }
83 }

```

Figura 14 – Algoritmo em C para o atuador.