UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
ENG. DE CONTROLE E AUTOMAÇÃO

NERY DA SILVA MARQUES NETO - 00205989

DESENVOLVIMENTO E TESTE DE PLATAFORMA DE COMUNICAÇÃO DE BAIXO CUSTO PARA PLANTA LABORATORIAL EMPREGANDO OPC-UA

Porto Alegre
2020

NERY DA SILVA MARQUES NETO - 00205989

DESENVOLVIMENTO E TESTE DE PLATAFORMA DE COMUNICAÇÃO DE BAIXO CUSTO PARA PLANTA LABORATORIAL EMPREGANDO OPC-UA

Trabalho de Conclusão de Curso (TCC-CCA) apresentado à COMGRAD-CCA da Universidade Federal do Rio Grande do Sul como parte dos requisitos para a obtenção do título de *Bacharel em Eng. de Controle e Automação* .

ORIENTADOR:
Prof. Dr. Pedro Rafael Bolognese Fernandes

COORIENTADOR:
Prof. Dr. Marcelo Götz

Porto Alegre
2020

NERY DA SILVA MARQUES NETO - 00205989

DESENVOLVIMENTO E TESTE DE PLATAFORMA DE COMUNICAÇÃO DE BAIXO CUSTO PARA PLANTA LABORATORIAL EMPREGANDO OPC-UA

Este Trabalho de Conclusão de Curso foi julgado adequado para a obtenção dos créditos da Disciplina de TCC do curso *Eng. de Controle e Automação* e aprovado em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____
Prof. Dr. Pedro Rafael Bolognese Fernandes, UFRGS
Doutor em Engenharia pela Universidade Dortmund, Alemanha

Banca Examinadora:

Prof. Dr. Pedro Rafael Bolognese Fernandes, UFRGS
Doutor em Engenharia pela Universidade Dortmund, Alemanha

Prof. Dr. Diego Eckhard, UFRGS
Doutor em Engenharia pela Univerdade Federal do Rio Grande do Sul

Prof. Dr. Marcelo Farenzena, UFRGS
Doutor em Engenharia pela Univerdade Federal do Rio Grande do Sul

_____
Marcelo Götz
Coordenador de Curso
Eng. de Controle e Automação

Porto Alegre, novembro de 2020.

RESUMO

Ao longo das últimas décadas, os sistemas supervisórios e de controle evoluíram junto com a eletrônica e sistemas de telecomunicação para atender as necessidades da indústria, e o computador teve um papel fundamental nesse processo. Conforme os sistemas tornaram-se mais complexos, houve a necessidade de criar um padrão de comunicação comum para os dispositivos de aquisição de dados e controle, surgiu assim o OPC. Esse trabalho trata da implementação de um sistema supervisório e de controle (SCADA) de baixo custo para a planta laboratorial de nível do DEQUI. Para distribuição de dados na rede, configura-se um servidor OPC UA utilizando uma placa Raspberry Pi 3, que também se comunica com a planta laboratorial por porta serial. O desenvolvimento do software é feito em Python com bibliotecas de código aberto. Por fim, é feito um ensaio de controle em malha fechada para validar o funcionamento do sistema.

Palavras-chave: SCADA, DCON, OPC, OPC UA, Raspberry Pi, Python.

ABSTRACT

During the last decades, supervisory and control systems have evolved along electronics and telecommunication systems in order to meet requirements of the industry, and the computer had a fundamental role in this process. As systems became more complex, the need for creating a common communication standard por control and data acquisition devices emerged. This academic work is about the implementation of a low costed supervisory control and data acquisition system (SCADA) for the laboratory level plant at DEQUI. For network data distribution, it is used an OPC UA server with a Raspberry Pi operating it, which also communicates with the laboratorial plant through a serial port. The development of the software is done in Python using open source libraries. Lastly, a closed-loop control test is done in order to validate the system.

Keywords: SCADA, DCON, OPC, OPC UA, Raspberry Pi, Python.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

1 INTRODUÇÃO

No início do século XX, o desenvolvimento da engenharia de aeronaves, assim como o estudo do clima, exigiram que informações fossem coletadas por equipamentos de difícil ou impossível acesso. Por exemplo, nos primeiros foguetes lançados não havia espaço suficiente para um piloto e suas viagens na maioria das vezes terminavam de maneira abrupta muito antes do esperado, sendo necessário coletar as informações de voo enquanto estavam disponíveis. Quando a tecnologia foi empregada na previsão do tempo, os cientistas perceberam que grandes volumes de dados seriam necessários para fazer previsões do tempo precisas. Mas pouca informação estava disponível onde as pessoas estavam localizadas, surgiu então a questão de como coletar e armazenar toda essa informação necessária (BOYER, 2009).

A resposta surgiu de um método de comunicação que estava sendo utilizado na indústria ferroviária para questões de segurança. Por algum tempo, sistemas de trilhos utilizaram comunicação por fio para monitorar a posição dos vagões e o estado das chaves que controlavam os caminhos pelos quais os trens trafegavam. Esse sistema de comunicação chamado de *telemetria*, permitia que uma central monitorasse os eventos que estavam acontecendo em localizações remotas (BOYER, 2009). Telemetria é definida como a transmissão automática e medição de dados de uma fonte remota por fio, rádio ou outros meios (ROBLES; KIM, 2010).

A telemetria a rádio e por fio evoluíram ao longo das décadas 40 e 50, melhorando sua confiabilidade e aumentando a densidade de dados que poderiam ser transmitidos. Foi então estabelecido um novo conceito de comunicação de duas vias, que permitiu não só monitorar as chaves de transição dos trilhos de trem, por exemplo, mas como também atuar nelas a distância. No começo da década de 60, monitoramento remoto e controle supervisório de diversas indústrias de processo já era uma tecnologia em desenvolvimento (BOYER, 2009).

Ao mesmo tempo, outra tecnologia eletrônica estava sendo desenvolvida. No começo dos anos 60, os computadores digitais também passaram a ser utilizados no monitoramento remoto e controle supervisório de sistemas. E foi então, no começo dos anos 70, que o termo *SCADA* surgiu e a palavra *telemetria* começou a cair em desuso para descrever sistemas de duas vias de comunicação (BOYER, 2009).

Recentemente, a tendência à padronização de elementos e de protocolos de comunicação vem reduzindo os custos associados à implantação dos sistemas de controle. E ainda, graças aos avanços da eletrônica, os dispositivos de controle têm maior velocidade e capacidade de processamento, mais funcionalidades, tamanho reduzido, maior sensibilidade e precisão. Com isso tem-se conhecimento em tempo real do estado de operação de plantas remotamente, o que é muito vantajoso para organizações com operações separadas geograficamente (GUTIERREZ; PAN, 2008).

1.1 Motivação

Este trabalho trata do controle remoto de uma planta de seis tanques localizada no Departamento de Engenharia Química da Universidade Federal do Rio Grande do Sul (DEQUI), cujo sistema em operação depende de um computador instalado próxima da planta, já que se comunica com a mesma por cabo serial. Esse computador é responsável por fazer a leitura e a escrita de dados nos módulos de comunicação e transmitir esses dados aos computadores de sala de aula remota onde são ministradas aulas práticas de controle.

Esses módulos possuem um protocolo proprietário de comunicação chamado DCON e os drivers de comunicação para esse protocolo são disponibilizados apenas para o sistema operacional Windows. A distribuição dos dados na rede, atualmente, é feita utilizando o software Elipse E3 através do protocolo OPC-DA.

Deseja-se tornar o sistema mais flexível, com drivers de comunicação de código aberto e também com distribuição de dados na rede independente de sistema operacional. Isso torna o sistema mais propenso a futuras melhorias e em melhores condições para ser utilizado em outros trabalhos ou pesquisas. Também é interessante reduzir a dependência de um único computador, caso haja falha no servidor, o supervisório é muito mais fácil de ser reinstalado, utilizando-se hardware de baixo custo e softwares de licença livre.

1.2 Objetivo

O objetivo deste trabalho é o projeto, implementação e teste de um sistema supervisório e de controle para a planta de seis tanques situada no DEQUI e comparar os tempos de resposta em relação ao sistema em uso. O sistema deve ser de código aberto, flexível e de baixo custo, compatível com diferentes sistemas operacionais e capaz de operar todos os tanques da planta simultaneamente.

Para tanto, utiliza-se a estrutura já disponível de atuadores, instrumentação, dispositivos de comunicação e aquisição de dados. O sistema projetado é implementado em uma placa *Raspberry Pi Model 3 B* como o servidor e computador do sistema, e a distribuição dos dados na rede é feita utilizando o protocolo OPC UA.

2 REVISÃO BIBLIOGRÁFICA

2.1 SCADA

O acrônimo SCADA significa controle supervisório e aquisição de dados (do inglês *Supervisory Control And Data Acquisition*). Como o nome sugere, o conceito de SCADA não abrange um controle completo, é focado apenas no nível supervisório. Portanto se trata de um software que faz a comunicação entre um computador e uma rede de automação, cuja interface se dá normalmente por CLPs(controladores lógicos programáveis) ou outros módulos de hardware comerciais (MOHAMED; ABBAS, 2011)(DANEELS; SALTER, 1999) (TECHINICAL INFORMATION BULLETTIN, 2004)

Os sistemas SCADA são componentes vitais da infraestrutura de muitas nações. Eles controlam linhas de produção, sistemas de transporte, refinarias, plantas químicas e uma variedade de processos de manufatura. A tecnologia da automação produz grande volume de dados que representam as variáveis físicas dos processos como, por exemplo, temperatura, corrente, pressão ou outros dados de produção como número de unidades. Os dados vêm de diversas fontes como controladores, medidores de nível, pressostatos e termômetros. Esse informação é toda processada por vários tipos de softwares e pode ser exibida em tela ou dispositivos HMI (interface homem-máquina) (MOHAMED; ABBAS, 2011).

Um sistema SCADA consiste em:

- Dispositivos de interface de dados, normalmente UTRs(unidades de terminal remotas) ou CLPs, que fazem a interface com o campo e dispositivos de sensoriamento e também com os atuadores e dispositivos de controle;

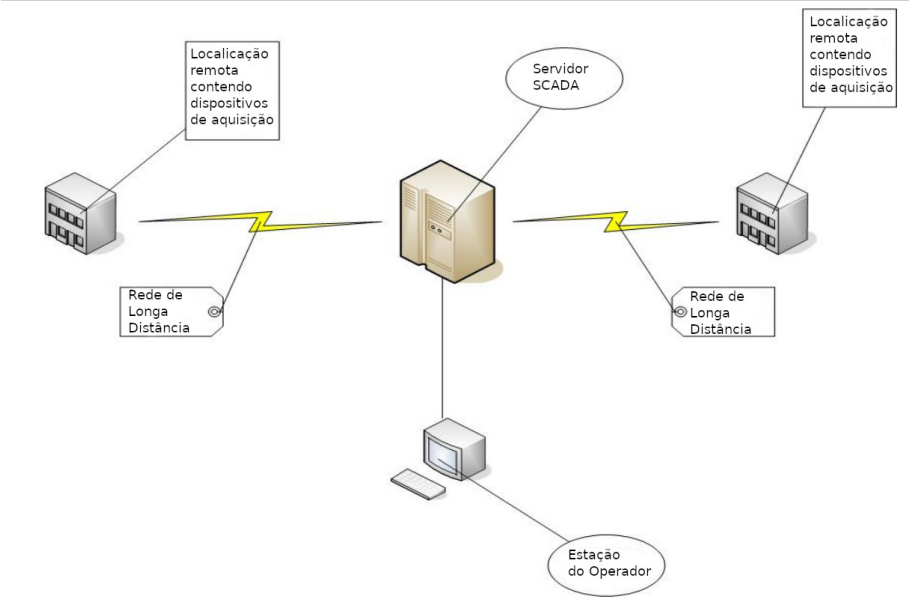
- Um sistema de comunicação que faz a transferência de dados entre os dispositivos de campo, as unidades de controle e os computadores na sede do sistema SCADA. Esse sistema de comunicação pode ser rádio, telefone, cabos, satélite, entre outros, como também qualquer combinação destes;

- Um servidor central ou servidores, também chamado de Unidade Terminal Mestre (MTU);

- Uma coleção de softwares padronizados ou customizados para prover à central e ao operador terminal uma interface a fim de monitorar e controlar remotamente os dispositivos de campo (TECHINICAL INFORMATION BULLETTIN, 2004).

Uma típica configuração de um sistema SCADA pode ser vista na Figura 1.

Figura 1: *Configuração típica de um sistema SCADA*



Fonte: Techinical Information Bullettin (2004)

2.2 DCON

O protocolo DCON é utilizado pelos dispositivos de comunicação presentes na planta. É um protocolo serial e utiliza o RS-232 ou RS-485 como meio físico. Os caracteres são codificados em ASCII, sendo semelhante ao modbus, porém utiliza outro grupo de comandos (TECHNOLOGY, 2020).

A cada dispositivo é atribuído um endereço único na rede e todo comando contém o endereço do módulo a que se refere. São mostrados a seguir dois quadros ilustrando, respectivamente, o formato de comando e resposta:

Caractere Inicial	Endereço do Módulo	Comando	Soma de Cheque	CR

Caractere Inicial	Endereço do Módulo	Dados	Soma de Cheque	CR

A soma de cheque é representada por dois caracteres e é composta pela soma de todos caracteres ASCII no comando, com exceção do *carriage return* (CR) e utiliza uma máscara de dois dígitos. A função de soma de cheque está desabilitada nos dispositivos de comunicação da planta laboratorial.

2.3 OPC

Sob o nome de OPC compreende-se uma série de padrões e especificações utilizados na telecomunicação industrial. Originalmente, significava *OLE (Object Linking and Embedding) for Process Control* quando surgiu em 1996, e estava restrito ao sistema operacional Windows (OPCFOUNDATION, 2019).

O OPC surgiu da necessidade dos fornecedores da indústria de automação industrial, buscando a padronização da comunicação com o sistema operacional Windows. Antes do

OPC, cada fornecedor precisava desenvolver seu próprio driver. Foi então feita uma força-tarefa pelas empresas fornecedoras para o desenvolvimento desse padrão de comunicação (MAHNKE; LEITNER; DAMM, 2009).

Juntamente com o OPC, foi criada a organização sem fins lucrativos chamada OPC Foundation que é responsável pelo padrão. Quase todos os provedores de sistemas para automação industrial se tornaram membros da organização.

Desde sua criação, o OPC expandiu suas funcionalidades e tecnologias utilizadas para transmissão de dados. Por esse motivo, OPC passou a significar *Open Platform Communications* e é utilizado atualmente até fora do contexto industrial inicialmente proposto.

2.3.1 OPC Clássico

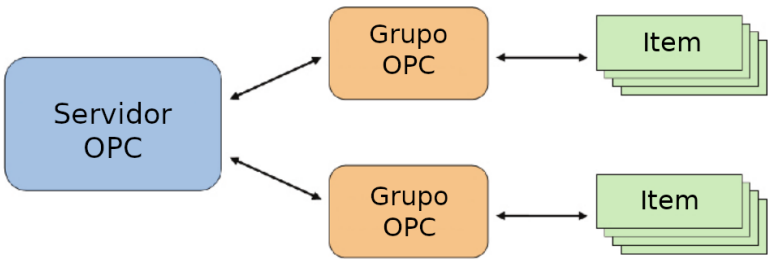
As especificações do OPC Clássico são baseadas na tecnologia COM/DCOM (*Distributed Component Object Model*) da Microsoft e são dividas em três grupos: OPC DA (*Data Access*), que define a leitura e escrita de variáveis de processo, OPC AE (*Alarms & Events*), que define a troca de dados para mensagens de alarmes e eventos, e OPC HDA (*Historical Data Access*), que define métodos aplicados para dados do histórico.

O conceito de cliente e servidor é utilizado na troca de dados. O servidor captura as informações do processo e as disponibiliza através de sua interface, já o cliente conecta-se ao servidor OPC e pode acessar e consumir os dados oferecidos.

O OPC DA permite leitura, escrita e monitoramento de variáveis correntes de processo. Seu principal uso é transmitir dados de PLCs, dispositivos de aquisição de dados e dispositivos de controle para IHMs e telas de clientes. Por isso é a interface mais importante do OPC Clássico (MAHNKE; LEITNER; DAMM, 2009).

Os clientes OPC DA selecionam os itens que necessitam ler, podem agrupá-los caso tenham parâmetros idênticos em um objeto de grupo. Quando definido um grupo, os itens podem ser todos lidos ou escritos de uma só vez pelo cliente. A Figura 2 mostra diferentes objetos que um cliente cria no servidor (MAHNKE; LEITNER; DAMM, 2009).

Figura 2: *Objectos criados por um cliente OPC para acessar dados*



Fonte: Adaptado de Mahnke, Leitner e Damm (2009)

No entanto, uma forma mais eficiente para monitorar os itens do grupo é receber apenas valores que sejam alterados no servidor. Pode-se definir uma frequência de atualização para um grupo de interesse e o servidor verifica ciclicamente mudanças nesses valores. Ao fim de cada ciclo, o servidor envia apenas os valores modificados para o cliente (MAHNKE; LEITNER; DAMM, 2009).

As principais desvantagens desse protocolo são sua dependência com a plataforma Windows e problemas envolvendo o DCOM em relação à comunicação remota, além de que o DCOM apresenta uma configuração complexa.

Porém, desde sua criação, o protocolo atendeu bem a comunidade, estabelecendo um padrão para a comunicação de equipamentos de diferentes empresas. Foi com a evolução da tecnologia e surgimento de novas necessidades que novas especificações fizeram-se necessárias. Por esses motivos em 2008 a OPC Foundation lançou o OPC UA (*Universal Access*).

2.3.2 OPC UA

A principal motivação para o surgimento do OPC UA é quebrar a dependência do OPC com a tecnologia COM/DCOM, com a intenção de criar um novo protocolo que pudesse substituir todas as especificações do OPC Clássico sem perder nenhuma funcionalidade ou performance. Esse novo protocolo também deveria cobrir todos os requisitos para ser independente de plataforma e capaz de modelar sistemas complexos (MAHNKE; LEITNER; DAMM, 2009).

O OPC Clássico foi concebido como uma interface para drivers e passou a ser utilizado como uma interface para sistemas. Porém, a sua capacidade de comunicação por rede não é robusta ou independente de plataforma. O OPC UA é concebido para superar essas limitações, sendo a interoperabilidade entre sistemas o principal requisito (MAHNKE; LEITNER; DAMM, 2009).

Outro aspecto importante do protocolo é a modelagem de dados. O OPC UA define a estrutura para modelar a informação, que pode ser modelada pelos usuários. Essa modelagem permite incluir mais informações na aquisição de dados, como qual o tipo de dispositivo está enviando a informação. Como também utilizar técnicas orientadas a objeto para que clientes possam tratar todas informação do mesmo tipo da mesma forma (MAHNKE; LEITNER; DAMM, 2009).

Nesse protocolo, o transporte de informações é feito via serviços web, por padrões como SOAP (*Simple Object Access Protocol*), HTTP (*Hypertext Transfer Protocol*) ou ainda através de um protocolo binário TCP para comunicação de alta performance.

2.4 Trabalhos Relacionados

Nesta seção são apresentados abordagens de outros autores para aplicações semelhantes à utilizada neste trabalho.

O trabalho de Patil et al. (2012) trata da implementação em código aberto da aquisição de dados e controle do software LabVIEW. Para isso utiliza o GNURadio, OpenCV, SCilab, Xcos, e COMEDI no Linux, todos softwares de código aberto e licença livre.

Em sua tese de mestrado, Santos Neto (2013) propõe um sistema de emulação de ambiente industrial com características multivariáveis. A comunicação de seus sistema é feita utilizando o OPC clássico e o controle de processos utilizando o Scilab. Nesse trabalho são abordadas diferentes estratégias de controle e é implementada a comunicação do seu sistema com um módulo físico laboratorial.

Em seu trabalho, Mohamed e Abbas (2011) propõe um sistema SCADA baseado em serviços de web, utilizando o protocolo OPC DA e tenta contornar as limitações de conectividade e dependência de plataforma do OPC DA utilizando tecnologias como ASP.NET e AJAX. Apesar de o trabalho ter sido realizado após o lançamento de OPC UA, que também busca resolver essas questões, é apresentada uma abordagem que pode ser utilizada em sistemas já em operação e nos quais não é possível atualizar toda a arquitetura da rede.

3 METODOLOGIA

3.1 Materiais Utilizados

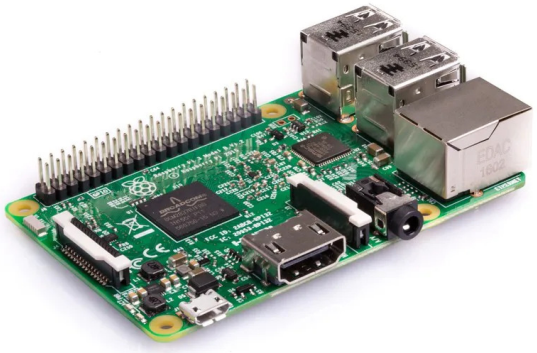
Essa seção descreve os materiais utilizados e justifica a escolha dos mesmos para o projeto.

3.1.1 Raspberry Pi 3

A Raspberry Pi é um computador de baixo custo, criado com o intuito de ensinar jovens a programar. Foi desenvolvido no Reino Unido e é mantido pela Raspberry Pi Foundation, uma instituição britânica sem fins lucrativos. Acabou por se tornar muito popular e foi adotada por públicos além do seu público alvo, é utilizada, por exemplo, em automação residencial, robótica e inúmeros projetos amadores (RASPBERRY..., 2020).

Sua escolha para esse projeto deu-se justamente pelo baixo custo da placa, é comercializada por R$ 440,00. O modelo em específico utilizado é o 3 B+, que pode ser visto na Figura 3. Apesar de suas especificações serem bastante modestas, esse mini computador atende às necessidades do projeto.

Figura 3: *Raspberry Pi Modelo 3 B+*



Fonte: www.raspberrypi.org

Essa placa utiliza um processador de 64 bits com quatro núcleos operando em 1.2GHz, arquitetura ARM e sistema operacional Linux, possui 1 giga de memória ram DDR3. Também apresenta uma porta ethernet de 100Mb/s, conexão wi-fi e bluetooth, quatro portas USB 2.0 e quarenta pinos de entrada e saída genéricos. A alimentação da placa é feita por uma porta micro USB de 5V e 2,5A. Mais especificações no anexo A.

3.1.2 Adaptador Serial

Apesar da Raspberry Pi contar com pinos de entrada e saída genéricos, optou-se por utilizar um adaptador de USB-serial para comunicação com os módulos da planta. Essa escolha simplifica a instalação da placa, pois os sinais de tensão na Raspberry Pi variam de 0V a +5V e os sinais do protocolo RS-232 variam de -15V a +15V, evitando-se assim a implementação de um circuito para o tratamento do sinal de comunicação. O adaptador pode ser visto na Figura 4.

Figura 4: *Adaptador USB-serial*

Fonte: www.mercadolivre.com.br

3.2 Python

Python é uma linguagem de programação de alto nível, interpretada e orientada a objeto. Sua concepção é focada na facilidade de leitura e manutenção dos códigos. É desenvolvido sob uma licença de código aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation (PYTHON..., 2020).

Foi a linguagem escolhida para o desenvolvimento do trabalho por sua natureza de código aberto e pela disponibilidade de bibliotecas para variadas aplicações. Outros fatores importantes são a facilidade de instalação e compatibilidade com diversas plataformas e sistemas operacionais. Nas subseções 3.2.1 e 3.2.2 são abordadas as principais bibliotecas utilizadas neste projeto.

3.2.1 Pyserial 3.4

A biblioteca Pyserial oferece métodos para utilização da porta serial do computador em Python, selecionando automaticamente qual *backend* utilizar dependendo do sistema operacional.

A função *Serial* permite definir um objeto que especifica os parâmetros da comunicação serial, como por exemplo, qual a porta a ser utilizada, qual o *baudrate*, *timeout*, etc. E os métodos *read* e *write* de um objeto do tipo *serial* são os comandos utilizados para ler e escrever bytes na porta serial.

3.2.2 Python OPC-UA 1.15

A biblioteca Python OPC-UA é a implementação em python do protocolo OPC UA. A API desta biblioteca oferece métodos para enviar e receber todos os modelos de dados

definidos em OPC UA. Utilizando-a, é possível programar um cliente ou servidor com poucas linhas de código.

A classe *Server* é utilizada para criar um servidor com os valores padrão predefinidos. Essa classe possui métodos para configurar seus parâmetros, como a url do servidor, certificados, algoritmos de criptografia e etc. Também possui métodos para configuração de nós, objetos e variáveis.

A classe *Client* é utilizada para criar um cliente. Possui métodos para encontrar e se conectar a servidores. Também é possível ler e escrever variáveis no servidor e se subscrever para alterações em variáveis.

Outra biblioteca relacionada a esta é a *opcua-gui-client* que oferece um cliente OPC UA com interface gráfica.

3.3 Descrição da Planta

A planta é constituída por seis tanques de geometria esférica com vinte centímetros de diâmetro, que podem operar individualmente, com a vazão de saída retornando para o tanque reservatório, ou conectados, com a vazão do tanque mais alto fluindo para o mais baixo. Os dois tanques inferiores, que estão na parte posterior do painel, só recebem vazão de outros tanques. A Figura 5 apresenta uma visão geral da planta.

Figura 5: *Visão geral dos tanques de nível*

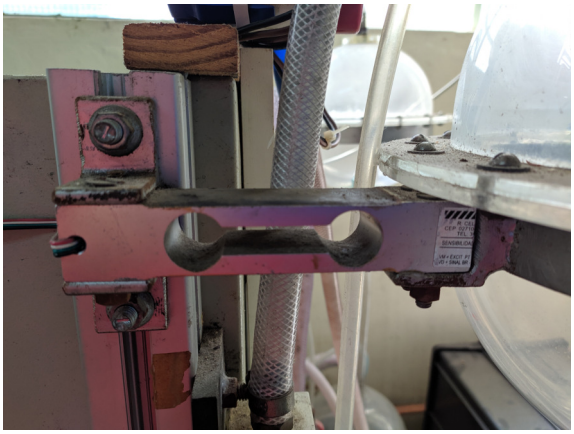
Fonte: Autor

O sistema dispõe de dois tipos de sensores: células de carga, para medição indireta do nível através do peso dos tanques e medidores de vazão. Imagens dos respectivos sensores são apresentadas na Figura 6.

Como atuadores são utilizadas quatro bombas de porão com capacidade de vazão 1360L/h a nível 0, ou então 1000L/h a 1m de altura. O atuador pode ser visto na Figura 7.

Figura 6: *Sensores da planta*

(a) *Célula de carga*



(b) *Medidor de vazão*

Fonte: Autor

Figura 7: *Bomba de porão utilizada na planta*

Fonte: www.mercadolivre.com.br

Para aquisição de dados, dispõe-se de dois módulos I-7017 e para controle de um módulo I-7024 fabricados pela ICPDAS, cada qual dispondo de oito canais de comunicação. Os módulos da série I-7000 providenciam conversão de sinal de analógico para digital ou de digital para analógico e são remotamente controlados por um grupo de comandos, chamados de protocolo DCON. A comunicação é feita através de barramento serial RS-232 e utiliza caracteres em código ASCII. Os módulos podem ser vistos na Figura 8, que mostra a parte interna do painel da planta. Nessa figura também é possível ver os circuitos de condicionamento dos sinais de saída para os atuadores.

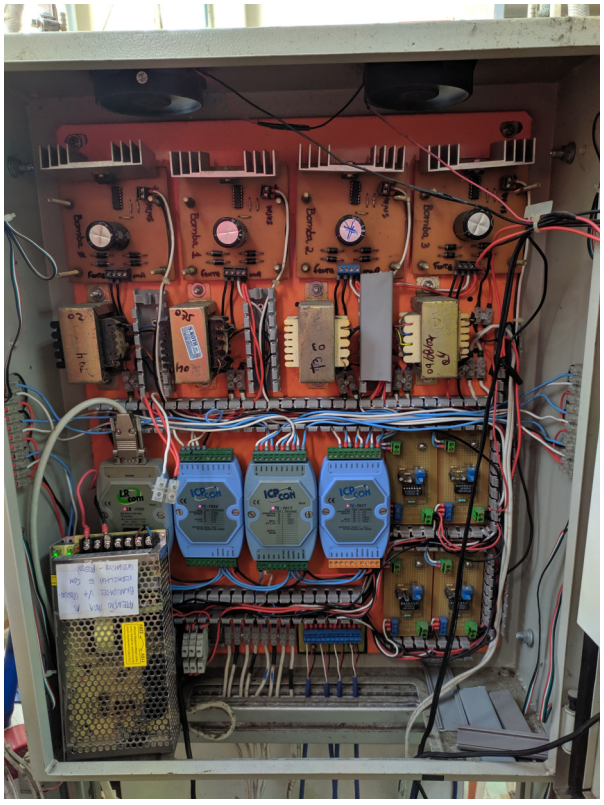


Figura 8: *Parte interna do painel*

Fonte: Autor

3.4 Arquitetura de Rede

Esta seção trata das diferenças de arquitetura de rede entre o sistema em uso e o proposto neste trabalho.

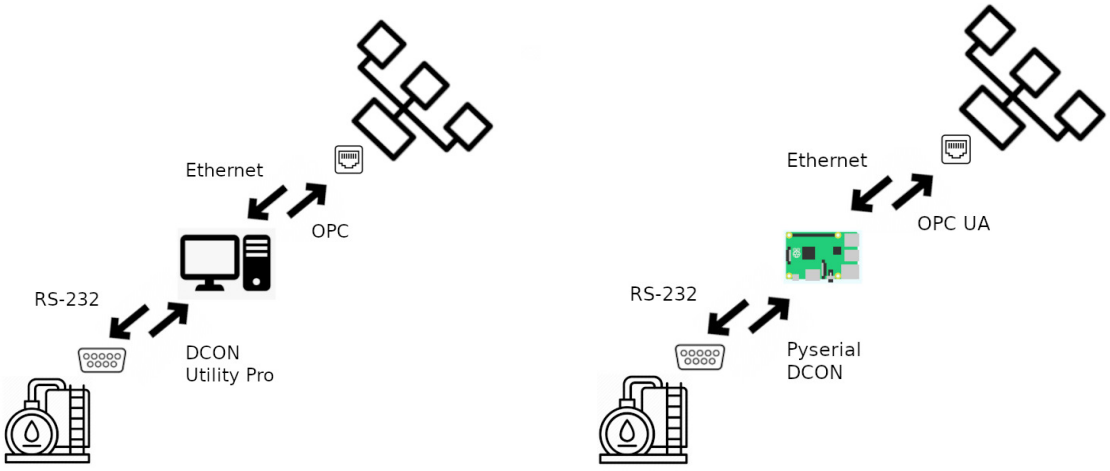
O sistema em uso utiliza o driver fornecido pela ICPDAS para realizar a comunicação serial e a distribuição dos dados na rede é feita pelo OPC-DA, utilizando o software E3 como SCADA. Um inconveniente desse sistema é que o computador servidor precisa estar com um usuário logado na rede Windows e também só aceita conexão de clientes Windows. Outro inconveniente é a necessidade de licença do programa, visto que o laboratório dispõe de apenas uma chave de licença completa, na maioria das vezes o servidor não suporta sessões de mais de uma hora.

O sistema proposto utiliza a Raspberry Pi como servidor, comunicando-se através da implementação em Python através dos comandos do protocolo DCON. A distribuição dos dados na rede é feita através do OPC UA, que os disponibiliza por TCP/IP, sendo assim

possível conectar qualquer tipo de cliente. A Figura 9 ilustra a comparação da arquitetura dos sistemas.

Figura 9: *Comparação da arquitetura dos sistemas*

(a) *Sistema em uso* **(b)** *Sistema proposto*



Fonte: Autor

3.5 Implementação do Protocolo DCON

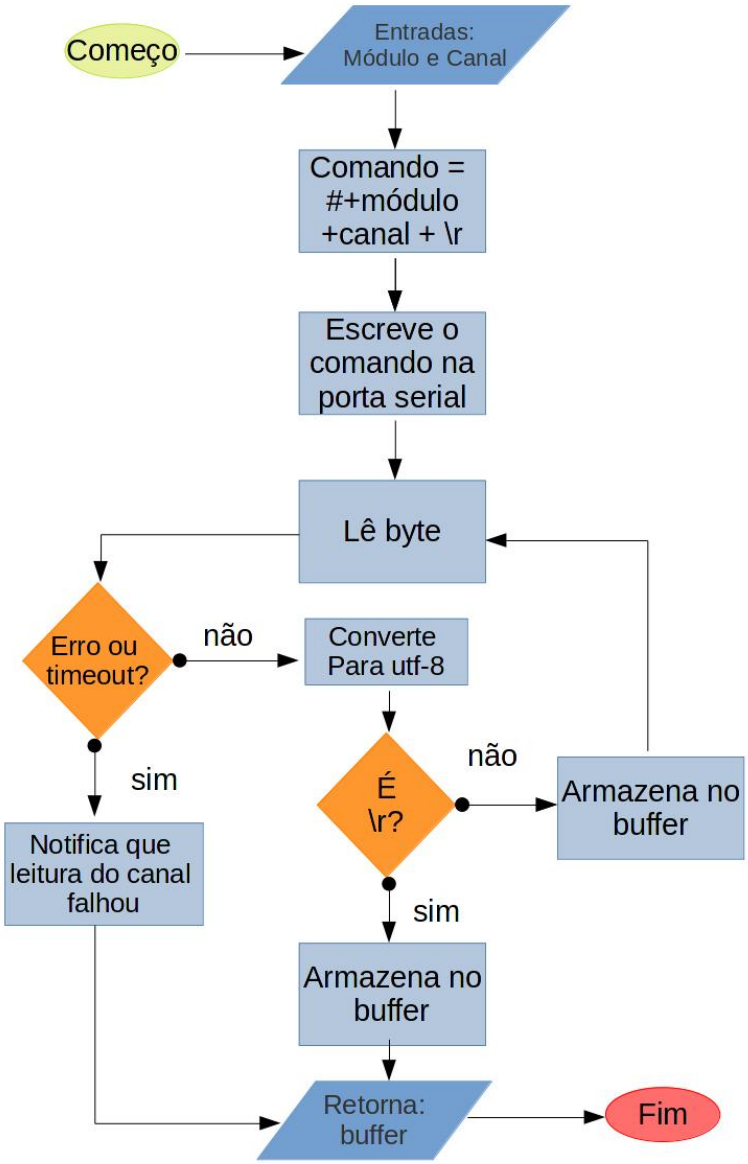
A implementação do protocolo DCON em Python é feita utilizando-se principalmente a biblioteca *pyserial*. A partir dela, são definidas funções para executar os comandos de leitura e escrita nos canais dos módulos que são suficientes para implementar o supervisório e controle. O protocolo dispõe de outros comandos, que são utilizados para a configuração dos módulos de comunicação.

É necessário implementar uma função auxiliar que é compartilhada tanto pela escrita quanto pela leitura. A função *read_data*, a qual lê caracteres da porta serial até encontrar o carácter *carriage return* e os converte de ASCII para utf-8, a codificação padrão do Python 3. Todas as mensagens do protocolo terminam com o carácter CR. Essa função trata eventuais erros na leitura serial como *time out* ou simplesmente falha na leitura, também ignora caracteres que não possam ser convertidos de ASCII para utf-8.

Define-se então a função *read_ch*, responsável pela leitura de um canal de módulo. Ela toma como argumentos o endereço do módulo, o canal a ser lido e a porta serial a ser utilizada. O comando é composto inserindo-se o carácter # no início da mensagem, o endereço, o canal e o CR para indicar fim da mensagem. O comando é enviado pela porta serial especificada e por fim a função *read_data* lê a resposta do módulo e retorna essa cadeia de caracteres. O fluxograma da Figura 10 ilustra o algoritmo da função.

A função *write_ch* é muito semelhante. A diferença consiste no módulo a que se destina, pois é um módulo de controle e não de aquisição, também é necessário incluir mais um caractere para especificar se o sinal de controle é positivo ou negativo e o valor que se pretende escrever no canal. E por fim a função também retorna uma cadeia de caracteres como confirmação de que o comando foi recebido.

Figura 10: *Fluxograma de comunicação DCON*



Fonte: Autor

3.6 Configuração do Servidor OPC UA

Para a configuração do servidor, utilizam-se as bibliotecas *opcua*, *pyserial*, *Multiprocessing* e *Threading*, as duas últimas a fim de paralelizar processos e executá-los periodicamente. Além disso, são utilizadas as funções definidas para o protocolo DCON. É necessário registrar a placa na rede interna da UFRGS com um endereço fixo de IP.

Define-se a url do servidor a partir do IP fixo da placa e uma porta livre para o serviço na rede, configura-se também parâmetros de segurança. As mensagens são criptografadas com o *Secure Hash Algorithm 256* (SHA-256) e o servidor utiliza um certificado e chave de acesso gerados pelo padrão X.509 para conexão de clientes.

Configuram-se então os objetos do servidor, sendo um para a medição de nível dos tanques, um para a medição de vazão e outro para as bombas e então insere-se as variáveis de cada canal nos respectivos objetos. As bombas são definidas como passíveis de escrita pelos clientes. A configuração com o endereço de cada variável no servidor é vista no tabela 1.

Tabela 1: *Endereço das variáveis*

Objeto	Variável	Nó	Item
Sensores de Nível	Nível 1	2	2
Sensores de Nível	Nível 2	2	3
Sensores de Nível	Nível 3	2	4
Sensores de Nível	Nível 4	2	5
Sensores de Nível	Nível 5	2	6
Sensores de Nível	Nível 6	2	7
Sensores de Vazão	Vazão 1	2	8
Sensores de Vazão	Vazão 2	2	9
Sensores de Vazão	Vazão 3	2	10
Sensores de Vazão	Vazão 4	2	11
Atuadores	Bomba 1	2	14
Atuadores	Bomba 2	2	15
Atuadores	Bomba 3	2	16
Atuadores	Bomba 4	2	17

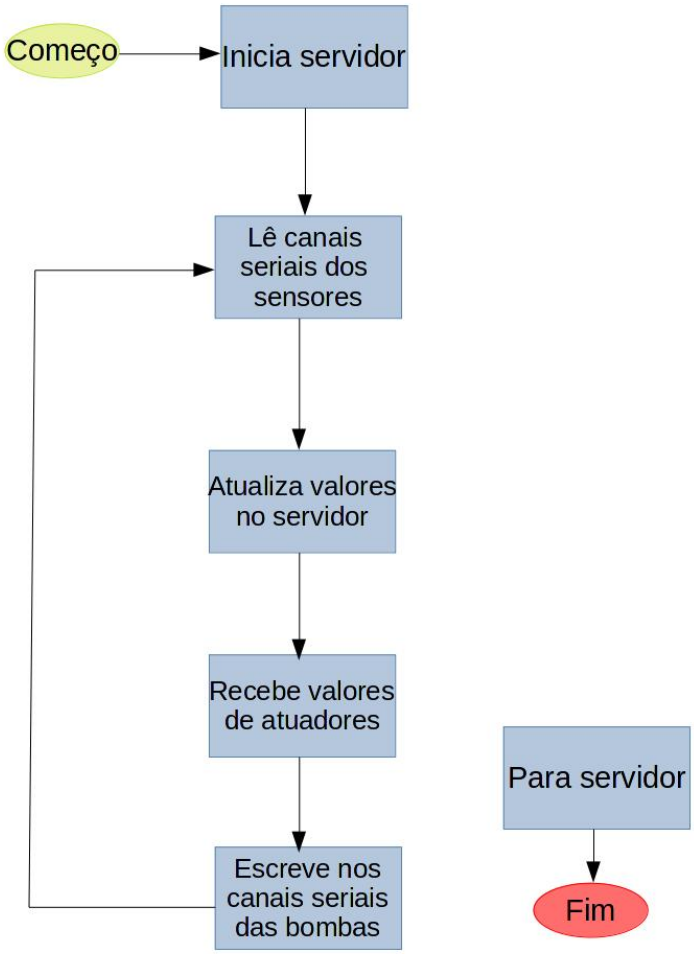
Fonte: Autor

O servidor também é responsável por atualizar o valor dessas variáveis e responder as solicitações dos clientes. Para isso, utiliza-se um laço que faz a leitura de todas as variáveis dos canais pela porta serial e atualiza seus valores, esse laço está configurado para ser repetido a cada segundo. Nesse mesmo laço também é feita a atualização do valor do sinal de comando enviado para as bombas através de comunicação serial. O valor informado ao módulo de controle está registrado no servidor, o qual normalmente é atualizado por um cliente que esteja realizando o controle de nível. A comunicação com os clientes é executada enquanto o servidor estiver operando, através da url definida. O fluxograma visto na Figura 11 representa o algoritmo.

3.7 Configuração do Cliente OPC UA

Configura-se o cliente de duas maneiras diferentes: a primeira é utilizando a biblioteca *opcua-client-gui* e a segunda é uma configuração por script.

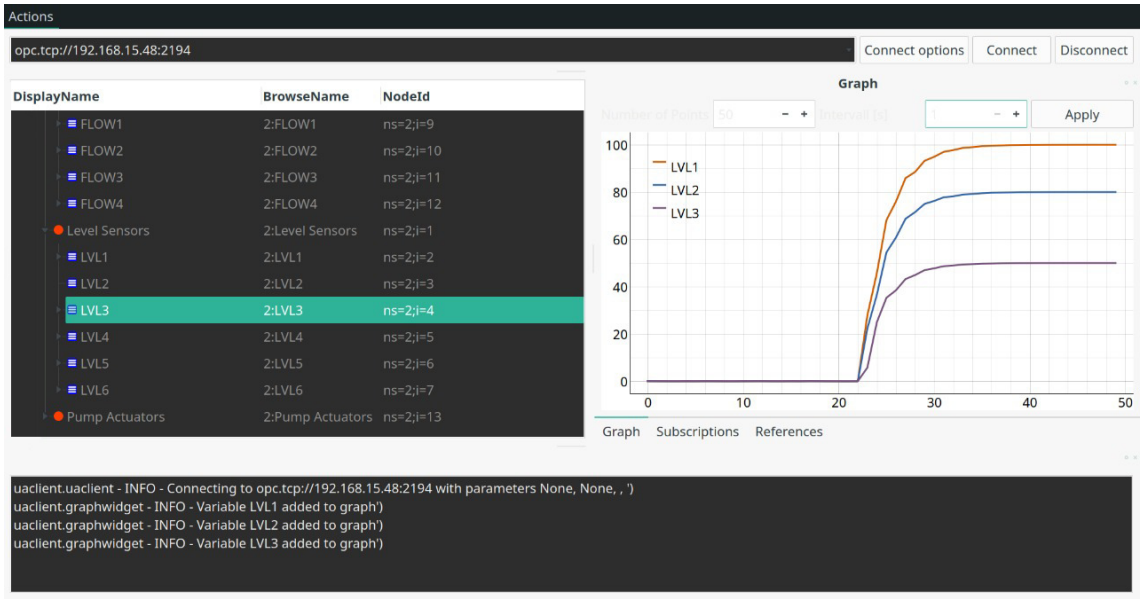
Figura 11: *Fluxograma de funcionamento do servidor*



Fonte: Autor

A biblioteca *opcua-client-gui* oferece uma interface gráfica e é útil para testar a conexão com o servidor e verificar a configuração de grupos e itens na rede. É possível também gerar gráficos de leitura de variáveis, contudo é uma interface simples, os gráficos são limitados a 100 pontos, portanto não serve como um historiador. Uma imagem dessa interface em pode ser vista na Figura 12.

Figura 12: *Interface Gráfica do Sistema*



Fonte: Autor

A segunda configuração, por script, é utilizada para realizar testes de controle em malha fechada com a planta. Configuram-se os parâmetros do servidor a ser conectado, os parâmetros de segurança e quais variáveis serão monitoradas. Nesse script também é definida a regra de controle utilizada.

3.7.1 Controlador

Como o escopo desse trabalho não envolve o desenvolvimento do controlador para a planta de nível, não foi feita uma análise para se obter o controlador o utilizado nos testes. Optou-se por utilizar um controlador disponível de experimentos anteriores em aula de graduação, que já havia sido testado.

O controlador utilizado é um PID na forma paralela padrão ISA, $Kp \cdot (1 + \frac{Ki}{s} + \frac{Kd \cdot s}{Kd \cdot \alpha \cdot s + 1})$, com a seguinte função de transferência:

$$C(s) = \frac{(Kp \cdot Kd \cdot \alpha) \cdot s^2 + (Kp \cdot Ki \cdot Kd \cdot \alpha + Kp) \cdot s + Kp \cdot Ki}{Kd \cdot \alpha s^2 + s} \tag{1}$$

Esse controlador é discretizado utilizando o método de retenção de ordem zero e 1 segundo de período de amostragem, é implementado digitalmente em script Python no cliente que faz o controle em malha fechada. Os parâmetros de controle e a função discretizada são apresentados no capítulo de resultados, pois foram utilizadas funções distintas para o ensaio em laboratório e fora de laboratório.

3.8 Ambiente de Simulação

Por conta de restrição do uso do laboratório, foi desenvolvido um ambiente de simulação . O comportamento do sistema simulado deve ser tal que funcione como a planta real, em termos de comunicação e não de comportamento dinâmico. A Raspberry Pi deve executar as mesmas rotinas que executaria no laboratório independente do dispositivo que está comunicando-se com ela.

Para isso definiu-se uma simulação numérica para o comportamento da planta. Simplifica-se o modelo para um sistema de primeira ordem e aproxima-se um ganho DC com base nos dados de ensaio. A dinâmica do sistema também é definida com base no tempo aproximado de resposta do sistema. Foi utilizada a função de transferência vista em (2).

$$G(s) = \frac{0.6385}{300 + s} \tag{2}$$

E quanto à comunicação, definem-se funções que tem o comportamento idêntico aos módulos de comunicação da planta. Elas correspondem aos módulos e canais do sistema real e o formato de sua resposta ou aquisição de dados também é a mesma. Os valores informados são provenientes da simulação numérica.

O servidor OPC UA também pode ser testado nesse ambiente, de fato, do ponto de vista de rede o ambiente simulado e o real não tem diferença. É possível realizar a simulação numérica da planta em um computador, realizar a comunicação serial com a Raspberry Pi e acessar os dados da simulação através de um terceiro dispositivo. Basta que nesse dispositivo seja possível configurar um cliente OPC UA, isso é possível em um aparelho Android ou um terceiro computador.

4 RESULTADOS

Nesta seção são apresentados os resultados do sistema proposto em funcionamento, tanto em simulação quanto em laboratório, e uma comparação com o sistema em uso através de um ensaio de controle. Essa comparação inclui a resposta do sistema em malha fechada e dados acerca do tempo de resposta do sistema proposto.

4.1 Tempo de resposta

Foi feita uma medição do tempo de leitura do cliente ao requisitar uma informação do servidor. Esse tempo foi medido utilizando a função *timeit* da biblioteca *time* em Python. Essa não é uma medição rigorosa e pode ser afetada por outros processos do sistema operacional.

O tempo médio de leitura em laboratório foi de 19,72 milissegundos, com um desvio padrão de 23,4 milissegundos entre 1318 amostras. Fora de laboratório, o tempo médio de leitura foi de 10,45 milissegundos, com um desvio padrão de 20,52 milissegundos em 3448 amostras. Essa diferença talvez se justifique pela capacidade de processamento do cliente ou pela estrutura da rede ethernet, por exemplo, na UFRGS as informações precisam passar pelo CPD (Centro de Processamento de Dados) e também o uso da rede por outros usuários pode afetar essa medição.

4.2 Ensaios de Controle

Com o objetivo de demonstrar o funcionamento do sistema projetado, foram realizados ensaios de controle em malha fechada. O controle foi feito por um terceiro computador no laboratório conectado como cliente, ora ao servidor OPC, ora ao servidor OPC UA.

Uma diretriz desse ensaio foi utilizar os sistemas nas condições mais semelhantes possíveis. Porém, em função de restrições de uso do laboratório isso não foi possível, conforme explicado em 4.2.1.1.

4.2.1 Ensaios em Laboratório

As imagens nessa subseção são referentes aos ensaios realizados em laboratório.

4.2.1.1 Ensaio com o Sistema na Configuração Proposta

Na Figura 13 é apresentada a resposta do ensaio de controle utilizando o sistema proposto, com o controle feito numericamente por script em Python e utilizando o servidor OPC UA. Após o ensaio, percebeu-se que houve uma diferença de configuração do controlador, mas não foi possível repetir o procedimento. Esse ensaio foi realizado com os

seguintes parâmetros:

- Proporcional, Kp = 2.5
- Integral, Ki = 0.1
- Derivativo, **Kd = 0.1**
- Filtro derivativo, $\alpha = .1$

Que gera a seguinte função de transferência:

$$C(s) = \frac{0.275 \cdot s^2 + 2.502 \cdot s + 0.25}{0.01 \cdot s^2 + s} \tag{3}$$

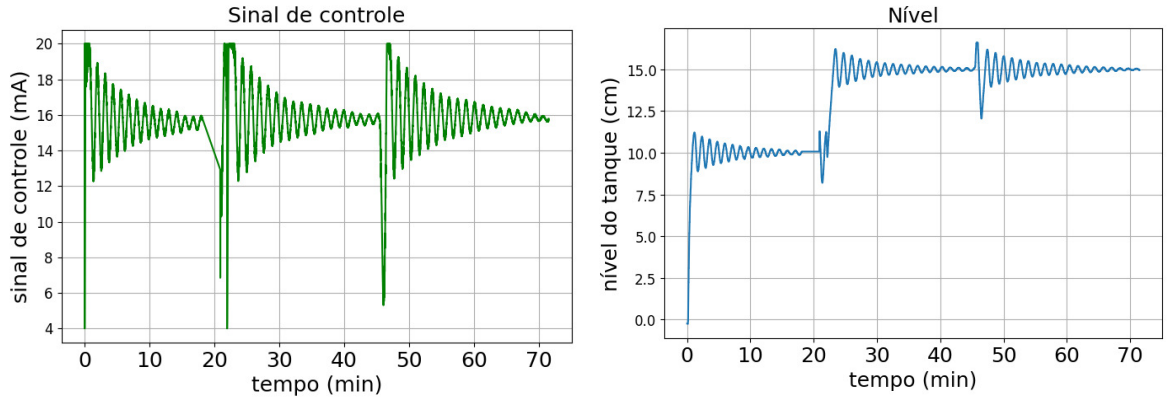
Foi feita então a discretização dessa função de transferência utilizando o método de amostragem de ordem 0 e tempo de amostragem de 1 segundo. Obtem-se a seguinte função discretizada:

$$C(z) = \frac{5 \cdot z^2 - 8.75 \cdot \ + 4}{z^2 - z + 4.54 \cdot 10^{-5}} \tag{4}$$

O controlador discretizado é facilmente implementado numericamente. Só é necessário definir uma função de laço e armazenar dois valores antecessores e definir as regras de atualização do sinal de controle conforme os termos do controlador. O mesmo código apresentado no apêndice A foi utilizado para todos os controladores em Python, alterando-se apenas os ganhos.

Figura 13: *Ensaio de controle em laboratório utilizando o sistema proposto com ganho derivativo de 0,1*

(a) *Ação de Controle* **(b)** *Resposta da Planta*



Fonte: Autor

O sistema apresentou uma resposta lenta, levou em torno de 20 minutos para atingir a referência e continuou apresentando um comportamento oscilatório indefinidamente. Uma hipótese que poderia ser levantada para justificar o desempenho do controlador seria que o atraso na comunicação pudesse estar afetando a resposta. Essa hipótese é descartada pois 20 milissegundos de atraso são irrelevantes comparados à dinâmica da planta que tem uma constante de tempo de 300 segundos.

4.2.1.2 Ensaio com o sistema na configuração atual

A Figura 15 mostra o sinal de controle e resposta de nível da planta controlada pelo sistema com a configuração atual, com o controle sendo processado no Matlab. Foram utilizados os seguintes parâmetros no controlador:

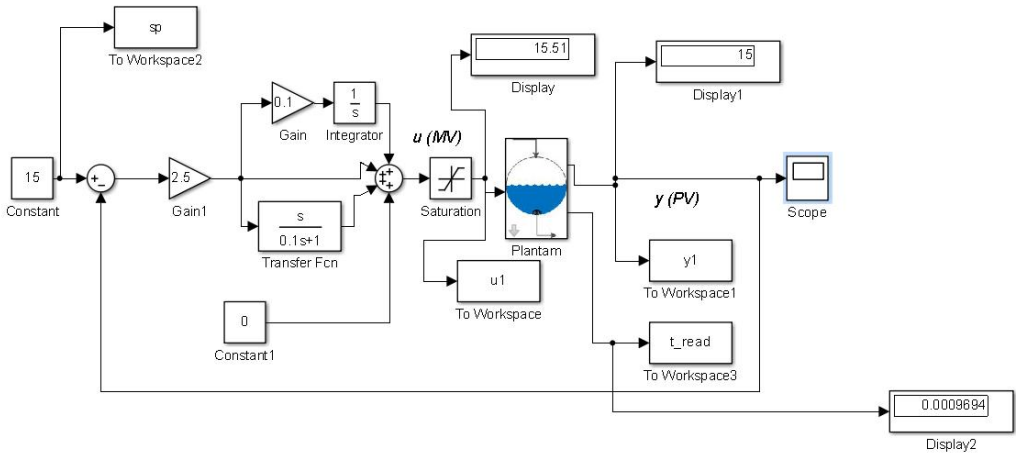
- Proporcional, Kp = 2.5
- Integral, Ki = 0.1
- Derivativo, **Kd = 1**
- Filtro derivativo, $\alpha = 0.1$

Que geram a seguinte função de transferência para o controlador:

$$C(s) = \frac{2.75 \cdot s^2 + 2.525 \cdot s + 0.25}{0.1 \cdot s^2 + s} \tag{5}$$

Esse controlador foi implementado em Matlab, utilizando o simulink, conforme a Figura 14.

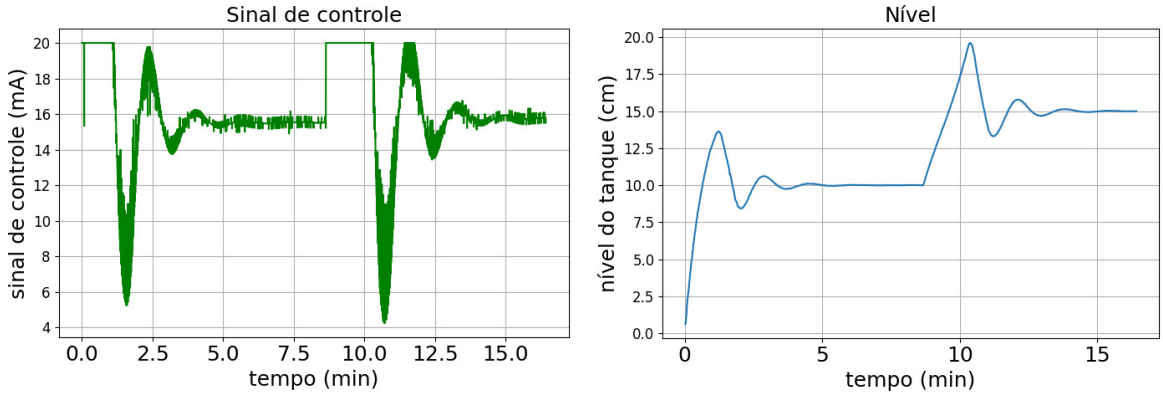
Figura 14: *Controlador para ensaio em laboratório*



Fonte: Autor

Figura 15: *Ensaio de controle em laboratório utilizando o sistema com a configuração atual e ganho derivativo de 1*

(a) *Ação de Controle* **(b)** *Resposta da Planta*



Fonte: Autor

O ensaio inicia com a referência em 10 centímetros, e após atingir essa referência, ela é atualizada para 15 centímetros. O sinal de controle saturou apenas no início do ensaio e na troca de referência, em seguida oscila com uma amplitude de 14 miliamperes até acomodar-se. O sistema levou aproximadamente 5 minutos para atingir a referência.

4.2.2 Ensaios com Planta Simulada

O ensaio com todos os tanques em operação foi feito com a planta simulada. Mesmo sendo um ambiente simulado, a malha de controle foi fechada remotamente por rede utilizando o protocolo OPC UA, tal como no ensaio em laboratório.

Foram utilizados os seguintes parâmetros no controlador:

- Proporcional, Kp = 2.5
- Integral, Ki = 0.1
- Derivativo, Kd = 1
- Filtro derivativo, $\alpha = .1$

Que gera a seguinte função de transferência:

$$C(s) = \frac{2.75 \cdot s^2 + 2.525 \cdot s + 0.25}{0.1 \cdot s^2 + s} \tag{6}$$

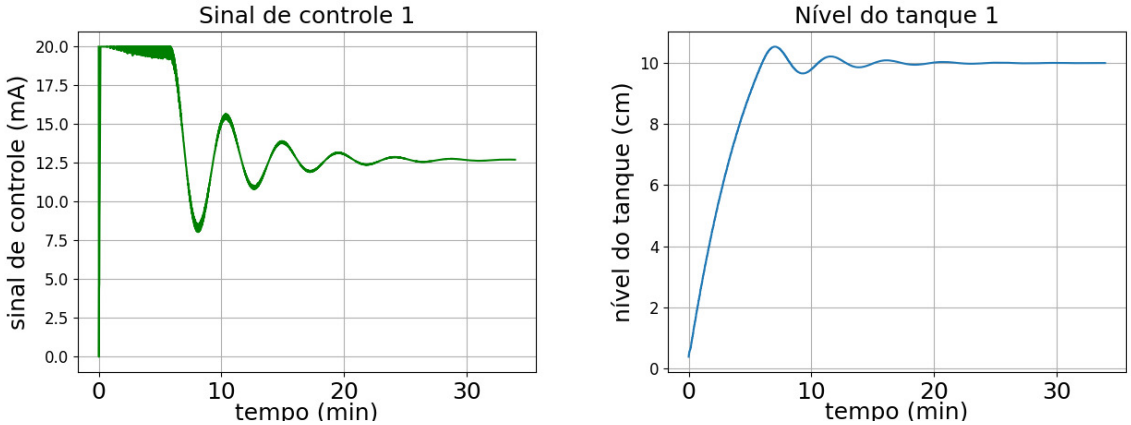
Foi feita então a discretização dessa função de transferência utilizando o método de retenção de ordem 0 e tempo de amostragem de 1 segundo. Obtem-se a seguinte função discretizada:

$$C(z) = \frac{27.5 \cdot z^2 - 52.25 \cdot \ +25}{z^2 - 3 \cdot z} \tag{7}$$

Foram utilizadas referências distintas em cada tanque, os sinais de controle e nível da planta para cada tanque podem ser vistos nas figuras 16, 17, 18 e 19. O sistema atingiu a referência em todos os casos, com exceção do tanque 4, por questões do modelo utilizado, mesmo com máximo sinal de controle não atingiu a referência.

Figura 16: *Tanque 1 - Ensaio com 4 tanques simultaneamente em planta simulada*

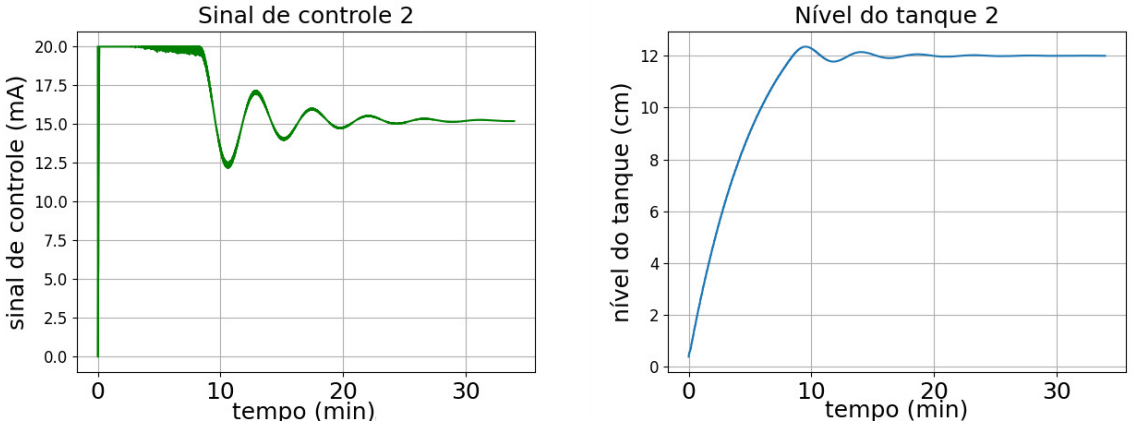
(a) *Ação de Controle* **(b)** *Resposta da Planta*



Fonte: Autor

Figura 17: *Tanque 2 - Ensaio com 4 tanques simultaneamente em planta simulada*

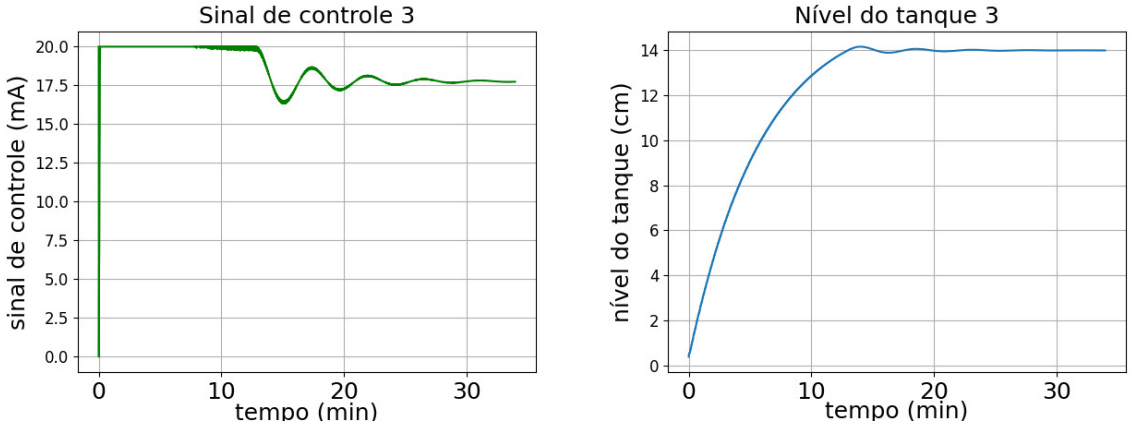
(a) *Ação de Controle* **(b)** *Resposta da Planta*



Fonte: Autor

Figura 18: *Tanque 3 - Ensaio com 4 tanques simultaneamente em planta simulada*

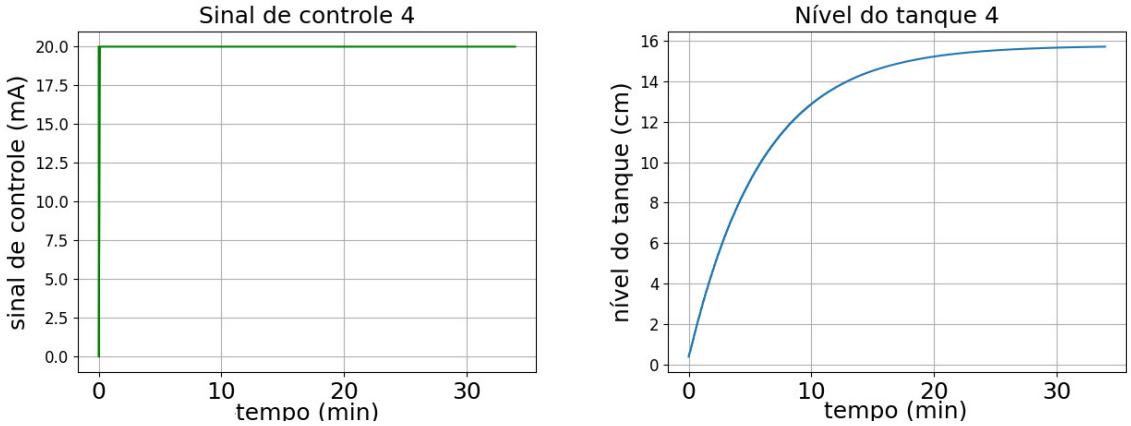
(a) *Ação de Controle* **(b)** *Resposta da Planta*



Fonte: Autor

Figura 19: *Tanque 4 - Ensaio com 4 tanques simultaneamente em planta simulada*

(a) *Ação de Controle* **(b)** *Resposta da Planta*



Fonte: Autor

5 CONCLUSÃO

Este trabalho teve como objetivo implementar um sistema supervisório de baixo custo e de código aberto e também testá-lo. Pode-se considerar que o projeto atendeu os seus requisitos, porém não foi possível fazer uma comparação direta dos dois sistemas de controle, em função das restrições de uso da planta. Apesar da alta histórica do dólar no momento em que esse trabalho é concluído, com o investimento de R$ 600,00 é possível comprar a placa, fonte, cartão de memória e cabo serial para implementar o sistema.

Analisando os dados obtidos, os tempos de resposta do sistema proposto apresentaram uma latência de aproximadamente 20 milissegundos. Contudo, para a aplicação da planta de nível laboratorial estudada neste trabalho, isso não é relevante, pois um tempo de resposta desa ordem é desprezível frente à dinâmica do processo, que tem uma constante de tempo de 300 segundos. Mas para uma aplicação em que a latência seja um fator crítico, e seja necessário utilizar controle de maneira remota, o OPC UA não seria a melhor opção.

O sistema implementado apresenta vantagens em relação à conectividade e compatibilidade com outros sistemas. É muito mais fácil conectar um novo cliente ao servido OPC UA do que ao servidor OPC Clássico, pois não há dependência de plataforma.

Esses sistemas de distribuição de dados, entretanto, não costumam ser utilizados para controle remoto, a menos que o processo tenha uma dinâmica lenta e não seja vantajoso ter um controlador no local de operação. A principal aplicação é distribuir os dados de processo, possibilitando o monitorando de processos de forma remota.

REFERÊNCIAS

BOYER, S. A. *SCADA: supervisory control and data acquisition*. Research Triangle Park, NC 27709, United States of America: International Society of Automation, 2009.

DANEELS, A.; SALTER, W. What is SCADA?, 1999.

ELINUX. *Raspberry Pi Genral History*. [S.l.: s.n.], 2020. Disponível em: <https://elinux.org/RPi_General_History>. (Acessado em 25 de out. de 2020).

GUTIERREZ, R. M. V.; PAN, S. S. K. Complexo eletrônico: automação do controle industrial. Banco Nacional de Desenvolvimento Econômico e Social, 2008.

MAHNKE, W.; LEITNER, S.-H.; DAMM, M. *OPC unified architecture*. [S.l.]: Springer Science & Business Media, 2009.

MOHAMED, A. M.; ABBAS, H. A. Efficient web based monitoring and control system. In: PROCEEDINGS of the Seventh International Conference on Autonomic and Autonomous Systems, ICAS. [S.l.: s.n.], 2011.

OPCFOUNDATION. *What is OPC?* [S.l.: s.n.], 2019. Disponível em: <https://opcfoundation.org/about/what-is-opc/>.

PATIL, J. Y. et al. GNURadio, Scilab, Xcos and COMEDI for Data Acquisition and Control: An Open Source Alternative to LabVIEW. *IFAC Proceedings Volumes*, Elsevier, v. 45, n. 15, p. 626–631, 2012.

PYTHON SOFTWARE FOUNDATION. *Python*. [S.l.: s.n.], 2020. Disponível em: <https://www.python.org>. (Acessado em 25 de out. de 2020).

ROBLES, R. J.; KIM, T.-H. Architecture for SCADA with mobile remote components. In: PROCEEDINGS of the 12th WSEAS international conference on Automatic control, modelling & simulation. [S.l.: s.n.], 2010.

SANTOS NETO, A. F. DOS. *AccacioFerreiradosSantosNetoAplicaçãodoProto coloAb erto-OPCedoFOSSScilabnoDesenvolvimentodeumMó duloLab oratorialparaControledePro cessosIndustriais*. 2013. Diss. (Mestrado) – Universidade Federal de Juiz de Fora.

TECHINICAL INFORMATION BULLETTIN, N. 04-1, National Communications System, Technical Information Bulletin 04-1. *Supervisory Control and Data Acquisition (SCADA) Systems*, p. 4–5, 2004.

TECHNOLOGY, I. A. *I-7017, I-7018, I-7019, M-7017, M-7018 and M-7019 Series User Manual*. [S.l.], 2020. March 31, 2020.

APÊNDICE A - CÓDIGOS EM PYTHON

```
# códigos disponíveis em https://github.com/Neryfoot/TCC
# implementação do protocolo DCON
def read_data(ser):  # lê bytes até ler \r(carriage return)
    buffer = ""
    while True:
        try:
            one_byte = ser.read(1)
        except:
            print("Serial read failed or took too long")
            return buffer
        if one_byte == b"\r":
            return buffer
        else:
            buffer += one_byte.decode(errors='ignore') # converte para utf-8


def read_ch(AA: bytes, N: bytes, ser):  # Lê o sinal do módulo AA canal N
    command = b"#" + AA + N + b"\r"  # command in hex array
    ser.write(command)
    data = read_data(ser)  # lê resposta
    return data


def write_ch(AA: bytes, N: bytes, data: bytes, ser):  # escreve dados no módulo AA c
anal N
    command = b"#" + AA + N + b"+" + data + b"\r"  # command in bytes array
    ser.write(command)  # exemplo de comando #AAN20.000, coloca 20mA de saída
    confirm = read_data(ser)
    return confirm
```

```
# códigos disponíveis em https://github.com/Neryfoot/TCC
# servidor OPC UA
import serial
import time
import threading
from multiprocessing import Process, Value
from opcua import ua, Server
import DCON
import re

url = "opc.tcp://143.54.96.127:2194" # endereço e porta do servidor
ser = serial.Serial('/dev/ttyUSB0', baudrate=9600, timeout=1.0, write_timeout=1.0)
# abre a prota serial
stop_flag = 0

server = Server() # cria servidor
server.set_endpoint(url) # define url
name = "OPC UA DEQUI"
addspace = server.register_namespace(name) # nome do servidor
server.set_security_policy([ua.SecurityPolicyType.Basic256Sha256_SignAndEncrypt]) #
criptografia
server.load_certificate("certificate.der") # certificado
server.load_private_key("key.pem") #chave

node = server.get_objects_node() # nó de objetos

lvl = node.add_object(addspace, "Level Sensors") # cria objeto para sensores de níve
l
lvl1 = lvl.add_variable(addspace, "LVL1", 0.0) # adiciona variável de nível
lvl2 = lvl.add_variable(addspace, "LVL2", 0.0) # adiciona variável de nível
lvl3 = lvl.add_variable(addspace, "LVL3", 0.0) # adiciona variável de nível
lvl4 = lvl.add_variable(addspace, "LVL4", 0.0) # adiciona variável de nível
lvl5 = lvl.add_variable(addspace, "LVL5", 0.0) # adiciona variável de nível
lvl6 = lvl.add_variable(addspace, "LVL6", 0.0) # adiciona variável de nível

flow = node.add_object(addspace, "Flow Sensors") # cria objeto para sensores de vazã
o
flow1 = flow.add_variable(addspace, "FLOW1", 0.0) # adiciona variável de vazão
flow2 = flow.add_variable(addspace, "FLOW2", 0.0) # adiciona variável de vazão
flow3 = flow.add_variable(addspace, "FLOW3", 0.0) # adiciona variável de vazão
flow4 = flow.add_variable(addspace, "FLOW4", 0.0) # adiciona variável de vazão

pump = node.add_object(addspace, "Pump Actuators") # cria objeto para bombas
pump1 = pump.add_variable(addspace, "PUMP1", 0.0) # adiciona variável de bomba
pump2 = pump.add_variable(addspace, "PUMP2", 0.0) # adiciona variável de bomba
pump3 = pump.add_variable(addspace, "PUMP3", 0.0) # adiciona variável de bomba
pump4 = pump.add_variable(addspace, "PUMP4", 0.0) # adiciona variável de bomba
pump5 = pump.add_variable(addspace, "PUMP5", 0.0) # adiciona variável de bomba
pump6 = pump.add_variable(addspace, "PUMP6", 0.0) # adiciona variável de bomba
pump1.set_writable() # define bombas como editáveis pelos clientes
pump2.set_writable() # define bombas como editáveis pelos clientes
pump3.set_writable() # define bombas como editáveis pelos clientes
pump4.set_writable() # define bombas como editáveis pelos clientes
pump5.set_writable() # define bombas como editáveis pelos clientes
pump6.set_writable() # define bombas como editáveis pelos clientes

# vetores para utilizar em laço
flows = [flow1, flow2, flow3, flow4]
lvls = [lvl1, lvl2, lvl3, lvl4, lvl5, lvl6]
pumps = [pump1, pump2, pump3, pump4, pump5, pump6]


def update_variable(variable, AA: bytes, N: bytes, ser):
    " função para atualização de variável
    lê o canal do modulo informado e escreve seu valor no servidor"
    data = DCON.read_ch(AA, N, ser)
    data = re.sub("[^0-9^.]", "", data)
    if len(data)>0:
        try:
            data = float(data)
        except:
            print("Skipped variable {}".format(variable))
    variable.set_value(data)
```

```
    print(variable)
    print(variable.get_value())



def communication(ser):
    "laço de comunicação"
    global flows
    global lvls
    for i in range(4):
        update_variable(flows[i], b"02", bytes(str(i), 'ascii'), ser)
    for i in range(6):
        update_variable(lvls[i], b"05", bytes(str(i), 'ascii'), ser)
    for i in range(6):
        data = bytes('{:06.3f}'.format(pumps[i].get_value()), 'ascii')
        DCON.write_ch(b"03", bytes(str(i), 'ascii'), data, ser)
    if stop_flag == 0:
        threading.Timer(1, communication, args=[ser,]).start()


server.start() #inicia o servidor
communication(ser) # inicia laço de comunicação
# stop_flag = 1
# stop_flag = 0
# server.stop()
```

```
# códigos disponíveis em https://github.com/Neryfoot/TCC
# implementação do controlador
import time
import threading
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from multiprocessing import Process, Value
from opcua import ua, Client

url = "opc.tcp://143.54.96.127:2194"

client = Client(url)

client.connect()
# Variáveis
yk_11 = 0
yk_12 = 0
e_11 = 0
e_12 = 0
stop_flag = 0
t_01 = 0
# vetores para armazenar os dados
PV1 = []
MV1 = []
REF1 = []
t1 = []
tread1 = []
twrite1 = []

# informações do servidor
lvl1 = client.get_node("ns=2;i=2")
lvl2 = client.get_node("ns=2;i=3")
lvl3 = client.get_node("ns=2;i=4")
lvl4 = client.get_node("ns=2;i=5")
lvl5 = client.get_node("ns=2;i=6")
lvl6 = client.get_node("ns=2;i=7")

flow1 = client.get_node("ns=2;i=9")
flow2 = client.get_node("ns=2;i=10")
flow3 = client.get_node("ns=2;i=11")
flow4 = client.get_node("ns=2;i=12")

pump1 = client.get_node("ns=2;i=14")
pump2 = client.get_node("ns=2;i=15")
pump3 = client.get_node("ns=2;i=16")
pump4 = client.get_node("ns=2;i=17")
pump5 = client.get_node("ns=2;i=18")
pump6 = client.get_node("ns=2;i=19")

ref1 = Value('f', 20) # referência
h1 = Value ('f',0) # nível do tanque 1
u1 = Value('f', 0) # sinal de controle do tanque 1


def controller1():
    global yk_11
    global yk_12
    global e_11
    global e_12
    global t_01, t1
    global PV1, MV1
    t_start = time.time()
    h = lvl1.get_value() # leitura em mA
    t_end = time.time()
    tread = t_end - t_start
    # h1.value = h
    h1.value = 0.0078*((h)**3) - 0.2790*((h)**2) + 4.3687*((h)**1) -12.724 # convert
e em cm
    print("referência {}".format(ref1.value))
    print("nivel {}".format(h1.value))
    e = ref1.value - h1.value
    yk = 27.5*e - 52.25*e_11 + 25*e_12 + 1*yk_11 - 5.54e-05*yk_12
```

```
    if yk > 20:
        yk = 20
    if yk < 0:
        yk = 0
    # atualiza as variáveis
    e_12 = e_11
    e_11 = e
    yk_12 = yk_11
    yk_11 = yk
    t_start = time.time()
    pump1.set_value(float(yk))
    t_end = time.time()
    twrite = t_end - t_start
    print("sinal de controle {}".format(yk))
    PV1.append(h1.value)
    MV1.append(yk)
    REF1.append(ref1.value)
    t = t_start - t_01
    t1.append(t)
    tread1.append(tread)
    twrite1.append(twrite)
    if stop_flag == 0:
        threading.Timer(1, controller1).start()

ref1.value=10
t_01 = time.time()
controller1()
# stop_flag = 1
# stop_flag = 0
ref1.value = 15

client.disconnect()

# salva resultados
results1 = np.vstack((t1, PV1, MV1, tread1))
results1 = results1.transpose()
np.savetxt('results1.csv', results1, delimiter=',')
np.savetxt('t1.txt', t1, delimiter=',')
np.savetxt('PV1.txt', PV1, delimiter=',')
np.savetxt('MV1.txt', MV1, delimiter=',')
np.savetxt('tread1.txt', tread1, delimiter=',')
np.savetxt('twrite1.txt', twrite1, delimiter=',')

# plota e salva figuras
fig, ax = plt.subplots()
ax.plot(t1, MV1)

ax.set(xlabel='tempo (s)', ylabel='sinal de controle em mA', title='MV')
ax.grid()

fig.savefig("MV1.png")
plt.show()

fig, ax = plt.subplots()
ax.plot(t1, PV1)

ax.set(xlabel='tempo (s)', ylabel='nível do tanque em cm',
       title='PV')
ax.grid()

fig.savefig("PV1.png")
plt.show()
```

```
# códigos disponíveis em https://github.com/Neryfoot/TCC
# Simulação numérica da planta

import time
from multiprocessing import Process, Value
import serial
import numpy as np

h = 0.1  # passo da solução numérica
yk_1 = 0  # condição inicial
stop_flag = 0 # flag para parar os processos

# Tanque 1
h1 = Value('f', 0)  # condição inicial de nivel
u1 = Value('f', 0.1)  # fluxo de entrada
f1 = Value('f', 0) # vazão de entrada
# Tanque 2
h2 = Value('f', 0)  # condição inicial de nivel
u2 = Value('f', 3.5)  # fluxo de entrada
f2 = Value('f', 1) # vazão de entrada
# Tanque 3
h3 = Value('f', 0)  # condição inicial de nivel
u3 = Value('f', 3.0)  # fluxo de entrada
f3 = Value('f', 2) # vazão de entrada
# Tanque 4
h4 = Value('f', 0)  # condição inicial de nivel
u4 = Value('f', 2.5)  # fluxo de entrada
f4 = Value('f', 3) # vazão de entrada

# Tanque 5 e 6 não têm medidor de vazão
h5 = Value('f', 0)  # condição inicial de nivel
u5 = Value('f', 2.0)  # fluxo de entrada
h6 = Value('f', 0)  # condição inicial de nivel
u6 = Value('f', 1.0)  # fluxo de entrada


ser = serial.Serial('/dev/ttyUSB0', 9600, timeout=1)  # Abre a porta serial


def read_data(ser):  # lê bytes até ler \r(carriage return)
    buffer = ""
    while True:
        one_byte = ser.read(1)
            return buffer
        else:
            buffer += one_byte.decode() # Converte para string


def communication(ser):  # rotina de comunicação
    buffer = b""
    while stop_flag == 0:
        one_byte = ser.read(1)
        if one_byte == b"\r":
            if buffer[0:3] == b"#02":
                device02(buffer, ser)
                buffer = b""
            if buffer[0:3] == b"#03":
                device03(buffer, ser)
                buffer = b""
            if buffer[0:3] == b"#05":
                device05(buffer, ser)
                buffer = b""
        else:
            buffer += one_byte


def device02(buffer, ser):  # simula dispositivo de comunicação
    case = int(chr(buffer[3]))
    if case == 0:
        data = b">" + bytes(str(f1.value), "ascii") + b"\r"
        ser.write(data)
    elif case == 1:
        data = b">" + bytes(str(f2.value), "ascii") + b"\r"
```

```
            ser.write(data)
        elif case == 2:
            data = b">" + bytes(str(f3.value), "ascii") + b"\r"
            ser.write(data)
        elif case == 3:
            data = b">" + bytes(str(f4.value), "ascii") + b"\r"
            ser.write(data)


def device05(buffer, ser):  # simula dispositivo de comunicação
    case = int(chr(buffer[3]))
    if case == 0:
        data = b">" + bytes(str(h1.value), "ascii") + b"\r"
        ser.write(data)
    elif case == 1:
        data = b">" + bytes(str(h2.value), "ascii") + b"\r"
        ser.write(data)
    elif case == 2:
        data = b">" + bytes(str(h3.value), "ascii") + b"\r"
        ser.write(data)
    elif case == 3:
        data = b">" + bytes(str(h4.value), "ascii") + b"\r"
        ser.write(data)
    elif case == 4:
        data = b">" + bytes(str(h5.value), "ascii") + b"\r"
        ser.write(data)
    elif case == 5:
        data = b">" + bytes(str(h6.value), "ascii") + b"\r"
        ser.write(data)



def device03(buffer, ser):  # simula dispositivo de comunicação
    case = int(chr(buffer[3]))
    data = buffer
    data = data[5:]
    data = data[:-1]
    data = float(data) #retirou caracteres do protocolo
    if case == 0:
        u1.value = data
    elif case == 1:
        u2.value = data
    elif case == 2:
        u3.value = data
    elif case == 3:
        u4.value = data
    elif case == 4:
        u5.value = data
    elif case == 5:
        u6.value = data
    ser.write(b">\r")


def dynamic1(): # simulação numérica
    global yk_1
    global h
    yk = h1.value
    steps = 0  # condição inicial
    tn_1 = float("{:.1f}".format(time.time()))  # condição inicial
    while stop_flag == 0:
        tn = float("{:.1f}".format(time.time()))  # tempo atual
        steps = (tn - tn_1)//h  # número de passos
        # print(steps)
        tn_1 = tn_1 + steps*h  # atualiza o tempo
        if steps > 0:
            while(steps > 0):  # executa atualização da função
                # yk = 0.0025*u1.value + 0.9973*yk_1
                yk = 0.0002128*u1.value + 0.99973*yk_1
                if yk < 0:  # nível não pode baixar de 0
                    yk = 0
                if yk > 20:  # nível não passa de 20
                    yk = 20
                yk_1 = yk
```

```
                steps -= 1
        h1.value = yk


dynamic_process1 = Process(target=dynamic1)
com_process = Process(target=communication, args=(ser,))
com_process.start()
dynamic_process1.start()
# stop_flag = 1
# stop_flag = 0
# dynamic_process1.terminate()
# com_process.terminate()
```

ANEXO A - ESPECIFICAÇÕES DA RASPBERRY PI MODELO 3 B+

Specifications

Processor:	Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz
Memory:	1GB LPDDR2 SDRAM
Connectivity:	■ 2.4GHz and 5GHz IEEE 802.11.b/g/n/ac wireless LAN, Bluetooth 4.2, BLE ■ Gigabit Ethernet over USB 2.0 (maximum throughput 300Mbps) ■ 4 × USB 2.0 ports
Access:	Extended 40-pin GPIO header
Video & sound:	■ 1 × full size HDMI ■ MIPI DSI display port ■ MIPI CSI camera port ■ 4 pole stereo output and composite video port
Multimedia:	H.264, MPEG-4 decode (1080p30); H.264 encode (1080p30); OpenGL ES 1.1, 2.0 graphics
SD card support:	Micro SD format for loading operating system and data storage
Input power:	■ 5V/2.5A DC via micro USB connector ■ 5V DC via GPIO header ■ Power over Ethernet (PoE)–enabled (requires separate PoE HAT)
Environment:	Operating temperature, 0–50°C
Compliance:	For a full list of local and regional product approvals, please visit www.raspberrypi.org/products/raspberry-pi-3-model-b+
Production lifetime:	The Raspberry Pi 3 Model B+ will remain in production until at least January 2023.