RAFAEL GUIMARÃES LUCENA

Dissertação de Mestrado apresentada ao Programa de Pós-graduação em Engenharia Industrial, da Universidade Federal da Bahia, como parte dos requisitos necessários à obtenção do título de Mestre em Engenharia Industrial.

Orientador: DANIEL DINIZ SANTANA

Salvador

Dezembro de 2020

S2320 LUCENA, RAFAEL GUIMARÃES

/RAFAEL GUIMARÃES LUCENA. – Salvador: UFBA, 2020.

XVII, 50 p.: il.; 29,7cm.

Orientador: DANIEL DINIZ SANTANA

Dissertação (mestrado) – UFBA/Programa de Engenharia Industrial, 2020.

Referências: p. 45 – 45.

1. Sistema Supervisório. 2. Python. 3. Sistema Open Source. 4. PyQt. I. SANTANA, DANIEL DINIZ. II. Universidade Federal da Bahia, Programa de Engenharia Industrial. III. Título.

CDD: 511

Dedicatória

Agradecimentos

Agradecimentos

Death is a promise, and your life is a lie

Oliver Sykes

I don't get mad, I get even

Dave Mustaine

Resumo da Dissertação apresentada à UFBA como parte dos requisitos necessários para a obtenção do grau de Mestre em Ciências (M.Sc.)

RESUMO

Este trabalho apresenta, dentro do âmbito acadêmico, o desenvolvimento de um sistema supervisório ou SCADA construído em linguagem Python. A proposta surgiu como uma alternativa a tecnologias existentes, e tenta sanar problemas delas advindos, como custo, incompatibilidade e obsolência. Além disto, visa fomentar o uso de tecnologias gratuitas na universidade. O objetivo foi de construir um programa que, além de plotar em tempo real sinais recebidos por comunicação serial, salve estas séries de dados e possibilite outros métodos de entrada de dados. Além disto, o processo de criação do sistema foi descrito detalhadamente, tornando este documento uma possível referência para outros trabalhos de construção de interface com Python. Os resultados apresentados validam o funcionamento da ferramenta e trazem resultados coerentes.

Palavras-chave. Python, SCADA, sistema supervisório, open-source

Abstract of Dissertation presented to PEI/UFBA as a partial fulfillment of the requirements for the degree of Master of Science (M.Sc.)

abstract

Keywords. Python, SCADA, open-source

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Capítulo 1

Introdução

1.1 Contextualização

O monitoramento remoto de processos é uma área da tecnologia surgida no século XX, quando os sistemas de controle passaram a ser aplicados na indústria. As interfaces homem-máquina passadas se baseavam em relés que acendiam lâmpadas e quadros sinóticos, representando alertas e mostradores. Como avanço dos meios de transmissão de dados, displays digitais e comunicação entre diferentes sistemas, os computadores se modernizaramo suficiente para tomar o espaço industrial, no que diz respeito ao controle e monitoramento de processos. (JUNIOR, 2019)

Neste âmbito da modernização industrial, surgiram os sistemas SCADA (*Supervisory Control And Data Acquisition*), também chamados de sistemas supervisórios. Geralmente são executados em computadores comuns e se comunicam com outros dispositivos eletrônicos via protocolos de comunicação como Modbus e Profibus. Com isso, podem enviar e receber dados de controladores de uma área industrial, e mostrar variáveis de processo em um monitor através de diversos recursos visuais, como gráficos, símbolos e tabelas. Esta dinâmica facilitam a um operador entender rapidamente o estado de suas máquinas e identificar tendências de falhas.

A grande vantagem do emprego destes softwares é a centralização da supervisão de uma forma muitas vezes mais barata que mostradores físicos espalhados numa planta, pois ocorre em um lugar só. Anos atrás, o monitoramento era local, e os operadores precisavem se deslocar a uma máquina para checar seu funcionamento. A comunicação entre o supervisório e controladores permite ainda que parâmetros intrínsecos ao processo sejam configurados dinâmicamente, a exemplo dos setpoints ou receitas.

Sistemas SCADA costumam vir já com interação programada com serviços de bancos de dados, através de drivers de comunicação dos fabricantes mais populares, como Oracle e SQL Server. Desta forma, o programador do sistema não utiliza de grandes conhecimentos de bancos de dados para criar um histórico de processo.

Segundo (JUNIOR, 2019), o valor de um sistema SCADA está relacionado com o conceito de software aberto. Resumidamente, ele deve ser o mais compatível possível com os hardwares com os quais se comunicaria e com os diferentes sistemas operacionais que podem executá-lo. Além disto, deve ser modular, executado em módulos, de forma que o mal-funcionamento de uma de suas partes não impacte negativamente a execução dos outros. Por fim, deve também ser escalável, e permitir a agregação de novos equipamentos e novas funcionalidades.

Frente à sua importância na área da automação, aspirantes a engenheiros devem se familiarizar com esta tecnologia. Atualmente existem ferramentas gratuitas focadas neste propósito, geralmente aplicadas no meio acadêmico. Softwares open source são gratuitos, e seus códigos-fontes podem ser compartilhados, modificados e aplicados à vontade. Normalmente são projetados para aplicações pequenas, mas a cultura open source tem se espalhado consideravelmente no meio tecnológico, e ganhado robustez, havendo inclusive empresas que fazem uso de licenças gratuitas.

O software livre é algo extremamente útil para o ensino de novas tecnologias, uma vez que, permite que todos tenham acesso ao conhecimento científico. Nas instituições de ensino superior, o software livre é uma das bases formadoras de conhecimento, pois permite que o aluno continue a desenvolver suas atividades em seu próprio computador pessoal, não impedido por recursos monetários ou dificuldades de acesso à informação. (SILVA et al., 2013)

1.2 Problema e Justificativa

Frente à importância de sistemas supervisórios na Automação e Controle de processos, e dos altos custos de licenças de alguns softwares como MATLab® e SIMATIC WinCC, surgiu a ideia da construção de uma plataforma que utilize uma linguagem gratuita e open-source e sirva de alternativa a aplicações deste ínterim. A mesma seria utilizada para comunicar-se com qualquer controlador conectado por porta serial

ao computador operante. Ele registraria em forma de gráficos variáveis inerentes a sistemas mecânicos, elétricos ou quaisquer outros, monitorados por sensores conectados a tal controlador, ou modelados por funções de transferências. Além disto, teria código livre e aberto, tanto para o estudo e aprendizado dos estudantes da universidade, como para futuras melhorias e inclusão de novas funcionalidades.

1.3 Objetivos

1.3.1 Objetivos Gerais

Desenvolver em linguagem Python uma GUI (Graphical User Interface) capaz de se comunicar com dispositivos externos que lhe fornecessem dados numéricos para plotagem em sua interface, de forma que seja possível a análise de respostas de sistemas diversos e comparação com modelagens formuladas pelos usuários deste sistema.

1.3.2 Objetivos Específicos

Com o foco nos objetivos supracitados, foram levantados os seguintes pequenos milestones a serem alcançados no decorrer do desenvolvimento do sistema-alvo:

- Definir os requisitos de comunicação
- Projetar a(s) tela(s) do sistema
- Programar os eventos que coordenam o funcionamento do sistema
- Configurar a comunicação com dispositivos externos (preferencialmente tomando o controlador Arduino como base)

1.4 Estrutura do Trabalho

Este trabalho foi divididos em:

1. No capítulo 1 o tema principal foi contextualizado de acordo com a tecnologia atual, o problema a ser solucionado foi apresentado, e os objetivos foram esclarecidos e segmentados em um passo-a-passo.

2. No capítulo 2 consta uma fundamentação teórica relativa aos assuntos que cercam o tema principal deste trabalho, a iniciar por uma descrição curta dos softwares SCADA existentes, das tecnologias que podem ser utilizadas para a criação de uma GUI e finalmente um resumo da linguagem selecionada para a construção da interface

3. O capítulo 3 trata da programação da plataforma em si, em linguagem Python, mostrando sua arquitetura e convenções empregadas com figuras e esquemas.

4. O capítulo 4 toma o sistema pronto e o emprega em uma aplicação real, com o intuito de validar seu funcionamento e exemplificar como o mesmo pode contribuir no ambiente acadêmico da universidade.

5. No capítulo 5 este trabalho se encerra em forma de um texto conclusivo, onde, entre outros assuntos, são abordadas possíveis futuras melhorias ao sistema criado.

Capítulo 2

Fundamentação Teórica

2.1 Modelagem de Processos

A dinâmica de muitos sistemas mecânicos, elétricos, térmicos, econômicos, biológicos ou outros pode ser descrita em termos de equações diferenciais. Estas equações são obtidas pelas leis físicas que regem dado sistema, por exemplo, as leis de Newton para sistemas mecânicos e as leis de Kirchhoff para sistemas elétricos. Um só modelo matemático, no entanto, não é o único para determinado sistema, pois ele pode ser representado por muitas maneiras diferentes e, portanto, por vários modelos matemáticos, dependendo da perspectiva a ser considerada. (KATSUHIRO, 2010)

Uma das formas mais comuns de representação de um sistema genérico é por funções de transferências. Sendo $y(t)$ a função que descreve uma saída de um processo, e $x(t)$ a de uma entrada, a função de transferência $G(s)$ traduz a influência de x em y. Assim, a função $G(s)$ advém das equações físicas descritivas de um processo e, de maneira geral se torna mais complexa quanto mais equações e parâmetros influentes são considerados. Como um exemplo, ao modelar a queda de um corpo livre, tomando como variável de saída sua posição, uma função de transferência simples consideraria apenas a influência da gravidade sobre o corpo, e a mesma pode se tornar mais complexa se considerasse o atrito e resistência com o ar.

Um processo, após modelado, pode se traduzir em um sistema linear ou não linear. Um sistema é dito linear se o princípio da superposição se aplicar a ele. Este princípio afirma que a resposta produzida pela perturbação simultânea de duas entradas é a soma das duas respostas individuais para cada entrada. A partir deste, princípio, é possível calcular soluções complexas a partir do cálculo de cada parte que a compõe. (KATSUHIRO, 2010)

Sistemas não lineares são, em geral, mais difíceis de modelar e de controlar. Para tais, podem ser empregadas algumas técnicas de trasformações em sistemas lineares equivalentes. Uma delas é a chamada linearização, que emprega a série de Taylor, truncanda no segundo termo, ou

$$f(x1, x2, .., xn) \simeq f(x1_0, x2_0, ..., x3_0) + \left(\sum_{i=1}^{n} \frac{df}{dxi}\Big|_{xi=xi_0} (xi - xi_0) \right), \qquad (2.1)$$

, obtendo assim, uma função linearizada em torno de um determinado ponto de operação do processo (x_i são os parâmetros da função descritiva f, e x_{i0} são os pontos de operação). Quanto mais as variáveis do sistema linearizado se afastarem deste ponto de operação, maior será o erro deste sistema, em relação ao sistema gerador não linear. Outra técnica é simplesmente partir o sistema não linear com uma entrada qualquer, e pela análise do gráfico de resposta projetar um sistema linear que seja o mais fiél possível ao primeiro.

Talvez o maior benefício da modelagem de processos para o setor industrial seja a possibilidade de projetar sistemas de controle mais eficientes, eliminando a necessidade de gastar tempo e dinheiro com testes em campo. Softwares como MATLAB e GNU Octave permitem que, a partir de funções modeladoras (no tempo ou no domínio s), um processo seja parametrizado, e seu comportamento, dada entradas também configuradas, seja simulado.

2.2 SCADA

Os sistemas supervisórios podem ser considerados como o nível mais alto de IHM, pois mostram o que está acontecendo no processo e permitem ainda que se atue neste. A evolução dos equipamentos industriais, com a introdução crescente de sistemas de automação industrial, tornou complexa a tarefa de monitorar, controlar e gerenciar esses sistemas. (MARTINS, 2007)

Sistemas SCADAs são responsáveis por buscar informações de controladores e equipamentos diversos de automação e manipular estas informações de diversas maneiras. As aplicações mais simples se constituem na visualização dinâmica destes dados através de objetos, mostradores, cores, entre outros meios dispostos em telas pré programadas. Uma estratégia, por exemplo, é representar todo um sistema por

imagens já embutidas na biblioteca do software editor, e incluir o máximo de informações importantes referentes ao mesmo em uma única tela detalhada. Outra seria agrupar as informações por temática, e mostrar diversas telas menos detalhadas, que alternem entre si de acordo com um temporizador interno. A Figura 2.1 ilustra um exemplo de tela para um sistema supervisório.

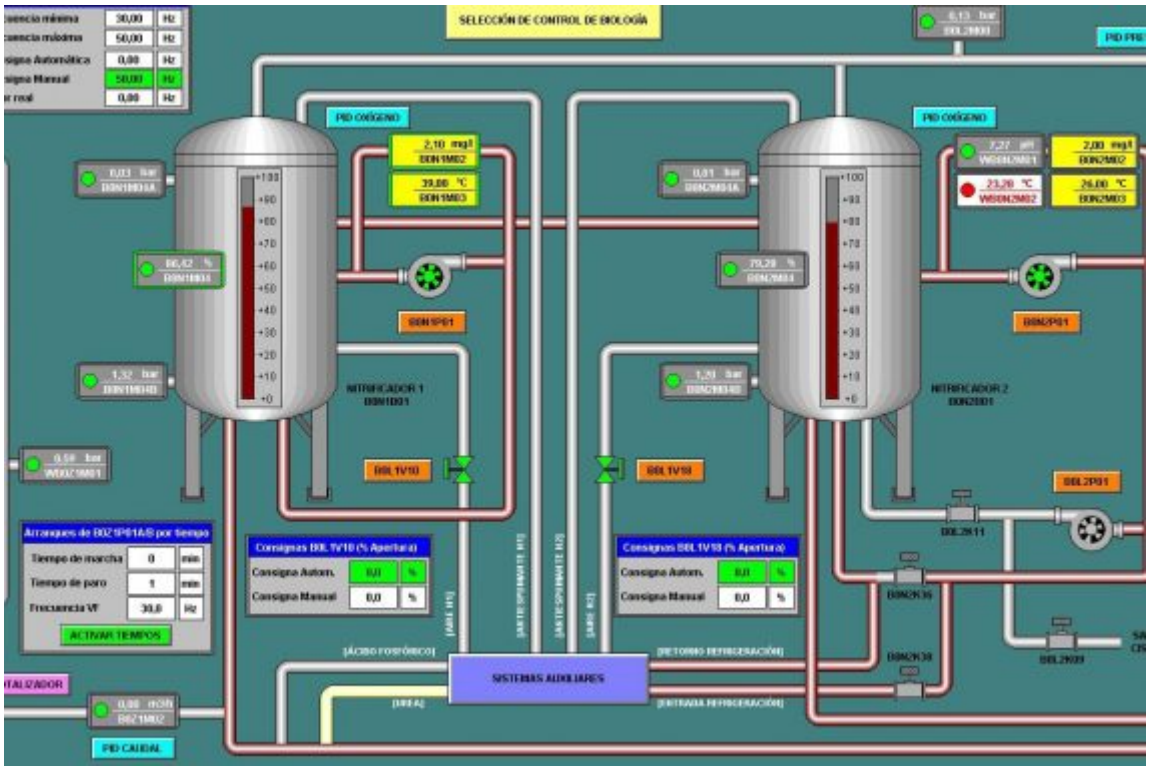

Figura 2.1: Tela exemplo de um sistema supervisório

Outro emprego comum de sistemas SCADA, é de serem responsáveis por informar valores de setpoint aos controladores acoplados a um processo, os quais podem advir de um cálculo computacional ou manual, por um operador. É possível ainda o sistema ser responsável pelo controle, enviando aos atuadores conectados somente o sinal de controle. Esta estratégia, no entanto, não é recomendada, por questões de confiabilidade da transmissão de dados e velocidade de processamento.

Por ser executado geralmente em um computador comum, a palavra chave de um sistema supervisório é flexibilidade. Um sistema SCADA deve ser capaz de se comunicar por diversos protocolos com diversos dispositivos, e adicionalmente disponibilizar os valores lidos para outros usuários, não somente os que têm acesso às telas. Por possuir esta funcionalidade, software SCADAs ultrapassam o nível 2 na pirâmide de automação (Figura 2.2), chegando ao nível de supervisão da produção,

pois agrupa as informações de diversos controladores e sensores em um só local, gerando suporte para ações de gerência.

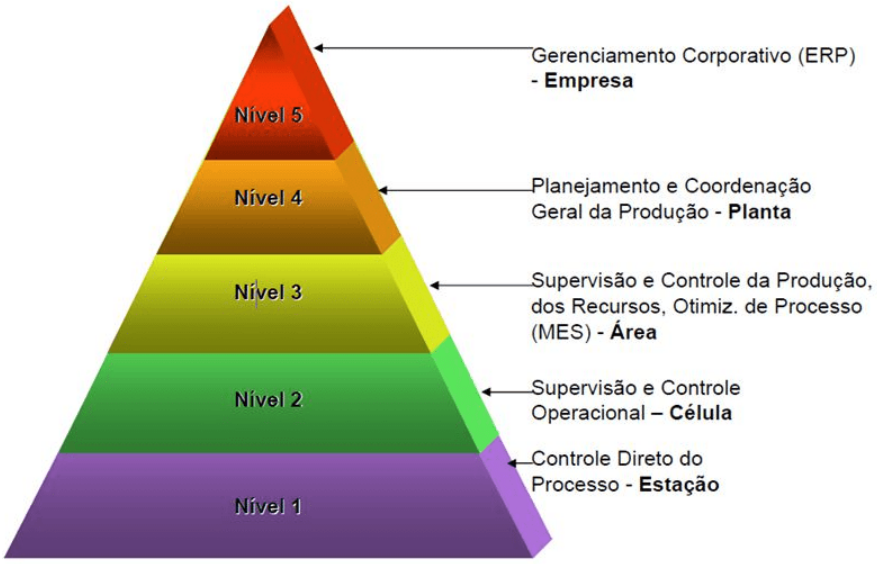

Figura 2.2: Pirâmide da automação

A fim de atingir uma maior compatibilidade com programas e usuários externos, a maior parte dos softwares supervisórios permitem de forma simples a criação de um banco de dados para os valores monitorados. Os mesmos podem ser exportados em forma de relatórios em layouts já embutidos e utilizados como insumos para tomada de decisões e cálculo de performance.

Segundo (ROGGIA; FUENTES, 2016), dentre os principais benefícios do uso de sistemas de supervisão podem-se citar: informações instantâneas, redução no tempo de produção, redução no custo de produção, precisão das informações, detecção de falhas, aumento da qualidade e aumento da produtividade.

2.3 Comunicação Serial

Comunicação serial é um meio simples de dois equipamentos trocarem informação em formato de uma série de bits. De acordo com (BOLTON, 2015), ela se dá através de um único cabo ou pino, transmitindo apenas um bit de cada vez. Apesar disto significar uma velocidade menor de envio de dados, a comunicação serial possui baixo custo, sendo popularmente utilizada na transmissão de dados por longas distâncias.Ainda existem outras tecnologias, como a chamada comunicação paralela,

que utilizam mais canais de comunicação e podem portanto transmitir vários bits simultaneamente.

Segundo (MAZIDI; MCKINLAY; CAUSEY, 2008), existem dois tipos de comunicação serial: assíncrona ou síncrona. Como sugerido por seus nomes, a comunicação síncrona transmite blocos de tamanhos definidos, em momentos definidos, enquanto que o outro tipo transmite bytes de dados em qualquer momento. Para contornar a necessidade de escrever trechos de código que lidem como os dois casos, muitos fabricantes utilizam chips de circuitos integrados que manipulam o fluxo de dados, facilitando a escrita de scripts de comunicação.

Diversos equipamentos eletrônicos utilizam a comunicação serial, como mouses e teclados. O conhecido controlador Arduino UNO também permite a troca de bits por meio de suas portas seriais, de números 0 ou 1, ou uma mais comumente utilizada entrada USB ((ARDUINO..., 2019)). Ela também está presente em alguns protocolos de comunicação modernos, como Ethernet e Profibus.

Como ilustrado na figura 2.3, e fundamentado por (MAZIDI; MCKINLAY; CAUSEY, 2008), um modelo de transmissão para bytes (8 bits) de informação pela porta serial se constituiria em uma onda digital cujos formato se traduz em um bits 1 ou 0. O primeiro bit marca o início da transmissão, seguido por 8 bits relativos à informação enviada, um bit opcional de paridade (acrescentado por fins de validação de dados), e um último bit que encerra o bloco de informação. Protocolos de comunicação diferentes podem acrescentar ou remover características à sequência transmitida, seja para assegurar a integridade da informação, acrescentar outras informações na cadeia, ou para aumentar a velocidade de comunicação.

Por comunicar dois equipamentos distintos, com diferentes arquiteturas, se fazem necessárias medidas que contornam a diferença entre os clocks de ambos, em outras palavras, a diferença entre a velocidade de transmissão e a de leitura dos bits recebidos. Uma delas é a utilização de um buffer, um espaço de memória na porta receptora, que guarde rapidamente os bits transmitidos, para posterior leitura e tratamento dos mesmos por parte do processador da máquina. Quando este buffer está próximo de encher, o receptor pode fechar o barramento serial via hardware ou software, para impedir a perda de informação durante sua transmissão.

Em seu artigo, (DENVER, 1995) sugere o emprego de threads para contornar

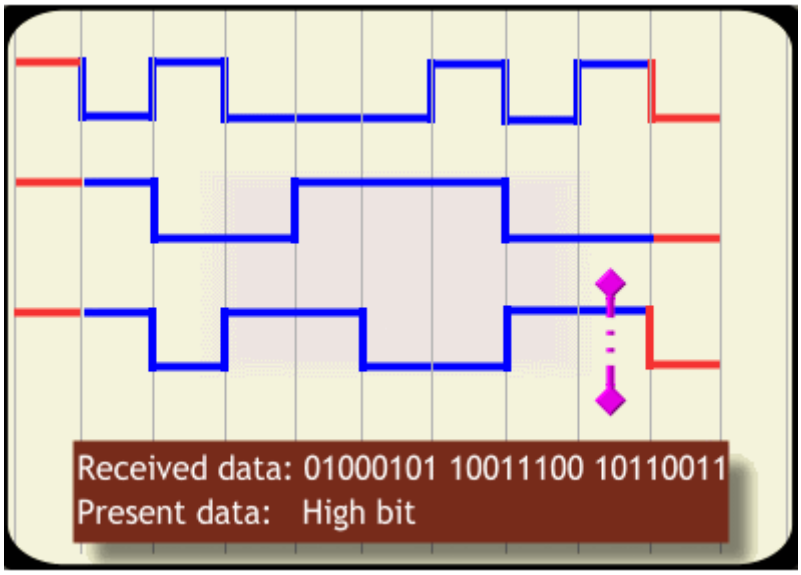

Figura 2.3: Exemplo de transmissão serial de uma sequência de 3 bytes

possíveis problemas na comunicação serial, como a espera para receber valores. Neste caso, threads impediriam que toda uma aplicação parasse até que a transmissão deste valor seja completada.

2.4 Qt em Python

Segundo seu criador, ??, Python foi criada no início da década de 90, com influências de outra linguagem na qual trabalhara, chamada ABC. O objetivo do projeto ABC era de criar uma linguagem que pudesse ser ensinada à usuários inteligentes de computadores, mas que não eram programadores nem desenvolvedores de softwares. Após ingressar em outro projeto, surgiu a necessidade de implementação de outra linguagem, onde Rossum teve a iniciativa de criar o Pyhton: uma linguagem simples e escalável, que permitisse a contribuição de terceiros.

Se tratando de sua arquitetura, Python é uma linguagem de alto nível, orientada a objetos e com tipagem dinâmica e forte. As definições de escopo e blocos de código são representadas por indentações, o que torna o código mais organizado e visualmente aprazível, dispensando a utilização de chaves para delimitar escopo. Além disto, permite interoperabilidade com outras linguagens. Por exemplo, utilizando a ferramenta Cython é possível, a partir de um código Python, gerar um código equivalente em C. Existem funções, inclusive, que são desenvolvidas em C, a fim de agilizar o processamento de grandes bases de dados, mas implementadas em

Python.

No âmbito acadêmico, Python apresenta boas vantagens. Não só é considerada simples fácil de aprender, como é gratuita e open source. Logo, seus usuários e clientes não têm custos com licenças e seus desenvolvedores podem usar livremente códigos publicados por terceiros, que geralmente se apresentam de fácil acesso na internet, e adaptá-los às suas necessidades. Em uma enquete realizada pelo conhecido fórum da comunidade de computação Stack Overflow, Python foi considerada a 3ª "linguagem mais amada" pelo público.

Pela facilidade de compartilhamento e comunidade crescente de usuários, existem diversas bibliotecas úteis de Python que podem ser baixadas diretamente de um repositório online e facilmente instaladas. Como alguns exemplo, cita-se as libs SQLAlchemy, que permite criação e acesso a bancos de dados leves; NumPy, uma poderosa ferramenta para cálculos matriciais; e PyQt5, que possui objetos e métodos para criação de interfaces gráficas.

Segundo sua documentação (LIMITED, 2020), a biblioteca PyQt5 veio da biblioteca de C++ "Qt", que implementa APIs para outras linguagens, permitindo-as implementar seus objetos em seus códigos. No seu site oficial, existe uma documentação extensa de todos os seus objetos em C++, e existem também muitos exemplos disponíveis online, tanto em sites oficiais do Qt como em fóruns de programadores.

Existem 3 módulos do PyQt julgados principais para criação de GUIs locais:

- QtWidgets: Engloba os objetos gráficos principais, como botões, textos e layouts. O objeto genérico QWidget, herdado por diversos outros deste módulo, tem funções cruciais relativas ao posicionamento, geometria, visibilidade e estilo dos objetos gráficos.

- QtCore: Este módulo lida com eventos dos objetos da GUI, como cliques de botões e edições de caixas de texto, e conecta estes eventos com funções definidas pelo programador. Ele também permite que ele configure a aplicação principal e coordena possíveis threads iniciadas por ela.

- QtGui: Contém objetos que possibilitam a edição de cores e bordas dos Widgets e também lida com eventos relacionados à atualização da posição e estética destes objetos.

Para iniciar a GUI, deve ser criado um objeto *QApplication*, que representa o núcleo da aplicação, juntamente com as janelas da mesma, que são objetos *QMainWindow*. Estes aceitam um objeto genérico *QWidget* como Widget central principal (através do método *setCentralWidget()*), cujas características de tamanho e layouts internos definem o tamanho da janela. Inicia-se a aplicação pelo comando *exec_ ()*, e as janelas pelo comando *show()*.

Numa interface criada com PyQt5, os objetos visíveis dispostos na tela são chamado de Widgets, e herdam da classe *QWidget*. Por conta da arquitetura em objetos da biblioteca, uma boa prática para programadores é colocar um Widget dentro do outro, no sentido de design e posicionamento. Assim, é possível definir melhor o espaço e as condições de redimensionamento dos objetos quando a janela for esticada ou comprimida, pois cada layout redimensionará apenas os Widgets que contém, dentro do espaço disponível.

A Figura 2.4 ilustra a organização padrão de uma aplicação em PyQt, listando também alguns objetos populares.

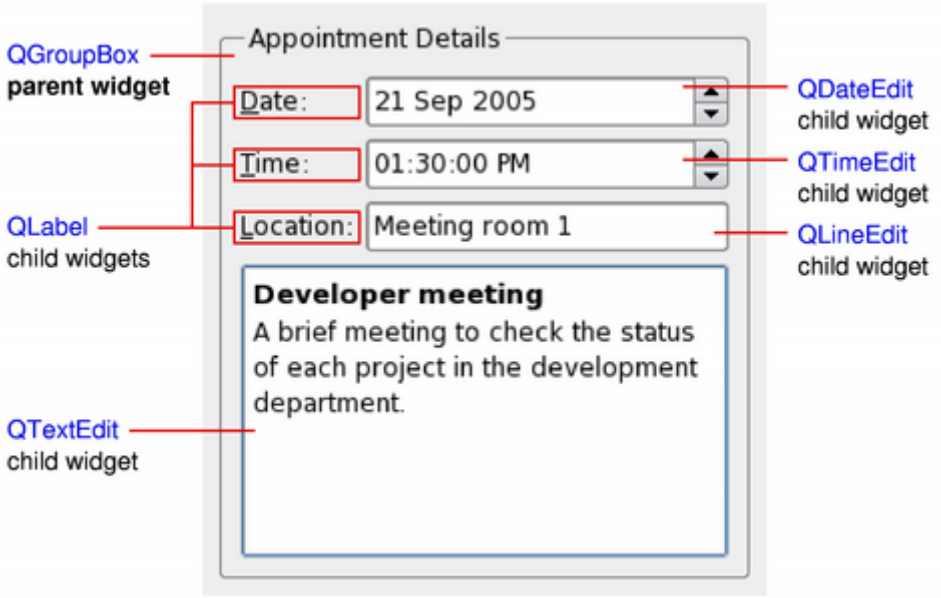

Figura 2.4: Janela *QMainWindow* com um *QGroupBox* como Widget central, contendo vários outros Widgets em um layout *QGridLayout* dentro de outro layout *QHBoxLayout*

Capítulo 3

Desenvolvimento do Sistema Supervisório

3.1 Requisitos do Sistema

Para a escolha da tecnologia empregada, foram levantados os seguintes requisitos do sistema:

- Deve ser gratuito para o desenvolvimento, simples e de código aberto no intuito de permitir análise por partes de interessados e facilitar melhorias e expansões;
- Deve ser capaz de plotar gráficos em tempo real, advindos de no mínimo porta serial. Opcionalmente pode ser compatível com arquivos nos formatos .csv, .tsv, .xls, e .xlsx, ou permitir modelagem direta no software, por funções de transferência
- O software deve rodar no sistema operacional windows (mínimo Windows 7)

3.2 Seleção das Tecnologias

Pelo primeiro requisito, em relação à gratuidade da tecnologia, pensou-se primeiramente em utilizar plataformas abertas para sistemas supervisórios, como o ScadaBR ou a versão demo do Elipse E3. Já houveram trabalhos, inclusive, utilizando a primeira tecnologia (referenciados na bibliografia deste documento).

Como atestado em (MORAES, 2016), o ScadaBR é construído em um servidor web, utilizando geralmente o Apache TomCat (como servidor web). Assim, julgou-se necessário um certo período para acostumar-se com a sua programação, além de

um conhecimento em redes para configuração da comunicação cliente-servidor.

Quanto à versão demo do Elipse E3, apesar do autor já possuir certo domínio da ferramenta, se trata de uma versão muito limitada. Segundo sua base de conhecimento, (ELIPSE..., 2019), a versão demo permite somente até 20 tags de dados e a aplicação roda por um máximo de 2 horas, tendo que ser reiniciada manualmente após este período. Finalmente, não se sabe as implicações legislativas em utilizar o software para trabalhos acadêmicos, nem sua utilização contínua no âmbito da universidade, mesmo se tratando de uma versão de demonstração.

Por cumprir todos os requisitos, e por ser considerado um método inovador, foi decidido construir um sistema SCADA em Python. Desta maneira, o código-fonte da aplicação seria aberto, sua programação não exigiria grande esforço por parte do programador, e este trabalho contribuiria na difusão da implementação de softwares gratuitos e open source no meio acadêmico. Além disto, Python é uma linguagem contemporânea, tendendo a acompanhar o avanço tecnológico. Desta forma o sistema desenvolvido seria compatível com uma vasta gama de tecnologias atuais e futuras.

3.2.1 Seleção das bibliotecas

Como já mencionado, a linguagem Python possui inúmeras bibliotecas, para os mais variados fins. No decorrer da criação do sistema, foram utilizadas as seguintes bibliotecas no Quadro 3.1.

Quadro 3.1: Bibliotecas Python utilizadas no desenvolvimento do supervisório didático

Biblioteca	Descrição
PyQt5	Usada para a construção de GUIs, contém diversos objetos úteis como botões, caixas de texto e rótulos. Também trata do posicionamento e direção destes objeto nas janelas principal e periféricas
matplotlib	Contém ferramentas que permitem a plotagem e design de gráficos variados, inclusive com objetos backend que fazem uma ponte com GUIs construídas com PyQt5
numpy	Biblioteca que lida com operações matriciais e cálculos avançados, com muitas funcionalidades similares ao MatLab. Também é capaz de gerar números aleatórios, que são úteis no teste do programa
pyserial	Permite a conexão com dispositivos externos pela porta serial e contém funções de escrita e leitura desta porta
python-control	Possibilita a criação de sistemas descritos em funções de transferência e espaço de estados, além de gerar a resposta simuladas para alguns formatos comuns de entradas, como degrau e impulso
pickle	Responsável pera serialização de objetos utilizados no código e armazenamento dos mesmos em um arquivo à parte. Lida também com a leitura e decodificação de objetos serializados

3.3 Criando a interface gráfica

A biblioteca PyQt funciona como uma linguagem orientada a objetos. Assim, esta arquitetura foi adotada no desenvolvimento da ferramenta. Foi utilizado uma folha de script que agrupasse a maior parte das classes implementadas, o form_objects.py, e outro script principal, main.py que inicia a execução do programa. Um terceiro script, realtime_objects.py, contempla objetos que rodam em tempo real e espera-se que futuros usuários também editem o código contido, por motivos explanados posteriormente neste documento.

No que concerne a interface gráfica do supervisório, imaginou-se um layout simplista. A aplicação conteria uma área para plotagem de gráficos, uma lista das séries de dados incluídas no programa pelos diversos métodos possíveis, e uma área para incluir séries novas. Os detalhes destes Widgets são listados abaixo, e sua disposição mostradas na Figura 3.1:

- *PlotManager*:representaria uma área de rolagem (QScrollArea) que conteria várias representações gráficas das séries de dados salvos na aplicação, com um

pequeno preview destes dados, e botões para sua plotagem, edição e exclusão da série.

- *MainPlotArea*: utiliza um objeto *FigureCanvas*, da lib matplotlib, para plotagem detalhada das séries selecionadas na lista de séries. O eixo y seria adimensional, dependendo da variável medida e observada, e o eixo x representaria o tempo.

- *DatasetConfig*: tem como Widget principal um seletor com abas (*QTabWidget*), que permitiria a inclusão de séries de dados novas no programa, pelas fontes já mencionadas, sendo cada aba responsável pelas diferentes métodos (serial, arquivo, função de transferência, etc)

- *SCADADialog*: se trata realmente de uma tela de supervisão, com uma área de plotagem atualizada em tempo real. Funciona somente quando os dados são importados por porta serial.

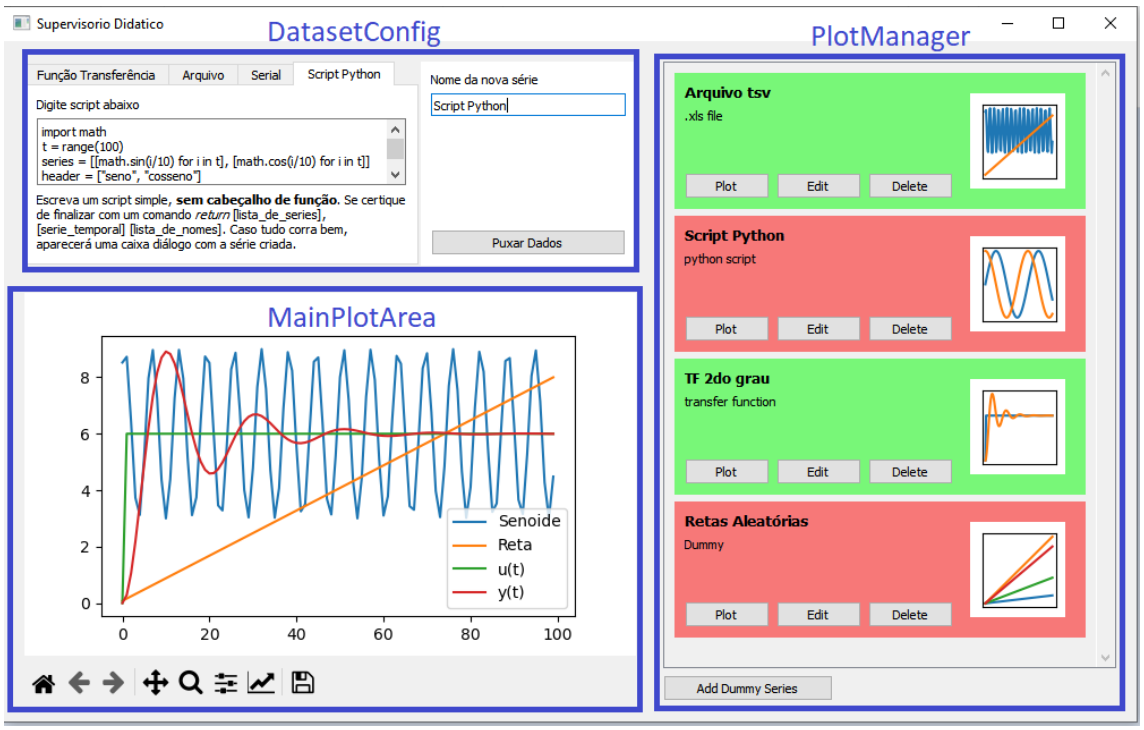

Figura 3.1: Supervisório didático e seus objetos principais

Além dos Widgets supracitados, para simplificar o código e troná-lo mais prático, foi criado um objeto abstrato *SeriesObject*. Ele armazena um agrupamento de série de dados que compartilham um mesmo eixo de tempo. Desta forma, foi mais fácil manipular as referências das séries pelo programa.

3.4 *Dataset Config*

A principal função deste objeto é de interface da aplicação com outros dispositivos ou programas, com a finalidade de puxar séries de dados destas fontes. A segunda função é de formatar estas séries, atribuindo a elas um nome e um cabeçalho, além de definir que legenda aparece no gráfico principal quando a série for plotada.

Na fase de idealização do software, levantou-se possíveis meios para adicionar séries de dados novas no programa. Como se trata de um sistema supervisório, deve haver uma funcionalidade que permita o recebimento de dados externos, no mínimo por comunicação serial, bastante utilizada por controladores didáticos como Arduino e Raspberry Pi. Como adicionais,, seria útil também a importação de séries por arquivos *.xls* ou *.xlsx*. Por último, caso o usuário deseje utilizar funcionalidades não ofertadas pelo software didático, ele pode escrever seu próprio script Python e levar os resultados para o programa.

Na engenharia de Controle e Automação, é comum a prática de modelar um processa antes de observar seu comportamento empírico. Visto isso, o programa deveria oferecer uma maneira simples de simular processos diversos e plotá-los em contraste com o sistema real, proveniente do controlador. Logo, foi incluída biblioteca que simulasse as resposta de funções de transferência, por padrão a uma entrada degrau, e fornecidos os meios para que o usuário encadeie em série variadas funções, de parâmetros quaisquer, pelo objeto *TransferFunctionConfig*.

Resumindo, existem quatro maneiras de importar ou gerar uma série de dados no software desenvolvido:

1. **Por arquivo**, nos formatos *csv* (Comma Separated Values), *tsv* (Tab Separated Values), *xls* (antigo arquivo Excel), *xlsx* (arquivo Excel);

2. **Por função de transferência**, informando o numerador e denominador de cada função, criadas em série e submetidas a uma entrada do tipo degrau, de valor definido pelo usuário;

3. **Por comunicação serial**, configurando alguns parâmetros (porta, baud rate e tempo para timeout), separando cada valor enviado pelo dispositivo por uma tabulação e linhas por quebras de linha;

4. **Por script Python**, escrevendo um código python funcional e retornando os valores das séries na ordem correta (lista dos valores das séries, série de valores de tempo, lista de nomes das séries, nesta ordem)

Caso não haja problemas na importação e as condições de formatação para cada método de entrada forem satisfeitas, o programa abrirá uma caixa diálogo (*QtWidgets.QDialog*) idêntica à da Figura 3.2 com uma preview dos dados e uma tabela com os valores numéricos. Através dele é possível editar cada valor separadamente, o nome da série e o cabeçalho.

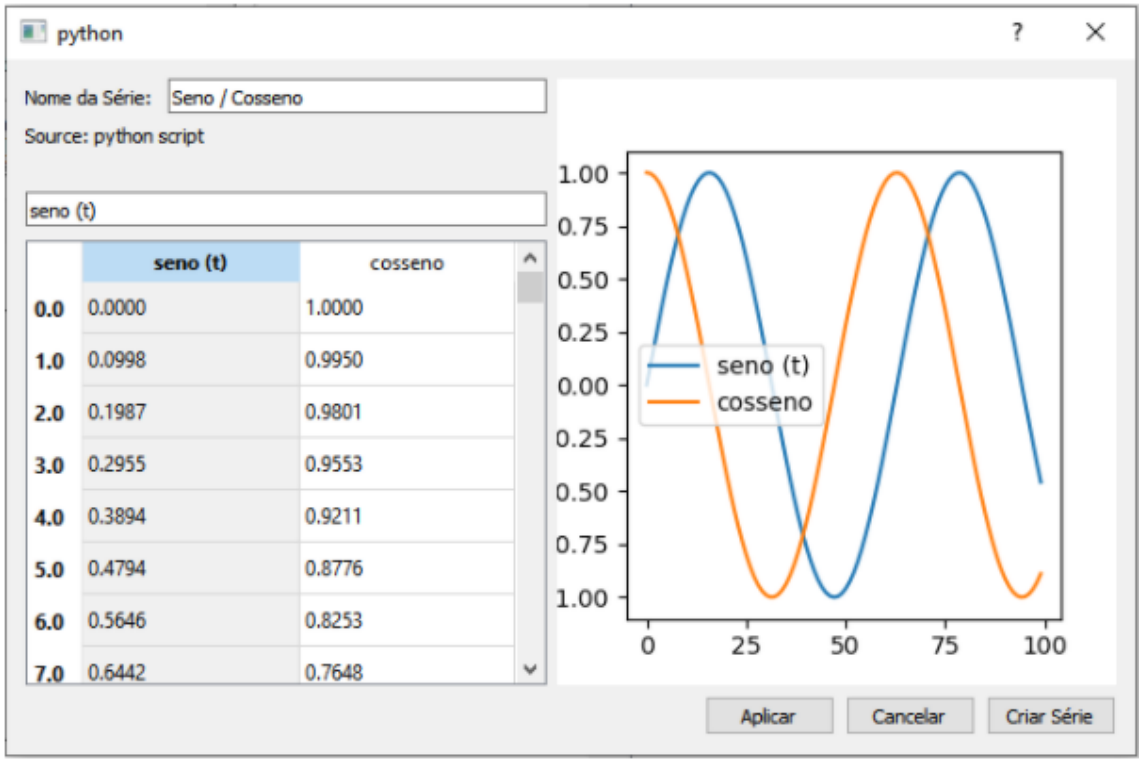

Figura 3.2: *ModelSeriesDialog*: Caixa diálogo para edição dos eixos e título das séries

A caixa diálogo contém um Widget bastante importante para a aplicação: o *FigureCanvas*, proveniente de um módulo da biblioteca matplotlib, que funciona como um plugin para o PyQt. Apesar dele não ser um Widget do PyQt, ele contém, se não todos, uma boa parte de seus módulos e é tratado como tal para fins de posicionamento, geometria e inclusão nos layouts de tela. O *FigureCanvas* faz uma interface com o objeto *Figure*, responsável pela plotagem de gráficos de linhas, barras, etc, e permite que ele seja incluído numa GUI construída em PyQt.

O objeto *Figure*, por sua vez, é bem similar ao objeto de mesmo nome no MATLab®, aceitando métodos como *plot()*, emphadd_subplot() e *clear()*. Sua do-

cumentação completa, juntamente com a de outros objetos relevantes consta no site da biblioteca matplotlib. Quando embutido numa GUI por um *FigureCanvas*, após a plotagem de gráficos e de formatações gerais, o segundo deve chamar o método *draw()* para que seja atualizada a imagem.

3.5 *PlotManager*

No lado direito da aplicação, se encontra uma lista das séries carregadas. Dado o espaço limitado, faz-se necessária uma área de rolagem. Assim, foi criado um objeto *GraphicPlotList* que implementa esta área ao herdar de *QtWidgets.QScrollArea*. Também existia originalmente um botão que criasse uma série de até 4 retas com inclinações aleatórias, para fins de teste.

Quando uma série nova é criada pelo *DatasetConfig*, ela fica armazenada em um objeto *SeriesObject*, que é representado graficamente na lista de séries *GraphicPlotList* por um outro objeto *GraphicPlotConfig* (Figura 3.3).

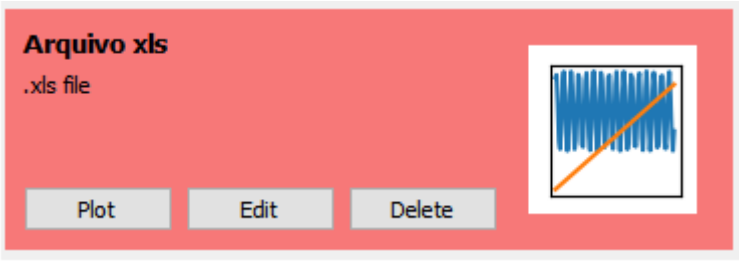

Figura 3.3: *GraphicPlotConfig* não plotado em *MainPlotArea*

A principal função deste objeto é identificar pelo nome e fonte a série que contém, e coordenar quando esta série deve ser plotada na área principal *MainPlotArea*, além de permitir sua edição ou deleção. Contribuindo com a identificação dos dados, foi incluída uma visualização simples das séries, o que torna o visual estético. A cor de fundo do *GraphicPlotConfig* alterna entre verde e vermelho, indicando se as séries contidas foram plotada ou não. Ao clicar no botão "Edit", uma caixa diálogo idêntica à de incluir uma série nova aparece permitindo que o usuário edite seus valores, cabeçalho e nome.

Em PyQt, os objetos de uma GUI são "pintados" na tela por um objeto *QtGui.QPainter*, de acordo com sua área, e paleta de cores. A primeira informação depende de alguns parâmetros, como o layout no qual ele está inserido ou sua polí-

tica de tamanho (QSizePolicy). Isto ocorre no método *paintEvent(event)*, chamado automaticamente quando o objeto é reposicionado ou quando sua aparência deve ser atualizada (cada caractere novo digitado numa caixa de texto, por exemplo).

Essa dinâmica torna necessário que o método seja sobrecarregado quando o programador deseje customizar a aparência de um botão ou o seu plano de fundo. A desvantagem é que, por sobrecarregar o método que pinta o objeto na interface, o objeto deve ser repintado manualmente. No caso de um botão, por exemplo, no mínimo o método *drawRectangle()* e *drawText()* deve ser invocado. Assim, quanto mais detalhado for um objeto, mais complicado se torna alterar suas cores e formatos internos. Existem maneiras que contornam a necessidade de criar um novo objeto e sobrescrever o método paintEvent, utilizando as chamadas stylesheets. Porém, para este trabalho, julgou-se mais simples a primeira opção, pelo fato dos objetos customizados possuírem um design pouco complexo.

O objeto *GraphicPlotConfig*, por alternar suas cores de fundo, teve seu evento de "pintura"sobrecarregado. Como sua tela de fundo é representada apenas por um retângulo, sua implementação não foi muito dificultosa.

3.6 *MainPlotArea*

Se tratando de análises de sistemas, a visualização dos dados é essencial. No canto inferior esquerdo da GUI, existe uma área de plotagem, implementada por um objeto *FigureCanvas* (Figura 3.4. Abaixo dele, uma faixa de ferramentas (*NavigationToolbar2QT*) permite que o usuário edite algumas propriedades do gráfico plotado, aproxime ou distancie a imagem e, principalmente, salve a figura em formato de imagem. Na lista de séries à direita, ao clicar no botão “Plot”, a série associada serão desenhadas no Canvas, com legenda.

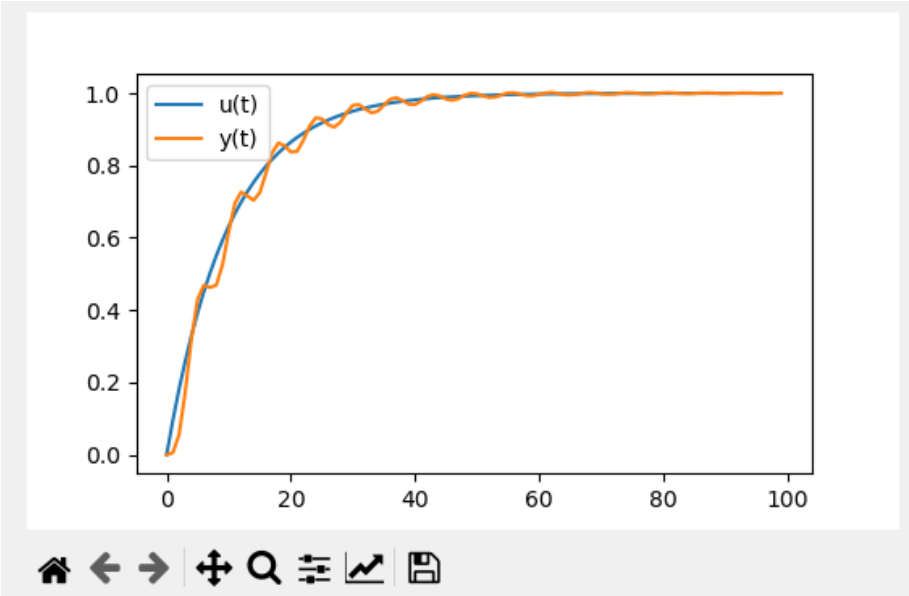

Figura 3.4: *MainPlotArea*

3.7 *SCADADialog*

Um sistema supervisório, como o apresentado neste documento, monitora dados de controladores geralmente industriais em tempo real. Por se tratar de um software didático, pensou-se em inclui no programa um suporte para futuras comunicações com um Arduíno ou qualquer outro microcontrolador ou até mesmo dispositivos que permitam comunicação serial. Assim, configurados os parâmetros de comunicação (*baud rate*, nome da porta e tempo de timeout), o usuário pode plotar dados enviados por um controlador em tempo real, desde que os mesmos estejam no formato esperado: tabulações para separar diferentes séries de dados e quebras de linhas para finalizar um registro.

Ao importar dados por fonte serial no *DatasetConfig*, sugere-se sempre testar a comunicação antes de clicar no botão "Puxar dados". quando o mesmo é clicado, uma caixa diálogo *DialogHeader* é aberta, para que o usuário edite os nomes das variáveis recebidas via serial. A primeira série é sempre o tempo, e o dispositivo conectado **deve** obedecer esta ordem. Ao clicar em OK nesta caixa, a comunicação com a porta serial será iniciada e uma janela de monitoramento surgirá, caso não tenha ocorrido nenhum erro de comunicação.

O código por trás da comunicação com periféricos implementa duas threads da biblioteca nativa emph_thread do Python. Threads são trechos de código que

rodam simultaneamente em um mesmo processo pai. Por conta disso, ao utilizar esta estrutura, alguns cuidados devem ser tomados pelo programador, no que se refere a acesso compartilhado à memória. Como não há controle de execução de cada thread separadamente, bugs podem ocorrer caso mais de um processo acesse e modifique o mesmo objeto ou variável simultaneamente.

Para contornar este problema, existem algumas estruturas que controlam o acesso de memória em programas que implementam threads, como monitores e semáforos. O último é, talvez, o mais trivial. Um semáforo possui dois estados: aberto ou fechado. Quando uma thread toma controle de um objeto no código, o semáforo é fechado, impedindo que outras threads façam o mesmo, até que o objeto seja liberado e o semáforo reaberto. Para Python, existem bibliotecas que implementam estruturas de controle, como a *asyncio*, porém, por ser uma estrutura simples e não utilizada mais que algumas vezes no código-fonte deste trabalho, uma variável booleana bastou.

Após a conexão bem sucedida com o dispositivo, o programa escreve uma mensagem na porta serial ("go", por padrão) e inicia suas duas threads, uma para ler dados da porta, e outra para atualizar o Canvas da caixa diálogo *SCADADialog*. Esta separação se fez necessária pois o método do Canvas que o atualiza (emphdraw()) é considerado custoso, e poderia travar a aplicação se fosse executada repetidas vezes. Outro motivo para implementação das threads é exemplificar a estudantes do código uma implementação simples de paralelismo, o que pode ser bastante util e que não é abordadas nos cursos tradicionais de Engenharia (excetuando computação).

Cada uma das threads tem um tempo de repetição, que define a periodicidade que elas executam suas instruções. O padrão definido foi de 0,1 segundos para ler dados da porta serial, e 1 segundo para atualizar o gráfico com os valores lidos. Quando a leitura da porta ocorre, os valores são armazenados em uma lista chamada emphto_be_plotted. Quando o gráfico é atualizado, os valores desta lista são movidos para outra, chamada emphplotted, que por sua vez é plotada, e o gráfico atualizado. A manipulação compartilhada da lista to_be_plotted justifica o emprego de um semáforo, pois uma thread escreve e outra lê.

Como já mencionado, o programa espera que os dados recebidos sejam espaçados por tabulações, separando diferentes variáveis, e quebras de linhas, separando

diferentes leituras ao longo do tempo de execução. O primeiro valor lido sempre é considerado o tempo, e deve vir do dispositivo conectado, pois o mesmo é quem dita o andamento do processo controlado. Caso o número de variáveis numa mesma linha lida seja diferente do configurado no objeto DatasetConfig, o programa considerará que houve um erro e toda a linha de dados será ignorada.

A figura 3.5 esquematiza a execução do código de monitoramento serial:

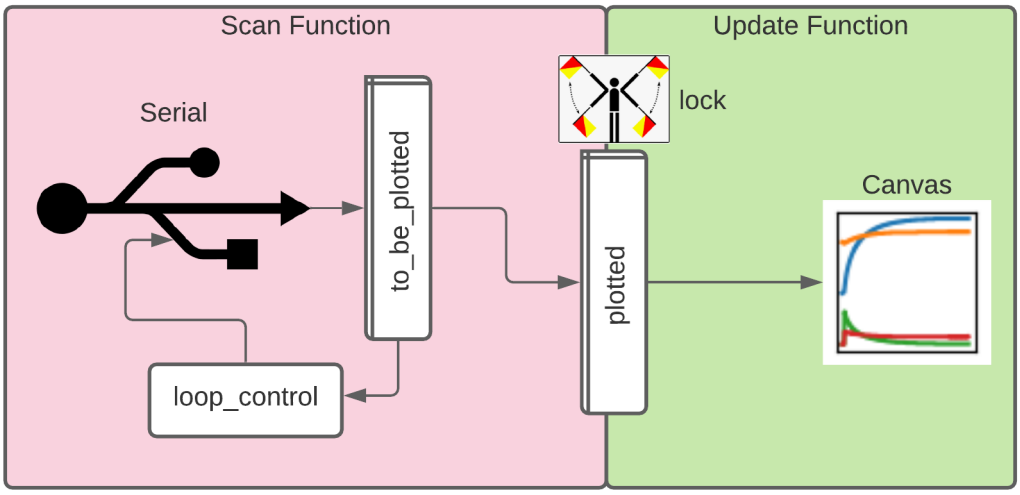

Figura 3.5: Esquema de leitura serial no supervisório didático

Para que o programa possa ser de fato implementado em aulas do curso, foi adicionada a possibilidade de envio de dados ao controlador a cada leitura da porta serial. Isto seria uma analogia a um controlador que atuasse num processo físico, medido por um dispositivo conectado. Devido à popularidade do microcontrolador Arduino, a implementação da rotina de controle foi projetada similarmente à sua programação. A rotina é feita em duas etapas, uma na função *setup_control*, chamada logo após uma conexão bem-sucedida, uma única vez, e outra na função *loop_control*, chamada logo após cada leitura da porta serial.

Numa aplicação de controle PID, por exemplo, a primeira função realizaria sua sintonia, enquanto que a segunda receberia como parâmetro a última linha lida da porta, calcularia a resposta do controlador PID, e a escreveria de volta. Obviamente, o dispositivo conectado deve ser programado para receber esta informação, o que requer certo conhecimento do usuário, tanto de Python como do dispositivo utilizado. Felizmente, códigos de exemplo se encontram disponíveis neste documento.

Durante o monitoramento e registro dos valores trazidos via serial, o gráfico irá atualizar numa janela de 20 segundos, contando do maior tempo registrado para trás. Isto se justifica no comportamento dinâmico da maioria dos sistemas, que atinge valores por vezes maiores que os estacionários. Esta medida impede que a escala do gráfico fique prejudicada, e variações pequenas relativas a um comportamento estacionário não sejam bem percebidas. O usuário pode, de todo modo, em tempo de execução clicar nos botões "Parar" e "Criar Série", salvar as séries de dados lidas e plotá-las no objeto MainPlotArea, visualizando todo seu comportamento histórico.

3.8 Salvamento Automático de Séries

Durante a realização do caso de teste, descrito posteriormente neste documento, percebeu-se que no decorrer da análise de processos muitos ajustes são realizados, seja na função de controle da aplicação ou no controlador. Quando o script Pyhton é iniciado, uma cópia dele é criada e compilada, tornando impossível que alterações no script original em tempo de execução tenham influências no sistema. Por isso, o usuário tem que fechar o supervisório didático sempre que desejar alterar as funções *setup_ control()* e *loop_ control()*, o que resultaria na perda das séries já salvas pelo programa.

Para amenizar este problema, foi incluída uma funcionalidade de salvamento automático na aplicação. Uma das bibliotecas nativas do Python, o *Pickle*, permite que um objeto ou variável do programa seja serializada em formato de arquivo, e salva em um diretório no computador. Isto ocorre através do comando *pickle.dump()* Desta forma, o arquivo pode ser restaurado (*pickle.load()*) e reincorporado ao código com os mesmos valores de atributos que possuía quando foi serializado.

Sempre que uma série é incluída, editada ou deletada, o arquivo "autosave.dat"é sobrescrito com a lista de séries da sessão (*listSeries*). Em contrapartida, quando o objeto *PlotManager* é inicializado, o arquivo "autosave.dat"é aberto e a lista de séries lá salva é restaurada.

3.9 Diagrama de Relações entre Objetos

A Figura 3.6 ilustra todos os objetos utilizados na construção do software, suas relações e principais métodos.

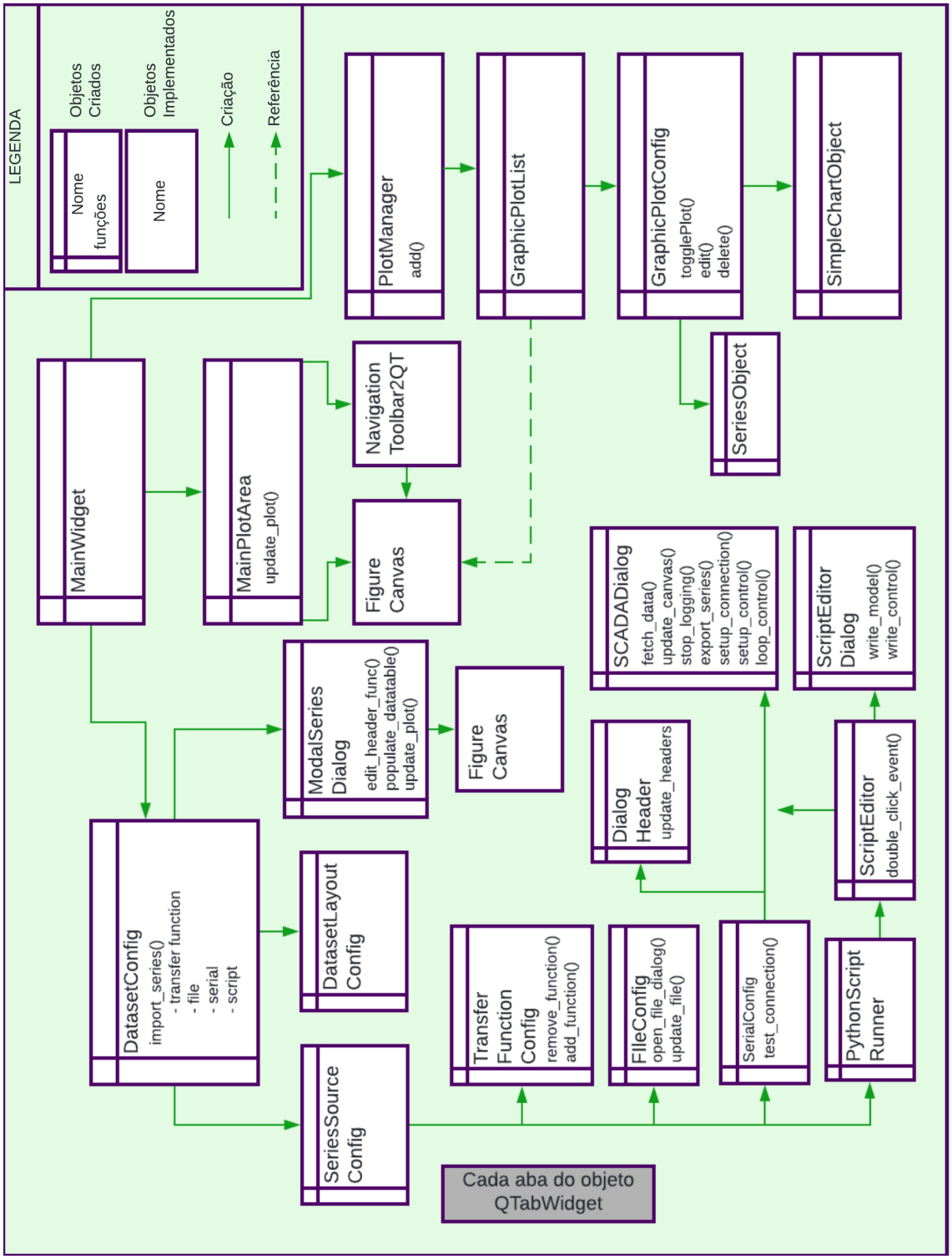

Figura 3.6: Diagrama de relações entre os objetos empregados e funções principais

Capítulo 4

Caso de Teste

4.1 Apresentação do Sistema

Para ilustrar o funcionamento do sistema supervisório, foi utilizado um Arduino UNO que simulasse o funcionamento de um processo com dois tanques de área variável acoplados. A Figura 4.1 o esquematiza:

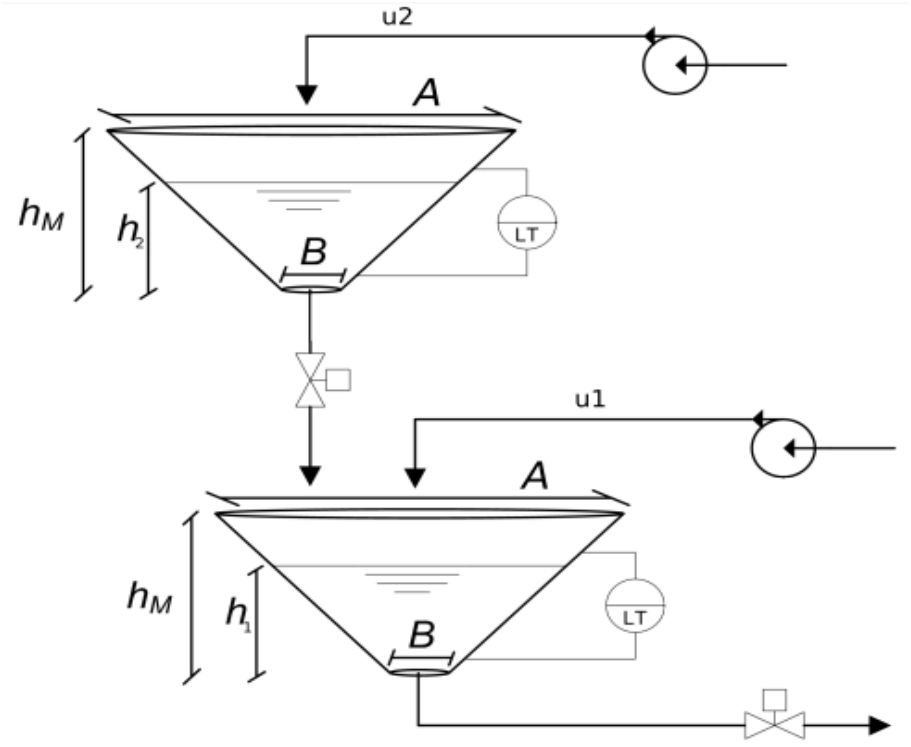

Figura 4.1: Esquema de leitura serial no supervisório didático
Fonte: Elaborado pelo Prof. Daniel Santana

Como percebido pela Figura 4.1, os tanques têm o formato de tronco de cone e as variáveis controladas são suas alturas. A variável de controle são as vazões de entrada de fluido em ambos os tanques.

As equações deste processo são descritas por

$$\frac{dh_1}{dt} = \frac{1}{\beta(h_1)}(u_1 - k\sqrt{\rho g h_1} + k\sqrt{\rho g h_2}) \frac{dh_2}{dt} = \frac{1}{\beta(h_2)}(u_2 - k\sqrt{\rho g h_2})\beta(h_i) = \frac{dV}{dh_i}V(h_i) = \frac{\pi\gamma^2}{3}(h_i + \frac{B}{2\gamma})^3 - \tag{4.1}$$

e seus parâmetros e variáveis são descritos e dimensionados na tabela 4.1:

Símbolo	Descrição	Valor (u.m.)
A	diâmetro superior	4 (m)
B	diâmetro inferior	1 (m)
hm	altura máxima	4 (m)
V	volume	- (m^3)
h	altura	- (m)
ρ	densidade do fluido	1000 (kg/m^3)
g	aceleração da gravidade	9,8 (m/s^2)
k	constante de descarga no tanque	0,001 (-)
u	vazão da bomba	- (m^3/s)

Tabela 4.1: Tabela de parâmetros e variáveis do sistema

4.2 Comportamento esperado

[]

Por se tratar de um sistema não linear, pouco pode-se dizer de seu comportamento dinâmico, partindo-se somente das esquações. Porém, é possível prever possíveis estados estacionários, onde a taxa de variação dos estados no tempo é nula.

Com as derivadas zeradas nas equções descritivas, tem-se que

$$h_{1ss} = (\frac{u_{1ss}}{k\sqrt{\rho g}} + \sqrt{h_{2ss}})^2 \tag{4.2}$$

$$h_{2ss} = \frac{u_{2ss}^2}{k^2 \rho g} \tag{4.3}$$

Como os estados estacionários são reais e finitos, o sistema é dito estável. Nota-se também que, caso a bomba 1 seja desligada, é possível manter as alturas em um ponto diferente de zero somente com a acão da bomba 2, e ambas assumirão os mesmos valores eventualmente ($h_{1ss} = h_{2ss}$).

4.3 Sintonia do Controlador

Segundo (LOURENçO, 1997), não é possível determinar o tipo de controlador a se usar numa determinada aplicação. Idealmente, o controlador mais simples, que satisfaça a "resposta desejada"deve ser ser escolhido. Porém, a escolha depende também das condições de operação do sistema e de performance, como o erro estacionário máximo, e o tempo de estabelecimento permitido.

Mantendo-se neste raciocínio, serão comparados dois controladores distintos neste caso de teste. No primeiro caso, o sistema será linearizado em torno de um ponto de operação e o resultado será utilizado para sintonizar um controlador PID. Em seguida o controlador será acoplado ao sistema. No segundo caso, um controlador mais simples será empregado, o chamado LQR, sintonizado por uma equação matemática, descrita na sua respectiva seção neste documento.

4.3.1 Linearização do Sistema

O processo de linearização de um sistema consiste na aplicação da série de Taylor em suas equações descritivas num determinado ponto de operação:

Aplicando a linearização no sistema de estudo, as novas equações do sistema, em desvio, serão então

$$\frac{dh_1}{dt} = -\Big(\frac{d\beta^{-1}(h)}{dh}\Big|_{h_{1ss}} k\sqrt{\rho g h_{1ss}} + \frac{\beta^{-}1(h_{1ss})k\sqrt{\rho g}}{2\sqrt{h_{1ss}}}\Big)\overline{h_1} + \Big(\frac{\beta^{-}1(h_{1ss})k\sqrt{\rho g}}{2\sqrt{h_{2ss}}}\Big)\overline{h_2} + \beta^{-1}(h_{1ss})\overline{u_1} \tag{4.4}$$

,

$$\frac{dh_2}{dt} = -\Big(\frac{d\beta^{-1}(h)}{dh}\Big|_{h_{2ss}} k\sqrt{\rho g h_{2ss}} + \frac{\beta^{-}1(h_{2ss})k\sqrt{\rho g}}{2\sqrt{h_{2ss}}}\Big)\overline{h_2} + \beta^{-1}(h_{1ss})\overline{u_2} \tag{4.5}$$

$$\beta^{-1}(h) = \frac{4}{4\pi(\gamma h)^2 + 4B\gamma h + B^2} \tag{4.6}$$

$$\frac{d\beta^{-1}(h)}{dh} = -4\frac{8\pi\gamma^2 h + 4\gamma B}{(4\pi(\gamma h)^2 + 4B\gamma h + B^2)^2} \tag{4.7}$$

Substituindos os parâmetros de acordo com a tabela 4.1 e em formato de espaço

de estados o sistema final linearizado será

$$\begin{pmatrix} \dot{\overline{h_1}} \\ \dot{\overline{h_2}} \end{pmatrix} = \begin{pmatrix} -0{,}063 & 0{,}046 \\ -0{,}063 & 0 \end{pmatrix} \begin{pmatrix} \overline{h_1} \\ \overline{h_2} \end{pmatrix} + \begin{pmatrix} 0{,}937 & 0 \\ 0 & 0{,}937 \end{pmatrix} \begin{pmatrix} \overline{u_1} \\ \overline{u_2} \end{pmatrix} \tag{4.8}$$

4.3.2 Sintonia do PI

O controlador PI é um tipo bastante utilizado na indústria. Possui um ganho proporcional (P) e um ganho integrativo (I), que incidem sobre a diferença entre a medição atual de uma saída de processo e seu valor desejado. Isto exige que o sistema de controle seja realimentado com um valor geralmente advindo de um sensor, constituindo-o num sistema de malha fechada. A sintonia de um PI se traduz na definição de P e I, e pode ser realizada, entre outras formas, por métodos como Ziegler-Nichols, Coher-Coon ou até mesmo tentativa e erro.

> "Nos métodos práticos de sintonia o primeiro passo na utilização dos controladores standard P, PI, PD, PID tem como principal decisão a escolha dos modos a utilizar (proporcional, derivativo, integral, ou uma combinação destes). Uma vez aquela tomada, procede-se ao ajustamento dos vários parâmetros do controlador. O ajustamento ou calibração do controlador (sintonização de controladores) consiste em deduzir, partindo da resposta do sistema, quando este é sujeito a entradas específicas, determinados valores que vão permitir o cálculo dos referidos parâmetros."(??)

Sendo $P(s)$ uma função de transferênciaentre a saída Y de um processo e sua única entrada u, um método simples de sintonia, que dispensa experimentação é o chamado síntese direta.

Tomando um sistema em malha fechada representado na figura 4.2

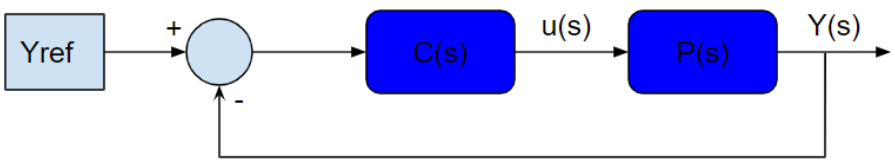

Figura 4.2: Processo 1x1 com feedback e controlador

sua função de tranferência entre a saída Y e entrada u será

$$\frac{Y(s)}{u(s)} = \frac{C(s)P(s)}{1 + P(s)C(s)} \tag{4.9}$$

sendo $C(s)$ a função do controlador.

Desta forma, caso deseje-se modelar $Y(s)$ como uma função de primeiro grau $\frac{1}{\tau s + 1}$, a equação do controlador $C(s)$ assumirá

$$C(s) = \frac{P^{-1}(s)}{\tau s} \tag{4.10}$$

sendo τ o tempo de resposta do sistema, mais precisamente o tempo que o sistema leva para atingir 62,3% do valor de seu estado estacionário.

Caso P seja uma função de primeiro grau, o controlador será um PI.

Para este método de sintonia, o cálculo dos ganhos do controlador de h_1 desconsidera a influência de h_2. Tomando como base o sistema linearizado e aplicando a equação da malha fechada, o mesmo será

$$\overline{h_1}(s) = \frac{0{,}937}{s + 0{,}063}\overline{u_1}(s) + \frac{0{,}046}{s + 0{,}063}\overline{h_2}(s) \tag{4.11}$$

$$\overline{h_2}(s) = \frac{0{,}937}{s + 0{,}063}\overline{u_2}(s) \tag{4.12}$$

Logo, os controladores, projetados para um tempo de resposta τ de 10 segundos, serão:

$$C_1(s) = C_2(s) = \frac{1}{10s}\frac{s + 0{,}063}{0{,}937} = 0{,}1067 + 0{,}0067\frac{1}{s} \tag{4.13}$$

4.3.3 Sintonia do LQR

O Regulador Linear Quadrático, ou LQR, é um controlador de sintonia mais simples que o PI. Seu setpoint é sempre a origem (todos os estados e entradas em zero), tornando necessárias manipulações matemáticas para controlar o sistema em outros pontos de operação. Quanto à sintonia do LQR, o controlador possui basicamente uma matriz Q e uma matriz R, que atribuem pesos, respectivamente, aos estados e

às entradas do sistema. Tomando um sistema genérico em espaço de estados:

$$\dot{x}(t) = A.x(t) + B.u(t) \tag{4.14}$$

sendo $\dot{x}(t)$ as derivadas dos estados, $x(t)$ os estados, A a matriz de estado, $u(t)$ as entradas e B a matriz de entrada; o LQR aplicado será sintonizado de forma a minimizar a função quadrada de custo 4.15:

$$J = \int_0^{\inf} (x'Qx + u'Ru)dt \tag{4.15}$$

Segundo (Argentim et al., 2013), a função de controle por feedback u, para o caso do LQR, é representada pela equação **??**

$$u = -Kx(t) \tag{4.16}$$

sendo K a matriz de ganho do feedback dos estados.

Logo, em suma, a sintonia do LQR se dá ao encontrar a matriz K que minimize a função de custo 4.15.

Para o presente caso de teste, partindo da equação 4.8, tem-se como matrizes A e B:

$$A = \begin{pmatrix} -0{,}063 & 0{,}046 \\ -0{,}063 & 0 \end{pmatrix}, B = \begin{pmatrix} 0{,}937 & 0 \\ 0 & 0{,}937 \end{pmatrix} \tag{4.17}$$

Assumindo que nenhum estado nem entrada deve levar maior importância em relação aos outros, tem-se como matrizes de ganhos Q e R

$$Q = \begin{pmatrix} 0{,}5 & 0 \\ 0 & 0{,}5 \end{pmatrix}, R = \begin{pmatrix} 0{,}5 & 0 \\ 0 & 0{,}5 \end{pmatrix} \tag{4.18}$$

Utilizando resursos computacionais (neste caso, o comando *controller_lqr()* da biblioteca Python *controlpy*, obtém-se para o sistema de teste a matriz de ganhos

$$K = \begin{pmatrix} 0.936 & -0{,}011 \\ -0{,}011 & 0{,}999 \end{pmatrix} \tag{4.19}$$

De acordo com a equação 4.16, as entradas do sistema serão então regidas pelas

equações

$$u_1 = 0.936h_{1lqr} - 0.011h_{2lqr} u_2 = -0.011h_{1lqr} + 0.999h_{2lqr} \tag{4.20}$$

Porém, como o LQR traz todas as variáveis para a origem, é preciso uma translaçãonas mesmas, para que seja possível controlar o sistema em outros pontos. Desta forma, para h_1 e h_2 são feitas as seguintes operações:

$$h_{1lqr} = h_1 - h_{1sp} h_{2lqr} = h_2 - h_{2sp} \tag{4.21}$$

sendo h_{isp} os setpoints das alturas dos tanques no caso de teste

4.4 Configuração do supervisório didático

Como já mencionado, o programa apresentado implementa duas funções que podem ser editadas pelo usuário, de forma que seja possível uma resposta ao dispositivo conectado. O usuário pode ainda escrever suas próprias funções e alterar os objetos do código-fonte como preferir, para atender às suas necessidades.

4.4.1 Controlador PI

Além da biblioteca *python-control*, existe outra chamada *simple-pid* que, como o nome indica, possui objetos que se comportam como controladores PID. Eles recebem como parâmetros seus respectivos ganhos, valores de setpoints, limites para a resposta e outros que são referenciados na página da biblioteca . Para implementá-los no programa, na função *setup_ control()*, criam-se tais objetos de acordo com o código 1.

Código 1

```
1 self.pids = []
2 self.pids.append(simple_pid.PID(0.1067, 0.0067, setpoint=2.5))
3 self.pids.append(simple_pid.PID(0.1067, 0.0067, setpoint=2.5))
```

O objeto PID da biblioteca simple-pid, quando chamado, retorna um valor numérico referente à resposta do controlador a partir de uma leitura, informada como

parâmetro. Este valor pode ser escrito diretamente na porta serial, e será capturado pelo dispositivo conectado, desde que o mesmo esteja preparado para recebê-lo. No Arduino, o comando ***Serial***.*parseFloat()* se mostrou satisfatório.

Implementa-se, então, na função *loop_ control()*, a lógica descrita acima através do código 2:

Código 2

```
1       if len(input_data) == 0:
2             return
3
4       signals = [self.pids[i](input_value) for i, input_value in
           enumerate(input\_data)]
5       for signal in signals:
6             self.porta.write('{:.2f}'.format(signal).encode('UTF-8'))
7       return
```

O resultados podem ser conferidos na seção de resultados.

4.4.2 LQR

Uma das alternativas para a implementação do LQR no programa é a de usar a mesma biblioteca *control* referenciada no Quadro 3.1, que simula processos por funções de transferência. Nela existe a função *lqr()* que retorna, entre outras saídas, a matriz de ganhos K que parametriza a função de controle mostrada na equação 4.16

Na função setup_control(), escreve-se o código 4, onde calcula-se primeiramente a matriz de ganhos K e a armazena como uma propriedade do objeto pai. Aqui também foram definidos os desvios das variáveis (para o caso dos estados, seus setpoints), que serão empregados no cálculo do sinal de controle.

Código 3

```
1 A = [[-0.063, 0.046],[-0.063, 0]]
2 B = [[0.937, 0],[0, 0.937]]
3 Q = [[1, 0],[0, 1]]
4 R = [[2, 0],[0, 2]]
```

```
K, P, V = control.lqr(A, B, Q, R)

rho = 1000
g = 9.8
k = 0.001

self.K = K
self.sp_h1 = 2.5
self.sp_h2 = 2.5
self.sp_u1 = 0
self.sp_u2 = k*math.sqrt(rho*g)

return
```

Já na função loop_control(), aplica-se a equação 4.20 para o cálculo de cada sinal de controle. Porém, antes disto, realiza-se a translação nas variáveis, como mostrado na equação 4.21.

Neste caso de teste, notou-se que a função *Serial.parseFloat()* não foi totalmente útil, pois lia apenas a parte decimal dos valores enviados por porta serial, ignorando a parte inteira. No caso anterior, isto não se constituiu num problema, pois os sinais de controle não ultrapassaram $1m^3/s$. Sendo assim, para o LQR, foi necessário dividir o sinal por 10, e multiplicá-lo novamente no Arduino, como constatado no código 4.

Uma outra alternativa seria normalizar o sinal de controle entre 0 e 1, e desnormalizá-lo no Arduino, considerando seus limites de operação.

Código 4

```
if len(input_data) == 0:
      print('No Read')
      return

input_data[0] = input_data[0] - self.sp_h1
input_data[1] = input_data[1] - self.sp_h2

signals = [-(input_data[0]*self.K[0][0] + input_data[1]*self.K[0][1]) +
    self.sp_u1,
```

```
9               -(input_data[0]*self.K[1][0] + input_data[1]*self.K[1][1])
                    + self.sp_u2]
10  for signal in signals:
11          resp = '{:.3f}'.format(float(signal)/10).encode('UTF-8')
12          self.porta.write(resp)
13  return
```

Exceto pela manipulação do sinal de controle, código utilizado no Arduino foi idêntico ao do caso do controlador PI, no anexo. Buscou-se modificá-lo o mínimo possível, pois numa aplicação real seria mais coveniente alterar as funções do supervisório, não as do controlador.

4.5 Resultados

4.5.1 Controlador PI

Com os códigos no Arduino e no supervisórios prontos, a série por via serial foi importada. Primeiramente, removeu-se as restrições de processo, para que a resposta seja mais fiel ao sistema modelado, em detrimento do sistema real. Os tanques foram partidos em $h1 = 1m$ e $h2 = 2m$.

Os resultados se encontram na Figura 4.3.

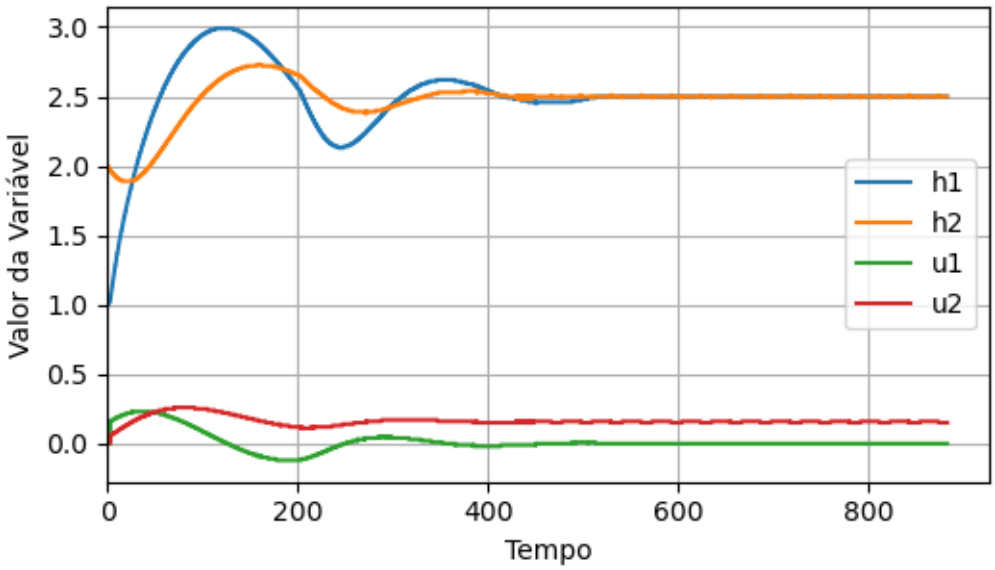

Figura 4.3: Resposta do processo para controlador PI, desconsiderando restrições de processo

Em seu estado estacionário, pode-se afirmar que o sistema se comporta satisfatoriamente. Como calculado na Seção 4.2, com os setpoints de altura iguais, a bomba 1 tem estado estacionário nulo, e o sistema se mantém pela ação única da bomba 2, que assume um valor constante de $k\sqrt{\rho g h_{2ss}} = 0{,}16$.

Como esperado, entretanto, o controle não é perfeito, pois não somente o sistema leva cerca de 10 minutos para estabilizar, como também tem comportamento dinâmico oscilatório, o que não corresponde com um sistema de primeira ordem. Porém, pelo controlador ter sido sintonizado considerando um sistema não linear, mas ter sido aplicado em outro não linear, o comportamento estacionário tem peso maior avaliação da eficiência do controle. Outro fator a ser considerado é acoplamento do sistema, que dificulta o controle preciso, já que cada controlador age somente sobre uma variável.

Apesar de nenhuma altura extrapolar o limite máximo de 4 metros, o sinal de controle assume, por vezes, valores negativos. Isto sigificaria que o fluido poderia ser retirado do tanque pelas bombas, o que não foi considerado possível neste caso. Assim, foram incluídas as seguintes restrições no processo:

1. As alturas h_{1ss} e h_{2ss} devem estar compreendidas no intervalo [0, 4]m

2. A entrada do sistema assume valores somente entre 0 e 1 m^3/s

Os resultados para um sistema fisicamente mais fiel são apresentados na Figura 4.4:

Neste caso, para valores negativos da variável de controle, os mesmo são por sua vez substituídos por 0. Isto faz com que a altura h_{1ss} leve mais tempo para diminuir seu valor, pois depende somente do escoamento por gravidade e da alimentação do tanque 2, o que se reflete num tempo maior de estabilização do processo, comparado ao caso anterior.

Contudo, ainda não houve transbordamentos e os sinais de controle não atingiram valores muito altos. Assim, pode-se concluir que a estratégia do PI sintonizado em um sistema linearizado se mostrou aplicável e razoável, pois é capaz de controlar o sistema com uma boa precisão, e com pouco esforço de controle.

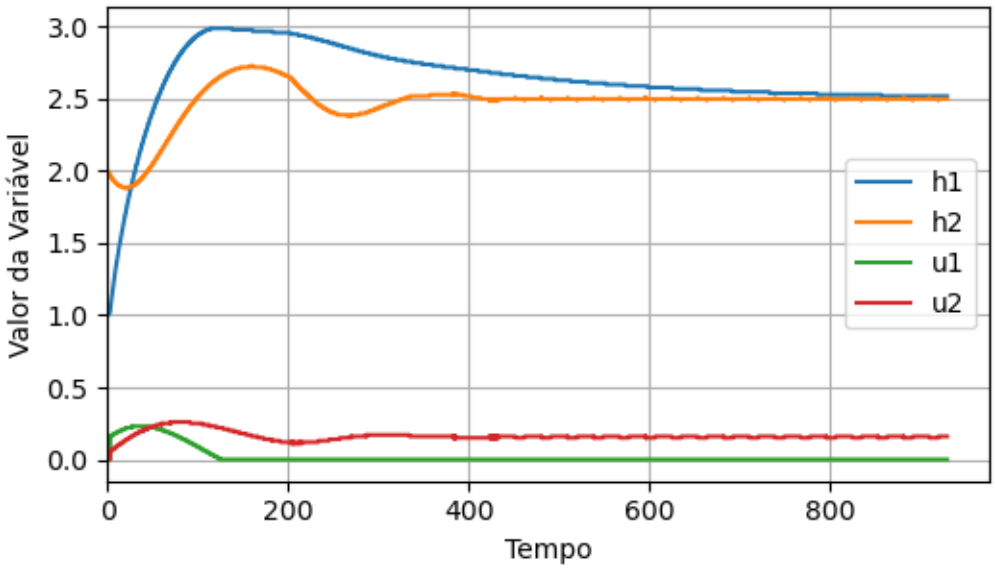

Figura 4.4: Resposta do processo para controlador PI, incluindo restrições de processo

4.5.2 LQR

Para a resposta do sistema com o LQR, o mesmo foi iniciado com as mesmas condições iniciais do caso anterior, $h_1 = 1m$, $h_2 = 2m$. Os resultados encontram-se na Figura 4.5

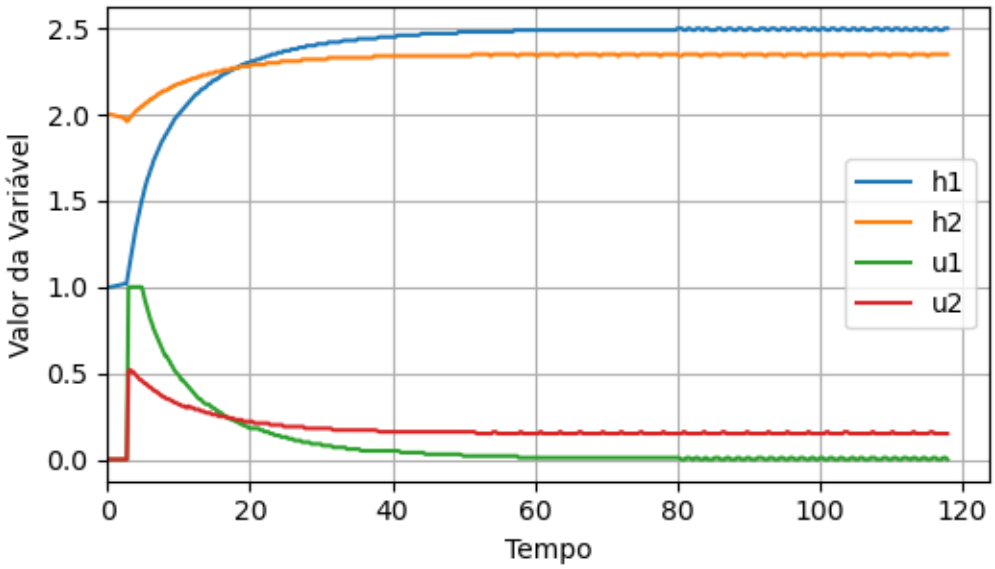

Figura 4.5: Resposta do LQR no primeiro caso de teste

De acordo com a resposta no tempo, assim que o controlador entra em operação, percebe-se um pico no sinal de controle, que ultrapassaria o limite estabelecido, caso não tivesse sido implementada uma salvaguarda no Arduino. A partir daí, as

variáveis suavemente se aproximam do setpoint estabelecido. No geral, o sistema responde mais rapidamente que com o controlador PI do caso anterior. Porém, existe um erro estacionário maior.

De forma a suavizar o pico inicial do sinal de controle, aumentou-se os ganhos da matriz Q para 2. Isto faz com que o controlador favoreça valores menores de controle, diminuindo sua eficiência, mas ganhando em um menor esforço das bombas do sistema. Como esperado e como constatado na Figura 4.6, o tempo de acomodação do sistema aumenta, mas o pico não é tão grande quanto no caso 1. A eficiência do controle também é prejudicada, pois os pesos dos estados do sistema ficaram menores em relação aos pesos dos sinais de controle.

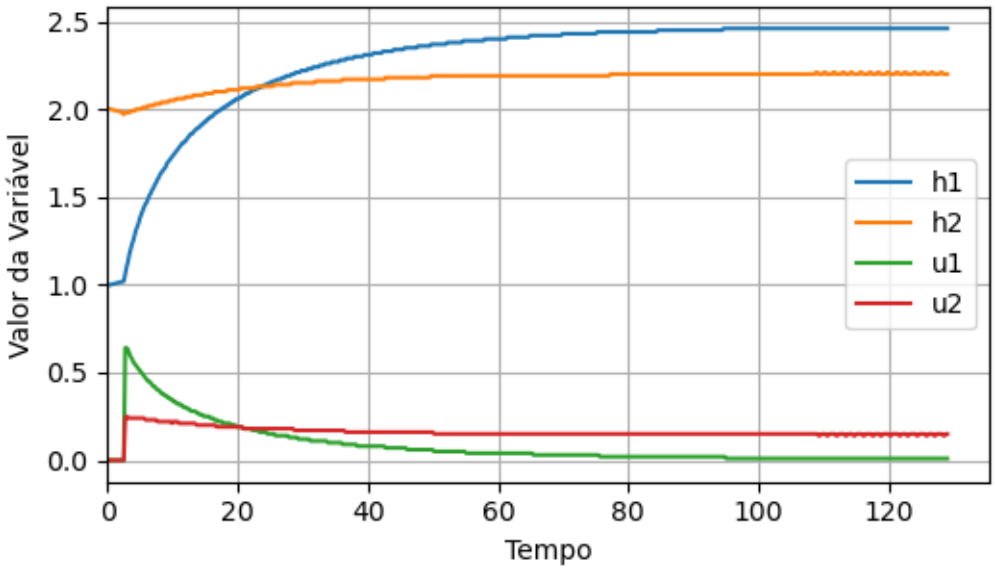

Figura 4.6: Resposta do LQR no caso 2

Caso se deseja-se uma melhor precisão, os pesos das matrizes de controle e de estados podem ser invertidos. O resultado disto é visto na Figura 4.7.

Como esperado, o sistema responde ainda mais rápido que no caso 1 e tem menor erro estacionário. A desvantagem é o pico do sinal de controle, que permanece mais tempo no limite máximo.

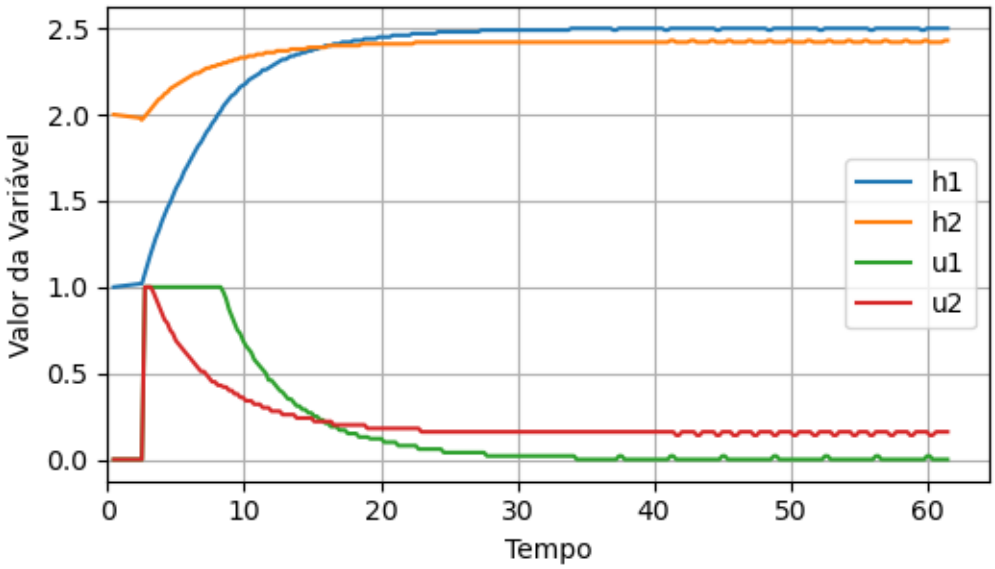

Figura 4.7: Resposta do LQR no caso 3

4.6 Considerações Finais

Neste capítulo, foram simulados e comparados dois diferentes métodos de controle num sistema de tanques acoplados. No caso do PI, apesar de haver certa demora para regular o sistema, o esforço de controle não chegou perto do limite máximo, e o estado estacionário atingido foi bem próximo dos setpoints escolhido. Em contrapartida, o LQR apresentou um erro estacionário visível, e demandou um esforço maior de controle. Por outro lado, a sintonia de ambos são diferentes, e outros valores podiam ter sido empregados para τ do sistema final modelado para o PID.

Assim como afirmado por (LOURENçO, 1997), não existe um controlador universalmente melhor ou pior para um dado sistema. Dados os resultados atingidos, a escolha do método de controle dependeria de alguns fatores externos, como potência das bombas empregadas no sistema, tolerância de erros e tempo de resposta. Em uma aplicação industrial, o custo monetário de cada controlador e atuador certamente também seria levado em consideração. Cabe ao engenheiro responsável tomar a decisão quanto ao que seria mais vantajoso de se implemantar.

Nota-se também que o supervisório didático cumpriu seu propósito em trazer os dados em tempo real e com confiança ao usuário. Não só isso, possibilitou que os gráficos de resposta fossem facilmente armazenados e exportados em formato de imagem. Um ponto de melhoria seria permitir que o setpoint fosse modificado

em tempo de execução, para que o comportamento do sistema fosse perturbado. Por outro lado, isto pode ser feito com programação Python na função do loop de controle.

Capítulo 5

Conclusão

Este trabalho teve como objetivo principal apresentar e testar um sistema supervisório didático construído em Python, que permitisse aos usuários monitorarem valores recebidos pela porta serial da máquina na qual fosse executado, e ainda a criarem suas próprias séries de dados. De outra perspectiva, este trabalho almejou fomentar o emprego de tecnologias open source no ambiente acadêmico, e encorajar os estudantes do ensino superior a se familiarizarem com linguagens de programação, sobretudo Pyhton, que vem ganhando notoriedade no cenário atual.

Frente aos objetivos levantados, pode-se afirmar que o presente trabalho alcançou seu propósito, como exemplificado nos casos de teste. Utilizando a ferramente desenvolvida, o estudante pode criar um sistema físico controlado por microcontrolador, e monitorar em tempo real o comportamento deste sistema. Ainda, pode salvar o gráfico da resposta no programa e compará-lo com outros comportamentos esperados, modelados também dentro do programa.

A seção de apresentação do sistema abordou algumas técnicas adotadas na programação do software, e representa um guia para construções de GUIs utilizando a bilbioteca PyQt. Apesar de haver diversos vídeos, exemplos e tutoriais na internet tratando desta biblioteca, este trabalho se destaca não só a aplicar Python no âmbito da automação, como a enriquecer a interface com outros recursos da linguagem, como a serialização de objetos e a paralelização do código em threads.

Espera-se que o produto entregue neste trabalho de conclusão de curso seja de grande valia para os estudantes, bem como para a universidade.

5.1 Possíveis aditivos

Por se tratar de um código aberto, qualquer interessado tem a possibilidade de alterar os scripts como desejar, incluindo novas funcionalidades ou modificando as existentes. Acredita-se que, por ter sido construído em uma linguagem open-source, o supervisório didático deixa muitas possibilidades de melhoria, como as citadas a seguir, ordenadas por complexidade:

1. Permitir que o usuário exporte os resultados em diferentes formatos de arquivos;

2. Adicionar um módulo no programa responsável pela escrita em um banco de dados das séries registradas, assim como feito em sistemas SCADA industriais;

3. Incluir novos meios de comunicação com periféricos, como por bluetooth ou rede Wi-Fi;

4. Acrescentar novas formas de simulação de sistemas, não somente por funções de transferência simples, mas por equações descritivas, ou até mesmo um ambiente que monte um diagrama de blocos, como implementado por ferramentas como o MATLab® e o SciLab;

5. Incluir ferramentas de análise de processos como perditor de Smith e estimadores e avaliadores de estado.

Referências

ARDUINO Serial Function Reference. 2019.

Argentim, L. M.; Rezende, W. C.; Santos, P. E.; Aguiar, R. A. Pid, lqr and lqr-pid on a quadcopter platform. In: *2013 International Conference on Informatics, Electronics and Vision (ICIEV)*. [S.l.: s.n.], 2013. p. 1–6.

BOLTON, W. *Programable Logic Controllers*. [S.l.: s.n.], 2015.

DENVER, A. *Serial Communication in Win32*. 1995.

ELIPSE Knowledge Base. 2019. Disponível em: <https://kb.elipse.com.br/kb29583-utilizando-o-e3studio-em-modo-demo/>.

JUNIOR, E. G. *Introdução a Sistemas de Supervisão, Controle e Aquisição de Dados: SCADA*. [S.l.]: Alta Books, 2019.

KATSUHIRO, O. *Engenharia de Controle Moderno*. [S.l.: s.n.], 2010.

LIMITED, R. C. *Documentation for PyQt*. 2020. Disponível em: <https://pypi.org/project/PyQt5/>.

LOURENçO, J. *Sintonia de Controladores*. 1997. Disponível em: <http://ltodi.est.ips.pt/smarques/CS/Pid.pdf>.

MARTINS, G. M. *Princípios de Automação Industrial*. 2007.

MAZIDI, A. M.; MCKINLAY, R. D.; CAUSEY, D. *PIC Microcontroller and Embedded Systems: Using Assembly and C for PIC18*. [S.l.: s.n.], 2008.

MORAES, A. S. *Desenvolvimento de sistema supervisório didático para controle de inversor de frequência acionando motor de indução trifásico*. Trabalho de conclusão de curso, 2016.

ROGGIA, L.; FUENTES, R. C. Apostila, *Automação Industrial*. 2016.

SILVA, M. R.; OLIVEIRA, n. R. de; CARMO, M. J. do; JUNIOR, L. O. d. A. Importância da ferramenta scadabr para o ensino em engenharia. In: . [S.l.: s.n.], 2013.

Apêndice A

Apendice 1

A.1 Código do arduino para Controle de Tanque com Área Variável

```
1
2  #include <stdlib.h>
3
4  //Variables
5  float dh1dt;
6  float dh2dt;
7  float h1;
8  float h2;
9  float u1;
10 float u2;
11 float t0;
12 float t;
13
14 //Parameters
15 float pi = 3.14159265358979323846264338327795;
16 float B = 1.;
17 float A = 4.;
18 float hM = 4.;
19 float gamma = ((A/2.)-(B/2.))/2.;
20 float k = 0.001;
21 float g = 9.8;
22 float rho = 1000;
23 float c = k*pow(rho*g,0.5);
24 //sample time in miliseconds
25 float tstep = 100;
```

```
26
27 //Espera por um sinal vindo do computador
28 void wait_for_comm() {
29       while (true) {
30             if (Serial.available() > 0) {
31                   break;
32             }
33       }
34       clearSerial();
35       return;
36 }
37
38 //Limpa a porta seria para nao atrapalhar futuras leituras
39 void clearSerial() {
40       char c;
41       while(Serial.available() > 0)
42             c = Serial.read();
43       return;
44 }
45
46 void setup() {
47       // put your setup code here, to run once:
48       Serial.begin(9600);
49
50       h1 = 1.0;
51       h2 = 2.0;
52       u1 = 0;
53       u2 = 0;
54
55       wait_for_comm();
56       t0 = millis();
57 }
58
59 void loop() {
60 // put your main code here, to run repeatedly:
61       if (Serial.available()) {
62             Serial.flush();
63             u1 = Serial.parseFloat();
64             u2 = Serial.parseFloat();
```

```
65        }
66
67        //Limitacao da entrada
68        if (u1 > 1) u1 = 1;
69        if (u1 < 0) u1 = 0;
70
71        if (u2 > 1) u2 = 1;
72        if (u2 < 0) u2 = 0;
73
74        t = millis() - t0;
75        t0 = millis();
76        for (int i = 0; i < ceil(t/tstep); i++) {
77              dh1dt = (1./(pi*pow(gamma,2)*pow(h1+(B/2)/gamma,2))*(u1+c*
                    pow(h2,0.5)-c*pow(h1,0.5)));
78              dh2dt = (1./(pi*pow(gamma,2)*pow(h2+(B/2)/gamma,2))*(u2-c*
                    pow(h2,0.5)));
79
80              h1 += dh1dt*tstep/1000;
81              h2 += dh2dt*tstep/1000;
82
83              //Limitacao dos estados
84              if (h2<0){
85                    h2 = 0;
86              } else if(h2>hM){
87                    h2 = hM;
88              }
89
90              if (h1<0.){
91                    h1 = 0;
92              } else if(h1>hM){
93                    h1 = hM;
94              }
95        }
96
97        Serial.print(t0/1000);
98        Serial.print('\t');
99        Serial.print(h1, 2);//+(float) random(-1,1)/80.,2);
100       Serial.print('\t');
101       Serial.print(h2, 2);//+(float) random(-1,1)/80.,2);
```

```
102         Serial.print('\t');
103         Serial.print(u1,2);
104         Serial.print('\t');
105         Serial.println(u2,2);
106 }
```