TÍTULO

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.

Orientadores: NOME SOBRENOME ÚLTIMO SOBRENOME
NOME SOBRENOME ÚLTIMO SOBRENOME

Salvador

Dezembro de 2020

S2320 LUCENA, RAFAEL GUIMARÃES

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

XVII, 35 p.: il.; 29,7cm.

Orientadores: NOME SOBRENOME ÚLTIMO SOBRENOME

NOME SOBRENOME ÚLTIMO SOBRENOME

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

Referências: p. **??** – **??**.

1. Sistema Supervisório. 2. Python. 3. Sistema Open Source. 4. PyQt. I. ÚLTIMO SOBRENOME, NOME SOBRENOME *et al.*. II. Universidade Federal da Bahia, Programa de Engenharia Industrial. III. Título.

CDD: 511

Dedicatória

Agradecimentos

Agradecimentos

"Nunca tenha certeza de nada, porque a sabedoria começa com a dúvida."

Freud

"A única coisa que me espera é exatamente o inesperado."

Clarice Lispector

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

Resumo.

Palavras-chave. palavra-chave 1, palavra-chave 2, palavra-chave 3

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. keyword 1, keyword 2, keyword 3

Sumário

Lista de Figuras

Lista de Tabelas

Lista de Quadros

Capítulo 1

Introdução

1.1 Contextualização

Incluir

1.2 Problema e Justificativa

Frente à importância de sistemas supervisórios no âmbito da 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 que a executasse, registrando em forma de gráficos e valores estatísticos 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 físicos diversos e comparação com modelagens criadas 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:

- 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 apresenta o sistema já pronto e aborda as lições aprendidas no decorrer de seu desenvolvimento, além de outras discussões relativas à sua programação.

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. Essas equações diferenciais 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 modelo matemático não é o único para determinado sistema. Um sistema pode ser representado de muitas maneiras diferentes e. portanto, pode ter vários modelos matemáticos, dependendo da perspectiva a ser considerada. (OGATA 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 deste processo, 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 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.

A modelagem de um processo pode traduzi-lo 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 aplicação simultânea de duas funções de determinação diversas é a soma das duas respostas individuais. Então, para o sistema linear, a resposta a diversas entradas pode ser calculada tratando uma

entrada de cada vez e somando os resultados. Esse é o princípio que permite construir soluções complicadas para equações diferenciais lineares a partir de soluções simples. (OGATA KATSUHIRO, 2010)

Sistemas não lineares são, em geral, mais difíceis de modelar e de controlar. Assim, podem ser empregadas algumas técnicas para transformá-los em sistemas lineares. Uma delas é a chamada linearização, que emprega a série de Taylor, truncando a série no segundo termo e obtendo assim, uma função linearizada em torno de um determinado ponto de operação do processo. Quanto mais as variáveis do sistema linearizado se afastarem deste ponto de operação, maior será o erro deste sistema, em relação ao sistema não linear. Outra técnica é partir o sistema não linear com uma entrada qualquer, e pela análise do gráfico de resposta projetar um sistema linear que seja o mais similar 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 com testes em campo. Softwares como MATLAB e GNU Octave permitem que, a partir de funções (no tempo ou de transferência) um processo seja modelado 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. (MACHADO MARTINS, 2012)

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

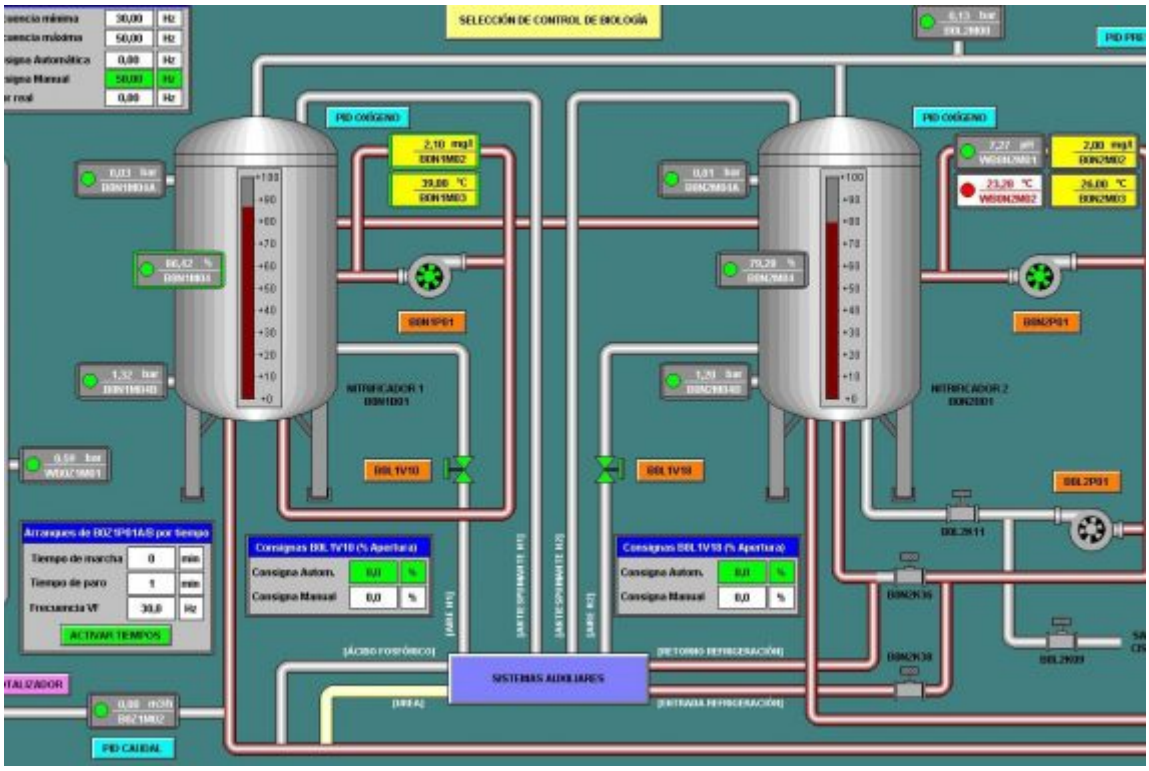


Figura 2.1: Tela exemplo de um sistema supervisório

agrupar as informações por temática, e mostrar diversas telas menos detalhadas, que alternem entre si de acordo com um temporizador interno.

Os dados adquiridos podem ser manipulados de modo a gerar valores para parâmetros de controle como "set-points". Os dados são armazenados em arquivos de dados padronizados, ou apenas utilizados para realização de uma tarefa. Esses dados que foram armazenados em arquivos poderão ser acessados por programas de usuários para realização de cálculos, alteração de parâmetros e de seus próprios valores. (MACHADO MARTINS, 2012).

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 manualmente, 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, referente à supervisão e controle de uma célula específica de produção, chegando ao nível seguinte, de supervisão da produção, pois agrupa as informações de diversos controladores e sensores em um só local, gerando suporte para ações de gerência.

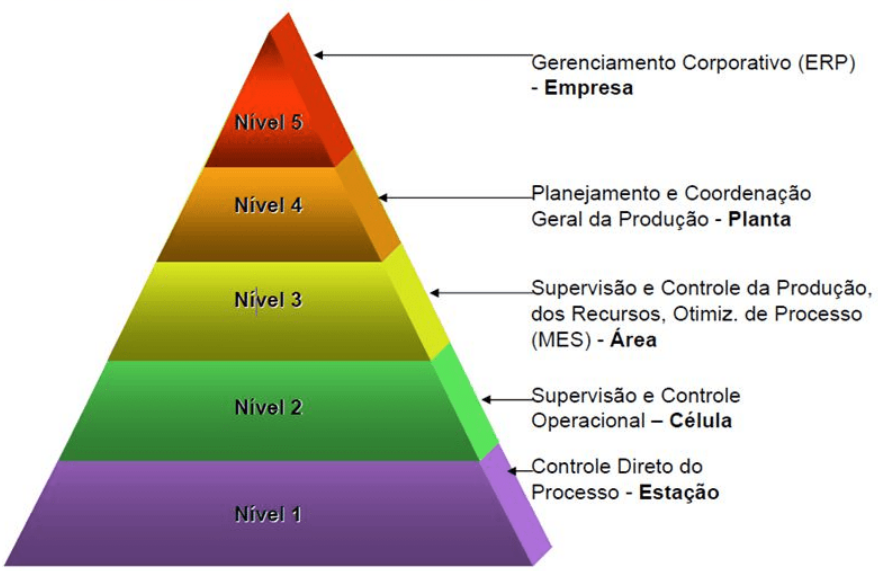


Figura 2.2: Pirâmide da automação

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 Leandro, et al, 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. Ela ocorre através de um único cabo ou pino

em um circuito integrado, sendo este um dos motivos da sua popularidade: o baixo custo necessário para a montagem da infraestrutura. A desvantagem disto é a perda em velocidade, pois transmite-se apenas um bit de cada vez, enquanto que existem tecnologias, como a chamada comunicação paralela, que utilizam mais canais de comunicação e podem transmitir vários bits simultaneamente.

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 uma porta serial. Ela também está presente em alguns protocolos de comunicação modernos, como Ethernet e Profibus.

Um modelo de transmissão para um byte (8 bits) de informação por porta serial se constitui 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 (que torna o dados mais confiáveis), 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 dos dados ou para aumentar a velocidade de comunicação.

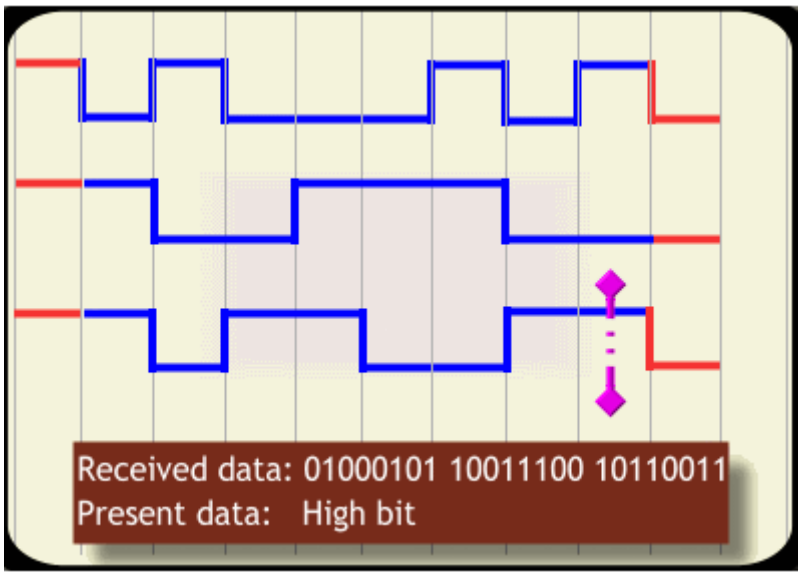


Figura 2.3: Exemplo de transmissão serial de uma sequência de 3 bytes

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.

2.4 Qt em Python

Com traços da linguagem C e Perl, Python foi criada por Guido van Rossum no início da década de 90. 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. 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, existem diversas bibliotecas de Python que podem ser baixadas com o simples comando no terminal pip install, configurado na instalação da linguagem. 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 gráficos e métodos para criação de interfaces gráficas.

A biblioteca PyQt5 veio da biblioteca de C++ "Qt", que implementa APIs para outras linguagens. 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 *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. Como dito, a maioria herda direta ou indiretamente do objeto *QWidget*. Por conta da arquitetura 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.

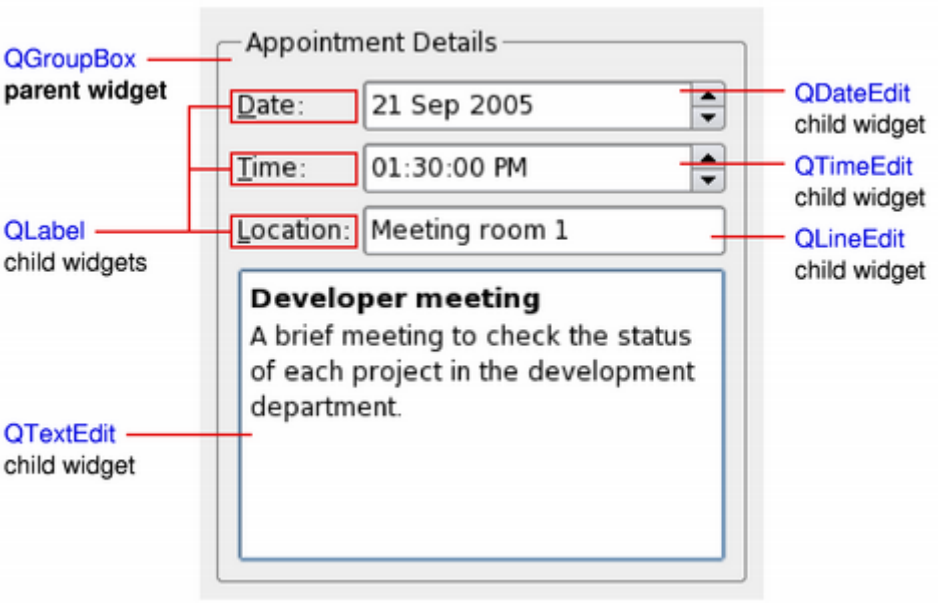


Figura 2.4: Janela *QMainWindow* com um *QGroupBox* como Widget central, contendo vários outros Widgets em um layout *QGridLayout* dentro de outro layout *QHBoxLayout*

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 e, opcionalmente, simples e de código aberto no intuito de permitir análise por partes de interessados e futuras melhorias e expansões;
- Deve ser capaz de plotar gráficos, advindos de no mínimo qualquer uma das seguintes fontes: via porta serial, arquivos nos formatos .csv, .tsv, .xls, e .xlsx, ou por modelagem direta no software, por funções de transferência
- Deve se capaz de monitorar sinais em tempo real e plotá-los em um gráfico, similarmente a um SCADA.
- 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 trabalho, inclusive, utilizando a primeira tecnologia (referenciados na bibliografia deste documento). Porém, o

ScadaBR é construído em um navegador de internet, utilizando ainda softwares periféricos como o Apache TomCat (como servidor web). Assim, julgou-se necessário um certo período para acostumar-se com a sua utilização. Quanto à versão demo do Elipse E3, apesar do autor já possuir certo domínio da ferramenta, se trata de uma versão muito limitada. Permite somente até 20 tags de dados e a aplicação roda por um máximo de 2 horas, tendo que ser reiniciada após este período. Finalmente, não se sabe das implicações legislativas em utilizar o software para trabalhos acadêmicos, nem sua utilização contínua no âmbito da universidade, mesmo se tratando de uma versão gratuita de uma plataforma licenciada.

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.

3.2.1 Seleção das bibliotecas

Como já mencionado, a linguagem Python possui inúmeras bibliotecas, para os mais variados fins. Para a programação do sistema, foram utilizadas as seguintes bibliotecas, com as seguintes descrições:

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

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

Tabela 3.1: Bibliotecas Python utilizadas no desenvolvimento do supervisório didático

incluir séries novas. Os detalhes destes Widgets são listados abaixo:

- emphPlotManager: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.

- emphMainPlotArea: 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, em segundos.

- emphDatasetConfig: 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 fontes (serial, arquivo ou função de transferência)

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

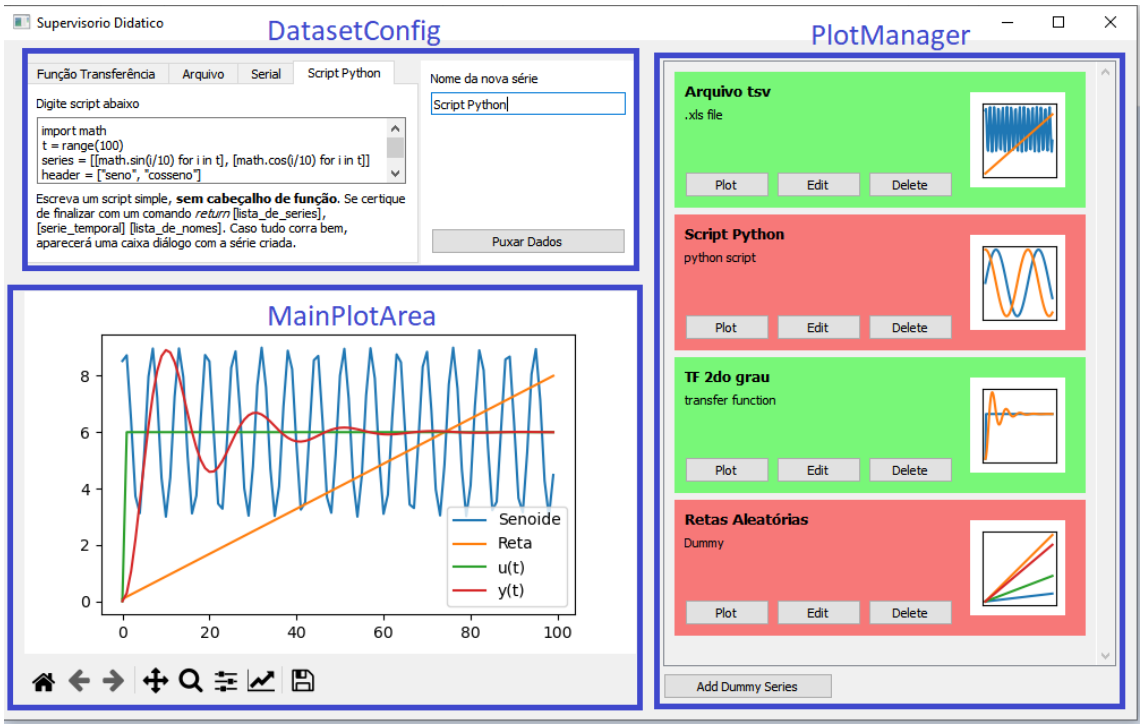


Figura 3.1: Supervisório didático e seus objetos principais

Além dos Widgets supracitados, para simplificar o código e troná-lo mais prático, foi criado um objeto abstrato emphSeriesObject. 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. O último é importante para escrita na legenda do gráfico principal, quando a série for plotada.

Durante o desenvolvimento do software, levantou-se meios para adicionar séries de dados 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. Sendo o Microsoft Excel um software popular para análise e plotagem de gráficos, seria útil também a importação de séries por arquivos *.xls* ou *.xlsx*.

Além disso, caso um estudante estiver fazendo uma modelagem de um processo monitorado, o programa deveria oferecer uma maneira simples de simular processos diversos e plotá-los em contraste com o sistema real. Sendo ele voltado para o curso de Controle e Automação então, o supervisório devia permitir a criação de funções de transferência e simular sua resposta.

Como um método alternativo, caso a função de comportamento de um processo não seja bem descrita por uma função de transferência, foi incluído também um campo para que o usuário escreva um pequeno script em python que produza uma série de dados como ele desejar e salve-a no programa. 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 de comunicação (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 (os valores recebidos são todos numéricos e as séries de valores e tempo possuem o mesmo tamanho), o programa abrirá uma caixa diálogo (*QtWidgets.QDialog*) 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.

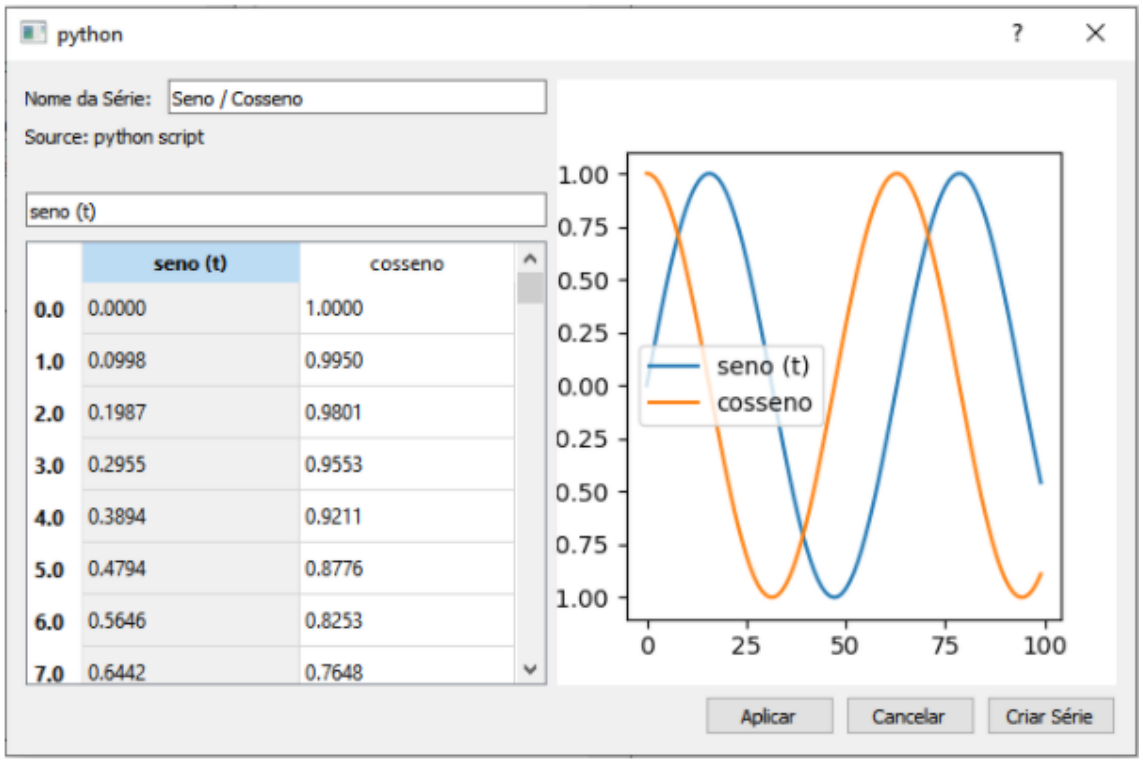


Figura 3.2: *ModelSeriesDialog*: Caixa diálogo para edição dos eixos e título das séries

A caixa diálogo contém um Widget bastante importante para a aplicação: o *FigureCanvas*, que vem de um módulo da biblioteca matplotlib, e 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 documentação 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.

Devo Focar no algoritmo de cada tipo de importação??

3.5 *PlotManager*

No lado direito da aplicação, se encontra uma lista das séries carregadas. À medida que forem inseridas, faz-se necessária uma área de rolagem. Assim, foi criado um objeto GraphicPlotList que herda de *QtWidgets.QScrollArea*. Também existia originalmente um botão que criasse uma série de até 4 retas com inclinações aleatórias, para fins de teste. Foi criado um objeto filho pelo motivo explicado abaixo.

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 política 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.

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*.

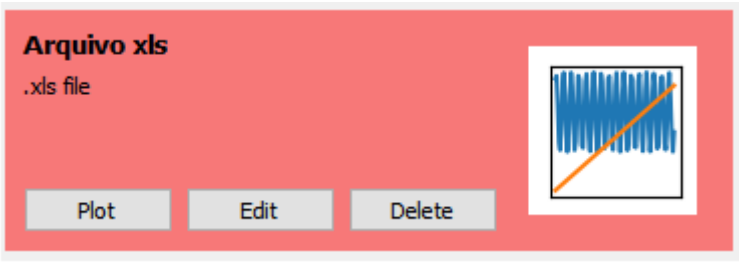


Figura 3.3: *GraphicPlotConfig* não plotado em *MainPlotArea*

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.

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*. Abaixo dele, uma faixa de ferramentas (*NavigationToolbar2QT*) permite que o usuário edite o gráfico plotado, aproxime ou distancie a imagem e salve a figura. Na lista de séries à direita, ao clicar no botão "Plot", as séries respectivas serão desenhadas no Canvas, com legenda.

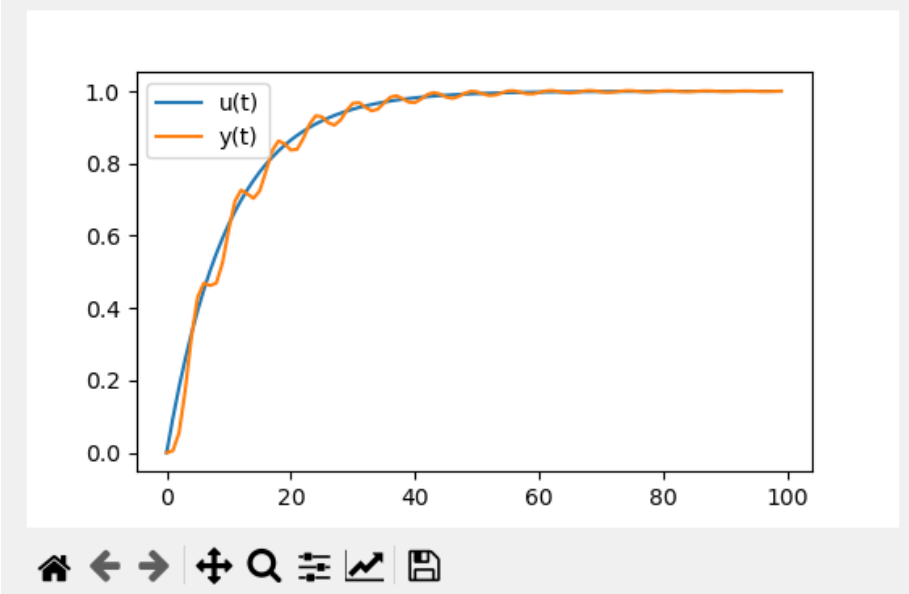


Figura 3.4: *MainPlotArea*

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 dispositivo que permita comunicação serial. Assim, configurados os parâmetros de comunicação (emphbaud 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 em "Puxar dados". Ao fazê-lo, uma caixa diálogo 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.

{Inserir Imagem}

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, principalmente 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 fica 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. Esta separação se fez necessária pois o método do Canvas que o atualiza (emphdraw()) é considerado custoso, e poderia apresentar problemas se operações importantes tivessem que aguardá-la. Outro motivo é exemplificar a estudantes do código uma implementação simples de emph_threads, estruturas que podem ser bastante úteis e que não são abordadas no curso tradicional de Engenharia de Controle e Automação.

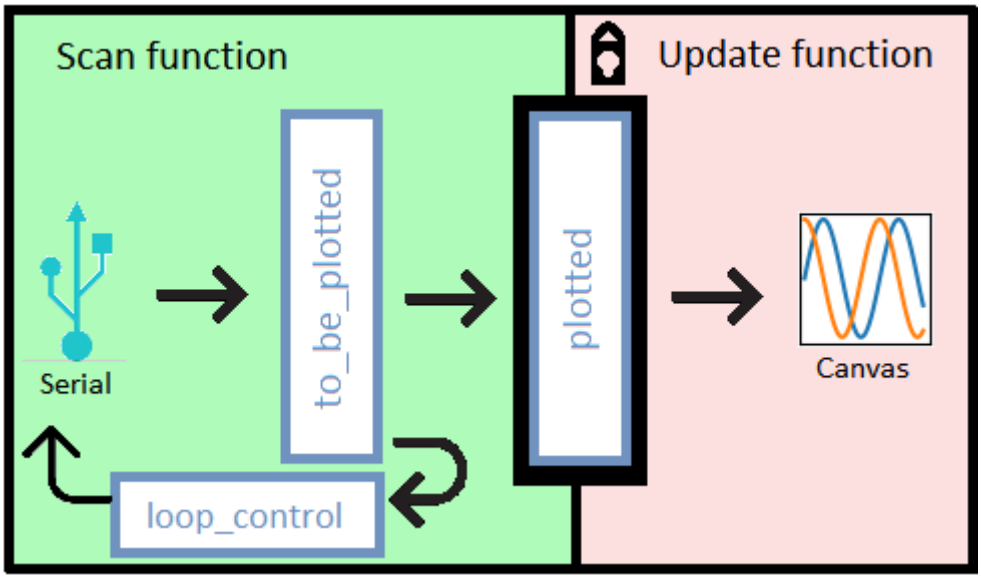
Cada uma delas tem um tempo de repetição, que define a periodicidade que executam suas instruções. O padrão definido foi de 0,1 segundos para ler dados da porta serial, e 1 segundo para atualizar o gráfico com os valores lidos. Quando a leitura da porta ocorre, os valores são armazenados em uma lista chamada emphto_be_plotted. Quando o gráfico é atualizado, os valores desta lista são movidos para outra, chamada emphplotted, que de fato é plotada, enquanto que a primeira é esvaziada. Inclusive esta manipulação compartilhada justifica o emprego de um semáforo.

Como já mencionado, o programa espera que os dados recebidos sejam separados 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, que deve vir do dispositivo conectado. 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 será desconsiderada.

A figura abaixo esquematiza a execução do código:

De forma que o programa possa ser de fato implementado em aulas do curso, foi adicionada a possibilidade de uma reposta 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.

Quadro 3.1: Esquema de leitura serial no supervisório didático



A rotina é feita em duas etapas, uma na função *setup_control*, chamada logo após uma conexão bem-sucedida, e outra na função *loop_control*, chamada logo após a leitura da porta serial, dentro da thread por padrão mais rápida.

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. Isto requer certo conhecimento do usuário, tanto de Python como do dispositivo utilizado. Porém serão disponibilizados códigos de exemplo como auxílio.

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 impede que valores de alta magnitude travem os limites do gráfico, e variações pequenas não sejam bem percebidas. Porém, o usuário pode 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.

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 abaixo o esquematiza:

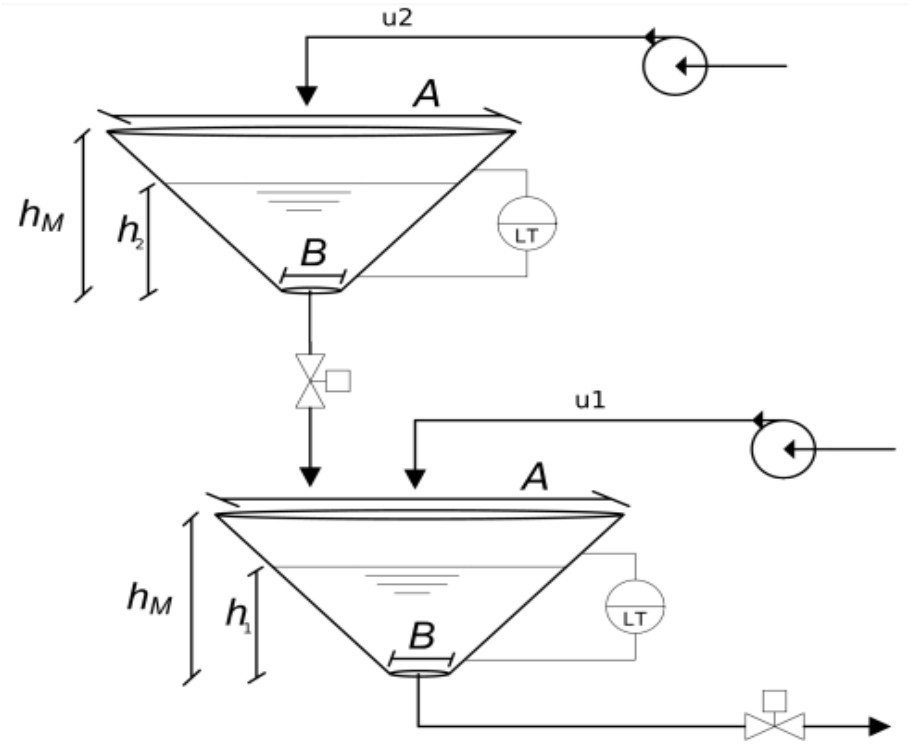


Figura 4.1: Esquema de leitura serial no supervisório didático
Fonte: Elaborado pelo Prof. Daniel Santana

Como percebido pelo esquema, os tanques têm o formato de tronco de cone e a variável controlada é a sua altura. O controle é feito por 2 bombas que aumentam ou diminuem a vazão de entrada de fluido em ambos os tanques. A medição é realizada por dois sensores de nível, sujeito a ruídos brancos.

As equações deste processo são descritas abaixo:

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

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:

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

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

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 ($h_{1ss} = h_{1ss}$).

4.3 Sintonia do Controlador

Por tratar-se de um sistema não linear, serão comparados dois controladores sintonizados por métodos distintos. 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, será utilizada uma lei de controle que anula a não-linearidade do sistema, transformando-o em uma sistema linear. Após este passo, também será utilizado um controlador PID para controlá-lo.

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.

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

Figura 4.2: Série de Taylor truncada na primeira ordem

sendo a expressão $(xi - xi_0)$ chamada de desvio e representada por $\overline{xi}$.

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

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

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

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

De forma a zerar a função $\frac{dh_i}{dt}$ presente na série de Taylor, define-se como um bom ponto de operação um dos estados estacionários do sistema, como por exemplo, $h_{1_0} = h_{1_{ss}} = 1, h_{2_0} = h_{2_{ss}} = 1, u_{1_0} = u_{1_{ss}} = 0, u_{2_0} = u_{2_{ss}} = k\sqrt{\rho g}$. Substituídos os parâmetros de acordo com a tabela 4.1 e em formato de espaço de estados o sistema final linearizado será

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

4.3.2 Sintonia do LQI

4.3.3 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.

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 4.3

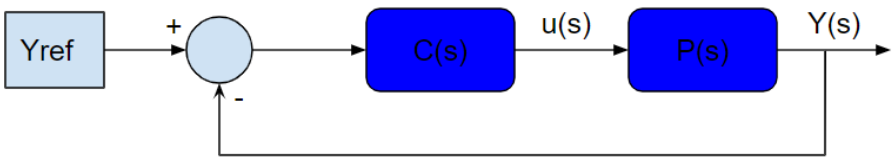


Figura 4.3: Processo 1x1 com feedback e controlador

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

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

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

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

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

$$\overline{h_1}(s) = \frac{0{,}937}{s + 0{,}063}\overline{u_1}(s) + \frac{0{,}046}{s + 0{,}063}\overline{h_2}(s)$$
$$\overline{h_2}(s) = \frac{0{,}937}{s + 0{,}063}\overline{u_2}(s)$$

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

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

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 as suas necessidades.

4.4.1 LQI

4.4.2 Controlador PI

Além da biblioteca *python-control*, existe outra chamada emphsimple-pid que, como o nome indica possui objetos que se comportam como controladores PID. Eles recebem como parâmetrosseus respectivos ganhos, valor de referência da variável controlada, 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 abaixo:

```
self.pids = []
self.pids.append(simple_pid.PID(0.1067, 0.0067, setpoint=2.5, output_limits=(None,
None)))
self.pids.append(simple_pid.PID(0.1067, 0.0067, setpoint=2.5, output_limits=(None,
None)))
```

O objeto 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 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.

Implementando na função *loop_ control()* a lógica descrita acima:

```
def loop_control(self, input_data):
    if len(input_data) == 0:
         return

    signals = [self.pids[i](input_value) for i, input_value in enumerate(input_data)]
    for signal in signals:
        self.porta.write(':.2f'.format(signal).encode('UTF-8'))
    return
```

Para este caso de teste, as restrições do processo, como valores possíveis para a entrada e limite de altura dos tanques, foram tratadas no Arduino. Seu código se encontra no anexo deste documento.

4.5 Resultados

4.5.1 LQI

4.5.2 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. Os tanques foram partidos em $h1 = 1m$ e $h2 = 2m$.

Os resultados se encontram na figura abaixo:

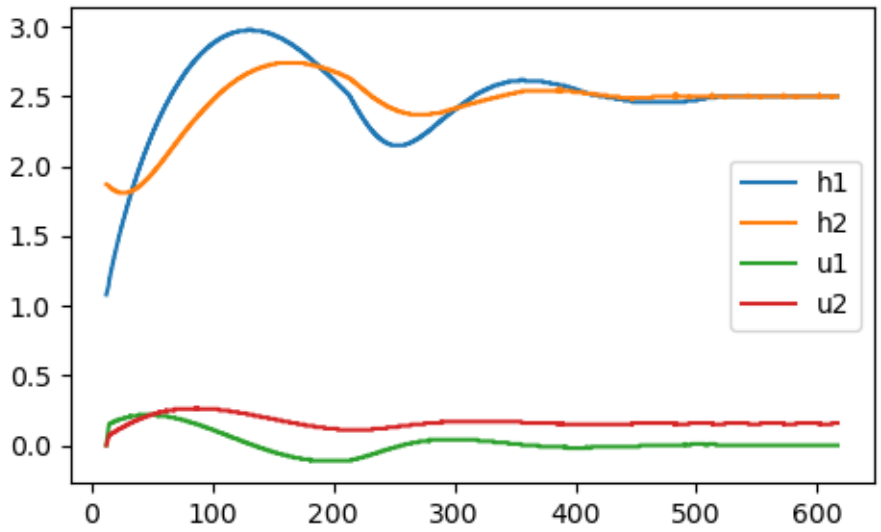


Figura 4.4: Resposta do processo para controlador PI, desconsiderando restrições de processo

Em seu estado estacionário, pode-se afirmar que o sistema se comporta satisfatoriamente. Como calculado no item 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 $k\sqrt{\rho g h_{2_{ss}}} = 0{,}16$.

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. Se possíveis, isto sigificaria que o fluido poderia ser retirado do tanque pelas bombas, o que não ocorre neste caso. Assim, foram incluídas as seguintes restrições no processo:

1. As alturas $h_{1_{ss}}$ e $h_{2_{ss}}$ 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 abaixo:

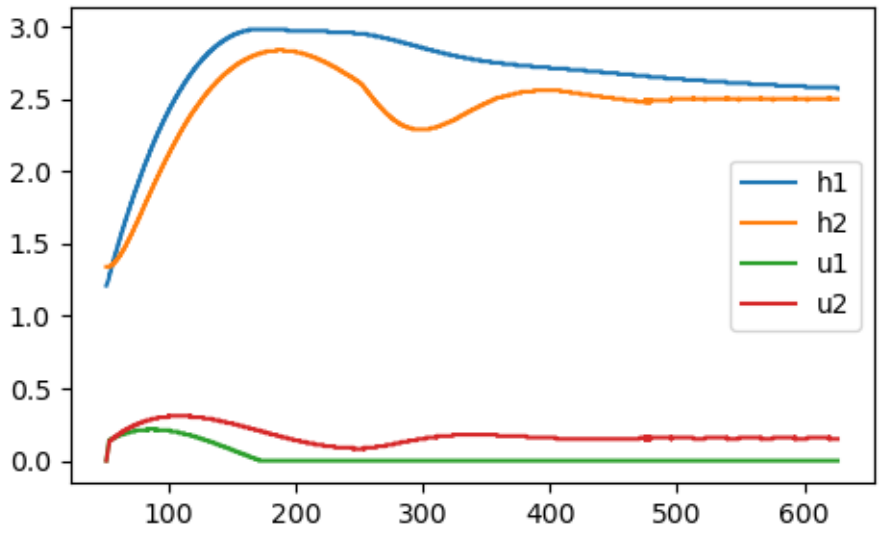


Figura 4.5: Resposta do processo para controlador PI, incluindo restrições de processo

Neste caso, para valores negativos da variável de controle, os mesmo são por sua vez substituídos por 0. Isto faz com que a altura $h_{1_{ss}}$ leve mais tempo para diminuir seu valor, pois depende somente do escoamento por gravidade e da alimentação do

tanque 2. Isso se reflete no tempo de estabilização do processo em si que também é aumentado

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 sem causar impactos negativos.

Apêndice A

Exemplo

A.1 Seção de exemplo

Aqui você escreve!

Para incluir uma abreviatura use: , assim que for utilizada pela primeira vez

Para incluir um símbolo use: , assim que ele for utilizado.