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

USO DE ARDUINO COM PROTROCOLO MODBUS TCP PARA CONTROLE DE NÍVEL DA BANCADA MULTIPROCESSOS

RELATÓRIO DA DISCIPLINA DE INTEGRAÇÃO DE REDES INDUSTRIAIS
Prof. Frederico Menine Schaf

Davi Calil
Douglas Maciel
Marcos Silveira
William Rocha

Santa Maria, RS, Brasil

2018

SUMÁRIO

INTRODUÇÃO

O trabalho foi realizado no laboratório do NUPEEDE (Núcleo de Pesquisa e Desenvolvimento em Engenharia Elétrica) da UFSM (Universidade Federal de Santa Maria). E baseia-se no controle de nível da bancada didática multiprocessos, utilizando o protocolo Modbus TCP e o microcontrolador Arduino Mega, como mestre.

Para tornar isso possível, foi utilizado um *Shield* Ethernet (com o intuito de conectar o Arduino à rede Ethernet). E, além disso, foram utilizados um display LCD para mostrar o "status" do controle do processo, botões (*start* e emergência) e um potenciômetro (visando configurar o *setpoint*).

CAPÍTULO 1 REVISÃO TEÓRICA

1.1 Protocolo Modbus

O protocolo Modbus é uma estrutura de mensagem aberta desenvolvida pela Modicon na década de 70. A Modicon foi posteriormente adquirida pela Schneider e os direitos sobre o protocolo foram liberados pela Organização Modbus. Esse protocolo se caracteriza pela padronização de mensagens e é baseado no modelo Mestre-Escravo com os escravos impedidos de se comunicarem entre si. Muitos equipamentos industriais utilizam o Modbus como protocolo de comunicação, e graças às suas características, este protocolo também tem sido utilizado em uma vasta gama de aplicações.

O meio físico do Modbus não é especificado, a estrutura dos quadros funciona sobre RS-232, RS-485, Ethernet, Fibra Óptica e RF. O padrão funciona apenas com dois tipos de quadros: de consulta (*request*) e de resposta (*reply*). Também permite mensagens em *broadcast*. Na Figura 1 mostra o quadro do Modbus e na Figura 2 pode ser visto o *frame* do MODBUS.

Figura 1 – Quadro do Modbus. (Fonte: material de aula).

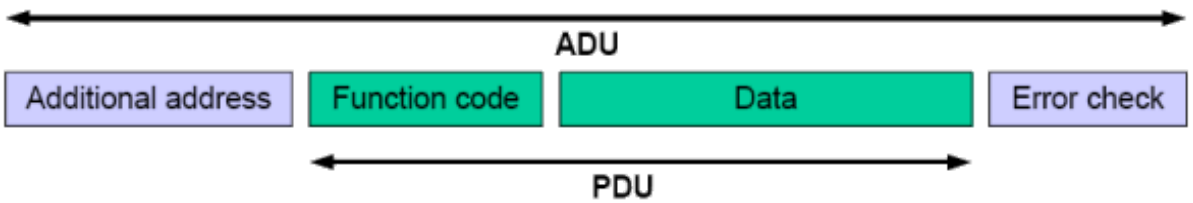

Figura 2 – Frame do Modbus. (Fonte: material de aula).

O meio físico utilizado neste projeto é o Ethernet, uma variação do protocolo, o Modbus TCP/IP. Os dados são encapsulados em quadros Ethernet, pacotes IP e segmentos TCP. A porta TCP utilizada é a padrão do protocolo, porta 502. Dentro do protocolo Modbus, existem tipos de variáveis, *Discrete Inputs, Coils, Input Registers* e *Holding Registers*.

1.1.1 Estrutura do protocolo

Para entendermos a estrutura do protocolo Modbus, podemos fazer uma relação com o modelo ISO/OSI que se trata de um modelo de rede de computador, divididos em sete camadas, com o objetivo de ser um padrão para protocolos de comunicação. O modelo OSI não se trata de uma arquitetura de rede, mas serve como base para delimitar o que cada camada deve fazer. A Figura 3 apresenta as sete camadas do modelo OSI, bem como o modelo de referência do TCP/IP no qual o Modbus TCP/IP se baseou. Já a Figura 4 mostra a pilha de comunicação do Modbus associado ao modelo ISO/OSI.

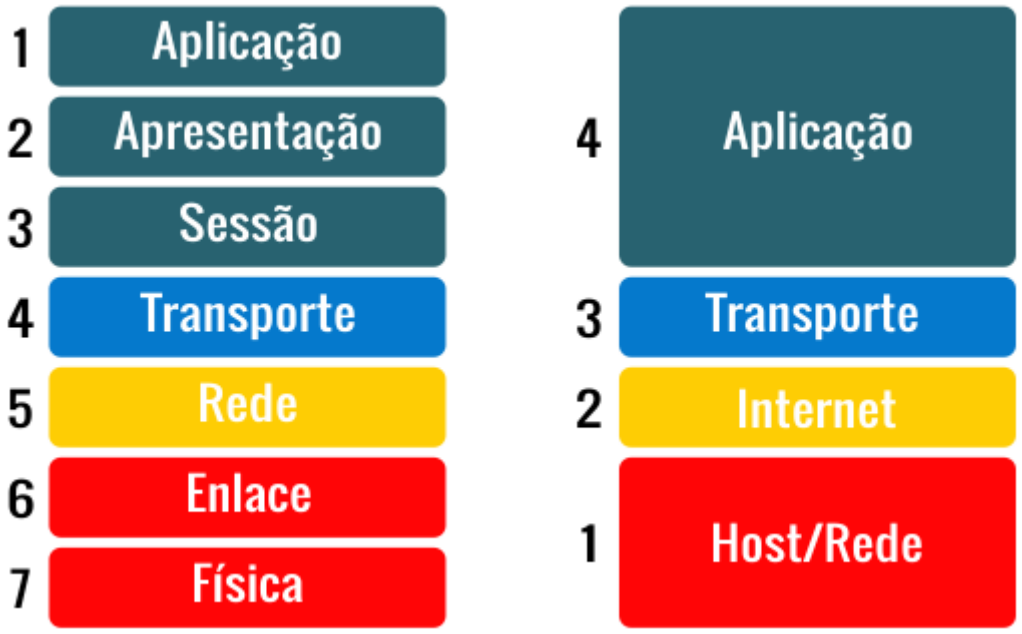

Figura 3 - Relação entre modelo OSI e TCP/IP. (Fonte: https://www.uniaogeek.com.br/arquitetura-de-redes-tcpip/).

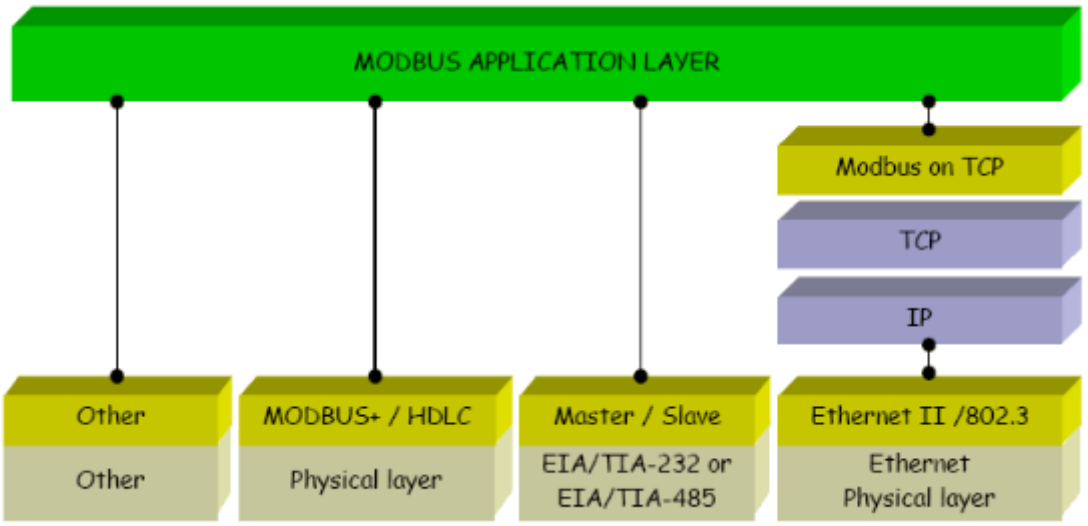

Figura 4 - Pilha de comunicação do Modbus associado ao modelo ISO/OSI. (Fonte: material de aula).

Não há a necessidade de um protocolo implementar todas as camadas do modelo OSI. Dessa forma, o Modbus TCP, baseado no protocolo TCP/IP, combina aspectos das camadas de apresentação e de sessão dentro da sua camada de aplicação, além de combinar as camadas física e de enlace em uma só camada.

Definido a estrutura do Modbus TCP, a Figura 2 da seção anterior mostra o frame do Modbus para o envio de mensagens. Neste frame é definida uma única PDU, que independe do meio físico e possui um tamanho máximo de 253 bytes. Além da PDU o Modbus adiciona na ADU 7 bytes chamados de MBAP (*MODBUS Application Protocol*) formados pelos campos: *Transaction Identifier, Protocol Identifier, Lenght e Unit Identifier*. Assim, o modelo de mensagem do Modbus TCP/IP é descrito pela Figura 5.

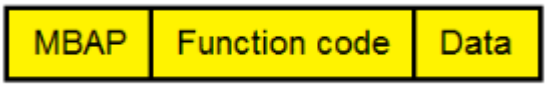

Figura 5 - Modelo de mensagem Modbus TCP/IP. (Fonte: https://www.embarcados.com.br/protocolo-modbus/).

O campo de checagem de erro não é implementado devido ao fato do frame ethernet utilizar o CRC-32 para checagem de erros, tornando desnecessário outro campo de checagem.

O código da função é onde o mestre especifica o tipo de serviço ou função solicitada ao escravo (leitura, escrita, etc.) e onde cada função é utilizada para acessar um tipo específico de dado.

A Figura 6 contém informações sobre alguns códigos de função e a Figura 7 mostra os tipos de dados mais comuns.

				Function Codes		
				code	*Sub code*	*(hex)*
Data Access	Bit access	Physical Discrete Inputs	Read Discrete Inputs	02		02
		Internal Bits Or Physical coils	Read Coils	01		01
			Write Single Coil	05		05
			Write Multiple Coils	15		0F
	16 bits access	Physical Input Registers	Read Input Register	04		04
		Internal Registers Or Physical Output Registers	Read Holding Registers	03		03
			Write Single Register	06		06
			Write Multiple Registers	16		10
			Read/Write Multiple Registers	23		17
			Mask Write Register	22		16
			Read FIFO queue	24		18
	File record access		Read File record	20		14
			Write File record	21		15
Diagnostics			Read Exception status	07		07
			Diagnostic	08	00-18,20	08
			Get Com event counter	11		OB
			Get Com Event Log	12		0C
			Report Slave ID	17		11
Other			Read device Identification	43	14	2B
			Encapsulated Interface Transport	43	13,14	2B

Figura 6 - Códigos de função. (Fonte: material de aula).

Primary Tables	Object Type	Type of
Discrete Input	Single bit	Read-Only
Coils	Single bit	Read-Write
Input Registers	16-bit word	Read-Only
Holding Registers	16-bit word	Read-Write

Figura 7 - Tipos de dados no Modbus. (Fonte: material de aula).

Destes códigos de função, os mais utilizados são os *holding registers,* que são registradores universais usados para escrita e leitura dos valores dos registradores de memória, e podem ser mapeados para entradas, saídas e dados de configuração, onde cada registrador são 2 *bytes*, ou seja, 16 *bits*.

A Modicon tem uma convenção para o endereço lógico dos registradores, sendo para os *holdings registers* valores de 40001 a 49999. Na prática todos os endereços lógicos variam

de 0 a 9998 e sua identificação está associada ao tipo do serviço, por exemplo: para os *coils* o endereço lógico varia de 00001 a 09999.

Antes da transmissão de dados, eles podem ser formatados de 4 modos: o RTU é o mais utilizado pelo fato de compor o frame em binário, onde cada byte contém dois caracteres de 4 bits cada, o que difere e ocupa menos recursos da rede que o ASCII que é outro modo de transmissão, pois um byte do formato ASCII é formado por 7 bits. Há ainda o formato Modbus TCP/IP cujo os dados são encapsulados em pacotes enviados pela rede Ethernet via TCP, utilizando o CSMA/CD como método de acesso ao meio que gerencia as possíveis colisões dando taxas de prioridade ao pacote enviado. E por último o Modbus Plus de propriedade da Schneider Electric que necessita de licença de uso.

A Figura 8 mostra as diferenças entre pacotes RTU e TCP.

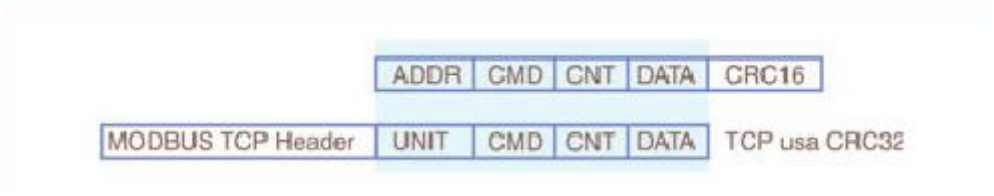

Figura 8 - Diferenças entre pacotes RTU e TCP. (Fonte: http://www.newtoncbraga.com.br/.../12089-como-funciona-o-protocolo-modbus-r0001).

Nota-se que os pacotes Modbus TCP não utilizam um campo para checagem de erros e empregam o cabeçalho MBAP, como dito anteriormente.

A Figura 9 mostra um exemplo de comunicação entre mestre e escravo.

Requisição do Mestre
(Master Request)

Nome do campo	Hexa	ASCII	RTU
Cabeçalho		:	Nenhum
Endereço do escravo	06	06	0000 0110
Código da função	03	03	0000 0011
Endereço de início HI	00	00	0000 0000
Endereço de início LO	6B	6B	0110 1011
Número de reg. HI	00	00	0000 0000
Número de reg. LO	03	03	0000 0011
Controle de erro	CRC	LRC(2)	CRC(2)
Trailer		CR LF	Nenhum

Resposta do Escravo
(Slave Reply)

Nome do campo	Hexa	ASCII	RTU
Cabeçalho		:	Nenhum
Endereço do escravo	06	06	0000 0110
Código da função	03	03	0000 0011
Quantidade de bytes	06	06	0000 0110
Dado HI	02	02	0000 0010
Dado LO	2B	2B	0010 1011
Dado HI	00	00	0000 0000
Dado LO	00	00	0000 0000
Dado HI	00	00	0000 0000
Dado LO	63	63	0110 0011
Controle de erro	CRC	LRC(2)	CRC(2)
Trailer		CR LF	Nenhum

Figura 9 - Frame de comunicação entre mestre e escravo. (Fonte: material de aula).

Percebemos no frame de requisição do mestre que ele possui um cabeçalho, um campo para o endereço do escravo, outro campo para o código da função, o endereço de início de leitura dos registradores, separado em parte alta (HI) e parte baixa (LO) e o número de registradores que serão lidos também com parte alta e parte baixa. Assim, o mestre requisita a leitura de *holding register* (código da função 3) de três registradores (parte alta :00 e parte baixa: 03) começando pelo endereço (6B = 107 + 40001) 40108 do primeiro registrador.

E o escravo responde repetindo o endereço e o código da função, envia a quantidade de bytes e os valores dos registradores que são divididos em parte alta e parte baixa. Lembrando que um registrador possui 16 *bits* que equivale a 2 *bytes* que são divididos em parte alta e parte baixa. Desta forma o escravo respondeu com 6 *bytes* (3 registradores como solicitado pelo mestre), e com os dados de cada registrador separados em parte alta e parte baixa.

1.2 Arduino

O microcontrolador Arduino, Figura 10, é uma plataforma de prototipagem livre de placa única, as unidades são constituídas por controlador Atmel AVR de 8 *bits*, pinos digitais e analógicos de entrada e saída, entrada USB ou serial, que permite a comunicação com

computadores e possui código aberto. O Arduino não possui recursos de rede, mas pode ser combinada com outras plataformas criando extensões chamadas de *Shields*.

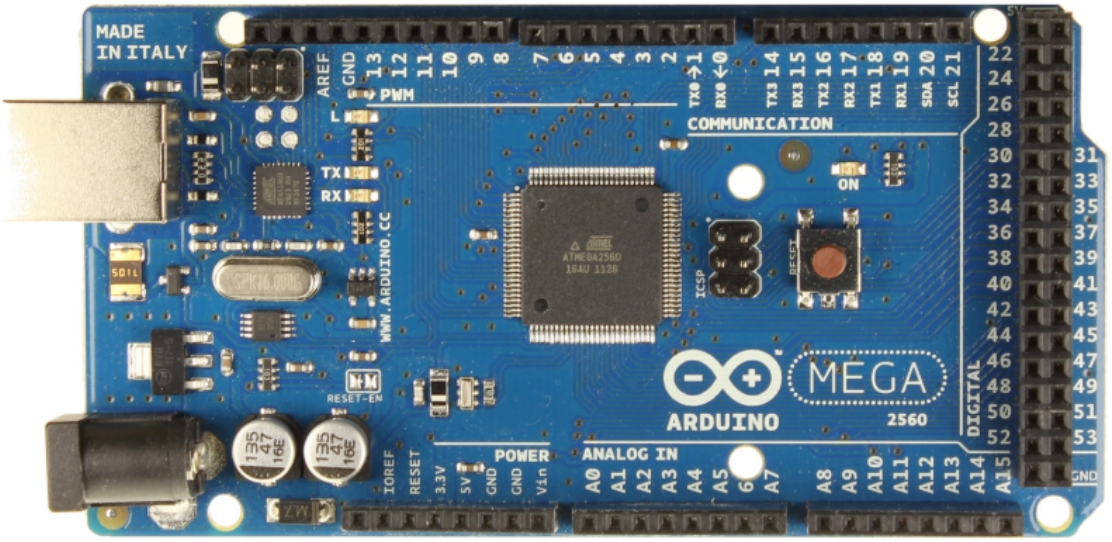

Figura 10 – Arduino Mega 2560.

1.2.1 Shield Ethernet

O S*hield* Ethernet permite conectar uma placa Arduino a uma rede Ethernet de forma que suas funções possam ser controladas pela rede, como pode ser visto na Figura 11. Este *shield* baseia-se no chip WIZnet Ethernet W5100, que fornece acesso à rede (IP) nos protocolos TCP ou UDP e é facilmente utilizado usando as bibliotecas Ethernet Library e SD Library.

Figura 11 – Shield Ethernet W5100.

1.3 Bancada Multiprocessos

Consiste em uma bancada didática localizada no laboratório do NUPEEDE na UFSM, que tem como intuito principal a medição de nível de água no processo. Porém, não se restringe apenas a isso. A bancada, Figura 12, também apresenta instrumentos de medição de outras variáveis, como por exemplo: temperatura, pressão e vazão.

Figura 12 – Bancada Multiprocessos

1.3.1 Esquemático da malha de nível

O esquemático da malha de nível, Figura 13, permite obtermos um mapeamento do processo a ser estudado. No esquema, são evidenciados, além dos tanques, o CLP (Controlador Lógico Programável) utilizado, assim como os instrumentos de medição, o inversor de frequência e a bomba. Ainda, é descrito através de um padrão os caminhos que a água irá percorrer, através dos capilares de medição e canos, e os caminhos da corrente elétrica, a qual, fornece energia aos equipamentos do processo (CLP, bomba, instrumentos e inversor).

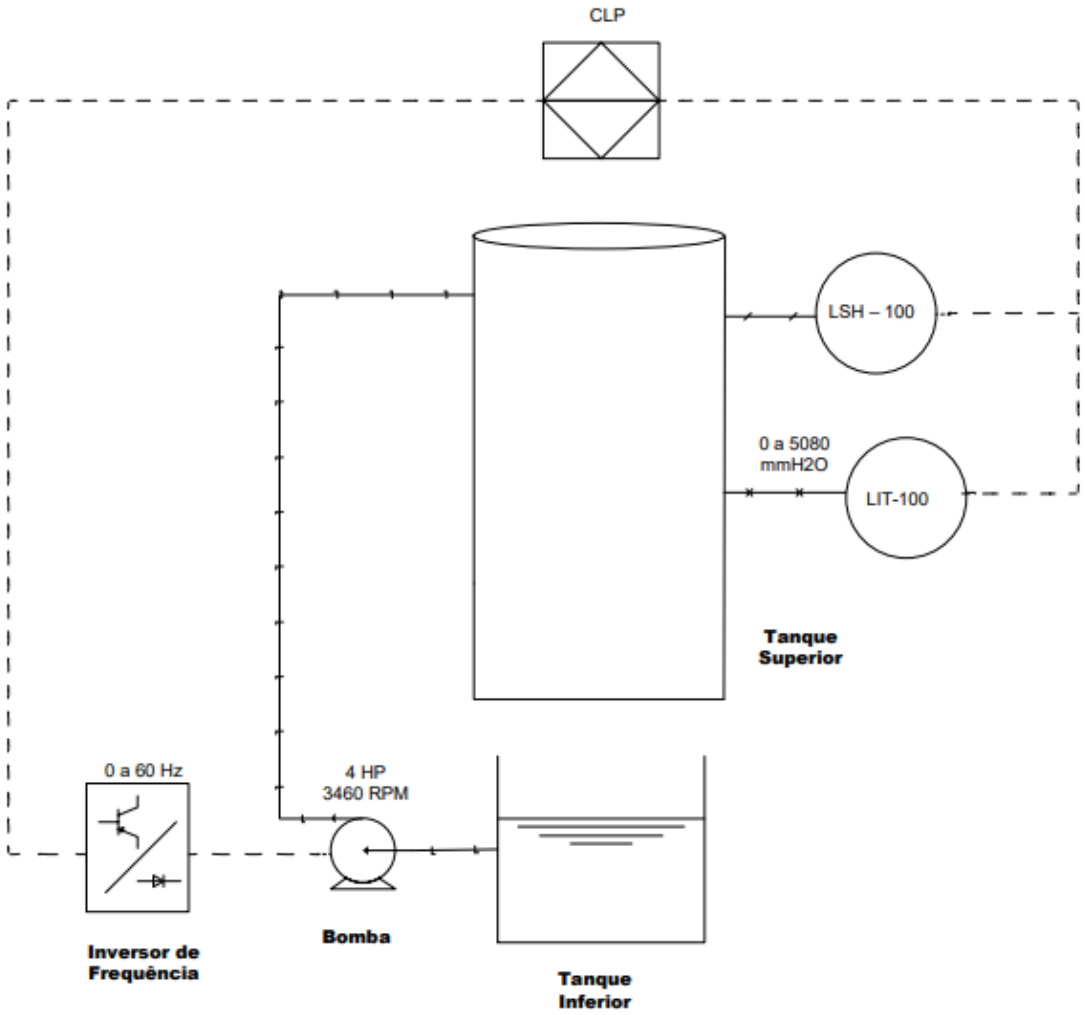

Figura 13 – Esquemático da malha de nível

1.4 Controle de nível

O controle do nível de água do processo é realizado através de um controlador PI (controlador proporcional integral), aliado à planta, o qual recebe dados de entrada (referentes ao nível) do *setpoint* do processo. Tal esquemático de controle, é descrito pela Figura 14.

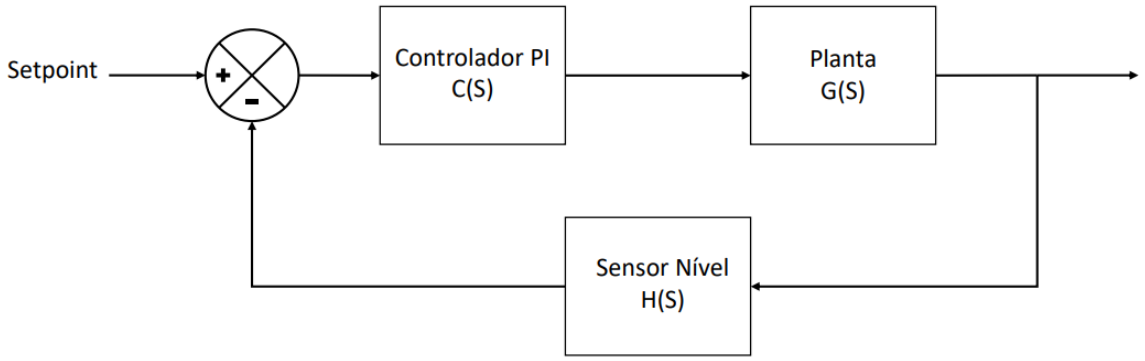

Figura 14 – Controle de nível em malha fechada.

CAPÍTULO 2 METODOLOGIA

O trabalho consiste no controle em malha fechada da bancada de nível localizada no NUPEDEE da Universidade Federal e Santa Maria (UFSM), empregando o microcontrolador Arduino Mega, um *Shield Ethernet*, um *Switch* e uma *Protoboard,* para realizar o controle do processo via protocolo de comunicação Modbus TCP estudado na disciplina de Integração de Redes Industriais do curso de Engenharia de Controle e Automação.

2.1 CLP (Aquisição dos *holding registers*)

A primeira tarefa feita na bancada foi obter o endereço das variáveis necessárias para o controle de nível em malha fechada da bancada. Desta forma, o primeiro passo foi localizar o endereço lógico dos registradores que possuem os dados do sensor de nível, os dados do inversor de frequência responsável por atuar no motor-bomba. Para tal, foi utilizado o sistema

supervisório da bancada e o *software Radzio Modbus Master Simulator*, que é um simulador de mestre Modbus, para a localização dos endereços dos registradores e para testar as variáveis escrevendo em seus registradores. A aquisição dos endereços dos registradores é um dos fatores principais para substituirmos o controle realizado pelo CLP para o novo mestre que é o Arduino. A Tabela 1 mostra a relação dos endereços lógicos dos registradores encontrados e suas respectivas variáveis.

Tabela 1 – Endereços lógicos dos registradores e sua respectiva função.

Endereço lógico	Função
9100	Liga/Desliga
52000 e 52001	Sensor de nível
52400 e 52401	MV de frequência do inversor
52402 e 52403	Set Point de frequência do inversor

Nota-se que os endereços lógicos escolhidos não são os endereços padronizados pela Modicon para leitura de *holding registers* nos casos dos sensores de nível, MV e Set Point. O endereço padrão está entre 40001 e 49999. Isto deve-se a escolha dos responsáveis por programar o CLP que resolveram utilizar os endereços para funções especiais que estão, conforme o padrão da Modicon, entre os valores 50001 e 59999. Ainda, percebe-se dois endereços de registradores nas variáveis utilizadas, exceto para liga/desliga. Isto é devido ao formato em ponto flutuante das variáveis empregadas. Como um único registrador possui 16 bits, para poder representar uma variável em ponto flutuante é necessário a utilização de 2 registradores, assim, para a manipulação destas variáveis deve-se ler/escrever nos dois endereços de registradores.

2.2 Materiais e Métodos (Esquemático *protoboard*)

Para a construção e funcionamento do projeto de controle da malha de nível da bancada multiprocessos foram utilizados os seguintes componentes:

- Microcontrolador Arduino Mega baseado no microprocessador ATmega328P;
- *Shield* Ethernet W5100;
- *Shield* LCD Display;
- CLP;

- Inversor de Frequência;
- Switch de rede OvisLink;
- Cabo de rede com conectores RJ-45;
- Dois potenciômetros de 50 kΩ;
- Jumpers;
- Protoboard.

O método de construção foi realizado em duas partes: primeiramente a montagem do projeto, cabos de comunicação e componentes na protoboard e, posteriormente, a programação para realizar a comunicação e controle.

A montagem da parte física conforme a Figura 15, evidencia a visualização lógica da comunicação.

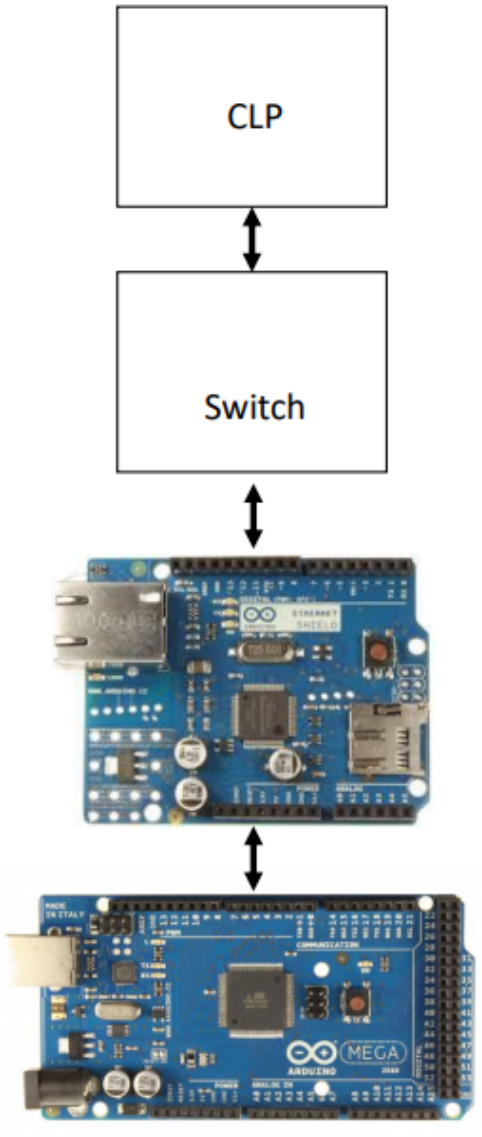

Figura 15 – Lógica de comunicação.

E na figura 16 abaixo, pode-se visualizar a disposição dos componentes na protoboard ligado ao arduino.

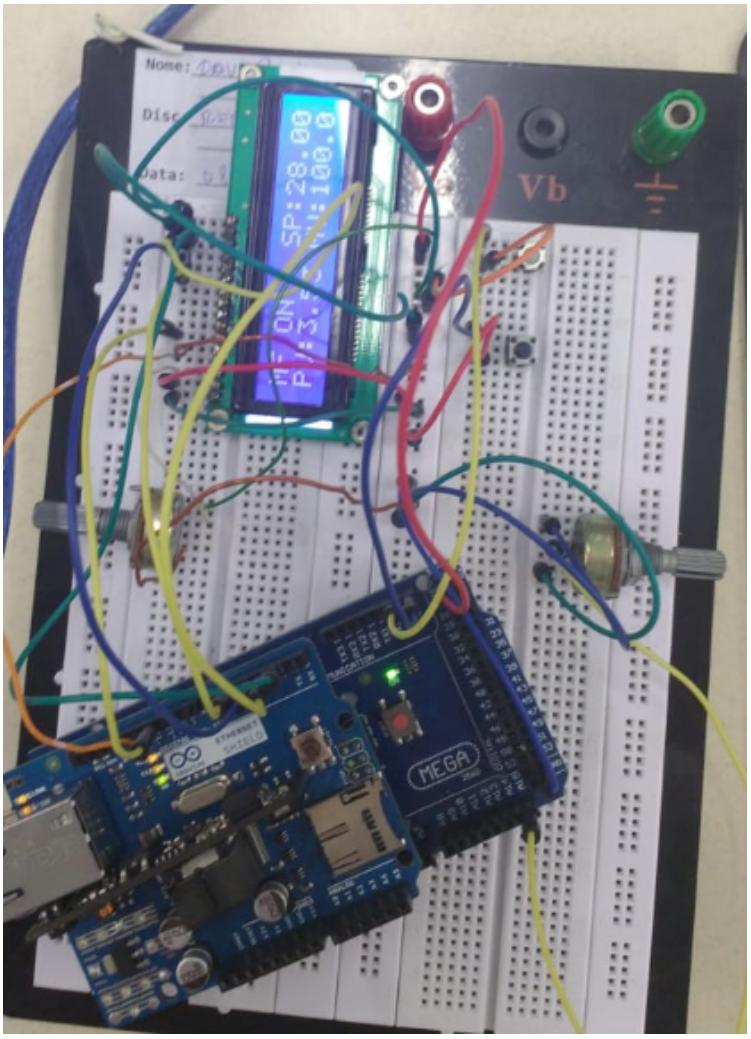

Figura 16 – Disposição dos componentes na protoboard

2.3 Programação Arduino

Através do Arduino que toda a malha de controle de nível é desenvolvida. Para o auxílio deste controle um *Switch* e um *Shield Ethernet* são empregados para fazer a interface do Arduino com a bancada de nível. Para a programação do Arduino o primeiro passo foi importar as bibliotecas necessárias para a programação do *shield Ethernet*, do LCD e da biblioteca Modbus TCP. Posteriormente, são configurados os endereços de IP do CLP e do *Shield Ethernet,* bem como o endereço MAC do *shield.* Além disso, o campo *unit identifier* que corresponde ao campo de endereço do escravo foi definido como 255 o que garante o caráter *broadcast* de comunicação entre o mestre e os escravos. Estes parâmetros são mostrados na Figura 17.

```
#define WIZNET_W5100 1 // Modelo do Shield Ethernet
#include <Ethernet.h>  // Bibliotecas Ethernet
#include <LiquidCrystal.h> // Bibliotecas Display LCD
#include <ModbusTCP.h> // Bibliotecas ModbusTCP

LiquidCrystal lcd(12, 11, 5, 4, 3, 2); // Iniciar o Display LCD
IPAddress ModbusDeviceIP(192, 168, 0, 244);  // Endereco IP do CLP
IPAddress moduleIPAddress(192, 168, 0, 243);  // Endereco do Shield Ethernet
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xE1 }; // MAC Adress do Shield Ethernet
ModbusTCP node(255);   // Unit Identifier do CLP

// Configuração de variaveis globais
int pinBotao_liga = 21; // Declara pino do botao liga
int pinBotao_emergencia = 20; // Declara pino do botao emergencia
int pinPotenciometro = A15; // Declara pino potenciometro
bool ligado = false;
float kp = 31.2;  // Constante proporcional do PID
float ki = 0.167; // Constante de integral do PID
float kd = 0;  // Constante derivativa do PID
double loopTime; // Variaveis de tempo
double lastProcess;
```

Figura 17 – Configações iniciais das bibliotecas e variáveis globais.

Foram também definidos os ganhos necessários para implementar o código de controle PI no Arduino. Esses ganhos foram retirados do sistema supervisório presente na bancada. Outro fator importante na implementação do algoritmo de controle, foi conseguir manipular os dados provenientes dos *holding registers*. Para isto, foi utilizado uma função que lê dois valores inteiros sem sinal vindo dos dois registradores (por se tratar de um valor em ponto flutuante) da variável lida e retorna o valor destes dois registradores em uma variável em ponto flutuante através de uma *union.* Este processo é empregado no algoritmo de controle para obtenção dos valores lidos pelo sensor de nível e também para mostrar os valores lidos no *display* LCD.

Para escrever os valores dos *holding registers* do atuador, ou seja, no inversor de frequência e do potenciômetro para alterar o *set point*, o processo inverso é realizado. A Figura 18 mostra como é feito este processo. E na figura 19 tem-se a configuração inicial das entradas e saídas digitais, assim como a inicialização dos Shields Ethernet e do visor LCD.

```
// Repartir o float e retornar o primeiro inteiro sem sinal (16 bits)
unsigned int f_2uint_int1(float float_number) {
  union f_2uint {
    float f;
    uint16_t i[2];
  };
  union f_2uint f_number;
  f_number.f = float_number;
  return f_number.i[0];
}

// Repartir o float e retornar o segundo inteiro sem sinal (16 bits)
unsigned int f_2uint_int2(float float_number) {
  union f_2uint {
    float f;
    uint16_t i[2];
  };
  union f_2uint f_number;
  f_number.f = float_number;
  return f_number.i[1];
}

// Reconstruir o float a partir de 2 inteiros sem sinal
float f_2uint_float(unsigned int uint1, unsigned int uint2) {
  union f_2uint {
    float f;
    uint16_t i[2];
  };
  union f_2uint f_number;
  f_number.i[0] = uint1;
  f_number.i[1] = uint2;
  return f_number.f;
}
```

Figura 18 – Funções auxiliares.

```
// Configuracoes basicas
void setup()
{
  lcd.begin(16, 2);
  lcd.clear();
  lcd.setCursor(0,0);
  lcd.print("MF OFF");
  pinMode(pinPotenciometro, INPUT);     // potenciometro
  pinMode(pinBotao_liga,INPUT);         // botao digital
  pinMode(pinBotao_emergencia,INPUT);   // botao digital
  delay(1000);
  // Iniciar Ethernet
  Ethernet.begin(mac, moduleIPAddress);
  node.setServerIPAddress(ModbusDeviceIP);
  delay(1000);                          // Tempo para inicializar
}
```

Figura 19 – Configuração inicial das entradas e saídas digitais.

Na figura 20 pode-se visualizar a lógica para enviar o sinal de liga e desliga para o CLP através de um botão digital (*push-button*).

```
void loop()
{
  int botao_liga = digitalRead(pinBotao_liga);
  int botao_emergencia = digitalRead(pinBotao_emergencia);
  float potenciometro = analogRead(pinPotenciometro);
  float setPoint = map(potenciometro, 0, 1023, 0, 100);

  if(botao_liga == 0)
  {
    node.writeSingleRegister(9100, 1); // Escrever em um unico registrador (Iniciar)
    ligado = true;
    lcd.setCursor(3,0);
    lcd.print("ON ");
  }

  if(botao_emergencia == 0)
  {
    node.writeSingleRegister(9100, 2); // Escrever em um unico registrador (Desativar)
    ligado = false;
    lcd.setCursor(3,0);
    lcd.print("OFF");
  }

  float nivel;
  float pid;
```

Figura 20 – Lógica para enviar o sinal de liga e desliga para o CLP.

O valor em ponto flutuante é divido em dois inteiros sem sinal, também empregando uma *union* dentro de uma função especificada. Posteriormente, estes valores são escritos nos respectivos registradores através dos endereços lógicos mostrados nas seções anteriores. A figura 21 mostra o algoritmo responsável pela leitura e escrita dos *holding registers* e implementar o controle PI. E na figura 22 tem o código para mostrar valores de controle no visor LCD.

```
if(ligado == true)
{
    uint8_t result;
    result = node.readHoldingRegisters(52000, 2);    // Ler os Holding Registers
    unsigned int aux_nivel1,aux_nivel2;
    aux_nivel1 = node.getResponseBuffer(0);
    aux_nivel2 = node.getResponseBuffer(1);
    nivel = f_2uint_float(aux_nivel1, aux_nivel2);
    node.clearResponseBuffer();

    unsigned int aux_setPoint1,aux_setPoint2;
    aux_setPoint1 = f_2uint_int1(setPoint);
    aux_setPoint2 = f_2uint_int2(setPoint);
    node.writeSingleRegister(52402, aux_setPoint1);   // Escrever o setPoint
    node.writeSingleRegister(52403, aux_setPoint2);   // no supervisorio
                                                      // (nao utilizado na prática)
    //PID
    float error;
    error = setPoint - nivel;
    loopTime = millis();
    float deltaTime = (loopTime-lastProcess)/1000.0;
    lastProcess = millis();
    float P, I;
    //Proporcional
    P = error*kp;
    //Integral
    I = I+(error*ki)*deltaTime;
    pid = P+I;
    if(pid >= 100) // Checagem de seguranca
    {
      pid = 100;
    }
    else if (pid <= 0)
    {
      pid = 0;
    }

    unsigned int aux_pid1,aux_pid2;
    aux_pid1 = f_2uint_int1(pid);
    aux_pid2 = f_2uint_int2(pid);
    node.writeSingleRegister(52400, aux_pid1); // Escrever o MV de freq do inversor
    node.writeSingleRegister(52401, aux_pid2);
  }
```

Figura 21 – Leitura e escrita dos holding register do controle PI.

```
    // Mostrar informacoes no Display LCD
    lcd.setCursor(8, 0);
    lcd.print("SP:");
    lcd.print(setPoint);
    lcd.setCursor(0, 1);
    lcd.print("PV:");
    lcd.print(nivel);
    lcd.setCursor(7, 1);
    lcd.print(" MV:");
    lcd.print(pid);
}
```

Figura 22 – Mostrar valores de controle no visor LCD.

CONCLUSÃO

A partir do trabalho, foi possível aplicar na prática os conhecimentos teóricos aprendidos em sala de aula, principalmente referentes ao Modbus TCP (baseado no protocolo TCP/IP). Além disso, houve um acréscimo não somente de aprendizado por parte dos alunos, como também de experiência (devido ao trabalho em grupo, à determinação de tarefas e à resolução de problemas), a qual é imprescindível para a formação de futuros engenheiros capacitados a atuar no mercado de trabalho.

Ainda, foi importante a aplicação de aprendizados advindos de outras disciplinas do curso de Engenharia de Controle e Automação, como: Circuitos Elétricos, Redes Industriais, Microcontroladores, entre outras.

Referências Bibliográficas

Shaff, Frederico M., Material de aula da disciplina de Integração de Redes Industriais. UFSM, 2018