UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA ELÉTRICA
PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA
THIAGO DO AMARAL FELIPE

DESENVOLVIMENTO DE UMA FERRAMENTA COMPUTACIONAL APLICADA NA SIMULAÇÃO DE PROCESSOS INDUSTRIAIS

Uberlândia
2014

THIAGO DO AMARAL FELIPE

DESENVOLVIMENTO DE UMA FERRAMENTA COMPUTACIONAL APLICADA NA SIMULAÇÃO DE PROCESSOS INDUSTRIAIS

Dissertação de mestrado apresentada à Universidade Federal de Uberlândia, como exigência parcial para a obtenção do título de mestre em Ciências.

Área de concentração: Sistemas de Energia

Orientador: Dr. Fábio Vincenzi Romualdo da Silva

Banca Examinadora :
Fábio Vincenzi Romualdo da Silva, Dr. (UFU)
Márcio José da Cunha, Dr. (UFU)
Henrique José Avelar, Dr. (CEFET/MG)

Uberlândia

2014

THIAGO DO AMARAL FELIPE

DESENVOLVIMENTO DE UMA FERRAMENTA COMPUTACIONAL APLICADA NA SIMULAÇÃO DE PROCESSOS INDUSTRIAIS

Dissertação de mestrado apresentada à Universidade Federal de Uberlândia, como exigência parcial para a obtenção do título de mestre em Ciências.

Prof. Fábio Vincenzi Romualdo da Silva

Orientador

Prof. Edgard Afonso Lamounier Júnior, Dr.

Coordenador da Pós-Graduação

Uberlândia

2014

Dedico este trabalho aos meus pais Cleber e Márcia, à minha esposa Rafaella, ao meu irmão Cleber, à minha família e amigos.

AGRADECIMENTOS

Aos meus pais Cleber e Márcia pelo carinho, amor, compreensão e pelo apoio para me proporcionar uma boa formação acadêmica.

À minha esposa Rafaella, por ter me apoiado e incentivado durante todo o trajeto. Obrigado pelo amor, carinho, conselhos e por estar sempre ao meu lado.

Aos professores Fábio Vincenzi Romualdo da Silva e Márcio José da Cunha pela disponibilidade, ensinamentos, paciência, ajuda e dedicação.

Aos meus amigos, colegas de mestrado, e a todos que contribuíram de forma direta ou indireta na realização deste trabalho.

“O sucesso é a soma de pequenos esforços repetidos dia após dia.”

(Robert Collier)

RESUMO

O uso de simulação em ambientes industriais permite ao usuário realizar testes e definir estratégias de controle aplicadas a processos, onde a configuração da rede, a sintonia da malha e o estudo das restrições de temporização no controle podem ser avaliados antes da aplicação na planta real. Neste contexto, existe a necessidade de serem desenvolvidos protocolos de redes industriais simulados, interfaces de modelagem de sistemas, além de uma possível integração com sensores físicos disponíveis. Neste trabalho foi apresentada uma ferramenta para simulação de um processo industrial, que será simulado em tempo real e fará a troca de dados com um servidor através do protocolo Modbus e os dados poderão ser visualizados e manipulados por um sistema supervisório.

Palavras-chave: Simulação Industrial, Protocolo Modbus, Processos Industriais, Redes Industriais, Sistemas em tempo real.

ABSTRACT

The use of simulation in industrial environments allows the user to perform tests and define applied control strategies to cases where the network configuration, tuning the loop of control and the study of the timing constraints on the control can be evaluated before the implementation in the real plant. In this context there is the need to be developed simulated industrial networks, systems modeling interfaces protocols and possible integration with physical sensors available. This work presents a tool for real time simulation of an industrial process and data exchange with a server via Modbus protocol. That data can be viewed and manipulated by a supervisory system.

Keywords: Industrial Simulation, Modbus protocol, Industrial Processes, Industrial Networks, Real-Time Systems.

LISTA DE FIGURAS

FIGURA 1 - MENSAGEM MODBUS 22
FIGURA 2 - FLUXOGRAMA DAS AÇÕES DO SERVIDOR MODBUS 23
FIGURA 3- AMBIENTE INDUSTRIAL 32
FIGURA 4 - SERVIDOR MODBUS 33
FIGURA 5 - CLIENTE MODBUS 34
FIGURA 6 - VARIÁVEIS E CONFIGURAÇÃO DO *INDUSOFT* 36
FIGURA 7 - CRIAÇÃO DA TELA DE SUPERVISÃO 37
FIGURA 8. PLANTA INDUSTRIAL DE NÍVEL DE UM TANQUE 39
FIGURA 9. DIAGRAMA DE BLOCOS PARA CONTROLE DE NÍVEL 40
FIGURA 10. DIAGRAMA DE BLOCOS DO SISTEMA SIMULADO 43
FIGURA 11 - SERVIDOR MODBUS EM EXECUÇÃO 45
FIGURA 12 - *EXCEPTION CODES* 45
FIGURA 13 - CLIENTE MODBUS 46
FIGURA 14. TELA DO SISTEMA SUPERVISÓRIO – TANQUE ENCHENDO 46
FIGURA 15. TELA DO SISTEMA SUPERVISÓRIO – TANQUE NA ALTURA DESEJADA 47
FIGURA 16. TELA DO SISTEMA SUPERVISÓRIO – VÁLVULA DE SAÍDA DESLIGADA 47
FIGURA 17. PACOTES DA REDE MODBUS 49
FIGURA 18 - WIRESHARK - WRITE MULTIPLE REGISTERS – REQUEST 50
FIGURA 19- WIRESHARK - WRITE MULTIPLE REGISTERS – RESPONSE 51
FIGURA 20- WIRESHARK - WRITE MULTIPLE REGISTERS – COMPRIMENTO INCORRETO 53
FIGURA 21 - WIRESHARK - WRITE MULTIPLE REGISTERS – ENDEREÇO INCORRETO 54
FIGURA 22 - WIRESHARK – FUNÇÃO INCORRETA 55
FIGURA 23- WIRESHARK - WRITE MULTIPLE REGISTERS – COMPRIMENTO INCORRETO – RESPONSE 56
FIGURA 24- WIRESHARK - WRITE MULTIPLE REGISTERS – ENDEREÇO INCORRETO – RESPONSE 57
FIGURA 25 - WIRESHARK – FUNÇÃO INCORRETA – RESPONSE 58

LISTA DE TABELAS

TABELA 1 - READ COILS – MENSAGEM DE REQUEST 25
TABELA 2 - READ COILS – MENSAGEM DE RESPONSE 25
TABELA 3 - READ COILS - EXEMPLO DE ESTADOS 25
TABELA 4 – READ COILS – BYTES DE DADOS 26
TABELA 5 - READ HOLDING REGISTERS - MENSAGEM DE REQUEST 26
TABELA 6 - READ HOLDING REGISTERS - MENSAGEM DE RESPONSE 26
TABELA 7 - READ HOLDING REGISTERS - EXEMPLO DE REGISTROS 27
TABELA 8 – READ HOLDING REGISTERS – BYTES DE DADOS 27
TABELA 9 – WRITE SINGLE COIL - MENSAGEM DE REQUEST E RESPONSE 27
TABELA 10 – WRITE SINGLE REGISTER - MENSAGEM DE REQUEST E RESPONSE 28
TABELA 11 – WRITE MULTIPLE COILS – MENSAGEM DE REQUEST 29
TABELA 12 - WRITE MULTIPLE COILS – MENSAGEM DE RESPONSE 29
TABELA 13 - WRITE HOLDING REGISTERS - MENSAGEM DE REQUEST 30
TABELA 14 - WRITE HOLDING REGISTERS - MENSAGEM DE RESPONSE 30
TABELA 15 - EXCEPTION CODES - MENSAGEM DE RESPONSE 31
TABELA 16 - VARIÁVEIS DO SISTEMA SIMULADO 44

LISTA DE ABREVIATURAS E SÍMBOLOS

A	Área do fundo do tanque
A_0	Área de seção transversal da tubulação
ADU	Application Data Unit
ASCII	American Standard Code for Information Interchange
C	Coeficiente de descarga
CAN	Control Area Network
CLP	Controlador Lógico Programável
CRC	Cyclic Redundancy Check
entrada_analog1	SetPoint Altura
entrada_analog2	Ganho proporcional do controlador
entrada_analog3	Ganho integral do controlador
entrada_analog4	Abertura da válvula de saída
entrada_digital1	Acionamento da Válvula de Entrada
entrada_digital2	Acionamento da Válvula de Saída
F_0	Vazão de saída
F_i	Vazão de entrada (bomba)
g	Aceleração da gravidade
G(s)	Função de transferência da planta
h	Nível de líquido
IP	Internet Protocol
K_d	Produto dos ganhos K_{driver} e $K_{motor+bomba}$
K_{driver}	Ganho estático do driver
$K_{motor+bomba}$	Ganho estático do motor e da bomba
K_s	Ganho estático do sensor
LAN	Local Area Network
LRC	Longitudinal Redundancy Check
MBAP	Modbus Application Header
OPC	Ole for Process Control
OSI	Open Systems Interconnection
PDU	Protocol Data Unit
PWM	Pulse Width Modulation

RTU	Remote Terminal Unit
saida_analog1	Altura do Tanque
saida_analog2	Vazão de entrada
saida_analog3	Vazão de saída
TCP	Transmission Control Protocol

SUMÁRIO

1 INTRODUÇÃO **14**
2 DESENVOLVIMENTO **17**
2.1 INTRODUÇÃO **17**
2.2 SIMULAÇÃO INDUSTRIAL **17**
2.3 PROTOCOLO MODBUS **21**
2.3.1 Definições *21*
2.3.2 Funções Modbus *24*
2.4 CONCLUSÃO **31**
3 METODOLOGIA **32**
3.1 INTRODUÇÃO **32**
3.2 SERVIDOR MODBUS **32**
3.3 CLIENTE MODBUS **33**
3.4 SUPERVISÓRIO **35**
3.5 CONCLUSÃO **37**
4 RESULTADOS **39**
4.1 INTRODUÇÃO **39**
4.2 MODELAGEM DO PROCESSO **39**
4.3 RESULTADOS **44**
4.3.1 Servidor *44*
4.3.2 Cliente Modbus *45*
4.3.3 Supervisório *46*
4.3.4 Análise dos pacotes na rede *47*
5 CONCLUSÃO **59**
REFERÊNCIAS **61**
APÊNDICE A – CÓDIGO SERVIDOR MODBUS **63**
APÊNDICE B – CÓDIGO CLIENTE MODBUS **76**

1 INTRODUÇÃO

A simulação de ambientes industriais é considerada uma ferramenta muito importante na indústria, pois através dela pode-se reduzir custos em testes realizados em plantas industriais e aquisição de novos equipamentos de campo, onde é possível determinar quais dispositivos são necessários para realizar uma determinada função e a forma correta de sua utilização (VIEIRA, 2006).

Nos ambientes acadêmicos, a simulação de processos industriais possui um papel fundamental para o ensino e a capacitação de profissionais, pois é capaz de mostrar a realidade de um ambiente industrial, sem que haja riscos de situações críticas, possibilitando a alteração de parâmetros de um determinado sistema industrial e a visualização e análise de variáveis do processo. Com a simulação também é possível alterar o método de controle da planta e os seus dispositivos (LOBÃO; PORTO, 1996). Esses fatos são considerados como vantagens, pois em plantas didáticas físicas isso não é possível, porque normalmente as plantas são desenvolvidas para uma finalidade específica (por exemplo, uma planta didática de controle de nível e vazão) de um determinado processo.

Diversas empresas, como Toyota, Motorola, Exxon Mobil, The Gillette Company, Caterpillar, Siemens, FIAT, 3M, Intel, Honeywell, FedEx, Ford, American Express, United Technologies, Kraft, Volkswagen, DaimlerChrysler, Delta, General Motors, Hewllet Packard, General Electric e Cymer, têm utilizado (ou utilizaram) simulação computacional para melhorar seus processos, produtos e/ou serviços (VIEIRA, 2006).

Como os ambientes industriais são formados por diversos dispositivos, tais como os sensores, atuadores e Controladores Lógicos Programáveis (CLP), é necessário que exista uma forma de comunicação entre os mesmos, de forma que seja possível a troca de dados, controle e supervisão do processo. Os sensores fazem o papel de aquisição de dados de um sistema, como pressão, nível, temperatura. Os atuadores realizam algum tipo de ação no sistema, como por exemplo, produzir uma força ou movimento, como motores e válvulas. Os CLPs realizam o controle do processo, através do recebimento das informações contidas nos sensores, que passam por alguma lógica de controle e fará o comando para que os atuadores realizem suas ações.

Para fazer a comunicação entre dispositivos existem diversos protocolos de redes industriais utilizados, dentre eles o Modbus, FOUNDATION Fieldbus, Profibus, Control Area Network (CAN), entre outros (REYNDERS; MACKAY; WRIGHT, 2005).

O protocolo Modbus é um protocolo de mensagens da camada de aplicação, posicionado na camada 7 do modelo OSI (*Open Systems Interconnection*). Foi criado em 1979 pela Modicon, hoje Schneider Electric e possui um vasto campo de aplicação. É normalmente utilizado devido a sua simplicidade, especificação livre e grande variedade de produtos existentes em diversas empresas (GUARESE *et al.*, 2012).

A utilização de redes e protocolos digitais prevê um significativo avanço nas seguintes áreas (CASSIOLATO, 2012):

- Custos de instalação, operação e manutenção;
- Procedimentos de manutenção com gerenciamento de ativos;
- Fácil expansão e upgrades;
- Informação de controle e qualidade;
- Baixos tempos de ciclos;
- Várias topologias;
- Padrões abertos;
- Redundância em diversos níveis;
- Menor variabilidade nas medições com a melhoria das exatidões;
- Medições multivariáveis.

Já existem pesquisas relacionadas ao desenvolvimento e pesquisa de simulação de alguns protocolos de redes industriais, tais como as desenvolvidas por Brandão (2005) e Torres (2013). Schneider *et al.* (2013) propôs o desenvolvimento de uma plataforma de simulação industrial, por meio do padrão de comunicação OPC (*Ole for Process Control*).

Nesse contexto, este trabalho apresenta o desenvolvimento de uma plataforma computacional de simulação industrial capaz de realizar a comunicação de um processo simulado em tempo real com um servidor. A simulação do processo industrial é feita no software Matlab®, por meio da ferramenta *Simulink*, responsável pela troca de informações com um servidor Modbus TCP computacional (*software*), tornando possível a manipulação das variáveis de entrada e saída do processo. A supervisão dos pontos da planta simulada é feita por um sistema supervisório de mercado utilizado nas indústrias. O foco da aplicação dessa pesquisa se dá em instituições de ensino que possuem dificuldades de aquisição de plantas industriais reais, em centros de treinamento dedicados para a automação industrial, escolas técnicas e tecnológicas, dentre outras.

Esse documento está dividido da seguinte forma: no Capítulo 2 são apresentadas as definições e regras sobre temas que serão abordados ao longo do trabalho, como a simulação

industrial e o protocolo Modbus. No Capítulo 3 é apresentada a metodologia utilizada para o desenvolvimento do ambiente industrial, incluindo os softwares criados e utilizados. No Capítulo 4 é apresentada a modelagem de um processo industrial e os resultados obtidos através deste processo e dos softwares desenvolvidos. Por fim, no Capítulo 5 são apresentadas as conclusões e sugestões para trabalhos futuros.

2 DESENVOLVIMENTO

2.1 Introdução

Este capítulo apresenta a definição de simulação industrial, mostrando quais são as suas vantagens e desvantagens, e possíveis formas de aplicação.

Além disso, é descrito o protocolo Modbus, incluindo suas principais versões e as diferenças entre elas, o formato das mensagens, o papel de cada dispositivo na rede, as ações do servidor e o encapsulamento da mensagem Modbus no protocolo TCP. São apresentadas também as funções do protocolo Modbus *Write Multiple Registers, Read Holding Registers, Read Coils, Write Multiple Coils, Write Single Coil e Write Single Register*, que foram utilizadas no projeto.

Para cada uma das funções, são descritas as utilizações, especificações dos campos da mensagem, as especificações dos campos da mensagem de *Request* e *Response* e exemplos de mensagens para melhor compreensão de sua estrutura.

Por fim, serão descritos os *Exception Codes*, citando quando ocorrem e como é a mensagem de resposta do servidor.

2.2 Simulação Industrial

A simulação é definida como um processo que permite projetar e conduzir experimentos em um modelo computacional tendo como base um sistema real com o propósito de entender seu comportamento e/ou avaliar estratégias para sua operação (PEDGEN; SHANNON; SADOWSKI, 1991). Portanto, a simulação tem como objetivo descrever o comportamento de um sistema real computacionalmente, através do qual é possível analisar os detalhes do sistema, realizar testes sem a necessidade de equipamentos físicos, economizar tempo e recursos no desenvolvimento de projetos.

Dentre as vantagens da simulação industrial, podem ser citadas:

- Identificação de problemas e situações críticas, quando utilizados modelos que reproduzam corretamente o comportamento do sistema real;
- Possibilidade de realização de testes sem interromper o processo real em execução, inclusive de forma mais rápida ou mais lenta do que ele ocorre na prática;
- Estudo dos dispositivos a serem utilizados;

- Treinamento dos usuários e operadores;
- Estudos de eficiência para redução de custos, através da redução de máquinas, matéria-prima, entre outros;
- Novos dispositivos e sistemas podem ser testados sem serem comprados;
- Compreender melhor as variáveis do processo, analisando o seu comportamento e influencia no sistema. O conhecimento adquirido pode ser de grande importância em casos de necessidade de aprimoramento ou mudança do projeto.

Como desvantagens, é possível citar:

- A modelagem de sistemas muito complexos pode não ter resultados satisfatórios ou de difícil interpretação;
- Dificuldade de reprodução de todas as variáveis interagindo no sistema;
- A análise e a modelagem do sistema podem ser demoradas e de custo elevado;
- Uma modelagem correta precisa de um vasto conhecimento do processo.

De acordo com o descrito por BANKS *et al.* (2005), a simulação pode ser usada para os seguintes propósitos:

- Quando ocorrem mudanças de informação, organizacional ou ambiental;
- Quando é necessário analisar os valores das saídas do sistema quando os valores de entrada são alterados;
- Como uma ferramenta pedagógica, quando se deseja reforçar os estudos teóricos, verificar soluções analíticas ou para treinamento;
- Quando se deseja mostrar a operação de um sistema através de uma animação;
- Quando se deseja testar novas configurações e sistemas antes da implementação, para a preparação do que pode ocorrer durante o seu funcionamento;

Além de citar onde a simulação deve ser utilizada, BANKS *et al.* (2005) apresenta 10 regras para avaliar se a simulação não é apropriada. São elas:

1. Quando o problema pode ser resolvido por senso comum;
2. Se o problema puder ser resolvido analiticamente;
3. Se realizar experimentos diretamente for mais fácil do que realizar a simulação;
4. Se os custos excederem o limite de renda disponível para esse fim;
5. Quando recursos não estão disponíveis;

6. Quando tempo não é disponível;
7. Quando dados não estiverem disponíveis, nem estimativas;
8. Se não há tempo ou pessoal para a verificação e validação do modelo;
9. Se os gerentes ou diretores têm expectativas irracionais, estão pedindo muito com pouco tempo de pesquisa ou os resultados da simulação estão sendo superestimados;
10. Se o sistema tem comportamento muito complexo ou não pode ser definido.

Especificamente no Brasil, aplicações de simulação têm ocorrido em inúmeras áreas, como apresentado por Vieira (2006):

- Logística & Supply Chain
  - Estudo de expansão do Canal do Panamá;
  - Dimensionamento de frotas de abastecimentos;
  - Estudo de abastecimento de matéria prima;
  - Estudo da logística de manutenção;
  - Estudo da logística de carga/descarga de produtos;
  - Dimensionamento da frota de transporte terceirizada;
  - Dimensionamento de supply-chain;
  - Distribuição de bebidas em escala nacional;
  - Comparativo entre operadores logísticos;
  - Movimentação e esvaziamento de pátio de veículos;
  - Avaliação de sistema de armazenagem automatizado;
  - Distribuição de produto alimentício perecível;
- Ferrovias
  - Malha ferroviária ligando São Paulo, Minas e Rio de Janeiro;
  - Pátio ferroviário do porto de Tubarão;
  - Estrada de ferro Vitória-Minas;
  - Malha ferroviária da região sul do Brasil;
  - Definição da melhor composição para transporte ferroviário urbano;
- Siderurgia
  - Armazenagem e transporte de matérias primas por sistema de correias em siderúrgica;
  - Laboratório de análise de amostras da Aciaria;
  - Forjaria de cilindros;

  - Aciaria e expansões da Aciaria;
  - Modernização da linha de fabricação de fios;
  - Expansão da coqueria;
  - Implantação de novo forno magnesiano;
  - Fabricação de telas de aço;
  - Estudo de movimentação para expansão de aciaria;
  - Dimensionamento da frota de transporte interno;
- Manufatura
  - Dimensionamento do abastecimento em linha de montagem automotiva;
  - Avaliação de mudanças em linha de montagem de motores;
  - Estudo de alternativas para linha de montagem;
  - Análise de impacto do mix produtivo;
  - Dimensionamento de mão de obra em célula de pintura;
  - Linha de pintura de chassis;
  - Operadores trabalhando em célula;
  - Estudo de simultaneidade em linha de vulcanização de pneus;
  - Implantação de linha de montagem de eixos semiautomática;
  - Estudo de desempenho de linha produtiva;
  - Simulação completa de nova fábrica automotiva;
  - Estudo de investimento em maquinário;
  - Dimensionamento da frota de empilhadeiras;
  - Estudo do impacto de tempos de setup em linha de usinagem;
  - Análise e balanceamento entre setores produtivos;
  - Comparação entre diferentes layouts de montagem;
  - Identificação do mix ótimo para célula de termoformagem;
  - Estudo de dimensionamento de equipamentos;
  - Dimensionamento de linha de montagem;
  - Balanceamento macro entre setores fabris;
  - Estudo de investimento em linha de montagem robotizada;
  - Dimensionamento de mão de obra em células de produção;
  - Comparação entre diferentes conceitos de produção e layouts de montagem;
  - Melhoria da estamparia;

- Petróleo
    - Simulador de cronograma de exploração de campo;
    - Estudo de risco no cronograma de desenvolvimento e exploração de campo petrolífero marítimo;
    - Dimensionamento de frota de rebocadores;
- Celulose
    - Logística de acesso rodoviário;
    - Sistema logístico de suprimento de madeira para a indústria de celulose;
    - Logística aquaviária;
    - Dimensionamento portuário;
    - Operações entre modais marítimo e ferroviário.

2.3 Protocolo Modbus

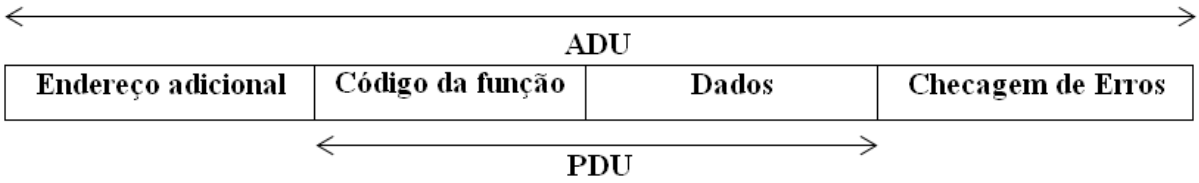
2.3.1 Definições

O Modbus é um protocolo de mensagens da camada de aplicação que fornece comunicação entre dispositivos conectados a diferentes tipos de barramentos e redes. O protocolo implementa uma arquitetura Cliente/Servidor e atua essencialmente em um modo '*Request/Response'*, independente do controle de acesso ao meio usado na camada de enlace (REYNDERS; MACKAY; WRIGHT, 2005). Como o protocolo Modbus atua na camada de aplicação, é necessária a associação com outros protocolos nas camadas inferiores. Para isso, foi utilizado o TCP na camada de transporte, IP na camada de rede e a Ethernet para realizar a troca de dados entre o cliente e o servidor na camada de enlace.

O protocolo Modbus tem duas principais versões: Modbus Serial e Modbus TCP (MODBUS ORGANIZATION, 2004) (MODBUS ORGANIZATION, 2012). No Modbus Serial as mensagens são transmitidas entre mestres e escravos usando os modos de transmissão *American Standard Code for Information Interchange* (ASCII) ou *Remote Terminal Unit* (RTU). Na camada de aplicação é gerado o *Protocol Data Unit* (PDU), que consiste em um código de função e os dados relacionados a ela. Nas camadas inferiores, essa mensagem será convertida em um *Application Data Unit* (ADU), que irá acrescentar um endereço adicional, que pode ser o endereço do escravo ao qual a mensagem se relaciona, e o

campo de checagem de erros. A Fig. 1 ilustra os campos da mensagem, mostrando cada um de seus campos.

Figura 1 - Mensagem Modbus



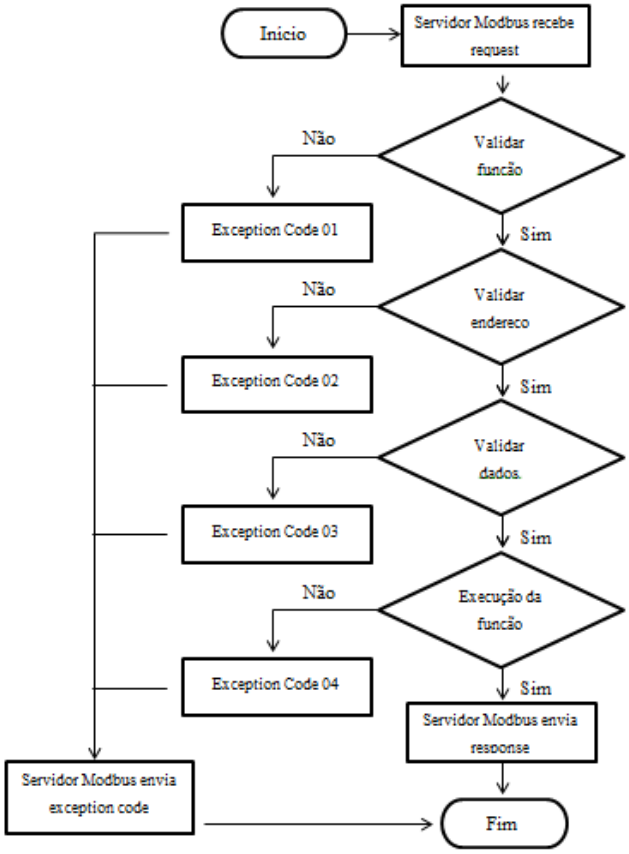
Fonte: Elaborado pelo autor

O endereço adicional identifica o receptor da mensagem, e pode assumir os valores de 1 a 247. O endereço 0 define que a mensagem será enviada em *broadcast* e os endereços de 248 a 255 são reservados. O PDU Modbus possui o código da função e seus parâmetros, com um limite de 252 bytes. O código da função define qual a ação será executada pelo dispositivo escravo e cada função possui seus parâmetros específicos. A checagem de erros é feita utilizando o *Cyclic Redundancy Check* (CRC), que garante que o dispositivo não reaja a mensagens que possam ter sido danificadas durante a transmissão (REYNDERS; MACKAY; WRIGHT, 2005).

O protocolo Modbus TCP fornece conectividade dentro de uma rede Modbus baseada em LAN (um mestre e seus escravos) e também para redes conectadas através de IP (múltiplos mestres, cada um com múltiplos escravos). O Modbus TCP possui a vantagem de habilitar os mestres a terem múltiplas transações em aberto e os escravos a participarem em comunicações simultâneas com múltiplos mestres (HUITSING *et al*., 2008).

Os escravos, que realizam o papel de servidores, ficam aguardando uma conexão TCP na porta 502. Essa conexão é requisitada pelos mestres, que fazem o papel de clientes da rede. É possível que um dispositivo escravo se torne um mestre e vice-versa, mas apenas se este dispositivo não estiver conectado a nenhum outro. A Fig. 2 apresenta um fluxograma que ilustra as ações do servidor durante a sua execução.

Figura 2 - Fluxograma das ações do servidor Modbus



Fonte: Elaborado pelo autor

As transações utilizando o Modbus TCP são semelhantes às do Modbus Serial, porém necessitam encapsular os dados contidos no PDU em uma mensagem TCP. Esse encapsulamento é feito através de um cabeçalho do tipo MBAP (*Modbus Application Header*), que possui os seguintes campos:

- *Transaction Identifier* – É utilizado para identificar cada *request* e seu tamanho é de 2 bytes. O *request* e a *response* são idênticos.
- *Protocol Identifier* – É utilizado para identificar o protocolo. Seu tamanho é de 2 bytes e, no caso do Modbus, o valor deverá ser 0x0000.
- *Length* – É utilizado para informar a quantidade de bytes da mensagem após esse campo e possui 2 bytes.
- *Unit Identifier* – É utilizado para identificar os dispositivos da rede. Possui 1 byte e a mensagem de *response* e *request* devem conter o mesmo valor.

A utilização do Modbus TCP possibilita duas vantagens ao sistema: a possibilidade de acesso remoto através da rede Ethernet ou a Internet e a possibilidade de qualquer nó acessar outro, permitindo assim conexão ponto a ponto.

2.3.2 Funções Modbus

Quando algum dispositivo cliente envia uma mensagem para o servidor, ele deve aguardar uma resposta, que pode ser de quatro tipos (MODBUS ORGANIZATION, 2006):

- Se o servidor receber uma requisição sem erro de comunicação e esta requisição puder ser executada, ele retorna uma resposta normal.
- Se o servidor não receber a mensagem devido a um erro de comunicação, nenhuma resposta será retornada. Ocorrerá então uma condição de *timeout* no cliente.
- Se o servidor receber a mensagem, mas detectar um erro (paridade, LRC, CRC,...) nenhuma resposta será retornada. Ocorrerá então uma condição de *timeout* no cliente.
- Se o servidor receber uma requisição sem erro de comunicação e esta requisição não puder ser executada (por exemplo, ler um local de memória não existente), o servidor irá retornar um resposta do tipo *exception*, informando a natureza do erro.

2.3.2.1 Read Coils – Função 1

Esta função é utilizada para a leitura de variáveis do tipo *coil*. A variável desse tipo pode assumir valor 0 ou 1, e podem ser lidas e escritas. A mensagem de *request* contém o código da função, o endereço inicial e a quantidade de variáveis a serem lidas. A mensagem de *response* contém o código da função, o contador de bytes e o estado das variáveis.

A quantidade de bytes será a quantidade de variáveis a serem escritas dividido por 8. Caso a quantidade de variáveis não seja múltipla de 8, deverá ser somado 1 byte no contador.

Os bytes de dados são estruturados de forma que o bit mais significativo represente o estado da variável de maior posição e o bit menos significativo o de menor posição.

Caso o campo de estado das variáveis tenha tamanho superior a 1 byte, as variáveis serão distribuídas de forma que os endereços apareçam de forma crescente, ou seja, os primeiros endereços estarão no primeiro byte e os últimos no último byte.

Por fim, se o numero de variáveis a serem lidas não for múltiplo de 8, o ultimo byte irá conter as variáveis restantes nos bits menos significativos, em ordem crescente da direita para a esquerda, e o restante dos bits deverão ser preenchidos com 0.

As Tabelas 1 e 2 identificam os campos, tamanhos e possíveis valores das mensagens de *Request* e *Response* da função *Read Coils*.

Tabela 1 - Read Coils – Mensagem de Request

Campo	Tamanho	Valor
Código da função	1 Byte	0x01
Endereço Inicial	2 Bytes	0x0000 até 0xFFFF
Quantidade de variáveis	2 Bytes	0x0001 até 0x07D0

Fonte: Elaborado pelo autor

Tabela 2 - Read Coils – Mensagem de Response

Campo	Tamanho	Valor
Código da função	1 Byte	0x01
Contador de Bytes	1 Byte	N
Estado das variáveis	N Bytes	Bytes de dados

Fonte: Elaborado pelo autor

Portanto, se for solicitado a leitura das variáveis da posição 0 até 10, o primeiro byte irá conter o estado das variáveis de 7 a 0, e o segundo byte irá conter os estados das variáveis de 10 a 8, e será complementada de 0 nos bits mais significativos. A Tabela 3 mostra um exemplo de estado de 11 variáveis.

Tabela 3 - Read Coils - Exemplo de estados

Variável	Estado
0	1
1	0
2	1
3	1
4	1
5	0
6	0
7	0
8	0
9	1
10	1

Fonte: Elaborado pelo autor

Neste caso, serão precisos 2 bytes para armazenar as informações das variáveis. A Tabela 4 mostra os valores de cada um dos bytes.

Tabela 4 – Read Coils – Bytes de dados

Bytes de dados	Valor (binário)
0	00011101
1	00000110

Fonte: Elaborado pelo autor

2.3.2.2 Read Holding Registers – Função 3

Esta função é utilizada para a leitura de variáveis do tipo *Holding Registers*. Esse tipo de variável pode assumir o valor de 0 a 65535, e pode ser lida e escrita. A mensagem de *request* contém o código da função, o endereço inicial e a quantidade de registros a serem lidos. A mensagem de *response* contém o código da função, o contador de bytes e os valores dos registros.

A quantidade de bytes será a quantidade de registros que foram lidos multiplicado por 2, pois cada variável de registro possui o tamanho de 2 bytes (0x0000 a 0xFFFF).

Os bytes de dados são estruturados de forma que os endereços apareçam de forma crescente, ou seja, os primeiros endereços estarão no primeiro byte e os últimos no último byte.

As Tabelas 5 e 6 identificam os campos, tamanhos e possíveis valores das mensagens de *Request* e *Response* para a função *Read Holding Registers*.

Tabela 5 - Read Holding Registers - Mensagem de Request

Campo	Tamanho	Valor
Código da função	1 Byte	0x03
Endereço Inicial	2 Bytes	0x0000 até 0xFFFF
Quantidade de registros	2 Bytes	0x0001 até 0x007D

Fonte: Elaborado pelo autor

Tabela 6 - Read Holding Registers - Mensagem de Response

Campo	Tamanho	Valor
Código da função	1 Byte	0x03
Contador de Bytes	1 Byte	N*2
Valores dos Registros	N*2 Bytes	Bytes de dados

Fonte: Elaborado pelo autor

Portanto, se for solicitado a leitura dos registros da posição 0 até 2, os bytes 0 e 1 irão conter o valor do registro 0, os bytes 2 e 3 irão conter o registro 1 e os bytes 4 e 5 irão conter o registro 2. A Tabela 7 cita um exemplo de 3 registros.

Tabela 7 - Read Holding Registers - Exemplo de registros

Registro	Valor (decimal)
0	55
1	258
2	1005

Fonte: Elaborado pelo autor

Neste caso, serão precisos 6 bytes para armazenar as informações dos registros. A Tabela 8 ilustra os valores de cada um dos bytes.

Tabela 8 – Read Holding Registers – Bytes de dados

Bytes de dados	Valor (binário)
0	00000000
1	00110111
2	00000001
3	00000010
4	00000011
5	11101101

Fonte: Elaborado pelo autor

2.3.2.3 Write Single Coil – Função 5

Esta função é utilizada para a escrita de apenas uma variável do tipo *coil*. As mensagens de *request* e *response* são idênticas e contêm o código da função, o endereço em que se deseja escrever e o valor a ser escrito. O valor a ser escrito deve ser 0x0000 caso se deseje que o *coil* fique em estado *OFF* (0) ou 0xFF00 para estado *ON* (1).

A Tabela 9 identifica os campos, tamanhos e possíveis valores das mensagens de *Request* e *Response* da função *Write Single Coil*.

Tabela 9 – Write Single Coil - Mensagem de Request e Response

Campo	Tamanho	Valor (hexadecimal)
Código da função	1 Byte	0x05
Endereço da variável	2 Bytes	0x0000 até 0xFFFF
Valor da variável	2 Bytes	0x0000 ou 0xFF00

Fonte: Elaborado pelo autor

2.3.2.4 Write Single Register – Função 6

Esta função é utilizada para a escrita de apenas uma variável do tipo registro. As mensagens de *request* e *response* são idênticas e contêm o código da função, o endereço em que se deseja escrever e o valor a ser escrito.

A Tabela 10 identifica os campos, tamanhos e possíveis valores das mensagens de *Request* e *Response* da função *Write Single Register*.

Tabela 10 – Write Single Register - Mensagem de Request e Response

Campo	**Tamanho**	**Valor (hexadecimal)**
Código da função	1 Byte	0x06
Endereço do Registro	2 Bytes	0x0000 até 0xFFFF
Valor do Registro	2 Bytes	0x0000 até 0xFFFF

Fonte: Elaborado pelo autor

2.3.2.5 Write Multiple Coils – Função 15

Esta função é utilizada para a escrita de variáveis do tipo *coil*. A mensagem de *request* contém o código da função, o endereço inicial, a quantidade de variáveis a serem escritas, o contador de bytes e os valores das variáveis. A mensagem de *response* contém o código da função, o endereço inicial e quantidade variáveis que foram escritas.

A quantidade de bytes será a quantidade de variáveis a serem escritas dividido por 8. Caso a quantidade de variáveis não seja múltipla de 8, deverá ser somado 1 byte no contador.

Os bytes de dados são estruturados de forma que o bit mais significativo represente o estado da variável de maior posição e o bit menos significativo o de menor posição.

Caso o campo de estado das variáveis tenha tamanho superior a 1 byte, as variáveis serão distribuídas de forma que os endereços apareçam de forma crescente, ou seja, os primeiros endereços estarão no primeiro byte e os últimos no último byte.

Por fim, se o numero de variáveis a serem escritas não for múltiplo de 8, o ultimo byte irá conter as variáveis restantes nos bits menos significativos, em ordem crescente da direita para a esquerda, e o restante dos bits deverão ser preenchidos com 0.

As Tabelas 11 e 12 identificam os campos, tamanhos e possíveis valores das mensagens de *Request* e *Response* da função *Write Multiple Coils*.

Tabela 11 – Write Multiple Coils – Mensagem de Request

Campo	Tamanho	Valor
Código da função	1 Byte	0x0F
Endereço Inicial	2 Bytes	0x0000 até 0xFFFF
Quantidade de variáveis	2 Bytes	0x0001 até 0x07B0
Contador de bytes	1 Byte	N
Valores das variáveis	N Bytes	Bytes de dados

Fonte: Elaborado pelo autor

Tabela 12 - Write Multiple Coils – Mensagem de Response

Campo	Tamanho	Valor
Código da função	1 Byte	0x0F
Endereço Inicial	1 Byte	0x0000 até 0xFFFF
Quantidade de variáveis	2 Bytes	0x0001 até 0x07B0

Fonte: Elaborado pelo autor

Caso seja utilizada a Tabela 3 de variáveis e estados, os bytes de dados serão idênticos à da função *Read Coils*, como mostra a Tabela 4.

2.3.2.6 Write Multiple Registers – Função 16

Esta função é utilizada para a escrita de variáveis do tipo registro. A mensagem de *request* contém o código da função, o endereço inicial, a quantidade de registros a serem escritos, o contador de bytes e os valores dos registros. A mensagem de *response* contém o código da função, o endereço inicial e a quantidade de registros.

A quantidade de bytes será a quantidade de registros que foram escritos multiplicado por 2, pois cada variável de registro possui o tamanho de 2 bytes (0x0000 a 0xFFFF).

Os bytes de dados são estruturados de forma que os endereços apareçam de forma crescente, ou seja, os primeiros endereços estarão no primeiro byte e os últimos no último byte.

As Tabelas 13 e 14 identificam os campos, tamanhos e possíveis valores das mensagens de *Request* e *Response* para a função *Write Holding Registers*.

Tabela 13 - Write Holding Registers - Mensagem de Request

Campo	Tamanho	Valor
Código da função	1 Byte	0x10
Endereço Inicial	2 Bytes	0x0000 até 0xFFFF
Quantidade de registros	2 Bytes	0x0001 até 0x007B
Contador de bytes	1 Byte	N*2
Valores dos registros	N*2 Bytes	Bytes de dados

Fonte: Elaborado pelo autor

Tabela 14 - Write Holding Registers - Mensagem de Response

Campo	Tamanho	Valor
Código da função	1 Byte	0x03
Endereço Inicial	2 Byte	0x0000 até 0xFFFF
Quantidade de registros	2 Bytes	0x0001 até 0x007B

Fonte: Elaborado pelo autor

Caso seja utilizada a Tabela 7 de registros e valores, os bytes de dados serão idênticos à da função *Read Holding Registers*, como mostra a Tabela 8.

2.3.2.7 Exception Codes

No projeto foram implementadas 4 *Exception Codes*, que são descritas a seguir.

- *Exception Code* 01 – *Illegal Function* - Ocorre quando um cliente faz uma requisição de uma ação que não é permitida ao servidor ou cliente. Pode ocorrer também caso a função só possa ser executada ou foi implementada apenas em alguns dispositivos.
- *Exception Code* 02 – *Illegal Data Address* - Ocorre quando a requisição contém um endereço que não pode ser acessado pelo servidor ou cliente. Isso pode ocorrer por tentar acessar um local não existente ou quando o endereço final de leitura não for acessível. Por exemplo, se um dispositivo possui 1000 endereços e é feito um pedido de acesso ao endereço 1005, ocorerrá um *exception code* 02. O mesmo ocorrerá caso o pedido de acesso seja feito ao endereço 950 e a quantidade de endereços a serem lidos seja de 60.
- *Exception Code* 03 – *Illegal Data Value* - Ocorre quando o valor contido no campo de dados não é permitido ao servidor ou cliente. Mas nem sempre o erro é referente ao campo de dados, pois é possível ocorrer esse *exception code* quando o campo de comprimento do campo de dados não esteja correto.

- *Exception Code* 04 – *Slave Device Failure* - Ocorre quando um erro irrecuperável acontece enquanto o servidor ou cliente está realizando a ação requisitada.

A Tabela 15 ilustra a mensagem de *response* quando ocorre um E*xception Code*.

Tabela 15 - Exception Codes - Mensagem de Response

Campo	Tamanho	Valor
Código do erro	1 Byte	Função + 0x80
Código de exceção	1 Byte	01, 02, 03 ou 04

Fonte: Elaborado pelo autor

2.4 Conclusão

Através do estudo realizado nesse capítulo, é possível concluir que a simulação industrial é uma ferramenta que possui muitas vantagens e é aplicada em diversas ocasiões, seja como uma ferramenta pedagógica ou para testes de novas soluções e dispositivos.

Em contrapartida, existem algumas limitações para a utilização da simulação, como em casos de sistemas muito complexos, onde a simulação pode não obter resultados satisfatórios, ou quando não se tem recursos ou tempo suficientes para a sua realização.

O protocolo Modbus TCP pode ser utilizado como forma de comunicação entre os dispositivos na rede, permitindo acesso remoto e ponto a ponto entre clientes e servidores.

Os campos das mensagens Modbus foram citados, incluindo os tamanhos, possíveis valores e cabeçalhos necessários para a sua transmissão através do protocolo TCP.

Por fim, foram descritas algumas funções utilizadas no trabalho e sua forma de utilização. Caso os campos dessas funções estejam com valores incorretos, serão enviados pelo servidor *Exception Codes*, de acordo com o erro ocorrido.

No próximo capítulo será apresentada a metodologia aplicada para a criação do ambiente industrial simulado, incluindo os softwares criados e utilizados.

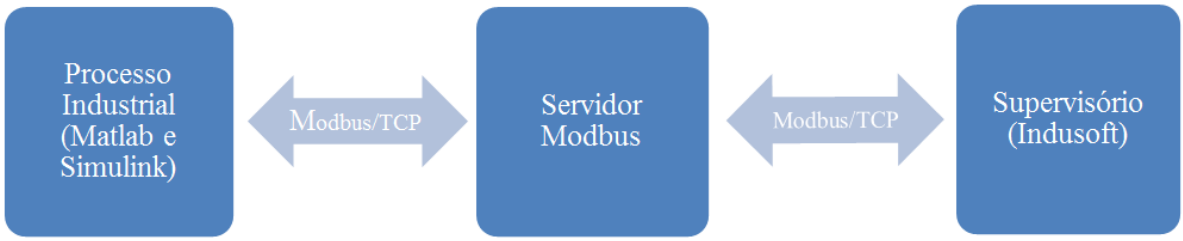
3 METODOLOGIA

3.1 Introdução

A simulação do ambiente industrial foi dividida em três partes, o servidor, o cliente e o supervisório. A função do servidor é gravar e disponibilizar os dados para o cliente e o supervisório. Esses dados podem ser do tipo *coil* ou registro. O cliente faz a simulação do processo industrial em tempo real e faz a troca de dados do processo com o servidor e o supervisório permite a visualização e alteração dos valores da planta. A comunicação entre as camadas é feita utilizando o protocolo Modbus/TCP.

O ambiente industrial é ilustrado pela Fig. 3, onde é possível visualizar suas camadas e a forma de comunicação entre elas.

Figura 3- Ambiente Industrial



Fonte: Elaborado pelo autor

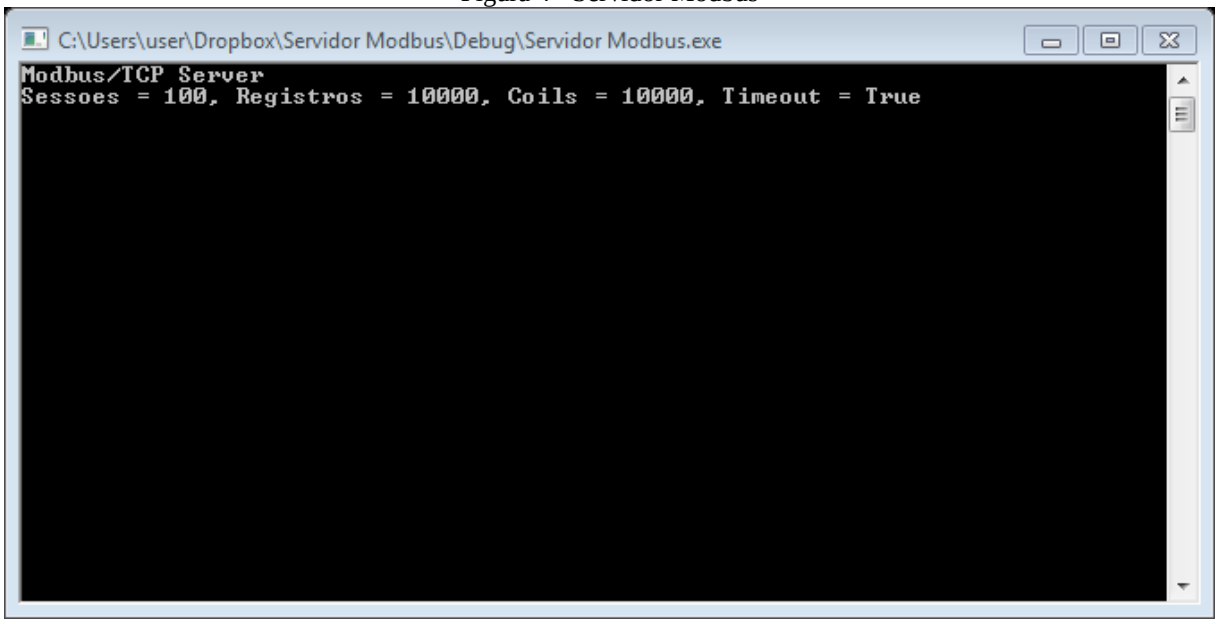
3.2 Servidor Modbus

O servidor utilizado é disponibilizado pela Modbus Organization (2013). Este servidor utiliza conexões via TCP/IP através de sockets, e permite a utilização de duas funções: *Write Multiple Registers* e *Read Holding Registers*. Ele foi implementado utilizando a linguagem C, aceita até 100 sessões simultâneas e possui 10000 valores de endereço disponíveis para a escrita e leitura de registros.

A fim de complementar este servidor, algumas modificações foram feitas. A primeira delas foi disponibilizar 10000 valores para as variáveis do tipo *coil*. Foram também implementadas as funções *Read Coils*, *Write Multiple Coils*, *Write Single Coil* e *Write Single Register*. Para cada uma delas, foram feitas suas respectivas respostas, incluindo as exceções, seja ela causada por tamanho irregular da mensagem ou tentativa de acesso a posições indisponíveis ou ilegais.

O servidor fica aguardando até que seja feita uma conexão. Ao receber uma mensagem Modbus TCP, ele irá processá-la e retornar a resposta correta. O código completo do servidor pode ser visualizado no Apêndice 1 e sua execução na Fig. 4.

Figura 4 - Servidor Modbus



Fonte: Dados do próprio autor

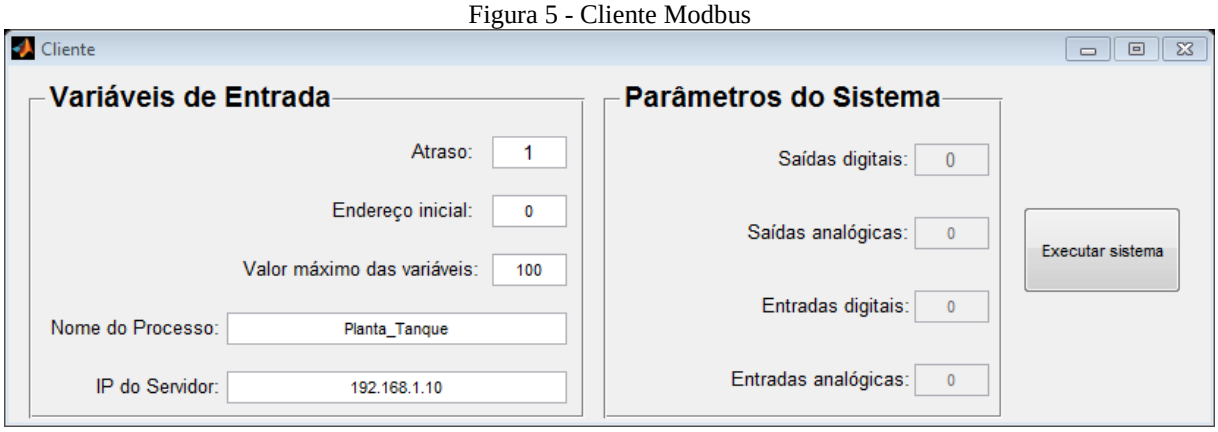
3.3 Cliente Modbus

A ferramenta *Simulink*, disponível no software Matlab®, foi utilizada para a simulação do ambiente industrial proposto. Assim é possível modelar todo o sistema e obter, em tempo real, as variáveis do processo, e manipulá-las.

Para realizar a conexão entre a planta simulada e o servidor, foi desenvolvido no Matlab® um programa que se conecta ao servidor através protocolo TCP/IP.

O programa desenvolvido no Matlab® foi dividido em duas etapas: aquisição dos dados do processo (saídas analógicas e digitais) e a leitura dos dados de entrada (*setpoints*, botões para acionamentos e parâmetros do controlador). A interface gráfica desenvolvida é ilustrada na Fig. 5 e o código fonte no Apêndice 2.

Figura 5 - Cliente Modbus



Fonte: Dados do próprio autor

A interface possui 5 entradas:

- Atraso: tempo de espera entre trocas de dados com o servidor;
- Endereço inicial: endereço inicial onde serão gravados os dados no servidor;
- Valor máximo das variáveis: valor máximo que as variáveis analisadas do processo podem atingir. Esse valor é utilizado para a conversão dos dados, pois os dados transmitidos deverão ser inteiros, variando de 0 a 65535, e os valores do processo são valores reais. Quanto menor este valor, maior precisão terão os valores aproximados.
- Nome do Processo: nome do arquivo do *Simulink* que será executado. É necessário que o projeto esteja localizado na mesma pasta que o cliente Modbus.
- IP do Servidor: endereço IP do servidor Modbus no qual o cliente fará a conexão.

São identificados como saídas todos os dispositivos da planta nomeados com o prefixo "saída_analog" ou "saída_digital" e como dados de entrada com o prefixo "entrada_analog" e "entrada_digital". A fim de designar uma ordem para a gravação dos dados nos registros do servidor, após cada um desses prefixos, deverá ser acrescido um número, começando com o numero 1 para o primeiro dispositivo, 2 para o segundo e assim sucessivamente.

Quando o botão "Executar Sistema" for pressionado, será inicialmente feito a configuração do objeto TCP, que requer como parâmetros o IP do servidor e a porta de acesso. Além disso, para habilitar a utilização de sockets, é necessário utilizar o parâmetro *'NetworkRole'*, seguido de *'client'*, para fazer a conexão como um cliente.

Feito a configuração do objeto TCP/IP, é necessário configurar a simulação e identificar o processo e suas variáveis. O modo de simulação é definido como sendo em tempo real e a simulação é iniciada. O cliente irá ler as variáveis de entrada da interface e irá fazer o reconhecimento das variáveis do processo simulado, identificando os blocos de entrada e saída.

Como os valores iniciais do servidor são desconhecidos, é feito uma leitura das variáveis de entrada do processo e estas são gravadas no servidor, através das funções *Write Multiple Registers* e *Write Multiple Coils*.

Enquanto o botão não for pressionado novamente, o cliente fará a leitura das variáveis de entrada (*setpoints*) do servidor através das funções *Read Coils* e *Read Holding Registers* e irá alterá-las no processo simulado. Em sequencia, fará a leitura dos valores de saída do processo e irá gravá-los no servidor através das funções *Write Multiple Registers* e *Write Multiple Coils*. Após cada ciclo é aguardado um tempo igual o atraso definido na interface gráfica.

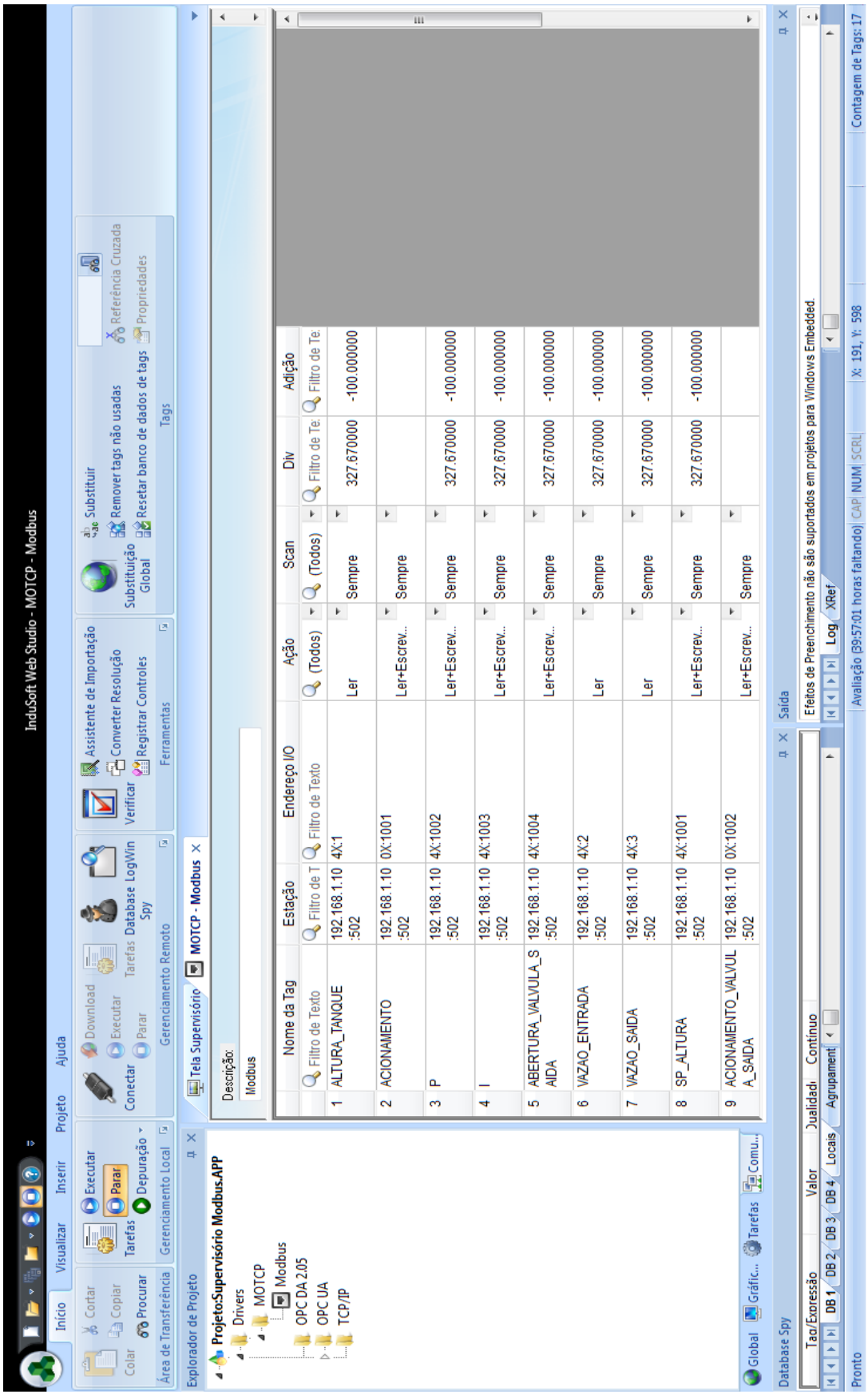
3.4 Supervisório

Para a leitura e alteração dos parâmetros do sistema, foi desenvolvido um sistema supervisório, através do software *Indusoft*®.

Nele inicialmente é feito a criação e a configuração do driver de conexão com o servidor. Para isso, é criado um *driver* do tipo MOTCP, que é um driver Modbus que utiliza o TCP/IP para a sua conexão. O *Indusoft*® utiliza as funções *Read Coils, Read Holding Registers, Write Single Coil e Write Single Register* para se comunicar com o servidor.

A configuração das variáveis é feita através dos endereços, onde as posições 4x são referentes aos registros e as 0x referentes aos *coils*. Cada variável necessita de uma *tag*, estação (IP do servidor), endereço (posições de memória do servidor), ações disponíveis (escrita, leitura ou ambos) e momentos para leitura. Caso seja necessário, os campos “Div” e “Adição” são utilizados para transformar os valores obtidos pela leitura do *software*, aplicando um valor para divisão ou soma, respectivamente. A Fig. 6 a seguir mostra as variáveis criadas para o problema proposto neste trabalho.

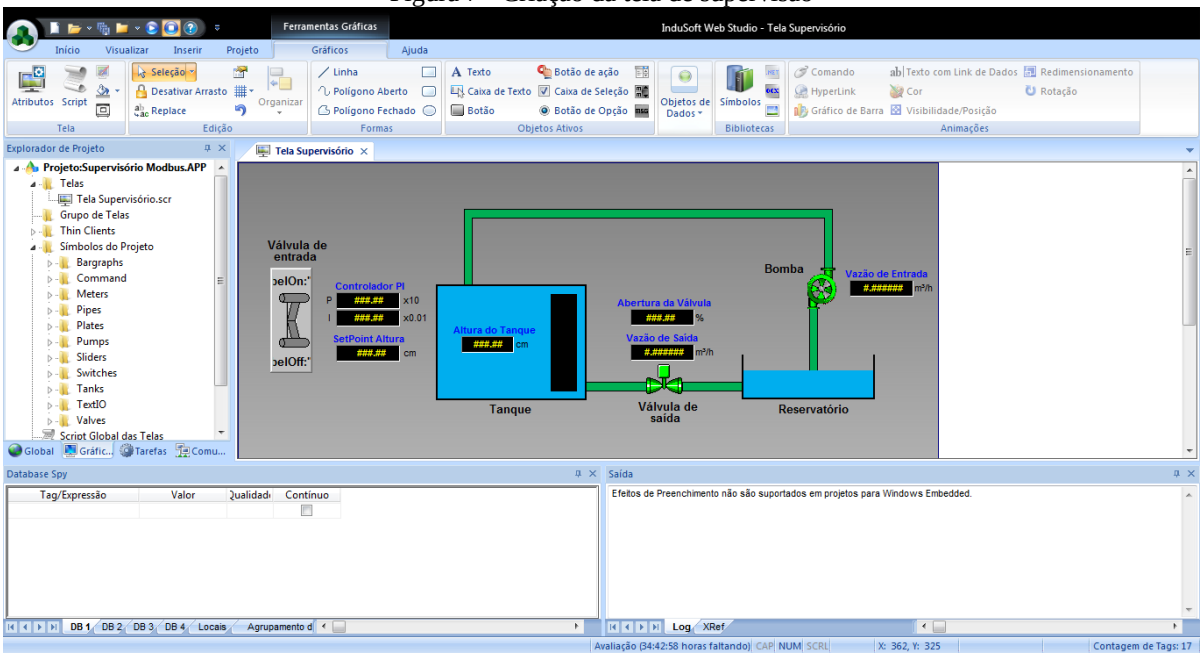
Figura 6 - Variáveis e configuração do *Indusoft*



Fonte: Elaborado pelo autor

A criação da tela de supervisão é feita utilizando símbolos fornecidos pelo programa, que podem executar animações ou alterar textos e/ou posição de acordo com o estado ou valor das variáveis relacionadas a eles. Utilizando esta ferramenta de relacionamento de variáveis a símbolos e animações é possível criar uma interface de fácil visualização e manipulação dos dados do processo. A Fig. 7 ilustra uma tela criada no software *Indusoft*.

Figura 7 - Criação da tela de supervisão



Fonte: Elaborado pelo autor

3.5 Conclusão

Nesse capítulo foi apresentada a metodologia utilizada para desenvolver o ambiente industrial simulado, além de cada software utilizado.

O servidor foi desenvolvido na linguagem C e permite que o cliente Modbus e o supervisório se conectem a ele através do protocolo Modbus TCP.

O cliente Modbus foi desenvolvido no Matlab®, que faz conexão com o servidor, realiza a criação das mensagens Modbus e utiliza a ferramenta *Simulink*, que permite a realização da simulação do processo em tempo real, tal como a modificação dos parâmetros da planta.

Através do supervisório é possível criar telas, acessar o servidor e manipular algumas de suas variáveis. Nas telas de supervisão podem ser adicionados elementos que interagem com as variáveis da planta, criando um ambiente de fácil entendimento e visualização.

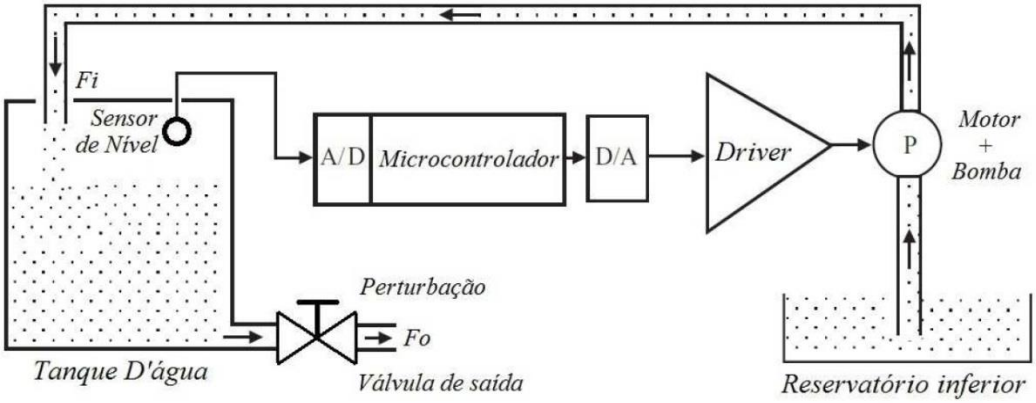
O próximo capítulo irá apresentar a modelagem de um processo e os resultados obtidos em cada um dos softwares utilizados.

4 RESULTADOS

4.1 Introdução

Este capítulo apresenta a modelagem de um processo industrial, que será uma planta de controle de nível de um tanque, e os testes realizados utilizando a ferramenta computacional desenvolvida. A modelagem realizada utiliza aproximações, devido ao fato de que o foco do projeto não é a modelagem do sistema e sim a construção do ambiente industrial simulado. O processo é ilustrado na Fig. 8.

Figura 8. Planta industrial de nível de um tanque



Fonte: Elaborado pelo autor

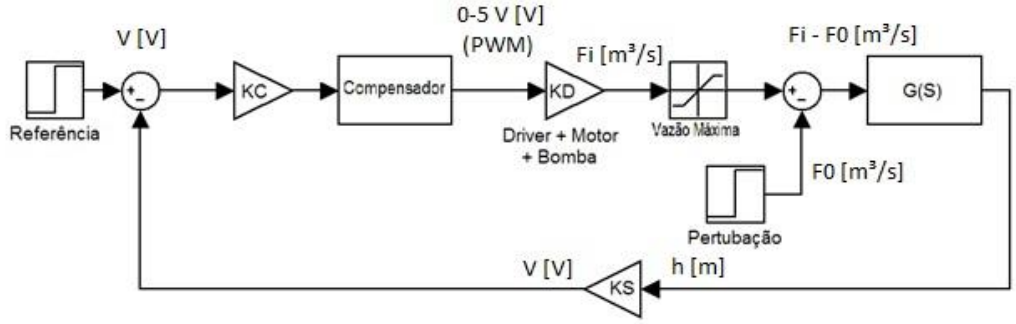
Os resultados serão validados através do software *Wireshark*, que possibilita a leitura dos pacotes que estão trafegando na rede.

4.2 Modelagem do processo

O diagrama de blocos para o controle do sistema leva em consideração que o sensor, o driver e o motor são apenas ganhos estáticos, sendo desprezados por apresentar funções de transferência muito mais rápidas que a do controle utilizado.

A Fig. 9 esquematiza a malha de controle, já com o compensador.

Figura 9. Diagrama de blocos para controle de nível



Fonte: Elaborado pelo autor

O micro controlador, que é representado pelo compensador e pelo ganho KC, receberá a referência em sua entrada que, comparada com o sinal proveniente do sensor de nível, enviará um sinal de erro para ser compensado. A saída é baseada no controle de PWM (P*ulse Width Modulation*), de forma a se controlar tensão no motor e, consequentemente, a vazão de saída necessária para o controle do nível de fluido do tanque. A função do driver será amplificar o sinal proveniente da saída do micro controlador de modo que o mesmo seja capaz de acionar o motor.

A relação da vazão de entrada pela altura de água no reservatório pode ser utilizada para o cálculo matemático da função de transferência do processo, é demonstrada na Eq. (1).

$$F_i = F_0 + A.\frac{dh}{dt} \tag{1}$$

F_i: Vazão de entrada (bomba) [m³/s]

F_0: Vazão de saída [m³/s]

A: Área do fundo do tanque [m²]

h: altura de líquido [m]

Utilizando a transformada de Laplace, obtemos a Eq. (2), para condição inicial nula.

$$F_i(s) - F_0(s) = A.s.h(s) \tag{2}$$

Como simplificação, a vazão de saída F_0 será considerada como uma perturbação do sistema, não sendo, portanto, considerada no cálculo da função da planta. Assim, a função de transferência é demonstrada pela Eq. (3).

$$G(s) = \frac{\text{saída}}{\text{entrada}} = \frac{h(s)}{F_i(s) - F_0(s)} = \frac{1}{A.s} \quad (3)$$

A vazão máxima de saída também pode ser estimada, considerando-se a maior coluna de fluido igual à maior altura de controle do tanque e a vazão máxima da válvula e é obtida através da Eq. (4).

$$F_0 = C.A_0.\sqrt{2.g.h} \quad (4)$$

F_0: Vazão de saída [m³/s]
A_0: Área de seção transversal da tubulação [m²]
h: altura de líquido [m]
g: aceleração da gravidade [m/s²]
C: coeficiente de descarga

O ganho estático do driver é determinado pela Eq. (5).

$$K_{driver} = \frac{\text{tensão de saída}}{\text{tensão de entrada}} \quad (5)$$

Os ganhos do motor e da bomba são considerados como um só e determinados através da Eq. (6).

$$K_{motor+bomba} = \frac{\text{vazão de saída}}{\text{tensão de entrada}} \quad (6)$$

O ganho K_d será o produto dos ganhos K_{driver} e $K_{motor+bomba}$.

Para determinar o ganho estático do sensor, é considerado que para a altura máxima de controle sua tensão de saída será máxima. Portanto, para o cálculo do ganho, é utilizada a Eq. (7).

$$K_s = \frac{\text{tensão de saída máxima}}{\text{altura máxima}} \quad (7)$$

Foram utilizados os seguintes parâmetros na simulação:

- Tensão máxima do PWM = 5 V

- Tensão máxima no motor = 24 V
- Área do tanque = 3.61 x 10^{-2} m²
- Altura máxima do tanque = 0.3 m
- Área de seção transversal da tubulação = 1.04 x 10^{-3} m²
- Vazão máxima de entrada = 2.5 x 10^{-4} m³/s
- Vazão máxima de saída = 1.51 x 10^{-4} m³/s
- Aceleração da gravidade = 9.8 m/s²
- Coeficiente de descarga = 0.6

Com esses parâmetros e as equações (3) (4) (5) (6) (7), tem-se que:

$$G(s) = \frac{1}{0.0361.s} \tag{8}$$

$$F_0 = 0.000151\ m^3/s \tag{9}$$

$$K_{driver} = \frac{24}{5} \tag{10}$$

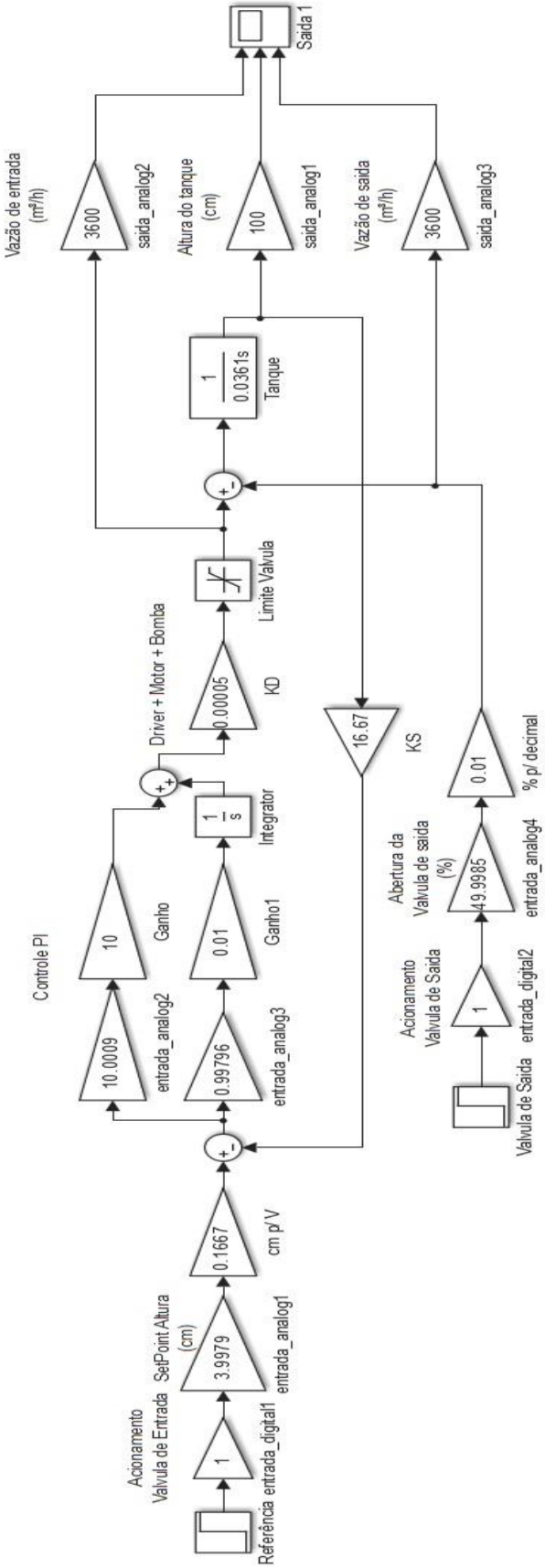
$$K_{motor+bomba} = \frac{0.00025}{24}\ m^3.V/s \tag{11}$$

$$K_d = 0.00005\ m^3.V/s \tag{12}$$

$$K_s = 16.67\ V/m \tag{13}$$

O diagrama de blocos para o sistema simulado é apresentado na Fig. 10.

Figura 10. Diagrama de blocos do sistema simulado



Fonte: Elaborado pelo autor

Nesse sistema, foram analisadas 9 variáveis, que são detalhadas na Tab. 16, que ilustra os tipos e a descrição de cada uma delas.

Tabela 16 - Variáveis do sistema simulado

Variável	Tipo da Variável	Descrição da Variável
entrada_digital1	Entrada Digital	Acionamento da Válvula de Entrada
entrada_digital2	Entrada Digital	Acionamento da Válvula de Saída
entrada_analog1	Entrada Analógica	SetPoint Nível (cm)
entrada_analog2	Entrada Analógica	Ganho proporcional do controlador
entrada_analog3	Entrada Analógica	Ganho integral do controlador
entrada_analog4	Entrada Analógica	Abertura da válvula de saída (%)
saida_analog1	Saída Analógica	Nível do Tanque (cm)
saida_analog2	Saída Analógica	Vazão de entrada (m³/h)
saida_analog3	Saída Analógica	Vazão de saída (m³/h)

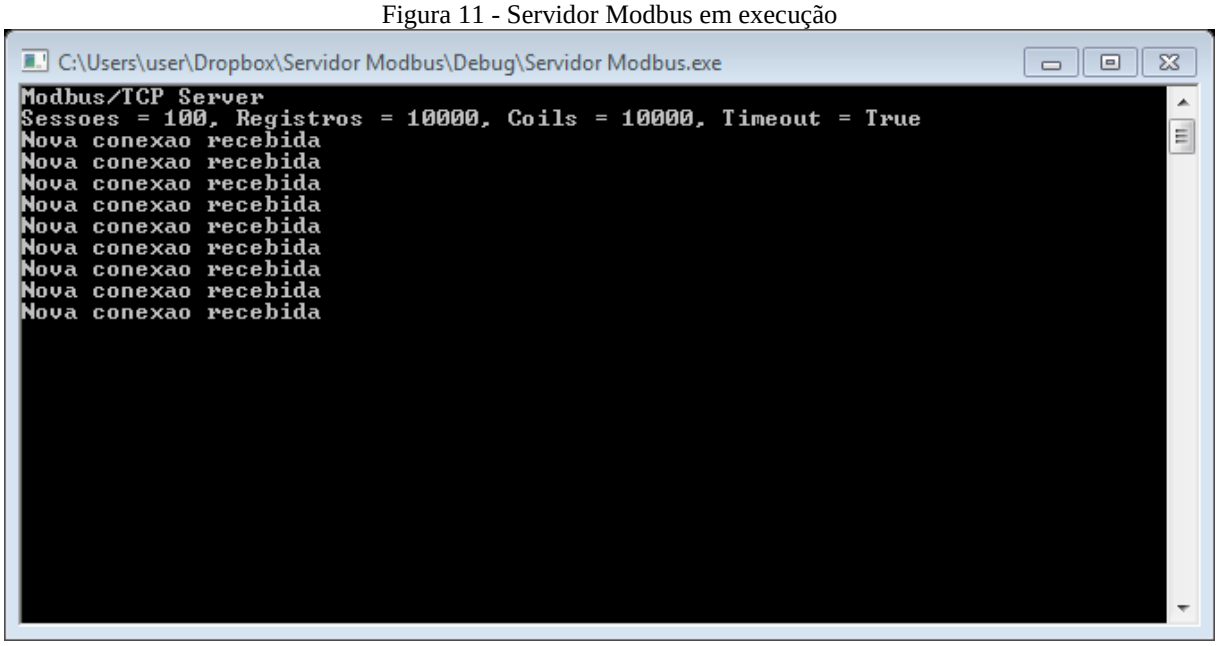
Fonte: Elaborado pelo autor

4.3 Resultados

4.3.1 Servidor

O *software* é executado em uma máquina virtual e aguarda o estabelecimento da conexão do cliente Modbus e do supervisório. A mensagem “Nova conexão recebida” é apresentada quando o servidor recebe uma conexão, como demonstrado na Fig. 11. Ele irá responder aos pedidos do cliente e do supervisório, processando a mensagem recebida e enviando a resposta, de acordo com as regras definidas pelo protocolo Modbus.

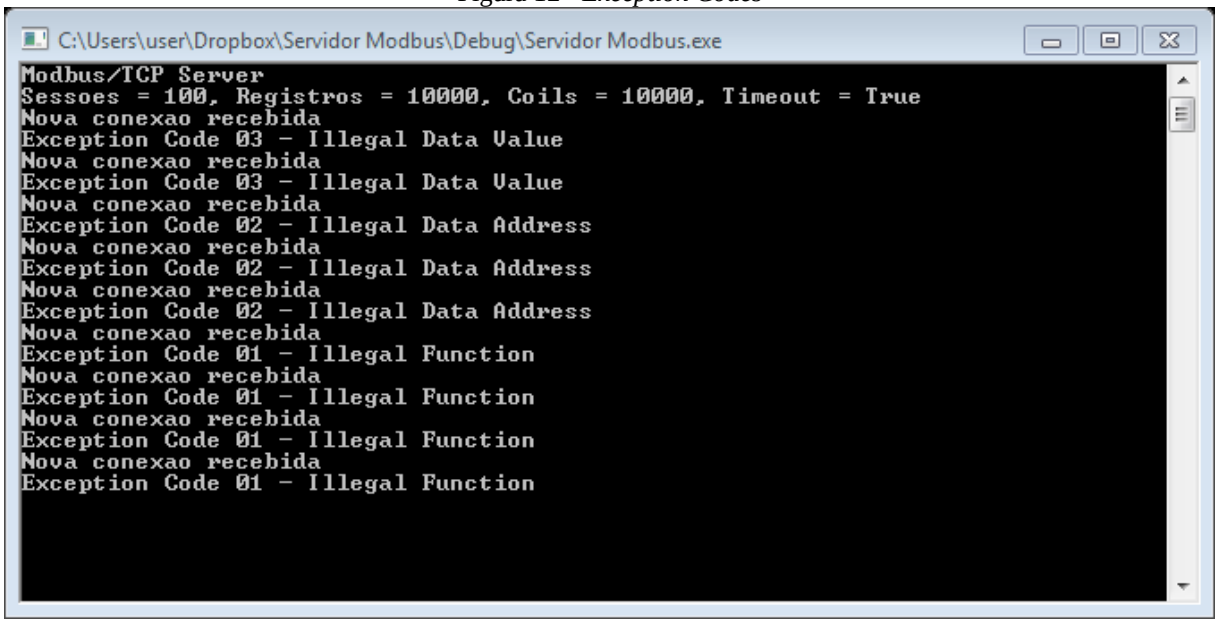
Figura 11 - Servidor Modbus em execução



Fonte: Elaborado pelo autor

Quando o pedido não for válido, o servidor irá retornar *Exception Codes*, como mostra a Fig. 12.

Figura 12 - *Exception Codes*



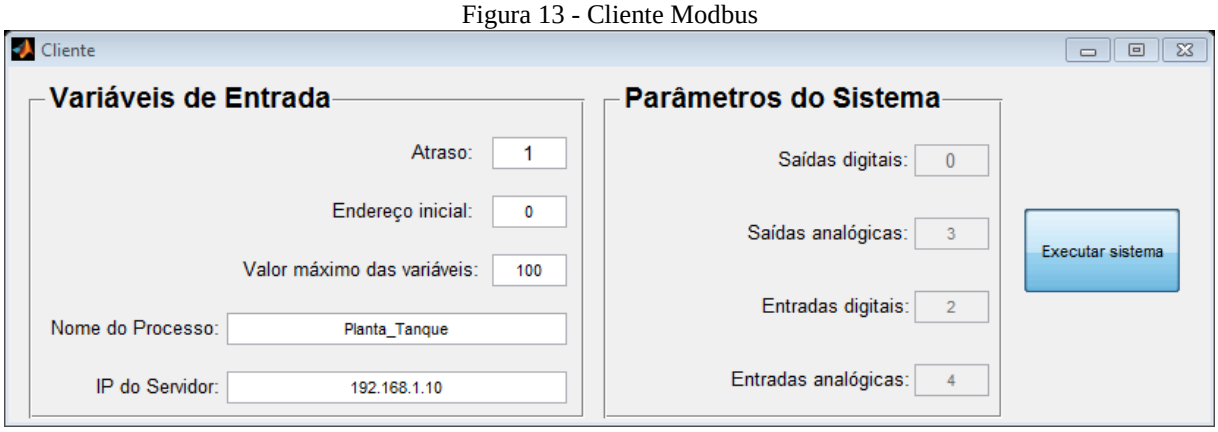
Fonte: Elaborado pelo autor

4.3.2 Cliente Modbus

O Cliente Modbus inicia a simulação da planta em tempo real e realiza a troca de dados entre o servidor e o processo simulado. A quantidade dos parâmetros do sistema é

atualizada automaticamente na interface do cliente durante a inicialização e é ilustrada, em conjunto com as variáveis de entrada, na Fig. 13.

Figura 13 - Cliente Modbus

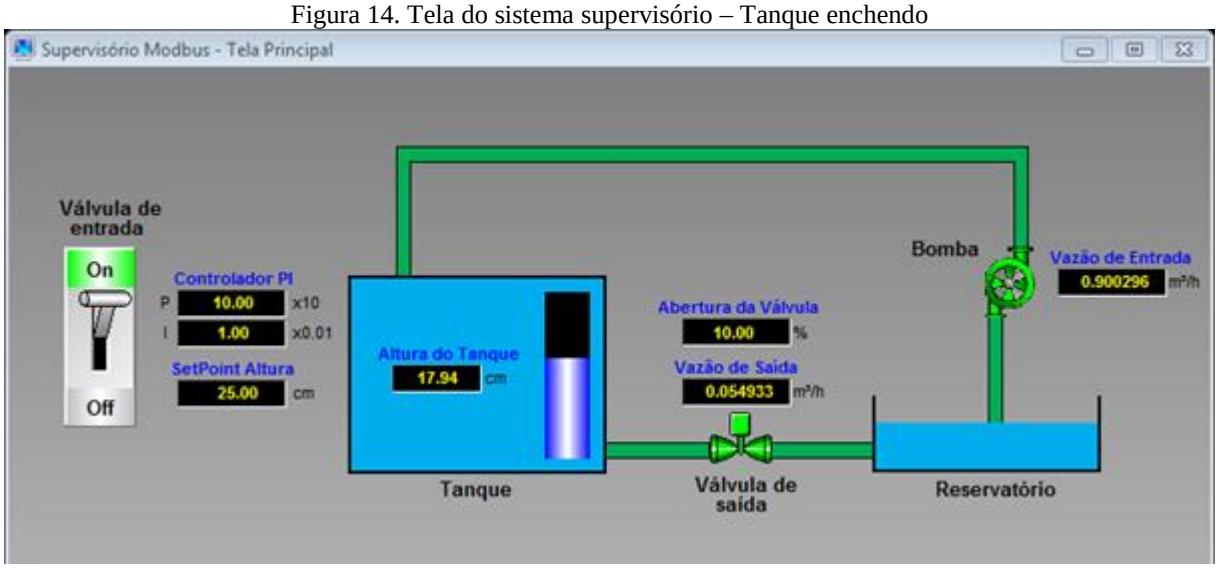


Fonte: Elaborado pelo autor

4.3.3 Supervisório

A tela de supervisão desenvolvida permite ao usuário a visualização dos dados da planta de nível, bem como a possibilidade de alteração dos valores de *setpoints* do processo e os parâmetros do controlador. Nela também é possível acionar ou desligar os dispositivos do sistema, como motores e bombas. A tela de supervisão em execução pode ser vista na Fig. 14.

Figura 14. Tela do sistema supervisório – Tanque enchendo



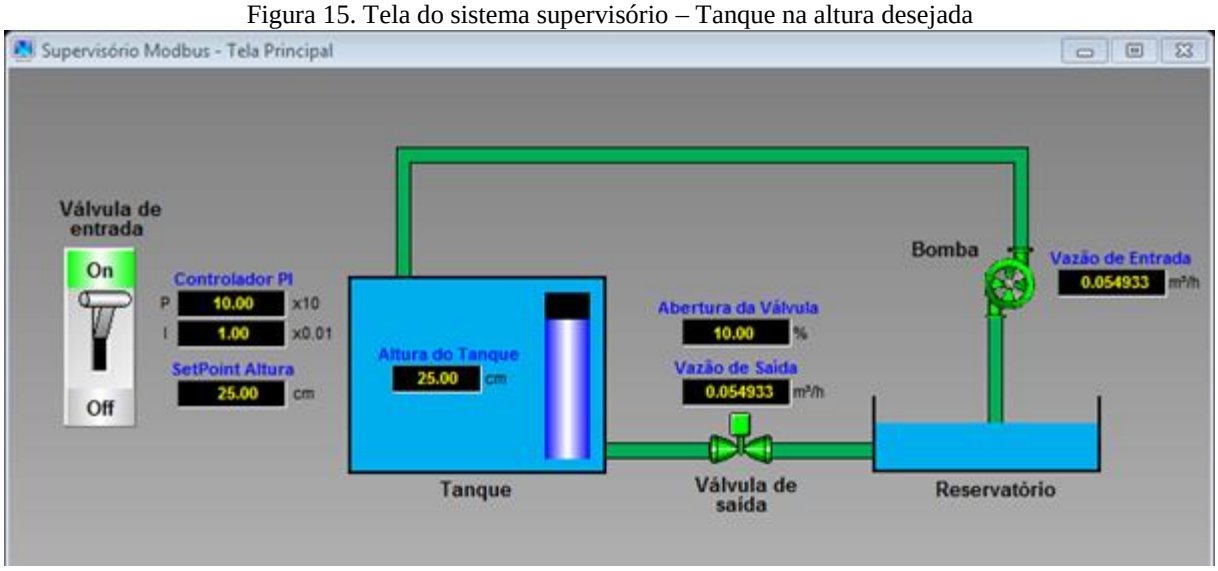
Fonte: Elaborado pelo autor

A válvula de entrada aciona a bomba que enche o tanque, utilizando a água do reservatório. Os valores das ações proporcional e integral do controlador também podem ser alterados clicando no valor atual das mesmas e digitando o valor desejado.

Através do Set Point, define-se a altura de água que o tanque deve atingir e nele é possível ver a altura atual de líquido. Quando o tanque está cheio, pode-se notar que a vazão

de saída é igual à vazão de entrada, mantendo assim o volume de água no tanque constante, e consequentemente, a altura. Esta situação é ilustrada pela Fig. 15.

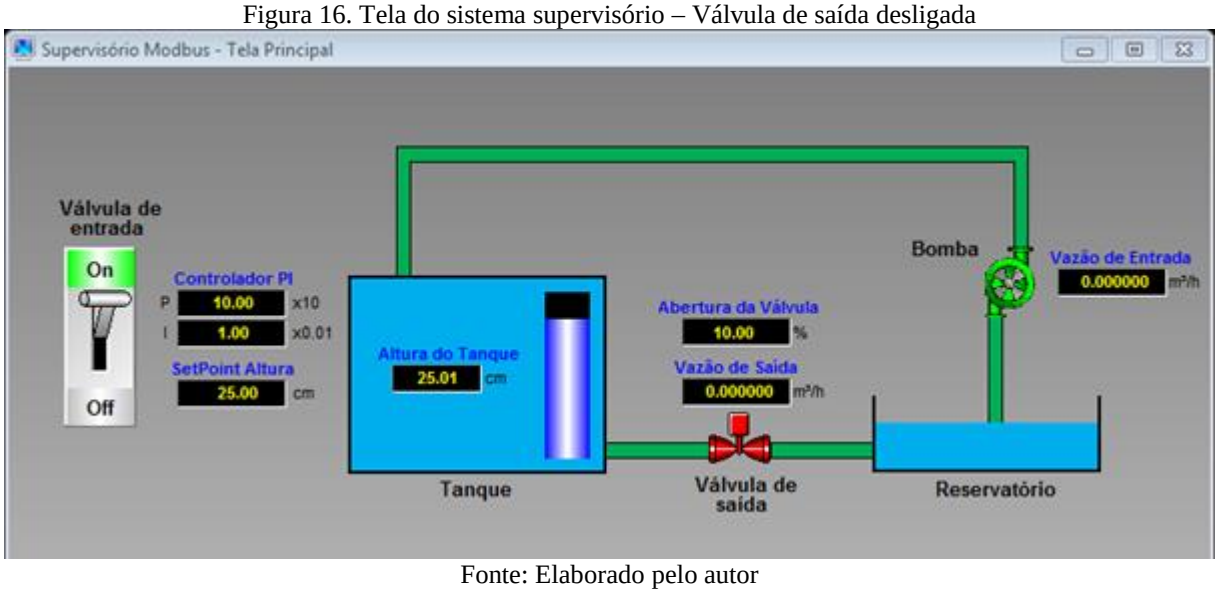
Figura 15. Tela do sistema supervisório – Tanque na altura desejada



Fonte: Elaborado pelo autor

Ao clicar na válvula de saída, ela é desativada e fica com a coloração vermelha, como ilustra a Fig. 16. A vazão de saída também é alterada variando a abertura da válvula. Um pequeno erro de regime permanente aparece ao se utilizar uma grande abertura na válvula de saída, devido às aproximações feitas durante a modelagem.

Figura 16. Tela do sistema supervisório – Válvula de saída desligada



Fonte: Elaborado pelo autor

4.3.4 Análise dos pacotes na rede

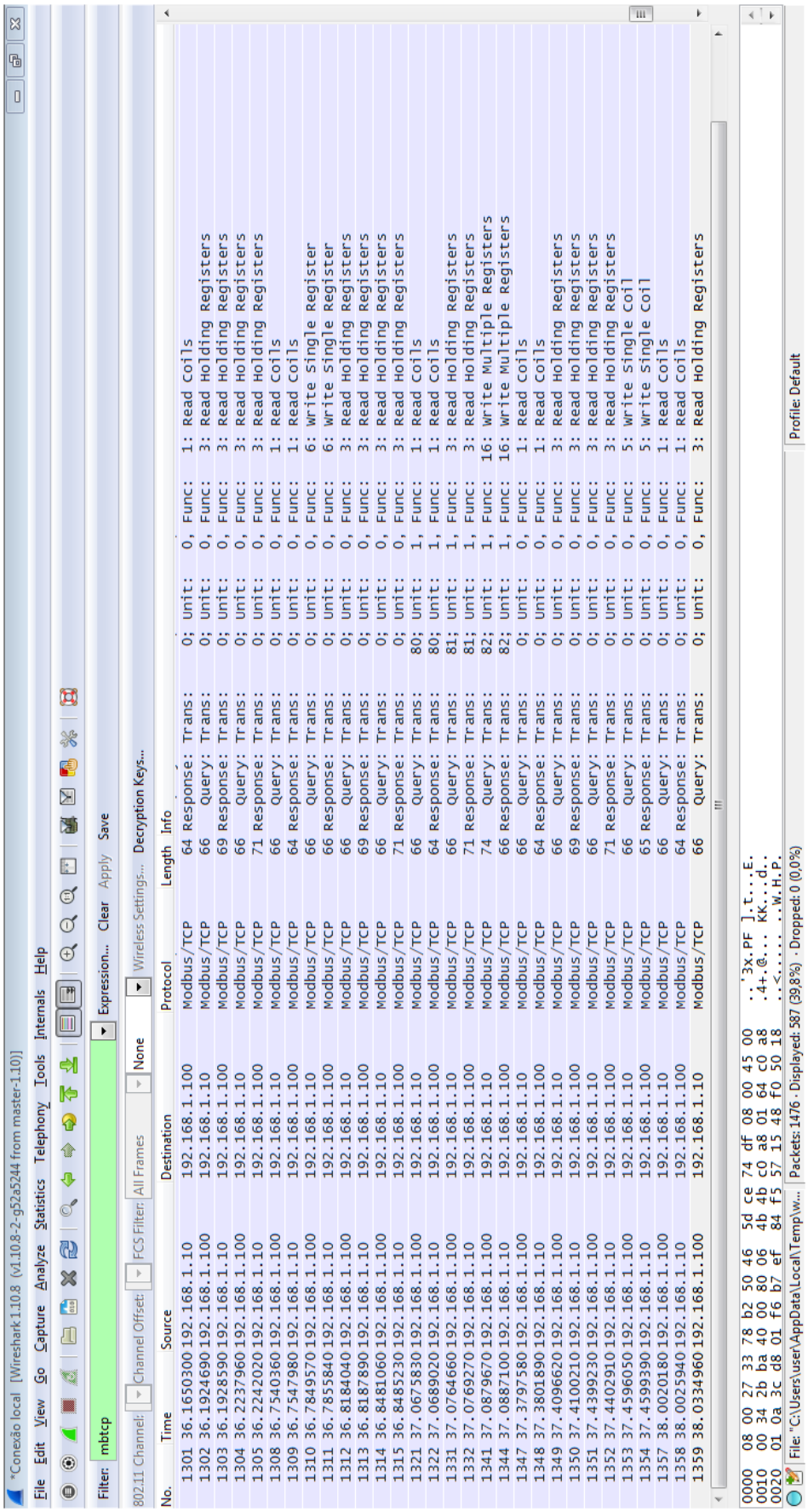
Para fazer a análise dos pacotes na rede, foi utilizado o *software Wireshark*®. Ele tenta capturar os pacotes da rede e mostrar os dados desses pacotes da forma mais detalhada

possível. Algumas das funcionalidades do *Wireshark* são (WIRESHARK FOUNDATION, 2004):

- Solucionar problemas na rede;
- Examinar problemas de segurança da rede;
- Testar implementações utilizando protocolos;
- Aprender sobre os protocolos.

A fim de comprovar o desenvolvimento das outras funcionalidades do protocolo Modbus TCP, foi utilizado o software *Wireshark*® para analisar os pacotes transmitidos na rede, através do qual é possível visualizar detalhadamente cada pacote e filtrar qual protocolo será monitorado. A Fig. 17 ilustra os pacotes Modbus/TCP que estão trafegando na rede durante a execução do sistema, onde é possível observar as funções *Write Multiple Registers*, *Read Coils*, *Read Holding Registers*, *Write Single Register* e *Write Single Coil* serem executadas.

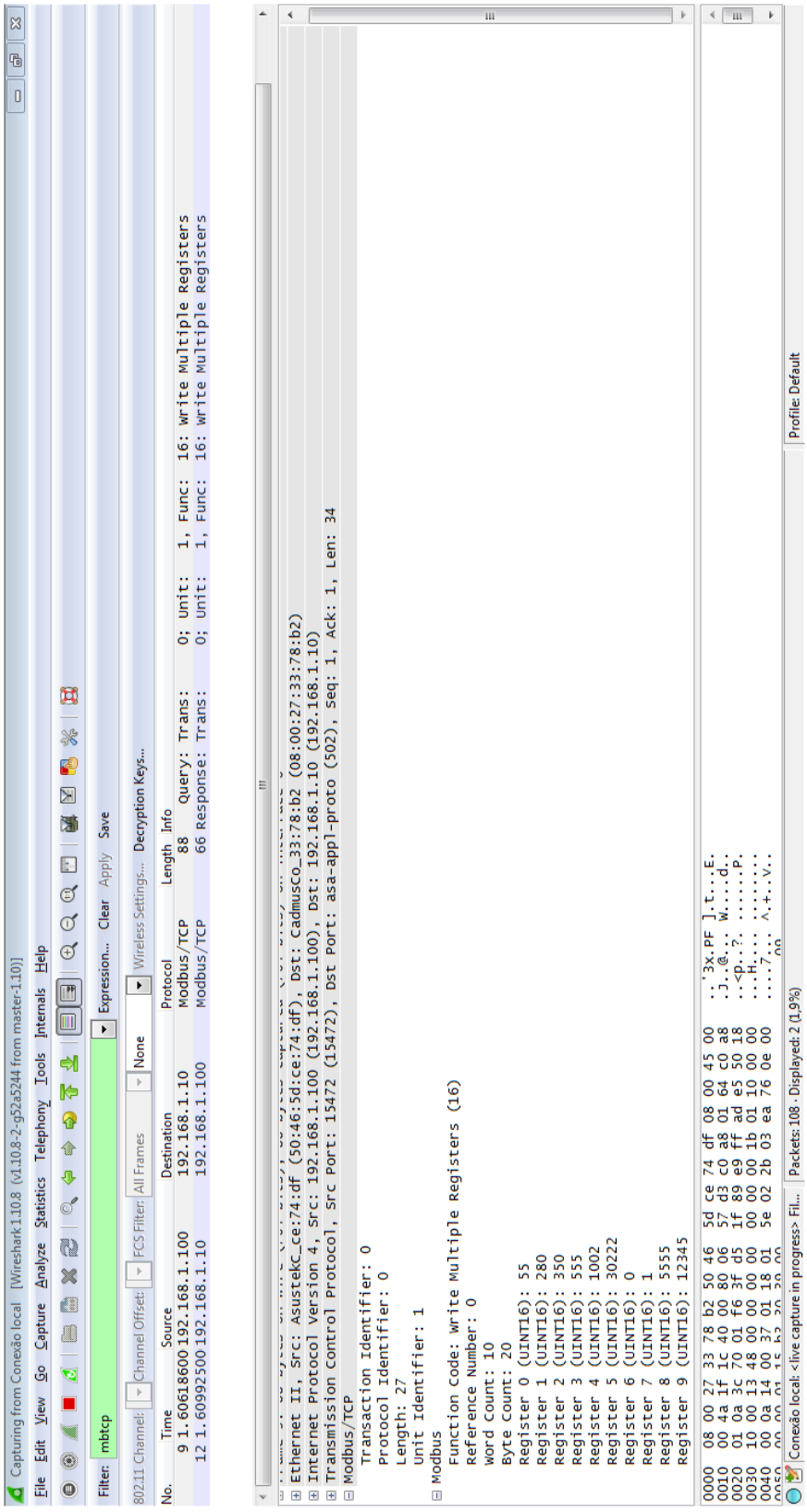
Figura 17. Pacotes da rede Modbus



Fonte: Elaborado pelo autor

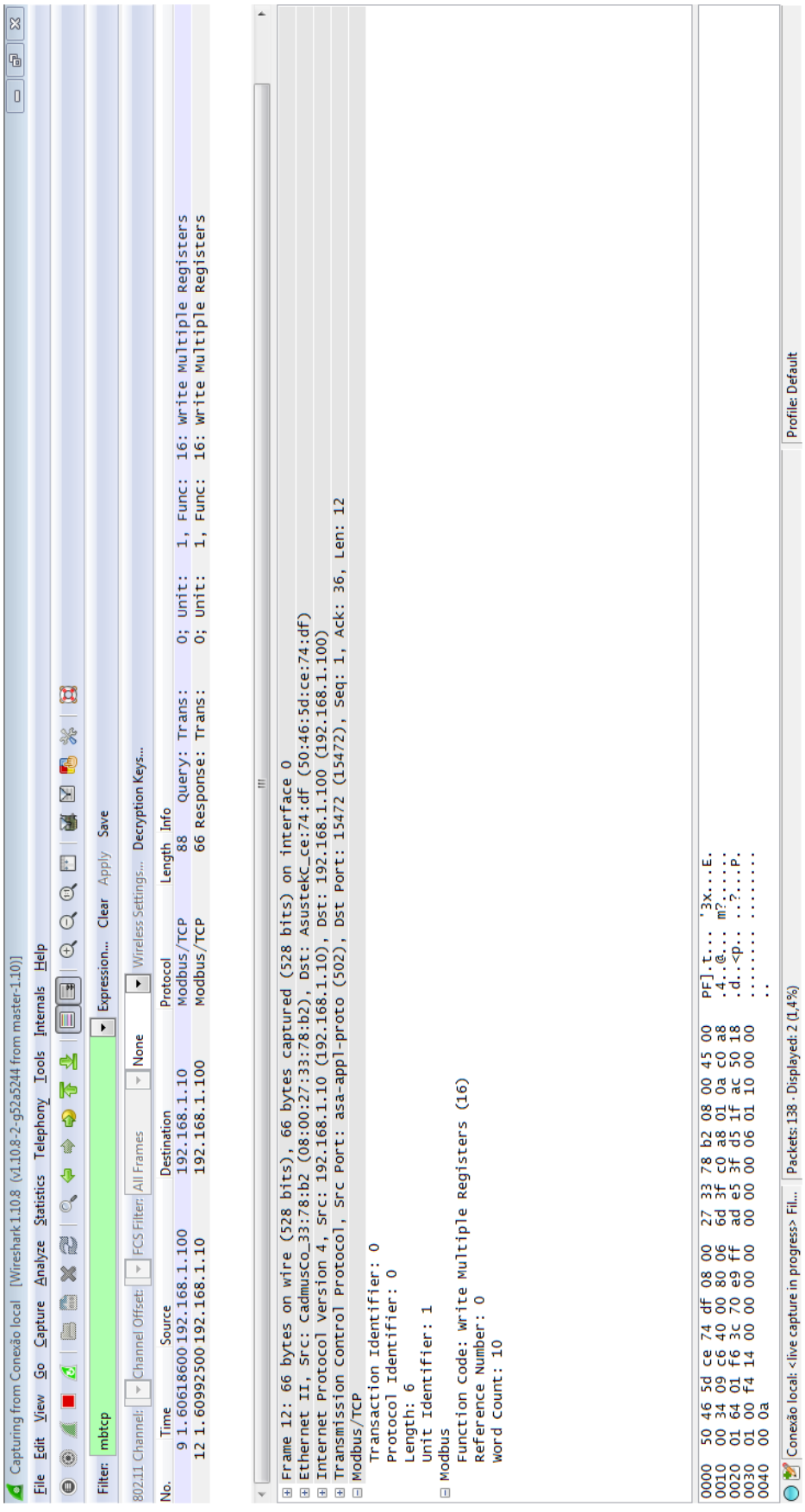
A Fig. 18 e Fig. 19 demonstram um pedido e uma resposta do servidor à função *Write Multiple Registers*, mostrando todos os campos e valores da mensagem Modbus.

Figura 18 - Wireshark - Write Multiple Registers – Request



Fonte: Elaborado pelo autor

Figura 19- Wireshark - Write Multiple Registers – Response

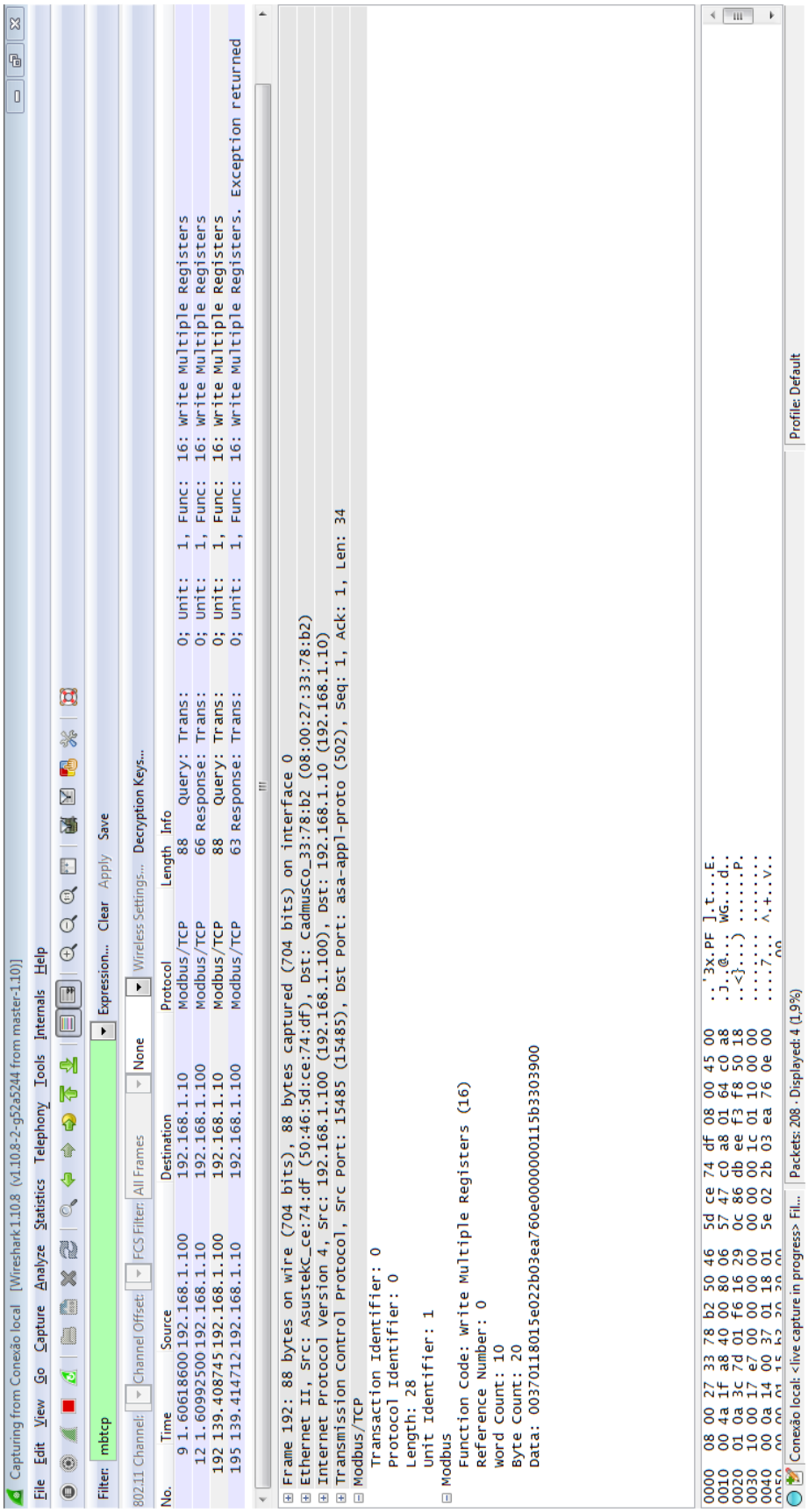


Fonte: Elaborado pelo autor

Com o objetivo de certificar a resposta do servidor nos casos de pedidos incorretos (*Exception Codes*), o código do cliente Matlab foi modificado, criando mensagens com parâmetros fora dos limites possíveis ou com valores incorretos.

Primeiramente, foi testado o envio de uma mensagem da função *Write Multiple Registers* com o campo de comprimento incorreto. Este campo deve possuir valor igual a 7 mais o dobro de número de registros. A mensagem enviada possui 10 registros a serem escritos no servidor e o campo de comprimento com o valor de 28. A Fig. 20 ilustra a mensagem enviada.

Figura 20- Wireshark - Write Multiple Registers – Comprimento incorreto



Fonte: Elaborado pelo autor

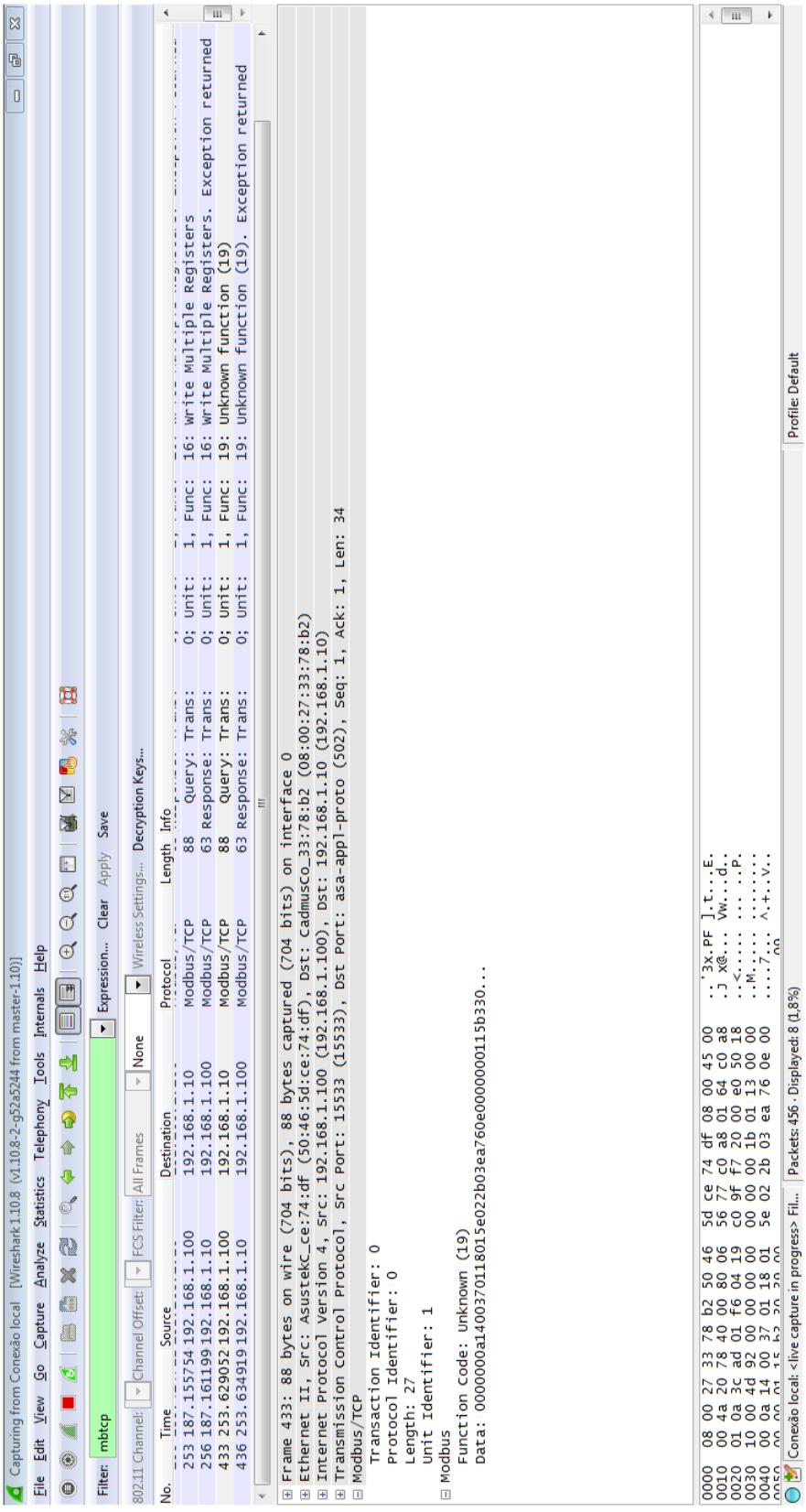
Em seguida, uma mensagem em que o endereço inicial em soma com a quantidade de registros a serem escritos excede o limite do servidor é ilustrada na Fig. 21.

Figura 21 - Wireshark - Write Multiple Registers – Endereço incorreto

Fonte: Elaborado pelo autor

E por fim, é solicitada uma função de código 19, que não existe no servidor, como ilustra a Fig. 22.

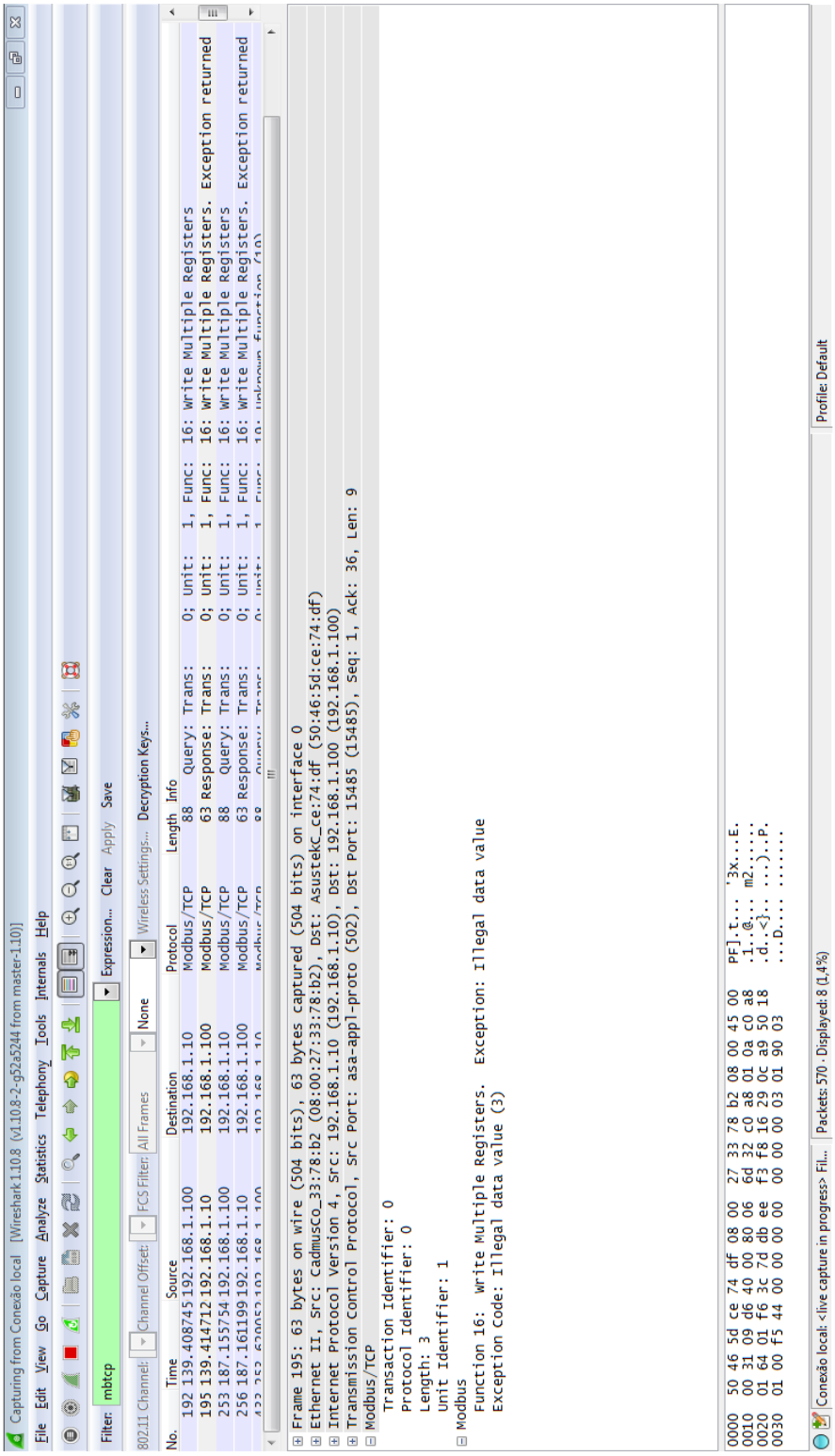
Figura 22 - Wireshark – Função incorreta



Fonte: Elaborado pelo autor

As respostas recebidas são, respectivamente, um *Exception Code* 03, 02 e 01, que são as respostas corretas nos casos citados, como apresentado anteriormente no Capítulo 2. A Fig 23, 24 e 25 ilustram as respostas do servidor ao cliente para as mensagens anteriores.

Figura 23- Wireshark - Write Multiple Registers – Comprimento incorreto – Response

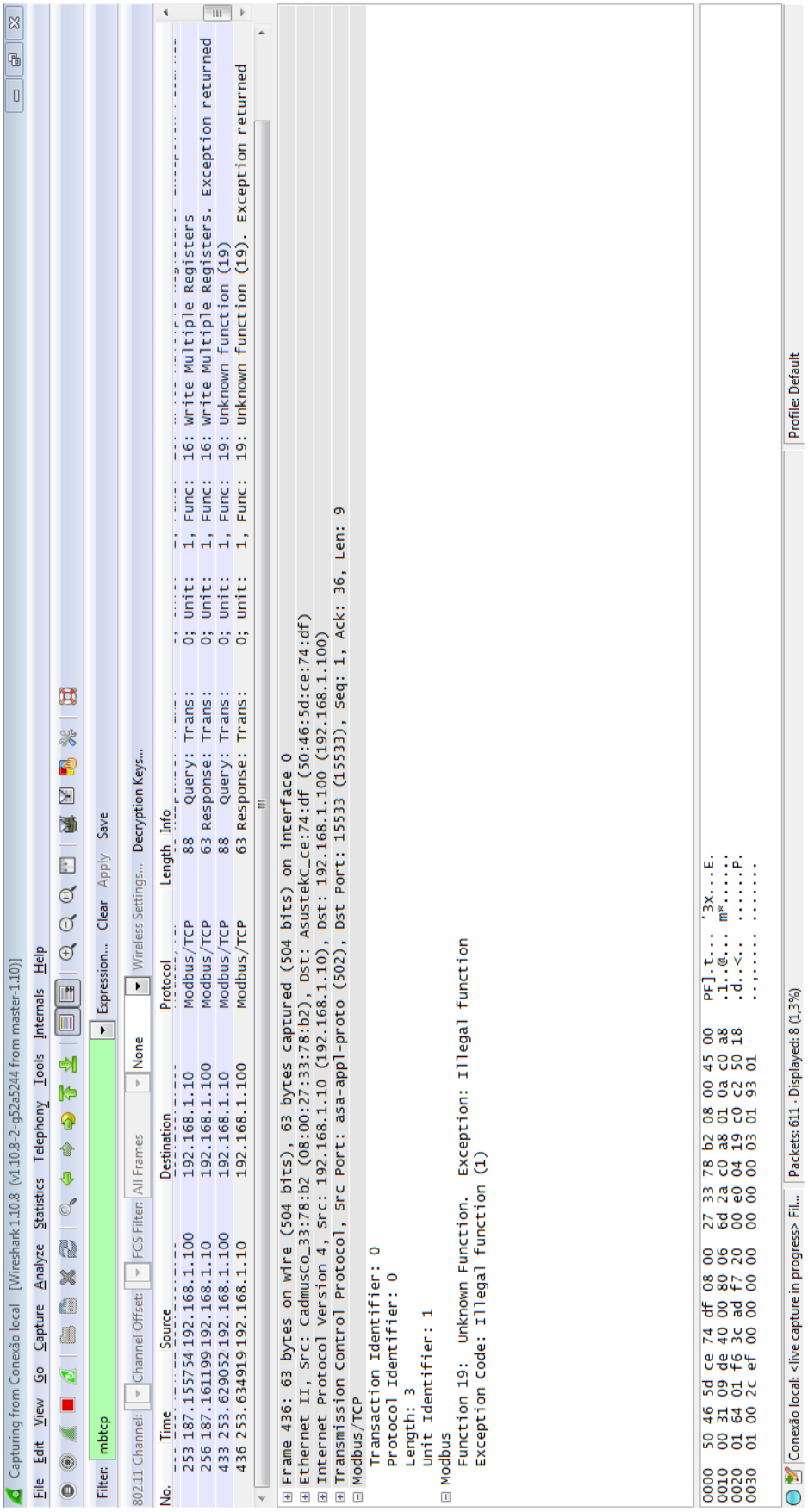


Fonte: Elaborado pelo autor

Figura 24- Wireshark - Write Multiple Registers – Endereço incorreto – Response

Fonte: Elaborado pelo autor

Figura 25 - Wireshark – Função incorreta – Response



Fonte: Elaborado pelo autor

5 CONCLUSÃO

Na introdução do trabalho foram apresentadas as aplicações e vantagens da simulação industrial, que se prova uma boa opção para indústrias que desejam minimizar os custos de operação e obter equipamentos adequados ao seu processo, além de auxiliar estudantes que desejam aprofundar seus conhecimentos sobre processos industriais.

Também foram citados alguns dos dispositivos presentes em ambientes industriais e a necessidade da comunicação entre eles, que pode ser feita através de diversos protocolos de redes industriais. Além disso, foram citadas pesquisas na área de desenvolvimento e pesquisas de protocolos de redes industriais.

Nesse contexto, tem-se como proposta de trabalho o desenvolvimento de uma ferramenta computacional aplicada na simulação de processos industriais, utilizando o software Matlab® e a ferramenta *Simulink*, que possibilita uma simulação em tempo real de uma planta industrial, que fará a conexão com um servidor através do protocolo Modbus/TCP. Este servidor permite o acesso de um sistema supervisório, que pode fazer a manipulação e visualização dos dados do processo.

No Capítulo 2 é apresentada uma definição de simulação industrial, indicando suas vantagens, desvantagens, situações onde ela deve ou não ser utilizada e áreas aonde vem sendo utilizada no cenário brasileiro. Em seguida, o protocolo Modbus é também definido, citando suas principais versões, a estrutura das mensagens, como são feitas as comunicações entre os dispositivos e as funções disponibilizadas pelo protocolo que foram utilizadas no trabalho.

No Capítulo 3 foram apresentadas as 3 partes em que foi dividida a ferramenta computacional desenvolvida: servidor Modbus, cliente Modbus e supervisório. O servidor Modbus desenvolvido disponibiliza seis funções do protocolo Modbus, que permitem a escrita e leitura de dados discretos e analógicos do sistema. O software do servidor mostra mensagens quando recebe conexões e quando recebe mensagens com erros, respondendo com os Exception Codes descritos no Capitulo 2.

O software Matlab®, em conjunto com a ferramenta *Simulink*, fazem o papel do cliente Modbus. Eles possibilitam a simulação do processo de forma precisa e rápida, onde é possível simular, realizar trocas de informações e atualizar os dados do processo em tempo real. A interface desenvolvida permite inserir o tempo entre as leituras realizadas no servidor, o endereço inicial a ser utilizado, a planta que será simulada e o IP do servidor.

O supervisório é feito através do software *Indusoft*®, que possibilita o monitoramento dos dados através de uma tela de supervisão criada, possuindo objetos que interagem com o usuário, de forma a facilitar o entendimento e visualização das variáveis do processo.

No Capítulo 4 é realizada a modelagem, de forma aproximada, de uma planta de controle de nível de um tanque. Esse sistema possui 9 variáveis, das quais 6 são variáveis de entrada e 3 variáveis de saída. São apresentadas as telas dos *softwares* utilizados em execução, mostrando o servidor recebendo conexões e mensagens incorretas, o cliente atualizando os parâmetros do sistema e o sistema de supervisão monitorando as variáveis do processo. Para validar os resultados obtidos, é utilizando um *software* de captura de pacotes da rede chamado Wireshark, o qual possibilita ver os dados das mensagens enviadas pelo cliente e as suas respectivas respostas geradas pelo servidor.

Portanto, a ferramenta desenvolvida permite a simulação de todo o processo industrial, possibilitando sua aplicação nas mais diversas áreas de aplicação, como o estudo comportamental de plantas industriais, desenvolvimento de novos dispositivos e sistemas e a realização de testes em situações extremas ou de risco. Isso é feito através de *softwares* com interfaces simples, de forma a facilitar para o usuário a sua utilização, onde as variáveis do processo são facilmente visualizadas e manipuladas pelo supervisório.

Como desvantagem da ferramenta computacional desenvolvida é que o software Matlab® possui requisitos elevados do computador para um bom funcionamento e a necessidade da nomeação correta dos blocos da planta a ser simulada.

Como propostas futuras, pretende-se estender esse trabalho na aplicação de outros protocolos de redes industriais e o mapeamento de blocos da planta sem a necessidade de nomes específicos, possibilitando uma plataforma computacional base para o desenvolvimento de novas tecnologias para a indústria e o controle e automação de processos industriais.

REFERÊNCIAS

BANKS, J.; CARSON, J.; NELSON, B. L.; NICOL, D. M. **Discrete-Event System Simulation**, 4 ed, Upper Saddle River, New Jersey: Prentice Hall, 2005, 624 p.

BRANDÃO, D. **Ferramenta de Simulação para projeto, estudo e treinamento de redes fieldbus**. 2005. Tese (Doutorado em Engenharia Mecânica) – Escola de Engenharia de São Carlos, Universidade de São Paulo, São Paulo, 2005

CASSIOLATO, C. Redes Industriais, SMAR Artigos Técnicos. 2012. Disponível em: <http://www.smar.com/brasil/artigostecnicos/artigo.asp?id=48>. Acesso em: 14 jul. 2014.

GUARESE, G. B. M.; SIEBEN, F. G.; WEBBER, T.; DILLENBURG, M. R.; MARCON, C. Exploiting Modbus Protocol in Wired and Wireless Multilevel Communication Architecture. In: BRAZILIAN SYMPOSIUM ON COMPUTING SYSTEM ENGINEERING, 2., 2012, Natal. **Anais**… 2012, p. 13-18.

HUITSING, P.; CHANDIA, R.; PAPA, M.; SHENOI, S. Attack taxonomies for the Modbus protocols. **International Journal of Critical Infrastructure Protection**, v. 1, p. 37-44, 2008.

LOBÃO, E. C.; PORTO A. J. V.. Proposta Para Sistematização de Estudo de Simulação, In: ENCONTRO NACIONAL DE ENGENHARIA DE PRODUÇÃO, 17., 1997, Gramado. **Anais Eletrônicos**... Disponível em: <http://www.abepro.org.br/biblioteca/ENEGEP1997_T1101.PDF>. Acesso em: 14 jul. 2014.

MODBUS ORGANIZATION. MBServer – Collection of Programs and Libraries Listed on Technical Resources Page. 2013. Disponível em: <http://modbus.org/tech.php>. Acesso em: 20 fev. 2014.

MODBUS ORGANIZATION. MODBUS Apllication Protocol Specification V1.1b3. 2012. Disponível em: <http://modbus.org/docs/Modbus_Application_Protocol_V1_1b3.pdf>. Acesso em: 20 fev. 2014.

MODBUS ORGANIZATION. MODBUS Messaging on TCP/IP Implementation Guide V1.0b. 2006. Disponível em: <http://modbus.org/docs/Modbus_Messaging_Implementation_Guide_V1_0b.pdf>. Acesso em: 20 fev. 2014.

MODBUS ORGANIZATION. Object Messaging Specification For The MODBUS/TCP Protocol, Version 1.1. 2004. Disponível em: <http://modbus.org/docs/Object_Messaging_Protocol_ExtensionsVers1.1.doc>. Acesso em: 20 fev. 2014.

PEGDEN, C. D.; SHANNON, R. E.; SADOWSKI, R. P. **Introduction to Simulation Using SIMAN**, New York: McGraw-Hill Higher Education, 1991, 610 p.

REYNDERS, D.; MACKAY, S.; WRIGHT, E. **Practical industrial data communications**: best practice techniques (Practical professional). Butterworth-Heinemann, 2005, 432 p.

SCHNEIDER, G.; LIMA, V. F.; SCHERER, L. G.; CAMARGO, R. F.; FRANCHI, C. M. SCADA System Applied To Micro Hydropower Plant. In: Annual Conference of the IEEE Industrial Electronics Society, 39., 2013, Vienna. **Anais Eletrônicos**... Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6700330>. Acesso em: 14 jul. 2014.

TORRES, R. V. **Simulador de redes Profibus**. 2013. Dissertação (Mestrado em Sistemas Dinâmicos) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2013. Disponível em: <http://www.teses.usp.br/teses/disponiveis/18/18153/tde-12112013-153414/>. Acesso em: jul. 2014.

VIEIRA, G. E.. Uma revisão sobre a aplicação de simulação computacional em processos industriais. In: SIMPÓSIO DE ENGENHARIA DE PRODUÇÃO, 13., 2006, Bauru. Anais... Bauru: Unesp, 2006.

WIRESHARK FOUNDATION. Wireshark User's Guide. 2004. Disponível em: <http://www.wireshark.org/docs/wsug_html_chunked/>. Acesso em: fev. 2014.

APÊNDICE A – CÓDIGO SERVIDOR MODBUS

```
// mbserver.c V2.1 1/18/01
// example multi-session Modbus/TCP server supporting class 0 commands

// This program should work under UNIX (as C or C++) or Win32 (as C or C++)

// the symbol WIN32 will be defined if compiling for Windows. Assume if it is not
// set that we are compiling for some form of UNIX

// V2.0 1/14/00 added 10 second idle timeout option
// V2.1 1/18/01 timeout was not being reset on successful traffic
// V2.2 7/27/01 defaults for EXE set to sessions = 100, listen() backlog set to same, timeout on

#ifndef WIN32

// various flavors of UNIX may spread their include files in different ways
// the set below is correct for Redhat Linux 5.1 and 6.1 and Ultrix V4.3A

#include <stdio.h> // printf
#include <errno.h> // errno
#include <unistd.h> // close
#include <sys/time.h> // timeval
#include <sys/socket.h> // socket bind listen accept recv send
#include <sys/ioctl.h> // FIONBIO
#include <netinet/in.h> // sockaddr_in sockaddr INADDR_ANY
#include <time.h>

typedef int SOCKET;
const int INVALID_SOCKET=(~(int)0);

// the following fake out the Winsock-specific routines

typedef struct WSAData { int w;} WSADATA;
int WSAStartup(int v, WSADATA *pw) { return 0;}
int WSACleanup() {}
int WSAGetLastError() { return (errno);}
int closesocket(SOCKET s) { close(s); }
int ioctlsocket(SOCKET s, long cmd, unsigned long *valp) { ioctl(s, cmd, valp); }
#define WSAEWOULDBLOCK EWOULDBLOCK

#else // must be WIN32

#include <winsock2.h>
//#include <ws2tcpip.h>
#include <stdio.h> // printf
#include <time.h> // time, difftime
#include <iostream>
#include <math.h>

using namespace std;

#pragma comment(lib, "ws2_32.lib")

#endif

////////////////////////////////////////////////////////////////////
// configuration settings
////////////////////////////////////////////////////////////////////

#define numSockets 100     /* number of concurrent server sessions */
#define num4xRegs 10000  /* number of words in the 'register table' maintained by this server */
```

```
#define IDLE_TIMEOUT 1              /* set if the session idle timeout to be enforced */

////////////////////////////////////////////////////////////////////
// data structure definitions
////////////////////////////////////////////////////////////////////

// maintain a data structure per session,into which can be stored partial msgs

struct fragMsg
{
   int fragLen;              // length of request assembled so far
   unsigned char fragBuf[261]; // request so far assembled
};

////////////////////////////////////////////////////////////////////
// global data definition
////////////////////////////////////////////////////////////////////

struct fragMsg frag[numSockets];
time_t openTime[numSockets];

// make a 'state table' to read and write into

unsigned short reg4x[num4xRegs];
unsigned short regCoil[num4xRegs];

////////////////////////////////////////////////////////////////////
// utility routines
////////////////////////////////////////////////////////////////////


// extract a 16-bit word from an incoming Modbus message

unsigned short getWord(unsigned char b[], unsigned i) // transforma 2 x 8 bits em 16 bits
{
   return (((unsigned short)(b[i])) << 8) | b[i+1];
}

// write a 16-bit word to an outgoing Modbus message

void putWord(unsigned char b[], unsigned i, unsigned short w)  // transforma 16 bits em 2 x 8 bits
{
   b[i] = (unsigned char)(w >> 8);
   b[i+1] = (unsigned char)(w & 0xff);
}

////////////////////////////////////////////////////////////////////
// process legitimate Modbus/TCP requests
////////////////////////////////////////////////////////////////////

int processMsg(unsigned char b[],   // message buffer, starting with prefix
            unsigned len)       // length of incoming messsage
                              // returns length of response
{

   // if you wish to make your processing dependent upon unit identifier
   // use b[6]. Many PLC devices will ignore this field, and most clients
   // default value to zero. However gateways or specialized programs can
   // use the unit number to indicate what type of precessing is desired

   unsigned i;

   // handle the function codes 3 and 16
```

```
switch(b[7])
{
		case 1: // Read Coils
		{
			int resto = getWord(b, 10) % 8;
			int pula_um;

			if (resto != 0)
			{
				pula_um = 1;
			}
			else
			{
				pula_um = 0;
			}

			unsigned regNo = getWord(b, 8);
			unsigned regCount = getWord(b, 10);

			if (len != 12 || regCount < 1 || regCount > 2000)
			{
				// Exception 3 - bad structure to message
				b[7] |= 0x80;
				b[8] = 3;
				b[5] = 3;
				printf("Exception Code 03 - Illegal Data Value \n");
				break;
			}
			if (regNo >= num4xRegs || (regCount + regNo) > num4xRegs)
			{
				// exception 2 - bad register number or length
				b[7] |= 0x80;
				b[8] = 2;
				b[5] = 3; // length
				printf("Exception Code 02 - Illegal Data Address  \n");
				break;
			}

			// OK so prepare the 'OK response'
			b[8] = (regCount / 8) + pula_um;
			b[5] = b[8] + 3;

			int dado[8] = { 0, 0, 0, 0, 0, 0, 0, 0 };
			int contador = 0;
			int contador_bytes = (int)((regCount / 8) + pula_um);

			for (int i = 0; i < contador_bytes; i++)
			{
				double soma = 0;

				if ((i == contador_bytes - 1) && (pula_um == 1))
				{
					for (int j = 7; j >= 8-resto; j--)
					{
						if (regCoil[regNo + contador] == 1)
						{
							soma = soma + pow(2, 7 - j);
						}
						contador++;
					}
				}
				else
				{
					for (int j = 7; j >= 0; j--)
```

```
                        {
                                if (regCoil[regNo + contador] == 1)
                                {
                                        soma = soma + pow(2, 7 - j);
                                }

                                contador++;
                        }
                }
                b[9 + i] = (unsigned char)soma;
        }
}

break;

case 15:   // Write coil
{
        int resto = getWord(b, 10) % 8;
        int pula_um;

        if (resto != 0)
        {
                pula_um = 1;
        }
        else
        {
                pula_um = 0;
        }

        unsigned regNo = getWord(b, 8);
        unsigned regCount = getWord(b, 10);

        if (regCount < 1 || regCount > 2000 || len != 13 + (regCount / 8) + pula_um ||
                b[12] != (regCount / 8) + pula_um)
        {
                // exception 3 - bad structure to message
                b[7] |= 0x80;
                b[8] = 3;
                b[5] = 3; // length
                printf("Exception Code 03 - Illegal Data Value\n");
                break;
        }
        if (regNo >= num4xRegs || (regCount + regNo) > num4xRegs)
        {
                // exception 2 - bad register number or length
                b[7] |= 0x80;
                b[8] = 2;
                b[5] = 3; // length
                printf("Exception Code 02 - Illegal Data Address  \n");
                break;
        }
        // OK so process the data

        for (i = 0; i<(regCount / 8) + pula_um; i++)
        {
                unsigned temp = (int)b[13+i];

                double temp2 = temp;
                int contador = 0;

                if (i == (((regCount / 8) + pula_um) - 1) && pula_um == 1)

                {
                        for (int j = resto-1; j >= 0; j--)
```

```
                                        {
                                                if (floor(temp2 / (pow(2, j)))>0)
                                                {
                                                        regCoil[8 * i + regNo + (resto - contador - 1)] = 1;
                                                        temp2 = temp2 - pow(2, j);
                                                }
                                                else
                                                {
                                                        regCoil[8 * i + regNo + (resto - contador - 1)] = 0;
                                                }
                                                contador++;
                                        }
                                }
                                else
                                {
                                        for (int j = 7; j >= 0; j--)
                                        {
                                                if (floor(temp2 / (pow(2, j)))>0)
                                                {
                                                        regCoil[8 * i + regNo + (7 - contador)] = 1;
                                                        temp2 = temp2 - pow(2, j);
                                                }
                                                else
                                                {
                                                        regCoil[8 * i + regNo + (7 - contador)] = 0;
                                                }
                                                contador++;
                                        }
                                }


                        }
                        // and the OK response is a copy of the request to byte 11
                        b[5] = 6;
                }

                break;

// read registers

case 3:
{
   unsigned regNo = getWord(b, 8);
   unsigned regCount = getWord(b, 10);
                        if (len != 12 || regCount < 1 || regCount > 2000)
   {
      // exception 3 - bad structure to message
      b[7] |= 0x80;
      b[8] = 3;
      b[5] = 3; // length
                                printf("Exception Code 03 - Illegal Data Value\n");
      break;
   }
   if (regNo >= num4xRegs || (regCount + regNo) > num4xRegs)
   {
      // exception 2 - bad register number or length
      b[7] |= 0x80;
      b[8] = 2;
      b[5] = 3; // length
                                printf("Exception Code 02 - Illegal Data Address \n");
      break;
   }
   // OK so prepare the 'OK response'
   b[8] = 2 * regCount;
```

```
    b[5] = b[8] + 3;
    for (i=0;i<regCount;i++)
    {
        putWord(b, 9+i+i, reg4x[i + regNo]);
    }
}
break;

// Write registers

case 16:
{
    unsigned regNo = getWord(b, 8);
    unsigned regCount = getWord(b, 10);

    if (len != 13 + regCount + regCount ||
                                        b[12] != regCount + regCount || regCount < 1 || regCount > 123)
    {
        // exception 3 - bad structure to message
        b[7] |= 0x80;
        b[8] = 3;
        b[5] = 3; // length
                                        printf("Exception Code 03 - Illegal Data Value\n");
        break;
    }
    if (regNo >= num4xRegs || (regCount + regNo) > num4xRegs)
    {
        // exception 2 - bad register number or length
        b[7] |= 0x80;
        b[8] = 2;
        b[5] = 3; // length
                                        printf("Exception Code 02 - Illegal Data Address \n");
        break;
    }
    // OK so process the data
    for (i=0;i<regCount;i++)
    {
        reg4x[i + regNo] = getWord(b, 13+i+i);
    }
    // and the OK response is a copy of the request to byte 11
    b[5] = 6;
}
break;

// Write Single Registers

                case 6:   /
                {
                        unsigned regNo = getWord(b, 8);
                        unsigned valor = getWord(b, 10);

                        if (valor < 0 || valor > 65535)
                        {
                                // exception 3 - bad structure to message
                                b[7] |= 0x80;
                                b[8] = 3;
                                b[5] = 3; // length
                                printf("Exception Code 03 - Illegal Data Value  \n");
                                break;
                        }
                        if (regNo >= num4xRegs)
                        {
                                // exception 2 - bad register number or length
                                b[7] |= 0x80;
```

```
                        b[8] = 2;
                        b[5] = 3; // length
                        printf("Exception Code 02 - Illegal Data Address \n");
                        break;
                }

                // OK so process the data

                reg4x[regNo] = valor;

                // and the OK response is a copy of the request to byte 11
                b[5] = 6;
        }
        break;

// Write single coil

        case 5:
        {
                unsigned regNo = getWord(b, 8);
                unsigned valor = getWord(b, 10);

                if (valor != 65280 && valor != 0)
                {
                        // exception 3 - bad structure to message
                        b[7] |= 0x80;
                        b[8] = 3;
                        b[5] = 3; // length
                        printf("Exception Code 03 - Illegal Data Value  \n");
                        break;
                }

                if (regNo >= num4xRegs)
                {
                        // exception 2 - bad register number or length
                        b[7] |= 0x80;
                        b[8] = 2;
                        b[5] = 3; // length
                        printf("Exception Code 02 - Illegal Data Address   \n");
                        break;
                }

                if (valor == 65280)
                {
                        regCoil[regNo] = 1;
                }
                if (valor == 0)
                {
                        regCoil[regNo] = 0;
                }

                // and the OK response is a copy of the request to byte 11
                b[5] = 5;
        }
        break;

default:

// generate exception 1 - unknown function code

b[7] |= 0x80;
b[8] = 1;
b[5] = 3; // length
```

```
                        printf("Exception Code 01 - Illegal Function   \n");
      break;
   }

   // return the total size of the MB/TCP response
   // notice that bytes 0-4 and 6 will be identical to those of the request

   return 6+b[5];
}

//////////////////////////////////////////////////////////////////
// main entry point for the program
//////////////////////////////////////////////////////////////////

int main(int argc, char **argv) // arguments ignored for now
{
   int i;
   SOCKET csa[numSockets];
   SOCKET s;
   struct sockaddr_in server;
   static WSADATA wd;

   unsigned long nbiotrue = 1;

   printf("Modbus/TCP Server\n"
                              "Sessoes = %u, Registros = %u, Coils = %u, Timeout = %s\n",
                              numSockets, num4xRegs, num4xRegs, IDLE_TIMEOUT ? "True" : "False");

   // initialize WinSock

   if (WSAStartup(0x0101, &wd))
   {
      printf("cannot initialize WinSock\n");
      return 1;
   }

   // set up an array of socket descriptors, initially set to INVALID_SOCKET (not in use)
   // and initialize the fragment buffer

   for (i=0;i<numSockets;i++)
   {
      csa[i] = INVALID_SOCKET;
      frag[i].fragLen = 0;
   }

   // set up listen socket

   s = socket(PF_INET, SOCK_STREAM, 0);
   server.sin_family = AF_INET;
   server.sin_port = htons(502); // ASA standard port

   //server.sin_addr.s_addr = INADDR_ANY; //localhost
  server.sin_addr.s_addr = inet_addr("10.240.1.157");
 // server.sin_addr.s_addr = inet_addr("192.168.1.10");

   i = bind(s, (struct sockaddr *)&server, sizeof(struct sockaddr_in));

   if (i<0)
   {
      printf("bind - error %d\n",WSAGetLastError());
      closesocket(s);
      WSACleanup();
      return 1;
   }
```

```
// set socket non-blocking in case a client has second thoughts about
// establishing a connection later on. In this case the accept() will return
// with an error rather than with a socket.

if (ioctlsocket (s, FIONBIO, &nbiotrue))
{
   printf("ioctlsocket - error %d\n", WSAGetLastError());
}

i = listen(s, numSockets);

if (i<0)
{
   printf("listen - error %d\n",WSAGetLastError());
   closesocket(s);
   WSACleanup();
   return 1;
}

// at this point, be prepared to handle incoming requests on socket s
// by doing accept() and processing them independently

for (;;)
{

   SOCKET cs;
   fd_set fds;
   int si;
   struct timeval tv;

   // set up a 1 second timeout to allow for (future) background activity

   tv.tv_sec = 1;
   tv.tv_usec = 0;

   // be prepared for incoming messages on the listen port and any active ports

   FD_ZERO(&fds);

   // wait for incoming connection

  FD_SET(s, &fds);

   // and add any Modbus sessions

   for (i=0;i<numSockets;i++)
   {
      if (csa[i] != INVALID_SOCKET)
      {
         cs = csa[i];
         FD_SET(cs, &fds);
      }
   }

   i = select(32, &fds, NULL, NULL, &tv); // read

   // note that the fd_set will have been updated to select only those sockets
   // requiring attention right now

   if (i<0)
   {
      printf("select - error %d\n",WSAGetLastError());
      closesocket(s);
```

```
            WSACleanup();
            //return 1;
        }

        // any listen work?


if (FD_ISSET(s, &fds))
        {
            printf("Nova conexao recebida\n");
            cs = accept(s,NULL,0);
            if (cs<0)
            {
                int e = WSAGetLastError();
                if (e != WSAEWOULDBLOCK)
                {
                    printf("accept - error %d\n",e);
                    closesocket(s);
                    WSACleanup();
                    return 1;
                } else
                {
                    // the connection is no longer pending so it must have been
                    // abandoned by the client
                }
            }
else
            {

                // add the newly opened socket to the list. If nowhere to put it, throw it
                // away

                for (i=0;i<numSockets;i++)
                {
                    if (csa[i] == INVALID_SOCKET)
                    {
                        csa[i] = cs;
                        frag[i].fragLen = 0;
                                                                openTime[i] = time(NULL);
                        break;
                    }
                }

                if (i >= numSockets)
                {
                    // nowhere to put it
                    printf("Conexão abandonada - maximo de conexoes simultaneas alcancado\n");
                    closesocket(cs);
                }

                // set socket non-blocking just in case anything happens which might make the
                // recv() operation block later on. This should not be necessary.

                if (ioctlsocket (cs, FIONBIO, &nbiotrue))
                {
                    printf("ioctlsocket - error %d\n", WSAGetLastError());
                }

            }
        }

        // any socket level work?

        for (si=0;si<numSockets;si++)
```

```
    {
      struct fragMsg *thisFrag;
      unsigned char *ibuf;

      cs = csa[si];

      if (cs == INVALID_SOCKET)
      {
        // nothing to do right now on this one
        continue;
      }

#if IDLE_TIMEOUT
                        if (10 <= difftime(time(NULL), openTime[si]))
                        {

                                // the 10 second idle timer has expired. Close the incoming session
                                printf("Sesssao %d fechada devido a timeout\n");
        // remove session from active list
        csa[si] = INVALID_SOCKET;
        // close connection
        closesocket(cs);
        continue; // abandon any further processing on this session
                        }
#endif // IDLE_TIMEOUT

      if (!FD_ISSET(cs, &fds))
      {
        // nothing to do right now on this one
        continue;
      }

#if IDLE_TIMEOUT
                        // ags 1/18/01
                        // update timeout so that shhutdown only occurs after 10 sec of IDLE
                        // and not arbitrarily 10 sec from session open!
                        openTime[si] = time(NULL);
#endif // IDLE_TIMEOUT

      cs = csa[si];

      // account for any fragment outstanding from a previous cycle
      // (this stupidity would not be necessary if all clients would send
      // their messages in one piece)

      thisFrag = &frag[si];
      ibuf = thisFrag->fragBuf;

      if (thisFrag->fragLen < 6)
      {
        // don't know the length yet, just read the prefix
        i = recv(cs, (char *)&thisFrag->fragBuf[thisFrag->fragLen], 6 - thisFrag->fragLen, 0);
        if (i <= 0)
        {
          // this session has been closed or damaged at the remote end
          // this may be a normal condition

          // remove session from active list
          csa[si] = INVALID_SOCKET;
          // close connection
          closesocket(cs);
          continue;
        }
```

```
    thisFrag->fragLen += i;

    // unfortunately, we are not sure if there are any more bytes
    // so continue this processing on the next cycle
    continue;
}

if (ibuf[2] != 0 || ibuf[3] != 0 || ibuf[4] != 0 || ibuf[5] < 2)
{
    // this is not legitimate Modbus/TCP
    // possibly your client is very confused
    // close down the connection

    // remove session from active list
    csa[si] = INVALID_SOCKET;
    printf("bad MB/TCP protocol - closing\n");
    // close connection
    closesocket(cs);
    continue;
}

// the real length is in ibuf[5]

if (thisFrag->fragLen < 6+ibuf[5])
{
    i = recv(cs, (char *)&thisFrag->fragBuf[thisFrag->fragLen], 6 + ibuf[5] - thisFrag->fragLen, 0);
    if (i <= 0)
    {
        // this session has been closed or damaged at the remote end

        printf("Exception Code 03 - Illegal Data Value \n");

        // remove session from active list

        csa[si] = INVALID_SOCKET;

        // close connection

        closesocket(cs);

        continue;
    }
    thisFrag->fragLen += i;
}

if (thisFrag->fragLen < 6+ibuf[5])
{
    // still waiting for completion of the message
    continue;
}

// if we get here, the message is complete and it looks like MB/TCP

// process the incoming request, generating a response
// note that there is no requirement to keep track of which connection
// the request was received on - you must only use the same one for
// sending the response

i = processMsg(ibuf, thisFrag->fragLen);

i = send(cs, (char *)ibuf, i, 0);

thisFrag->fragLen = 0;
```

```
        }
        // note that this outer loop will run forever unless cancelled
        // and that if it is cancelled, you must close outstanding sockets
        // and call WSACleanup()
    }
}
```

APÊNDICE B – CÓDIGO CLIENTE MODBUS

```
function varargout = Cliente(varargin)

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Cliente_OpeningFcn, ...
                   'gui_OutputFcn',  @Cliente_OutputFcn, ...
                   'gui_LayoutFcn',  [], ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
  gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
   [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
   gui_mainfcn(gui_State, varargin{:});
end

function Cliente_OpeningFcn(hObject, eventdata, handles, varargin)

handles.output = hObject;
guidata(hObject, handles);

function varargout = Cliente_OutputFcn(hObject, eventdata, handles)

varargout{1} = handles.output;

function executar_funcao_Callback(hObject, eventdata, handles)

if get(hObject,'Value')

   % Configuração objeto TCP

      ip = get(handles.entrada_ip,'String');
      obj = tcpip(ip, 502, 'NetworkRole', 'client');
      set(obj, 'InputBufferSize', 512);
      set(obj, 'Timeout', 15);
      obj.ByteOrder='bigEndian';
      global trans_identifier;
      trans_identifier = 0 ;

   % Configuração da planta e simulação

      planta = get(handles.entrada_nomeproc,'String');
      open_system(planta);
      set_param(planta,'SimulationMode','external');
      set_param(planta,'SimulationCommand','connect')
      set_param(planta,'SimulationCommand','Start');

      atraso = str2double(get(handles.entrada_atraso,'String'));
      endereco_inicial = str2double(get(handles.entrada_inicio,'String')); % endreço-1 no modscan
      entrada_max = str2double(get(handles.entrada_valormax,'String'));
```

```
todos_blocos = find_system(planta);
qtd_blocos_sistema = size(todos_blocos,1);

% Cria as variáveis do sistema

numero_saidas_analog = 0;
numero_saidas_digitais = 0;
numero_setpoints_analog = 0;
numero_setpoints_digitais = 0;

temp1 = strfind(todos_blocos,strcat(planta,'/saida_analog'));
temp2 = strfind(todos_blocos,strcat(planta,'/saida_digital'));
temp3 = strfind(todos_blocos,strcat(planta,'/entrada_analog'));
temp4 = strfind(todos_blocos,strcat(planta,'/entrada_digital'));

% Determina o valor das variáveis do sistema

for i=1:1:qtd_blocos_sistema
   if temp1{i}==1
      numero_saidas_analog = numero_saidas_analog + 1;
   end
   if temp2{i}==1
      numero_saidas_digitais = numero_saidas_digitais + 1;
   end
   if temp3{i}==1
      numero_setpoints_analog = numero_setpoints_analog + 1;
   end
   if temp4{i}==1
      numero_setpoints_digitais = numero_setpoints_digitais + 1;
   end
end

set(handles.parametro_digital,'String',num2str(numero_saidas_digitais));
set(handles.parametro_analog,'String',num2str(numero_saidas_analog));
set(handles.parametro_setpoint_analog,'String',num2str(numero_setpoints_analog));
set(handles.parametro_setpoint_digital,'String',num2str(numero_setpoints_digitais));

blocos_analog = cell(numero_saidas_analog);
for i=1:1:numero_saidas_analog
   blocos_analog{i} = strcat(planta,'/saida_analog',num2str(i));
end

blocos_digital = cell(numero_saidas_digitais);
for i=1:1:numero_saidas_digitais
   blocos_digital{i} = strcat(planta,'/saida_digital',num2str(i));
end

blocos_setpoint_analog = cell(numero_setpoints_analog);
for i=1:1:numero_setpoints_analog
   blocos_setpoint_analog{i} = strcat(planta,'/entrada_analog',num2str(i));
end

blocos_setpoint_digital = cell(numero_setpoints_digitais);
for i=1:1:numero_setpoints_digitais
   blocos_setpoint_digital{i} = strcat(planta,'/entrada_digital',num2str(i));
end

% Escrever no servidor os valores dos setpoints iniciais (Escreve os
% valores do matlab no servidor)
```

```
    if numero_setpoints_digitais~=0
        WriteMultipleCoils(obj,numero_setpoints_digitais,blocos_setpoint_digital,endereco_inicial+1000,0);
    end
    if numero_setpoints_analog~=0

WriteMultipleRegisters(obj,numero_setpoints_analog,blocos_setpoint_analog,endereco_inicial+1000,0,entrada_
max);
    end

    % Rodar as funções enquanto o botão estiver ativo

        while get(hObject,'Value')
            % Leio os setpoints no servidor e atualizo o matlab com estes
            % Leio as saídas do matlab e gravo as saídas no servidor
            if numero_setpoints_digitais~=0
                ReadMultipleCoils(obj,endereco_inicial+1000,numero_setpoints_digitais,blocos_setpoint_digital);
            end
            if numero_saidas_digitais~=0
                WriteMultipleCoils(obj,numero_saidas_digitais,blocos_digital,endereco_inicial,1)
            end
            if numero_setpoints_analog~=0

ReadMultipleRegisters(obj,endereco_inicial+1000,numero_setpoints_analog,blocos_setpoint_analog,entrada_m
ax);
            end
            if numero_saidas_analog~=0
                WriteMultipleRegisters(obj,numero_saidas_analog,blocos_analog,endereco_inicial,1,entrada_max)
            end

            % Pausa
            pause(atraso);
        end
        set_param(planta,'SimulationCommand','Stop');
end

% Funções Modbus

function ReadMultipleCoils(obj,endereco_inicial, b_count,blocos_setpoint_digital)
        % Read multiple Coils - Funcao 1 (Ler set points digitais)
        % Constantes
        global trans_identifier;
        funcao = 1;
        contadorbytes = 6;

        % Pacote Modbus
        trans_identifier = uint16(trans_identifier); % 2 bytes
        protocol_identifier = uint16(0); % 2 bytes
        length = uint16(contadorbytes); % 2 bytes
        unit_identifier = uint16(1); % 1 byte
        function_code = uint16(funcao); %1 byte
        unit_identifier = bitshift(unit_identifier,8); % desloca unit_identifier 8 posicoes para esquerda
        unit_and_function = bitor(unit_identifier,function_code); % soma unit_identifier e function_code para
formar 2 bytes
        reference_number = uint16(endereco_inicial); % 2 bytes
        bit_count = uint16(b_count); % 2 bytes
        message = [trans_identifier;protocol_identifier;length;unit_and_function;reference_number;bit_count];

        % Envia mensagem
        fopen(obj);
```

```
    fwrite(obj,message,'uint16'); % mensagem modbus deve ser int16

    % Lê a resposta
    tam_leitura = floor(b_count/8);
    resto = mod(b_count,8);
    if resto~=0
       byte_add = 1;
    else
       byte_add = 0;
    end
    contador = 1;
    message = fread(obj,(9+tam_leitura+byte_add));

    for i=1:1:tam_leitura+byte_add
       bin = fliplr(dec2bin((message(9+i)),8));
       if i==tam_leitura+byte_add &&  resto~=0 % Existe resto e está no byte do resto
          for i=1:1:resto
             set_param(blocos_setpoint_digital{contador}, 'Gain', bin(i));
             contador = contador + 1;
          end
       else
          for i=1:1:8
             set_param(blocos_setpoint_digital{contador}, 'Gain', bin(i));
             contador = contador + 1;
          end
       end
    end

    % Fecha a conexao
    fclose(obj);
    trans_identifier = trans_identifier + 1;
    if trans_identifier>=65536
       trans_identifier = 0;
    end

function ReadMultipleRegisters(obj,endereco_inicial, b_count,blocos_setpoint_analog,entrada_max)
   % Read multiple Registers - Funcao 3  (Ler set points analógicos)
   % Constantes
   global trans_identifier;
   funcao = 3;
   contadorbytes = 6; %fixo

   % Pacote Modbus
   trans_identifier = uint16(trans_identifier); % 2 bytes
   protocol_identifier = uint16(0); % 2 bytes
   length = uint16(contadorbytes); % 2 bytes
   unit_identifier = uint16(1); % 1 byte
   function_code = uint16(funcao); %1 byte
   unit_identifier = bitshift(unit_identifier,8); % desloca unit_identifier 8 posicoes para esquerda
   unit_and_function = bitor(unit_identifier,function_code); % soma unit_identifier e function_code para formar
2 bytes
   reference_number = uint16(endereco_inicial); % 2 bytes
   bit_count = uint16(b_count); % 2 bytes
   message = [trans_identifier;protocol_identifier;length;unit_and_function;reference_number;bit_count];

   % Envia mensagem
   fopen(obj);
   fwrite(obj,message,'uint16'); % mensagem modbus deve ser int16
```

```
    % Lê a resposta

    contador = 1;
    qtd_bytes = (b_count*2);
    message = fread(obj,(9+qtd_bytes));

    for i=1:2:qtd_bytes
        bin1 = dec2bin((message(9+i)),8);
        bin2 = dec2bin((message(10+i)),8);
        temp = bin2dec(strcat(bin1,bin2));
        temp = (temp-32767)*(entrada_max/32767);
        set_param(blocos_setpoint_analog{contador}, 'Gain', num2str(temp));
        contador = contador + 1;
    end

    fclose(obj);
    trans_identifier = trans_identifier + 1;
    if trans_identifier>=65536
        trans_identifier = 0;
    end

function WriteMultipleCoils(obj,numero_saidas_digitais,blocos_digital,endereco_inicial,in_or_out)
    % Write Multiple Coils - Funcao 15
    if in_or_out == 1 % in_or_out = 1 significa que está analisando a porta de saída do bloco
        for i=1:1:numero_saidas_digitais
        parametro_saida_digital(i) = get_param(blocos_digital{i}, 'RuntimeObject'); %
        saida_digital(i) = uint16(parametro_saida_digital(i).OutputPort(1).Data);
        end
    end
    if in_or_out == 0 % in_or_out = 0 significa que está analisando o ganho do bloco
        for i=1:1:numero_saidas_digitais
        parametro_saida_digital(i) = str2double(get_param(blocos_digital{i}, 'Gain'));
        saida_digital(i) = uint16(parametro_saida_digital(i));
        end
    end

    % Variaveis de entrada
    valor = fliplr(saida_digital); %valores dos coils, do maior equipamento pro menor

    % Outras variaveis
    contadorbits = size(valor,2); % até 56
    funcao = 15;
    N = ceil(contadorbits/8);
    contadorbytes = 7 + N;
    global trans_identifier;

    resto = mod(contadorbits,8);

    % O resto
    dado = [0,0,0,0,0,0,0,0];
    contador = 1;

    for i=9-resto:1:8
        dado(i) = valor(contador);
        contador = contador +1;
    end

    dado = num2str(dado);
```

```
    dado = bin2dec(dado);
    data8(N) = uint16(dado);

    if resto~=0
        pula_um = 1;
    else
        pula_um = 0;
    end

    % Os outros
    for i=1:1:N-pula_um
        dado = [0,0,0,0,0,0,0,0];
        for j=1:1:8
        dado(j) = valor(resto+j+(8*i-8));
        end
        dado = num2str(dado);
        dado = bin2dec(dado);
        data8(N-i+1-pula_um) = uint16(dado);
    end

    % Pacote Modbus
    trans_identifier = uint16(trans_identifier); % 2 bytes
    protocol_identifier = uint16(0); % 2 bytes
    length = uint16(contadorbytes); % 2 bytes
    unit_identifier = uint16(1); % 1 byte
    function_code = uint16(funcao); %1 byte
    unit_identifier = bitshift(unit_identifier,8); % desloca unit_identifier 8 posicoes para esquerda
    unit_and_function = bitor(unit_identifier,function_code); % soma unit_identifier e function_code para formar
2 bytes
    reference_number = uint16(endereco_inicial); % 2 bytes
    bit_count = uint16(contadorbits); % 2 bytes
    byte_count = uint16(N); % 1 byte
    byte_count = bitshift(byte_count,8); % desloca unit_identifier 8 posicoes para esquerda
    byte_and_data = bitor(byte_count,data8(1)); % soma unit_identifier e function_code para formar 2 bytes
    message =
[trans_identifier;protocol_identifier;length;unit_and_function;reference_number;bit_count;byte_and_data];

    qtd_data16 = ceil((N-1)/2);

    for i=1:1:qtd_data16
        if i==N-1
            message(7+i,1) =  bitshift(data8(i+1),8,'uint16');
        else
            message(7+i,1) = bitshift(data8(i+1),8,'uint16')+data8(i+2);
        end
    end

    % Envia mensagem
    fopen(obj);
    fwrite(obj,message,'uint16'); % mensagem modbus deve ser int16
    fclose(obj);
    trans_identifier = trans_identifier + 1;
    if trans_identifier>=65536
        trans_identifier = 0;
    end

function
WriteMultipleRegisters(obj,numero_saidas_analog,blocos_analog,endereco_inicial,in_or_out,entrada_max)
    %Write Multiple Registers - Funcao 16
```

```
    if in_or_out == 1 % in_or_out = 1 significa que está analisando a porta de saída do bloco
        for i=1:1:numero_saidas_analog
            parametro_saida_analog(i) = get_param( blocos_analog{i}, 'RuntimeObject');
            temp = (((parametro_saida_analog(i).OutputPort(1).Data))*(32767/entrada_max))+32767;
            saida_analog(i) = uint16(temp);
        end
    end

    if in_or_out == 0 % in_or_out = 0 significa que está analisando o ganho do bloco
        for i=1:1:numero_saidas_analog
            parametro_saida_analog(i) = str2double(get_param(blocos_analog{i}, 'Gain'));
            temp = ((parametro_saida_analog(i))*(32767/entrada_max))+32767;
            saida_analog(i) = uint16(temp);
        end
    end

    % Variaveis de entrada
    valor = saida_analog; %valores dos coils, do maior equipamento pro menor

    % Outras variáveis
    N = size(valor,2); % quantidade de registros para escrever
    dado = [0;0];
    funcao = 16;
    contadorbytes = 7 + (N*2);
    global trans_identifier;

    dado(1) = bitshift(valor(1),-8);
    for i=1:1:N
        if i==N
            dado(i+1) = bitshift(valor(i),8,'uint16');
        else
            dado(i+1) = bitshift(valor(i),8,'uint16') + bitshift(valor(i+1),-8,'uint16'); % ultimos 8 de i e primeiros 8 de
i+1
        end
    end

    % Pacote Modbus
    trans_identifier = uint16(trans_identifier); % 2 bytes
    protocol_identifier = uint16(0); % 2 bytes
    length = uint16(contadorbytes); % 2 bytes
    unit_identifier = uint16(1); % 1 byte
    function_code = uint16(funcao); %1 byte
    unit_identifier = bitshift(unit_identifier,8); % desloca unit_identifier 8 posicoes para esquerda
    unit_and_function = bitor(unit_identifier,function_code); % soma unit_identifier e function_code para formar
2 bytes
    reference_number = uint16(endereco_inicial); % 2 bytes
    bit_count = uint16(N); % 2 bytes
    byte_count = uint16(2*N); % 2*N byte
    byte_count = bitshift(byte_count,8); % desloca unit_identifier 8 posicoes para esquerda
    byte_and_data = bitor(byte_count,dado(1)); % soma unit_identifier e function_code para formar 2 bytes]
    message =
[trans_identifier;protocol_identifier;length;unit_and_function;reference_number;bit_count;byte_and_data];

    for i=1:1:N
        message(7+i,1) = dado(i+1);
    end

    % Envia mensagem
    fopen(obj);
```

```
    fwrite(obj,message,'uint16'); % mensagem modbus deve ser int16
    fclose(obj);
    trans_identifier = trans_identifier + 1;
    if trans_identifier>=65536
        trans_identifier = 0;
    end

% Outras funções

function parametro_digital_Callback(hObject, eventdata, handles)

function parametro_digital_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function parametro_analog_Callback(hObject, eventdata, handles)

function parametro_analog_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function parametro_setpoint_digital_Callback(hObject, eventdata, handles)

function parametro_setpoint_digital_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function parametro_setpoint_analog_Callback(hObject, eventdata, handles)

function parametro_setpoint_analog_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function entrada_atraso_Callback(hObject, eventdata, handles)

function entrada_atraso_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function entrada_inicio_Callback(hObject, eventdata, handles)

function entrada_inicio_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function entrada_valormax_Callback(hObject, eventdata, handles)

function entrada_valormax_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function entrada_nomeproc_Callback(hObject, eventdata, handles)

function entrada_nomeproc_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function entrada_ip_Callback(hObject, eventdata, handles)
% hObject    handle to entrada_ip (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    structure with handles and user data (see GUIDATA)

% Hints: get(hObject,'String') returns contents of entrada_ip as text
%        str2double(get(hObject,'String')) returns contents of entrada_ip as a double


% --- Executes during object creation, after setting all properties.
function entrada_ip_CreateFcn(hObject, eventdata, handles)
% hObject    handle to entrada_ip (see GCBO)
% eventdata  reserved - to be defined in a future version of MATLAB
% handles    empty - handles not created until after all CreateFcns called

% Hint: edit controls usually have a white background on Windows.
%       See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'), get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```