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

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.

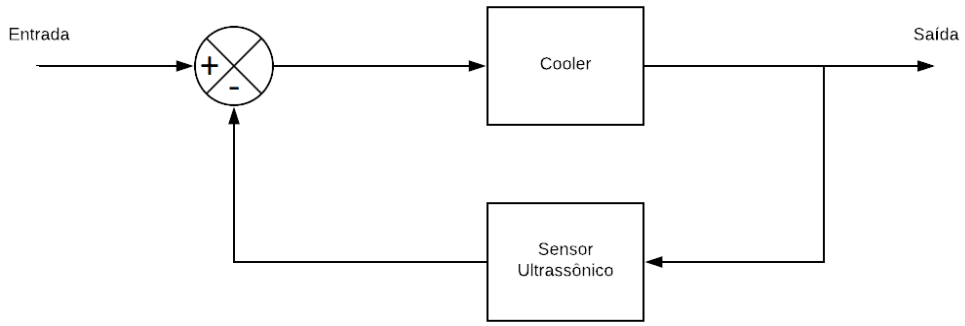

Figura 1 – Digrama de blocos em malha fechada

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.

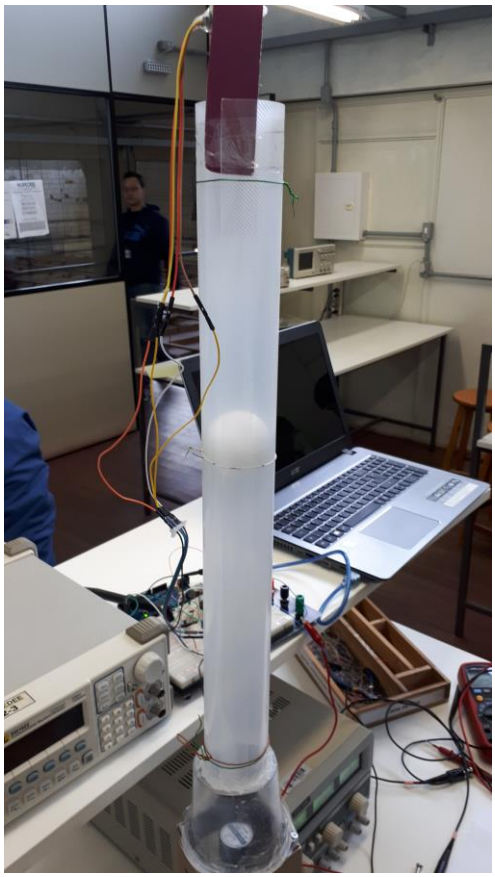

Figura 2 – Modelo físico do projeto

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.

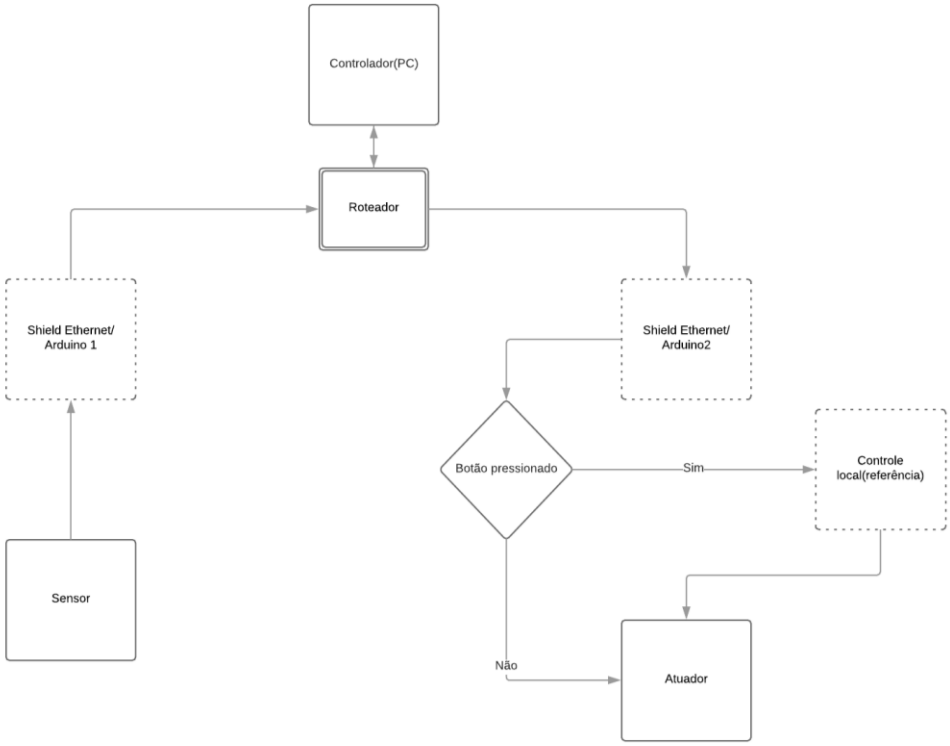

Figura 3 – Diagrama do funcionamento do sistema.

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.

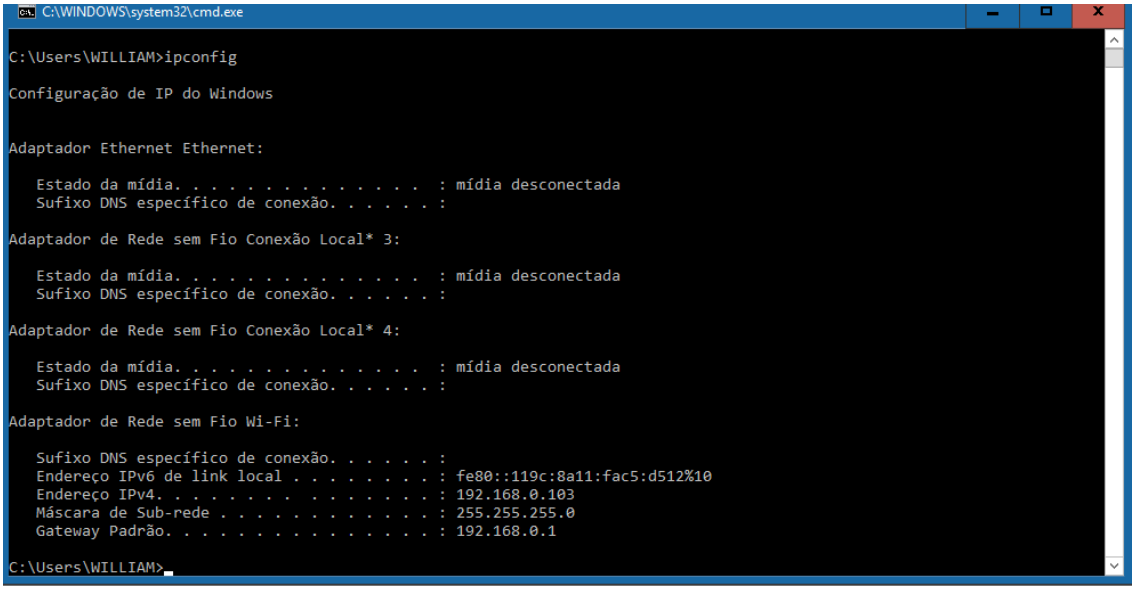

Figura 5 – Prompt de comandos do Windows mostrando configurações de IP.

Após entrarmos nas configurações do roteador utilizado o *IP Gateway Padrão*, devemos configurar os dispostivos 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.

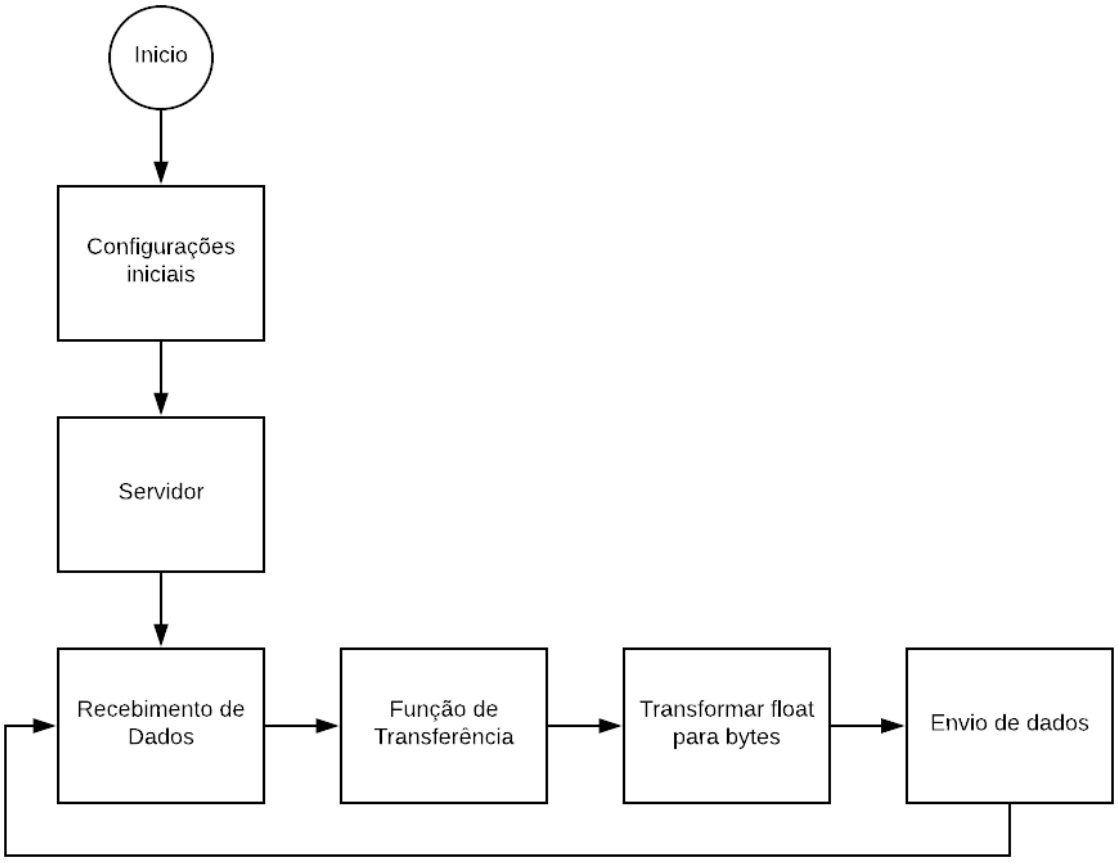

Figura 7 – Fluxograma representando código do controlador (PC).

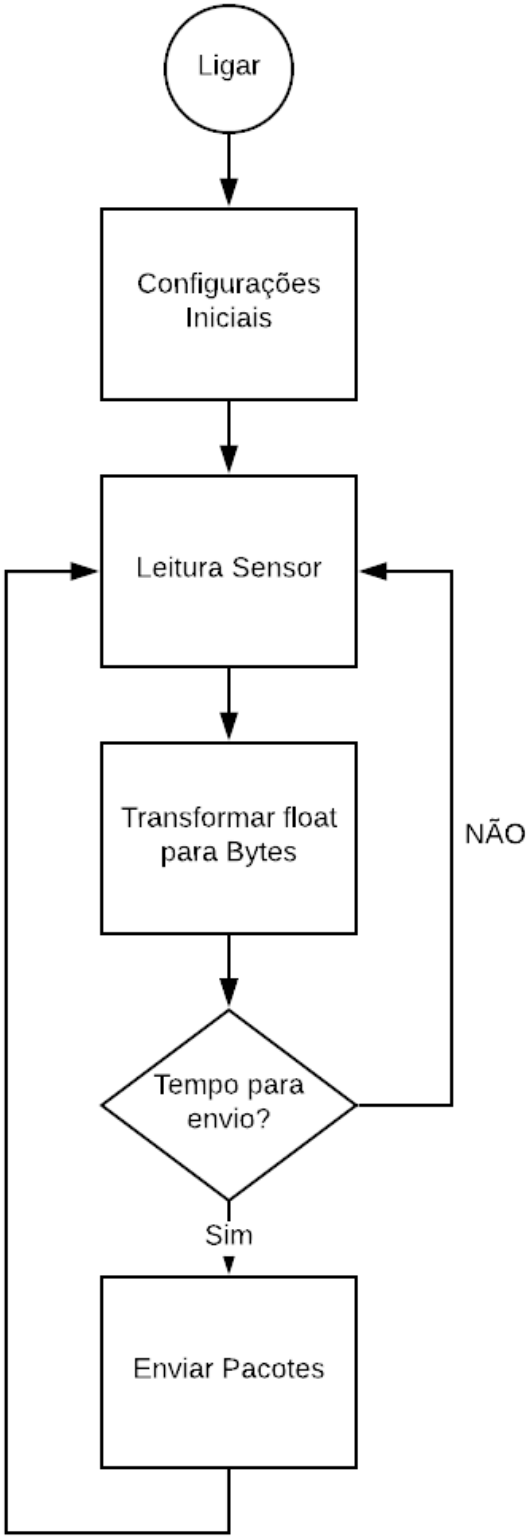

Figura 8 – Fluxograma representando o código do sensor (arduino 1).

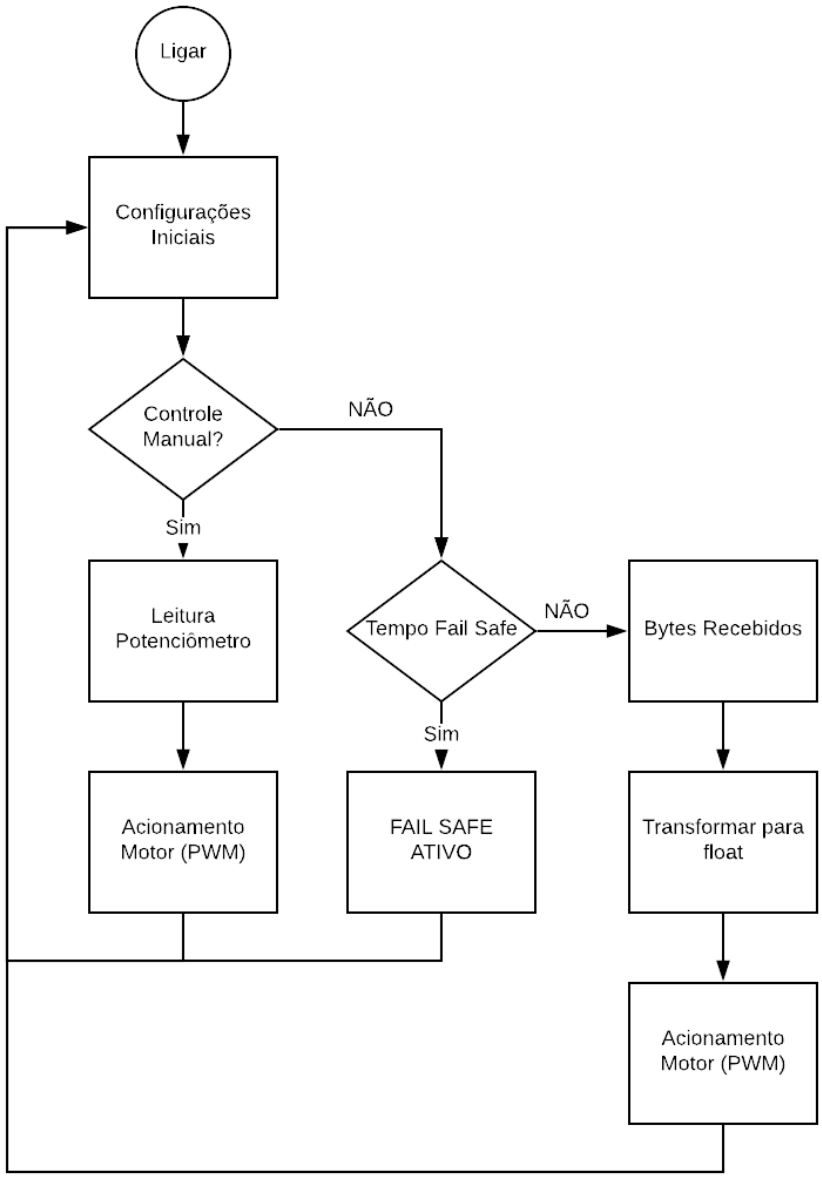

Figura 9 – Fluxograma representando o código do atuador (arduino 2).

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.

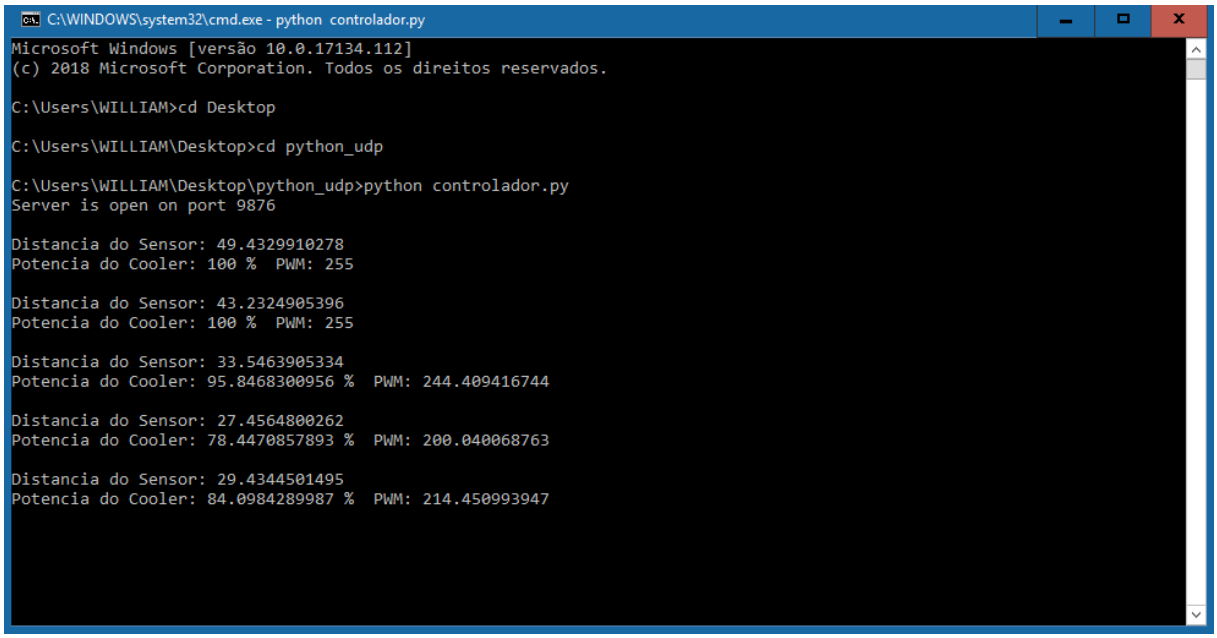

Figura 10 – Prompt de Comando do PC.

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.

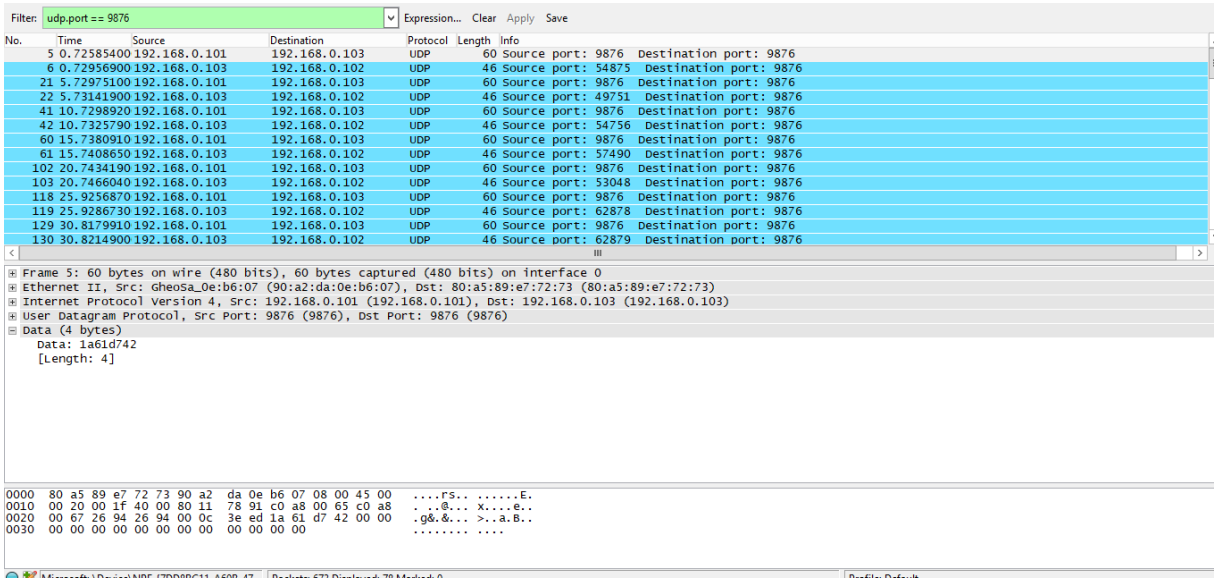

Figura 11 – Fluxo de dados através do Wireshark.

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.

```
""" Server Malha de Controle UDP"""

import socket #biblioteca socket para a utilização protocolo UDP
import struct #biblioteca para a conversão de bytes para float

UDP_IP = "192.168.0.103" #ip do servidor de controle
UDP_PORT = 9876 #porta de comunicação do servidor

UDP_IP_ARD = "192.168.0.102" #ip arduino atuação do sistema
UDP_PORT_ARD = 9876 #porta de comunicação do arduino de atuação
server_address_ARD = (UDP_IP_ARD, UDP_PORT_ARD) #configuração para a comunicação com o arduino

sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #criação do socket para UDP
server_address = (UDP_IP, UDP_PORT) #configuração do servidor de controle
sock.bind(server_address) #inicialização do servidor

print "Server is open on port", UDP_PORT

while True: #servidor roda continuamente
    data, addr = sock.recvfrom(4) #dados rebebidos e endereço de recebimento
    value = struct.unpack('f', data) #conversão de bytes para float

    pwm = value[0]
    pwm = pwm*255/35 #algoritmo de controle a partir dos dados recebidos de distância do sensor

    if pwm > 255: #item de segurança para nao enviar dados fora de escala
        pwm = 255
    elif pwm < 0:
        pwm = 0

    print "value message:", value[0]    #visualização de valores recebidos
    print "pwm:", pwm #visualização dos valores pwm que será enviados ao atuador

    pwmpack = struct.pack('f', pwm) #empacotamento de dados float em bytes para envio

    sock1 = socket.socket(socket.AF_INET, socket.SOCK_DGRAM) #criação do socket para envio para o arduino de atuação
    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.

```
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
#include <Ultrasonic.h>
// Configuração da Rede, MAC, IP e Porta
byte mac[] = {
  0x90, 0xA2, 0xDA, 0x0E, 0xB6, 0x07
};
IPAddress ip(192, 168, 0, 101);
IPAddress ip_pc(192,168,0,103);
unsigned int localPort = 9876;
char  ReplyBuffer[5];
int outputTiming = 1000;
Ultrasonic sensor_frente(39, 37);
EthernetUDP Udp;
typedef union //transformar float em array de bytes
{
  float number;
  uint8_t bytes[4];
} FLOATUNION_t;
void setup() {
  Ethernet.begin(mac, ip);
  Udp.begin(localPort);
  Serial.begin(9600);
  pinMode(41,OUTPUT);
  digitalWrite(41,HIGH);
  pinMode(35,OUTPUT);
  digitalWrite(35,LOW);
}
void loop() {
  static unsigned long  loopTime = 0; //Variáveis de tempo
  static unsigned long  time1 = 0;
  loopTime = millis();
  float dist = sensor_frente.timing(); //leitura do sensor e saida em centimetros
  dist = dist / 58;
  FLOATUNION_t myFloat;
  myFloat.number = dist;
  for(int i=0; i<4; i++)
  {
    ReplyBuffer[i] = myFloat.bytes[i]; //variavel em bytes para o envio na rede
  }

  if((loopTime - time1) >= outputTiming) //envio dos dados uma vez a cada segundo
  {
    time1 = loopTime;
    Udp.beginPacket(ip_pc, localPort); //criação do pacote
    Udp.write(ReplyBuffer); //escrita no pacote
    Udp.endPacket();    //finalização e envio do pacote
  }
}
```

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.

```
#include <AFMotor.h>
#include <SPI.h>
#include <Ethernet.h>
#include <EthernetUdp.h>
// Configuração da Rede, MAC, IP e Porta
byte mac[] = {0x90,0xA2,0xDA,0x0E,0xB6,0x06
};
IPAddress ip(192,168,0,102);
int localPort = 9876;
EthernetUDP Udp;
AF_DCMotor motor(4); //Seleciona o motor 1
//Variaveis Globais
float anterior = 2;
char packetBuffer[4];  //buffer para o envio de informacoes pela rede via UDP
int pinBotao = 40; //Declara pino do botao
bool fail = false; //caso erro de comunicacao ir para estado seguro
int outputTiming = 10000; //fail safe
void setup() { //start
  Serial.begin(9600);
  pinMode(A15, INPUT); //potenciometro
  pinMode(pinBotao,INPUT); //botao digital
  motor.setSpeed(255);
  motor.run(RELEASE);
  Ethernet.begin(mac, ip);
  Udp.begin(localPort);
  Serial.println("Start");
}
void loop() {
  //Variaveis de tempo
  static unsigned long  loopTime = 0;
  static unsigned long  time1 = 0;
  loopTime = millis();
  float potenciometro = analogRead(A15); // valor potenciometro para controle
  int estado_botao = digitalRead(pinBotao); //estado botao
  int packetSize = Udp.parsePacket(); //tamanho pacote recebido
  Udp.read(packetBuffer, 4); // leitura do pacote
  float pwm;
  pwm = *(float*)packetBuffer; //tranformacao de bytes para float
  if((loopTime - time1) >= outputTiming)
  {
    time1 = loopTime;
    if ((anterior - pwm) == 0)
    {
      if ((pwm != 0)||(pwm!=255))
      {
        fail = true;      //FAIL SAFE
      }
    }
    anterior = pwm;
  }
  if (estado_botao == HIGH)
  {
    fail = false;
    anterior = 0;
    //CONTROLE MANUAL
    pwm = map(potenciometro, 0, 1023, 0, 255);
    if (pwm > 255) //segurança para nao entrar fora de escala
    {
      pwm = 255;
    }
    else if (pwm < 0)
    {
      pwm = 0;
    }
    motor.setSpeed(pwm); //acionamento motores
    motor.run(FORWARD);
  }
  else
  {
    //CONTROLE PELA REDE
    if (fail == false)
    {
      motor.setSpeed(pwm);
      motor.run(FORWARD);
    }
  }
  if (fail == true) // FAIL SAFE
  {
    Serial.println("FAIL");
    motor.setSpeed(0);
    motor.run(RELEASE);
  }
}
```

Figura 14 – Algoritmo em C para o atuador.