



DESENVOLVIMENTO DE SISTEMA SUPERVISÓRIO *OPEN SOURCE*  
DIDÁTICO

RAFAEL GUIMARÃES DE ARAÚJO LUCENA

Orientador: DANIEL DINIZ SANTANA

Salvador  
Dezembro de 2020

## 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 obsolescência. 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

abstract

**Keywords.** Python, SCADA, open-source



---

# Sumário

---

<b>Lista de Figuras</b>	<b>i</b>
<b>Lista de Tabelas</b>	<b>k</b>
<b>Lista de Quadros</b>	<b>l</b>
<b>1 Introdução</b>	<b>1</b>
1.1 Contextualização e Justificativa . . . . .	1
1.2 Objetivos . . . . .	2
1.2.1 Objetivos Gerais . . . . .	2
1.2.2 Objetivos Específicos . . . . .	3
1.3 Estrutura do Trabalho . . . . .	3
<b>2 Fundamentação Teórica</b>	<b>5</b>
2.1 Modelagem de Processos . . . . .	5
2.1.1 Linearização de sistemas . . . . .	6
2.2 Controladores . . . . .	6
2.2.1 PID . . . . .	6
2.2.2 LQR . . . . .	7
2.3 SCADA . . . . .	8
2.4 Comunicação Serial . . . . .	10
2.5 Qt em Python . . . . .	12
<b>3 Desenvolvimento do Sistema Supervisório</b>	<b>15</b>
3.1 Requisitos do Sistema . . . . .	15
3.2 Seleção das Tecnologias . . . . .	15
3.2.1 Seleção das bibliotecas . . . . .	16
3.3 Criando a interface gráfica . . . . .	18
3.4 <i>Dataset Config</i> . . . . .	19

3.5	<i>PlotManager</i> . . . . .	21
3.6	<i>MainPlotArea</i> . . . . .	23
3.7	<i>SCADADialog</i> . . . . .	23
3.8	Salvamento Automático de Séries . . . . .	26
<b>4</b>	<b>validação do SUPervisório Didático</b>	<b>29</b>
4.1	Apresentação do Sistema . . . . .	29
4.2	Comportamento esperado . . . . .	31
4.2.1	Linearização do Sistema . . . . .	32
4.2.2	Sintonia do PI . . . . .	33
4.2.3	Sintonia do LQR . . . . .	33
4.3	Configuração do supervisório didático . . . . .	34
4.3.1	Controlador PI . . . . .	34
4.3.2	LQR . . . . .	35
4.4	Validação dos dados . . . . .	36
4.5	Considerações Finais . . . . .	40
<b>5</b>	<b>Conclusão</b>	<b>41</b>
5.1	Trabalhos futuros . . . . .	42
	<b>Referências</b>	<b>43</b>
	<b>Anexos e Apêndices</b>	<b>44</b>
<b>A</b>	<b>Códigos</b>	<b>45</b>
A.1	Código do arduino para Controle de Tanque com Área Variável . . . .	45
<b>B</b>	<b>Tutorial de utilização do supervisório didático</b>	<b>49</b>
B.1	Apresentação . . . . .	49
B.2	Primeira utilização . . . . .	49
B.3	Iniciando o programa . . . . .	50
B.4	Importando séries estáticas no programa . . . . .	50
B.4.1	Função de Transferência . . . . .	50
B.4.2	Entrada por arquivo . . . . .	51
B.4.3	Serial . . . . .	52

<i>SUMÁRIO</i>	g
B.4.4 Script Python . . . . .	52
B.5 Monitoramento em tempo real . . . . .	53
B.6 Salvando e editando séries de dados . . . . .	54
B.7 Plotando séries . . . . .	54
B.8 Editando e exportando gráficos . . . . .	54
B.9 Deletando séries . . . . .	55
<b>C Códigos</b>	<b>57</b>





---

# Lista de Figuras

---

2.1	Processo 1x1 com feedback e controlador . . . . .	7
2.2	Tela exemplo de um sistema supervisório . . . . .	9
2.3	Pirâmide da automação . . . . .	10
2.4	Exemplo de transmissão serial de uma sequência de 3 bytes . . . . .	11
2.5	Janela <i>QMainWindow</i> com um <i>QGroupBox</i> como Widget central, contendo vários outros Widgets em um layout <i>QGridLayout</i> dentro de outro layout <i>QHBoxLayout</i> . . . . .	14
3.1	Supervisório didático e seus objetos principais . . . . .	19
3.2	<i>ModelSeriesDialog</i> : Caixa diálogo para edição dos eixos e título das séries . . . . .	21
3.3	<i>GraphicPlotConfig</i> não plotado em <i>MainPlotArea</i> . . . . .	22
3.4	<i>MainPlotArea</i> . . . . .	23
3.5	Esquema de leitura serial no supervisório didático . . . . .	25
4.1	Esquema de leitura serial no supervisório didático . . . . .	30
4.2	Sistema partido no estado estacionário . . . . .	32
4.3	Interface com as séries salvas . . . . .	37
4.4	Resposta do processo para controlador PI, incluindo restrições de processo . . . . .	38
4.5	Resposta do LQR no caso 1 . . . . .	38
4.6	Resposta do LQR no caso 2 . . . . .	39
4.7	Resposta do LQR no caso 2 . . . . .	40
B.1	Objeto <i>TransferFunctionConfig</i> (à esquerda) . . . . .	51
C.1	Diagrama de relações entre os objetos empregados e funções principais	58



---

# Lista de Tabelas

---

4.1	Tabela de parâmetros e variáveis do sistema . . . . .	30
4.2	Tabela de valores de um dos estados estacionário do sistema . . . . .	31



---

# Lista de Quadros

---

3.1	Bibliotecas Python utilizadas no desenvolvimento do supervisorio di-	
	dático . . . . .	17



---

# Capítulo 1

## Introdução

---

### 1.1 Contextualização e Justificativa

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. Segundo Júnior (2019), 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 modernizaram o suficiente para tomar o espaço industrial, no que diz respeito ao controle e monitoramento de processos.

Neste âmbito da modernização industrial, surgiram os sistemas SCADA (*Supervisory Control And Data Acquisition*), também chamados de sistemas supervisórios. Dentre alguns exemplos de softwares da área, se encontram o Elipse E3 e o SIMATIC WinCC, que podem ser executados em computadores comuns e se comunicam com outros dispositivos eletrônicos via protocolos de comunicação como Modbus e Profibus ((MANUAL..., ), (SIEMENS, 1999)). 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.

Martins (2007) afirma que os dados adquiridos de controladores espalhados numa área industrial podem ser manipulados, de forma a servir de insumos para definição de *setpoints*. Num processo de aquecimento, por exemplo, a medição de temperaturas baixas pode enviar ao sistema de controle um setpoint maior que o desejado, com o objetivo forçar um pouco mais o sistema de controle, ao menos ate

atingir valores mais próximos do desejado.

Segundo Júnior (2019), o valor de um sistema SCADA está relacionado com o conceito de software aberto. O autor descreve este conceito, como um software o mais compatível possível com os hardwares com os quais se comunicaria e com os diferentes sistemas operacionais que podem executá-lo. Além disto, o software deve ser modular, executado em módulos, de forma que o mal-funcionamento de uma de suas partes não impacte negativamente a execução das outras. 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, como descrita por Moraes (2016). Os chamados softwares *open source* são gratuitos, e seus códigos-fontes podem ser compartilhados, modificados e aplicados à vontade.

Um exemplo software de modelagens computacionais, utilizado na engenharia, é o chamado MATLAB®. Ele oferece um pacote voltado para estudantes que contém, entre outros módulos, o *Simulink* para modelagem e simulação de processos e ferramentas para *machine learning*. O valor deste conjunto, segundo sua loja virtual em 03/12/20 era de USD 55,00. Se tratando de softwares supervisórios, o WinCC, fabricado pela Siemens é vendido por um de seus distribuidores, na versão Flexible por mais que R\$ 14.000,00 (<https://www.shmr.com.br/software1/>).

Dados os valores de softwares de engenharia, e a difusão de tecnologias *open source*, existe uma necessidade e oportunidade de instituições de ensino aderirem à utilização de ferramentas gratuitas, desenvolvidas pelos próprios estudantes. As vantagens não são somente financeiras, pois a construção destas ferramentas acrescenta profissionalmente a todos os envolvidos.

## 1.2 Objetivos

### 1.2.1 Objetivos Gerais

Desenvolver um sistema supervisório didático capaz de se comunicar com dispositivos externos que lhe fornecessem dados numéricos para plotagem em sua interface



em tempo real, além de possibilitar a simulação de sistemas diversos e comparação com modelagens formuladas pelos seus usuários.

### 1.2.2 Objetivos Específicos

Com o foco nos objetivos supracitados, foram levantados os seguintes objetivos específico direcionados desenvolvimento do supervisório didático:

- Definir os requisitos de comunicação;
- Projetar e desenvolver a interface gráfica e os eventos que coordenam o funcionamento do sistema;
- Mapear e documentar o software para facilitar futuras melhorias;
- Validar o funcionamento do supervisório com uma aplicação prática.

## 1.3 Estrutura do Trabalho

Este trabalho foi dividido 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, contendo uma descrição curta dos softwares SCADA existentes, das tecnologias que podem ser utilizadas para a criação de um supervisório e finalmente um resumo do método selecionado para a construção da ferramenta
3. O capítulo 3 trata da programação do supervisório em si, mostrando sua arquitetura e convenções empregadas com figuras e esquemas.
4. O capítulo 4 toma o sistema pronto e o emprega em uma aplicação prática, 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

Segundo Katsuhiko (2010), a dinâmica de muitos sistemas mecânicos, elétricos, térmicos, entre outros pode ser descrita em termos de equações diferenciais. Estas equações são obtidas pelas leis físicas que o regem, como a equação Bernoulli para dinâmica de fluidos. Para modelar um sistema por tais equações, levanta-se o nível de detalhes esperado, já que nem todas as variáveis de uma função tem impacto significativo na sua resposta. Um exemplo deste cenário seria a resistência do ar para corpos em queda livre, em alturas pequenas.

De acordo com Lathi (2007), um sistema, após modelado, pode traduzir-se 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 mais de uma entrada é a soma cada resposta se calculada individualmente. Isto possibilita solucionar equações complexas a partir do cálculo de partes da mesma, simplificando a solução.

Segundo Katsuhiko (2010), uma forma comum de representação de um sistema linear invariante no tempo é pelas chamadas funções de transferências. Sendo  $y(t)$  a função que descreve uma das saídas de um processo, e  $x(t)$  uma das entradas, a função de transferência  $G(s)$  traduz a influência de  $x$  em  $y$ . Este formato sempre relaciona uma saída a uma entrada, e é sempre representado pela razão entre a transformada de Laplace da primeira sobre a segunda, ou:

$$\frac{\mathcal{L}(y)}{\mathcal{L}(x)} = \frac{y(s)}{x(s)} \quad (2.1)$$

### 2.1.1 Linearização de sistemas

Enquanto que sistemas lineares tem propriedades que facilitam sua modelagem, de acordo com Smith e Corripio (2006), sistemas não lineares não possuem técnicas tão diretas para analisar suas dinâmicas. Assim, podem ser empregadas neles transformações para sistemas lineares equivalentes.

Uma delas é a chamada linearização, que emprega a série de Taylor, truncando no segundo termo, ou

$$f(x_1, x_2, \dots, x_n) \simeq f(x_{i0}, x_{20}, \dots, x_{n0}) + \left( \sum_{i=1}^n \frac{df}{dx_i} \Big|_{x_i=x_{i0}} (x_i - x_{i0}) \right) \quad (2.2)$$

, sendo  $x_i$  os parâmetros da função descritiva  $f$ , e  $(x_{i0})$  um ponto de operação. Obtém-se, assim, uma função linearizada em torno de um determinado ponto 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 gerador não linear.

## 2.2 Controladores

Segundo Lourenço (1997), não é possível determinar o tipo de controlador a se usar num determinado processo. Idealmente, o controlador mais simples, que satisfaça a "resposta desejada" deve 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.

Smith e Corripio (2006) descrevem o controle por realimentação como o emprego de um controlador que monitora uma variável (a variável controlada do processo), compara o valor lido com o valor desejado, o *setpoint*, e computa o sinal de controle a ser enviado para o sistema, através de uma variável manipulada.

### 2.2.1 PID

Segundo Katsuhiko (2010), a utilidade dos controladores do tipo PID (Proporcional, Integral e Derivativo) está na sua aplicabilidade geral à maioria dos sistemas. Por conta disto, é um controlador bem popular. Ele recebe como sinal o erro  $e$  de um

sistema, que é a diferença entre o valor da variável controlada e seu setpoint. A partir disto, calcula o sinal de controle como:

$$u(t) = K_p e(t) + K_i \int e(t) + K_d \frac{de(t)}{dt} \quad (2.3)$$

, a partir de seus ganhos  $K_p$ ,  $K_i$  e  $K_d$  (proporcional, integral, derivativo).

Ainda de acordo com Katsuhiko (2010), a definição dos ganhos de um controlador é um processo chamado sintonia, e pode ocorrer por vários métodos. Para um PID, exemplificam-se Ziegler-Nichols, otimização e resposta em frequência. Smith e Corripio (2006) descrevem um outro método, denominado síntese: dada uma função de transferência conhecida  $P(s)$  de primeira ordem para um determinado processo, a função de transferência entre o controlador e seu sinal de controle será, em malha fechada (Figura 2.1):

$$C(s) = \frac{u(s)}{e(s)} = \frac{1}{P(s)} \frac{1}{\tau_c s} \quad (2.4)$$

, com  $\tau_c$  sendo o único parâmetro de sintonização, e representando o tempo que o sistema controlado deve levar até atingir 62,3% do seu estado estacionário. Assim, se

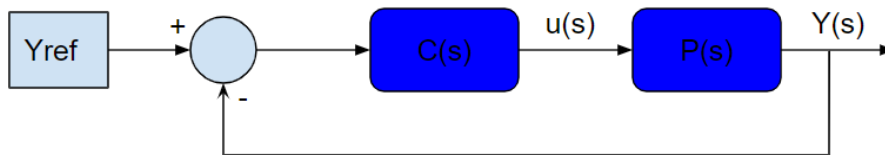
$$P(s) = \frac{G}{\tau s + 1} \quad (2.5)$$

, então

$$C(s) = K_p + K_i \frac{1}{s} = \frac{\tau}{G\tau_c} + \frac{1}{G\tau_c} \frac{1}{s} \quad (2.6)$$

, sendo  $\tau$  e  $G$  o tempo de resposta e o ganho natural do sistema, respectivamente.

Figura 2.1: Processo 1x1 com feedback e controlador



## 2.2.2 LQR

O Regulador Linear Quadrático, ou LQR, é um controlador de sintonia mais simples que o PID. Ele realiza um controle regulatório, ou seja, seu setpoint é sempre a origem (todos os estados e entradas em zero). Quanto à sua sintonia, segundo

control (2019), o controlador recebe 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:

$$\dot{x}(t) = A.x(t) + B.u(t) \quad (2.7)$$

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  $J$  2.8:

$$J = \int_0^{\infty} (x'Qx + u'Ru)dt \quad (2.8)$$

Segundo (Argentim et al., 2013), a função de controle por realimentação  $u$ , para o caso do LQR, é representada pela equação 2.9:

$$u = -Kx(t) \quad (2.9)$$

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 da equação 2.8.

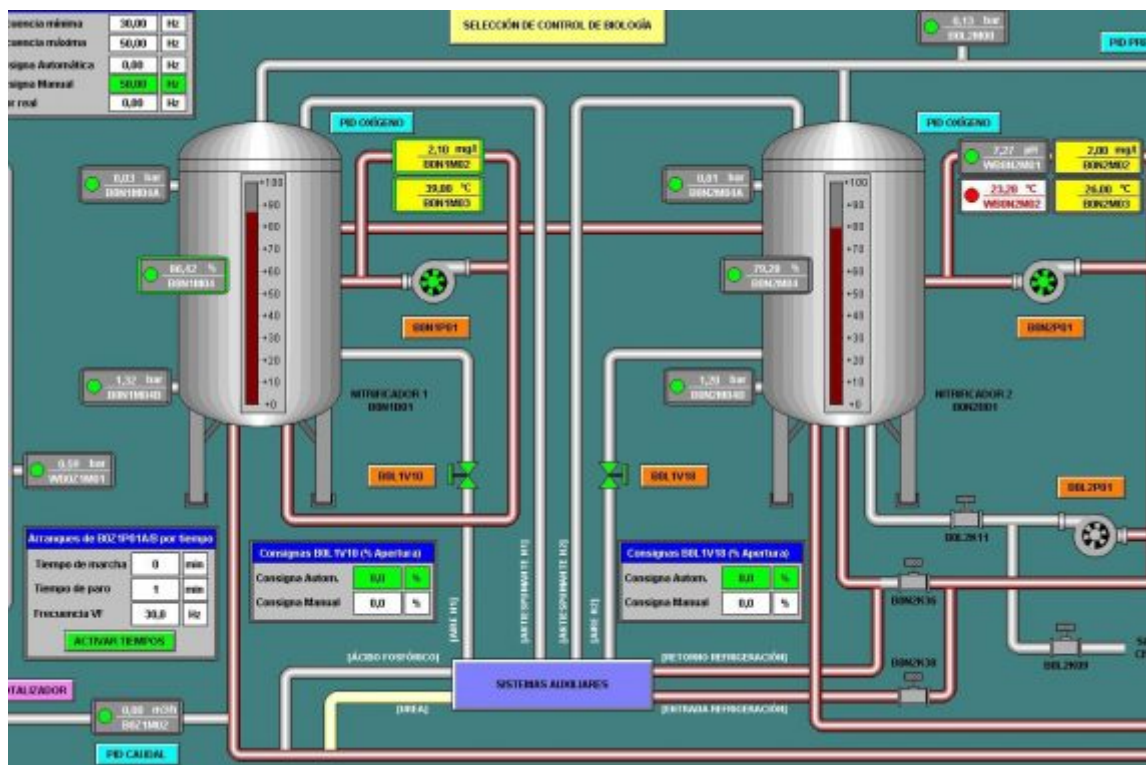
## 2.3 SCADA

De acordo com Martins (2007), 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.

Sistemas SCADAs são responsáveis por buscar informações de controladores e equipamentos diversos de automação, e manipular estas informações de acordo com o que foi programado. As aplicações mais simples se constituem na visualização dinâmica destes dados através de mostradores, cores, escalas, entre outros objetos gráficos em telas pré programadas. Uma estratégia, por exemplo, é a de literalmente

desenhar todo um sistema produtivo através imagens já embutidas na biblioteca do software editor, e incluir nela todas as medições acessíveis, mapeando-as por onde se encontram no desenho. Outra seria a de agrupar as informações por temática, e mostrar diversas telas menos detalhadas, que alternem entre si de acordo com um temporizador interno. A Figura 2.2 ilustra um exemplo de tela para um sistema supervisório.

Figura 2.2: Tela exemplo de um sistema supervisório

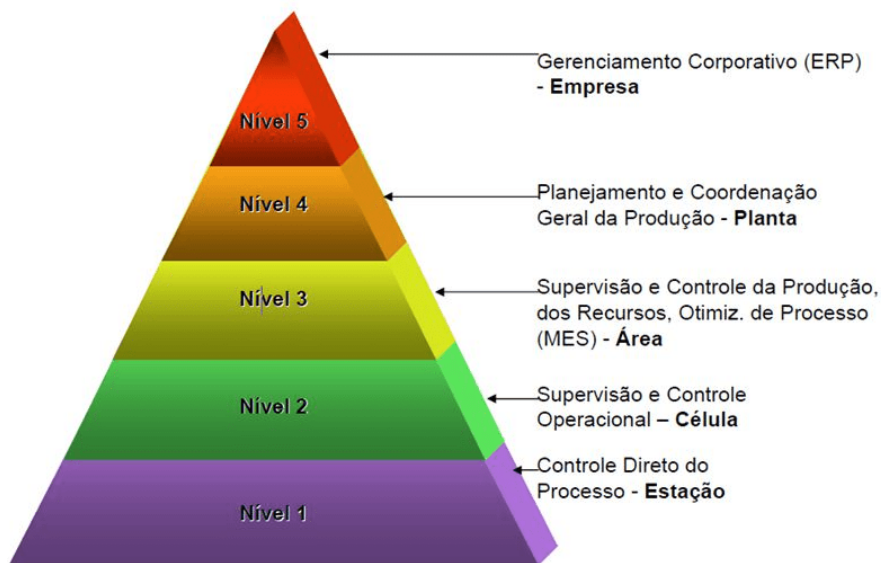


Fonte: <https://www.agaads.com/service/scada-system/>

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. Isto se relaciona com o conceito já mencionado de software livre, descrito por Júnior (2019). Por possuir estas funcionalidades, software SCADAs ultrapassam o nível 2 na pirâmide de automação (Figura 2.3), chegando ao nível de supervisão da produção, pois agrupa as informações de diversos controladores e sensores em um só local, e gera suporte para ações de gerência.

Segundo Roggia e Fuentes (2016), dentre os principais benefícios do uso de sis-

Figura 2.3: Pirâmide da automação



Fonte:

<https://www.logiquesistemas.com.br/blog/piramide-de-automacao-industrial/attachment/354/>

temas 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.

## 2.4 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 e 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

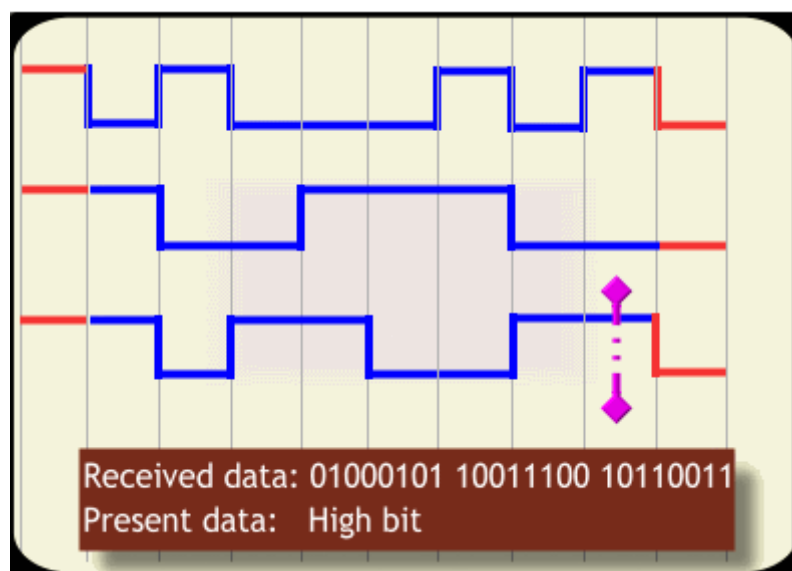


fabricantes utilizam chips de circuitos integrados que manipulam o fluxo de dados, facilitando a escrita de scripts de comunicação.

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 2.4, e fundamentado por Mazidi, Mckinlay e 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.

Figura 2.4: Exemplo de transmissão serial de uma sequência de 3 bytes



Fonte: <http://electrosofts.com/parallel/>

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

## 2.5 Qt em Python

Segundo seu criador (informação verbal, (ROSSUM, 2003)), 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 Python: 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 agradável, 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 sua enquete, Developer... () avaliou Python como a 2<sup>a</sup> “linguagem mais amada” pelo público, atrás de Rust.

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 exemplos, cita-se as libs SQLAlchemy (BAYER, 2012), que permite criação e acesso a bancos de dados leves; NumPy, uma poderosa ferramenta para cálculos matriciais (HARRIS et al., 2020); e PyQt5, que possui objetos e métodos para criação de interfaces gráficas (PYQT, 2020).

Segundo sua documentação (PYQT, 2020), a biblioteca PyQt5 veio da bibli-

oteca 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 utilizados para criação de GUIs locais:

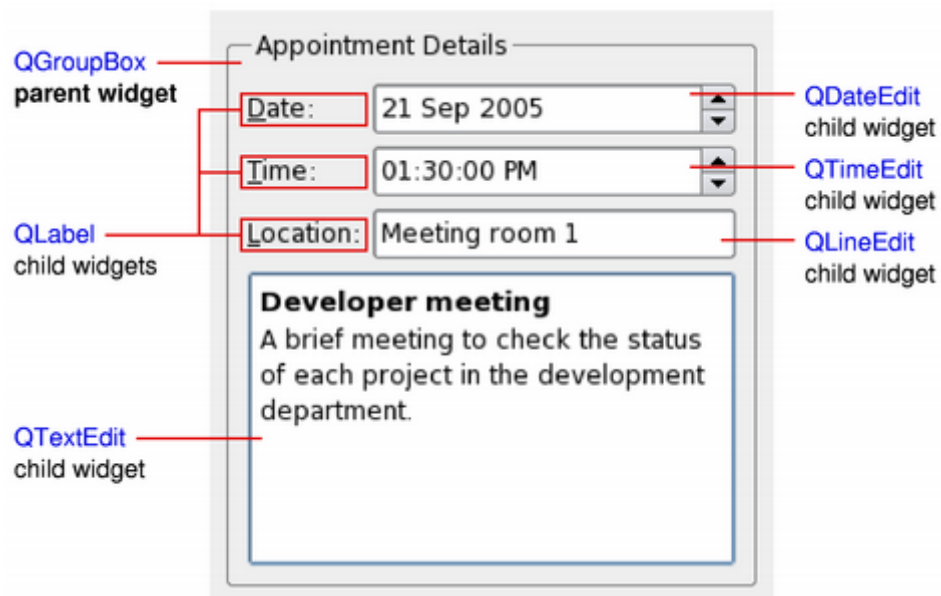
- **QtWidgets:** engloba os objetos gráficos principais, como botões, textos e layouts. O objeto genérico *QWidget*, herdado por diversos outros deste módulo, tem funções cruciais relativas ao posicionamento, geometria, visibilidade e estilo dos objetos gráficos.
- **QtCore:** este módulo lida com eventos 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 chamados de Widgets, e herdam da classe *QWidget*. Por conta da arquitetura em objetos da biblioteca, o desenvolvimento de supervisor didático seguiu a filosofia de incluir um objeto dentro de outro. Assim, foi possível definir melhor o espaço e as condições de redimensionamento dos objetos quando a janela for esticada ou comprimida, pois cada layout redimensiona apenas os Widgets que contém, dentro do espaço disponível.

A Figura 2.5 ilustra a organização padrão de uma aplicação em PyQt, listando também alguns objetos populares.

Figura 2.5: Janela *QMainWindow* com um *QGroupBox* como Widget central, contendo vários outros Widgets em um layout *QGridLayout* dentro de outro layout *QHBoxLayout*



Fonte: <https://doc.qt.io/qt-5/qwidget.html>

---

## 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 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 Eclipse E3. Já houveram trabalhos, inclusive, utilizando a primeira tecnologia (MORAES, 2016).

Como atestado em Moraes (2016), o ScadaBR é construído em um servidor web, utilizando geralmente o Apache TomCat (como servidor web). É um software *open source* e existem vários materiais *online*, tanto escritos como em vídeo orientando interessados em como utilizá-lo. Porém, os protocolos de comunicação

pré-configurados são Modbus (IP e Serial) e OPC. Para aplicações com arduino, o usuário teria que criar uma função que enviasse os dados ao sistema no formato correto.

Quanto à versão demo do Elipse E3, se trata de uma ferramenta muito limitada. Segundo 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. Adicionalmente, 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 mais facilmente escalável, em questão de tecnologias atuais, foi decidido construir um sistema SCADA em Python. Desta maneira, o código-fonte da aplicação seria aberto, compatível com diferentes sistemas operacionais, 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 modernas, como por exemplo, armazenamento em nuvem.

### **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 supervisor 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 objetos nas janelas principal e periféricas (PYQT, 2020)
PyQtChart	Conjunto de conectores de python para a biblioteca Qt
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 (HUNTER, 2007)
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 (HARRIS et al., 2020)
pyserial	Permite a conexão com dispositivos externos pela porta serial e contém funções de escrita e leitura desta porta (LIECHTI, 2020)
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 (CONTROL, 2019)
pickle	Responsável pela 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 esperam-se que futuros usuários também editem o código contido, por motivos explanados posteriormente neste documento.

No que concerne a interface gráfica do supervisor, 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 uma pequena *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 assumiria várias dimensões, a depender da variável plotada, 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.

Além dos Widgets supracitados, para simplificar o código e torná-lo mais prático, foi criado um objeto abstrato *SeriesObject*. Ele armazena um agrupamento de série



Figura 3.1: Supervisório didático e seus objetos principais



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 supervisorio, 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.

Como o supervisor foi desenvolvido para a área didática da automação, ele 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*.

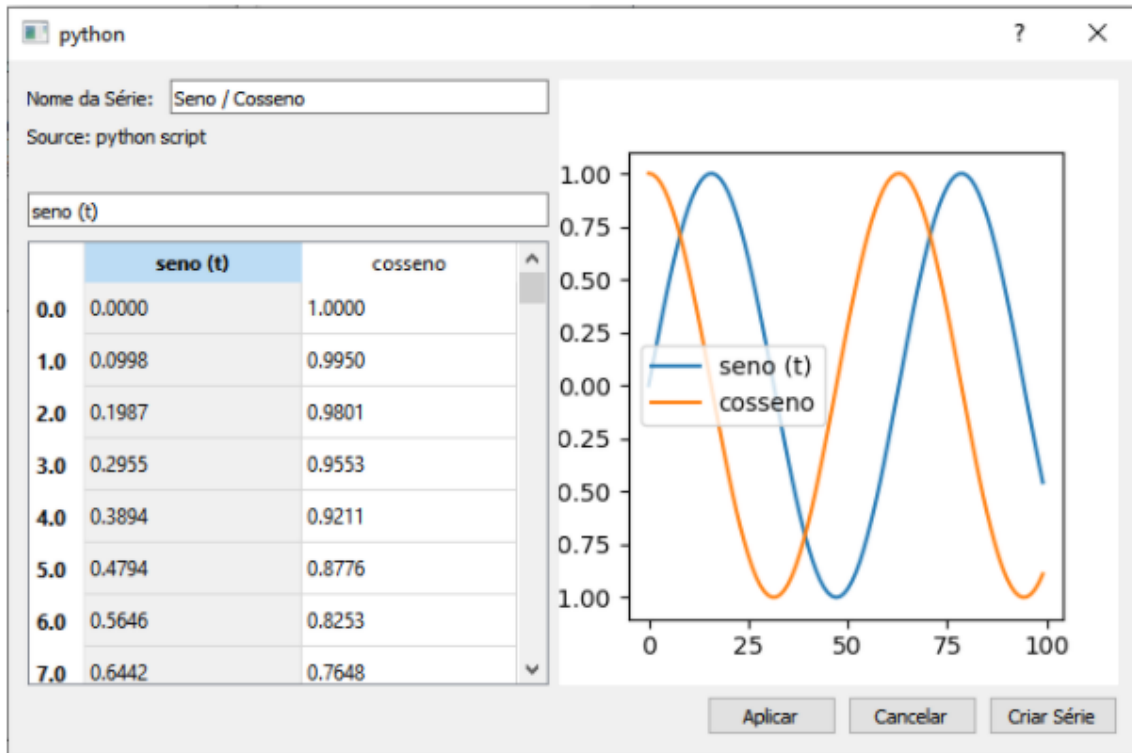
Resumindo, existem quatro maneiras de importar ou gerar uma série de dados no software desenvolvido:

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.

Esta 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.

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



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.

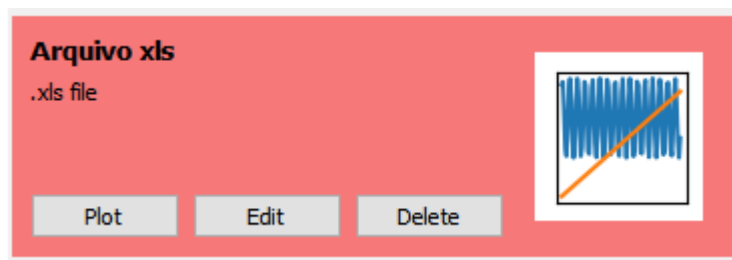
### 3.5 *PlotManager*

Como consta na Figura 3.1 o 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*.

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).

A principal função deste objeto é identificar pelo nome e fonte a série que contém,

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



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 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).

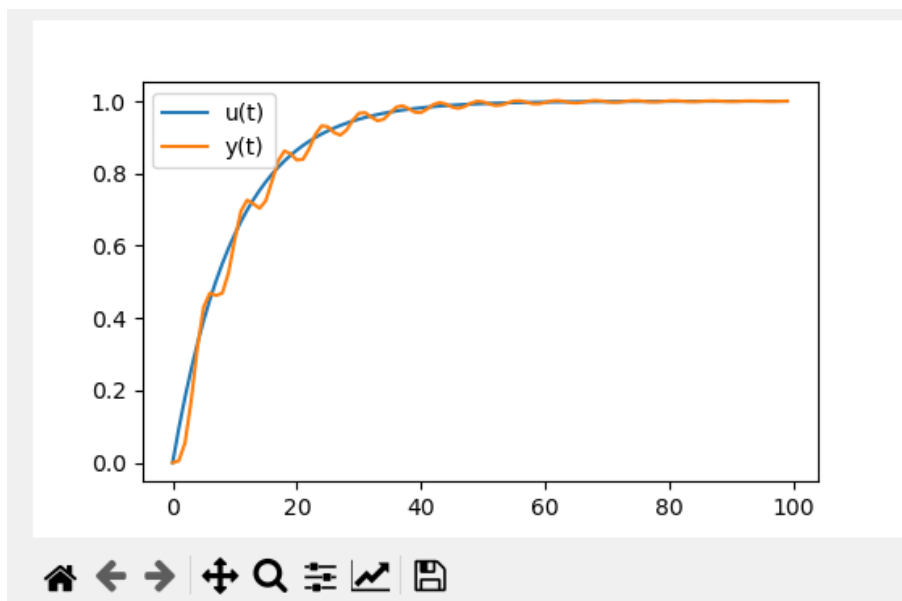
Essa dinâmica torna necessário que o método seja sobrecarregado quando o programador deseje customizar a aparência de um botão ou o seu plano de fundo. A desvantagem é que, por sobrecarregar o método que pinta o objeto na interface, o objeto deve ser repintado manualmente. No caso de um botão, por exemplo, 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.

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 difícil.

### 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 *SCADA Dialog*

Um sistema supervisório, como o apresentado neste documento, monitora dados de controladores geralmente industriais em tempo real. Por se tratar de um software didático, foi incluído no programa um suporte para futuras comunicações com um Arduino 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 da comunicação com periféricos implementa duas *threads* da biblioteca nativa `_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 útil e que não é abordadas nos cursos tradicionais de Engenharia (excetuando computação).

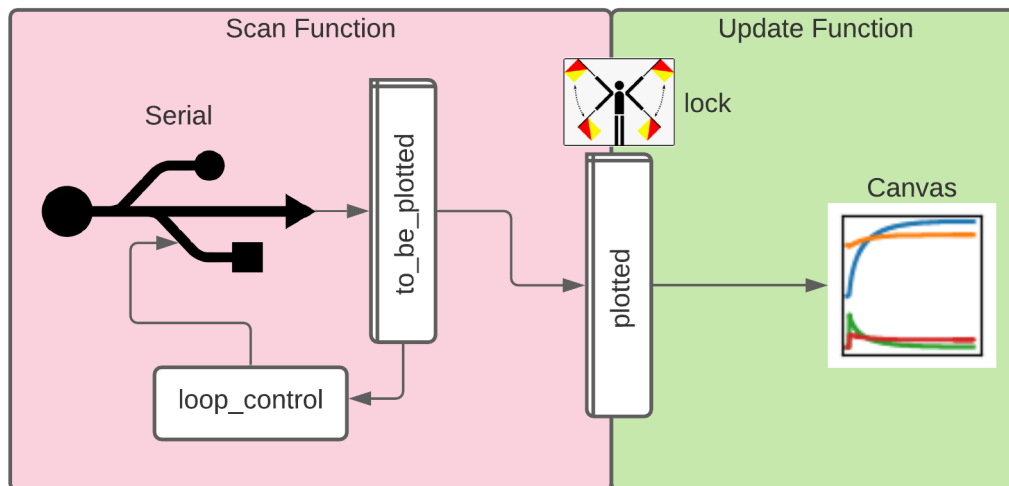
Cada uma das threads tem um tempo de repetição, que define a periodicidade que

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 `to_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 supervisor didático



Para que o programa possa ser de fato implementado em aulas, foram adicionadas duas funções que possibilitam de envio de dados ao dispositivo conectado a cada leitura da porta serial. Isto seria uma analogia a um controlador que atuasse num processo físico, medido por tal dispositivo. Desta forma, o último simularia o sistema em si, enquanto que parte do supervisor didático assumiria o papel de controlador.

Devido à popularidade do microcontrolador Arduino, a implementação da rotina de controle dentro do software 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 com o dispositivo, 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 ou informaria à "planta" os parâmetros de seus processos, enquanto que a segunda função receberia do supervisor a última linha lida da porta, calcularia a resposta do controlador PID, e escreveria o resultado 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 em si. 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 Python 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 supervisor 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.

A Figura C.1 no Apêndice C ilustra todos os objetos utilizados na construção do software, suas relações e principais métodos. No Apêndice B se encontra um tutorial de utilização do supervisor didático.



---

# Capítulo 4

## validação do SUPervisório Didático

---

Nesta seção, serão utilizados 2 controladores devidamente sintonizados em um sistema simulado, cujas equações descritivas são conhecidas. No supervisório didático, existem mecanismos que simulam um controlador, sendo a finalidade deste processo validar os valores retornados pelo arduino, como respostas dos sinais de controle, e concluir sobre a empregabilidade do software supervisório.

### 4.1 Apresentação do Sistema

O sistema simulado pelo arduino se trata de dois tanques de área variável acoplados e alimentados por uma vazão controlável. A Figura 4.1 os esquematiza.

Os tanques têm o formato de tronco de cone e as variáveis controladas são suas alturas em metro. As variáveis manipuladas são as vazões de entrada de fluido em ambos os tanques.

As equações deste processo são descritas por

$$\frac{dh_1}{dt} = \frac{1}{\beta(h_1)}(u_1 - k\sqrt{\rho gh_1} + k\sqrt{\rho gh_2}) \quad (4.1)$$

$$\frac{dh_2}{dt} = \frac{1}{\beta(h_2)}(u_2 - k\sqrt{\rho gh_2}) \quad (4.2)$$

$$\beta(h_i) = \frac{dV}{dh_i} \quad (4.3)$$

$$V(h_i) = \frac{\pi\gamma^2}{3}(h_i + \frac{B}{2\gamma})^3 - \frac{\pi}{3\gamma}(\frac{B}{2})^3 \quad (4.4)$$

$$\gamma = \frac{A - B}{2h_M} \quad (4.5)$$

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

Figura 4.1: Esquema de leitura serial no supervisório didático

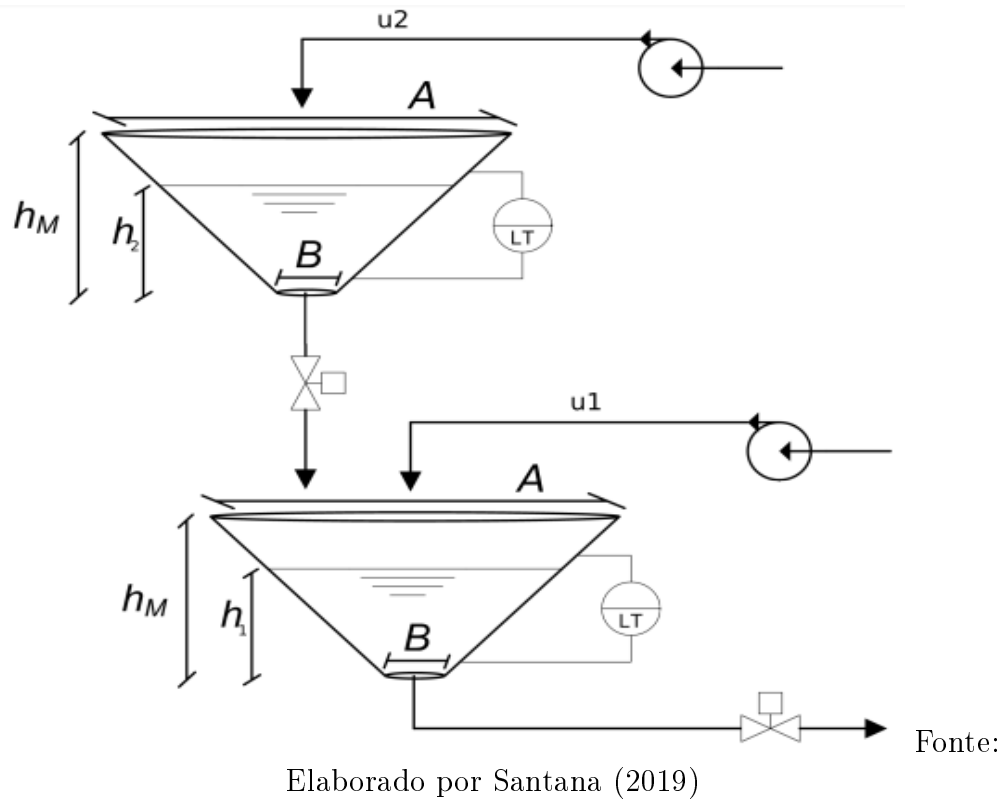


Tabela 4.1: Tabela de parâmetros e variáveis do sistema

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)
$\rho$	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	- ( $m^3/s$ )

## 4.2 Comportamento esperado

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

Com as derivadas zeradas nas equações descritivas 4.5, tem-se que:

$$h_{1ss} = \left( \frac{u_{1ss}}{k\sqrt{\rho g}} + \sqrt{h_{2ss}} \right)^2 \quad (4.6)$$

$$h_{2ss} = \frac{u_{2ss}^2}{k^2 \rho g} \quad (4.7)$$

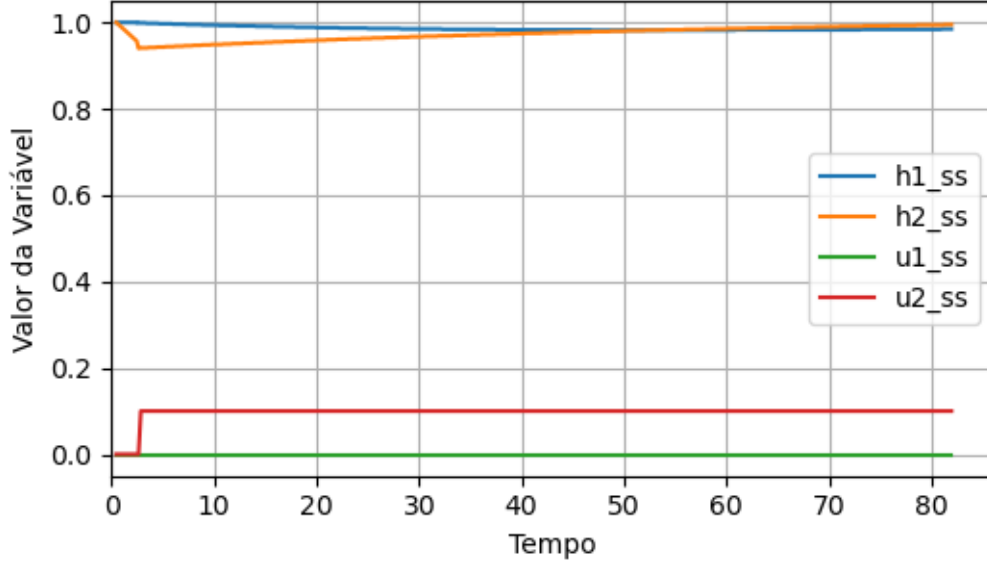
, o que resultam em infinitos valores representando estados estacionários deste sistema. Para o processo de linearização, contemplado na próxima subseção, ao menos um ponto deve ser definidos. Partindo de um valor 0 para a vazão de alimentação do tanque 2, no topo, e uma altura final de 1 metro para o mesmo, calculam-se o restante dos estados estacionários na tabela 4.2.

Tabela 4.2: Tabela de valores de um dos estados estacionário do sistema

Variável	Valor (u.m.)
$h_{1ss}$	1 (m)
$h_{2ss}$	1 (m)
$u_{1ss}$	0 (m <sup>3</sup> /s)
$u_{2ss}$	0,099 (m <sup>3</sup> /s)

Para validar estes valores, é possível usar o supervisor didático como auxílio. Para isto, programa-se o arduino para simular o sistema, partindo dos pontos na tabela 4.2, enquanto que o módulo do controlador no programa envia os valores constantes de  $u_{1ss}$  e  $u_{2ss}$ . Executada a simulação, a Figura 4.2 mostra o resultado. Nota-se a altura  $h_2$  do sistema cai ao iniciá-lo, mas isso se deu ao atraso de comunicação entre o arduino e o computador. Mesmo assim, o sistema retorna para uma região próxima à que foi iniciado, momento no qual não apresenta outras variações significativas.

Figura 4.2: Sistema partido no estado estacionário



#### 4.2.1 Linearização do Sistema

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

$$\frac{dh_1}{dt} = -\left(\frac{d\beta^{-1}(h)}{dh}\right)\bigg|_{h_{1ss}} \left(k\sqrt{\rho gh_{1ss}} + \frac{\beta^{-1}(h_{1ss})k\sqrt{\rho g}}{2\sqrt{h_{1ss}}}\right)\bar{h}_1 + \left(\frac{\beta^{-1}(h_{1ss})k\sqrt{\rho g}}{2\sqrt{h_{2ss}}}\right)\bar{h}_2 + \beta^{-1}(h_{1ss})\bar{u}_1 \quad (4.8)$$

$$\frac{dh_2}{dt} = -\left(\frac{d\beta^{-1}(h)}{dh}\right)\bigg|_{h_{2ss}} \left(k\sqrt{\rho gh_{2ss}} + \frac{\beta^{-1}(h_{2ss})k\sqrt{\rho g}}{2\sqrt{h_{2ss}}}\right)\bar{h}_2 + \beta^{-1}(h_{1ss})\bar{u}_2 \quad (4.9)$$

$$\beta^{-1}(h) = \frac{4}{4\pi(\gamma h)^2 + 4B\gamma h + B^2} \quad (4.10)$$

$$\frac{d\beta^{-1}(h)}{dh} = -4\frac{8\pi\gamma^2 h + 4\gamma B}{(4\pi(\gamma h)^2 + 4B\gamma h + B^2)^2} \quad (4.11)$$

$$(4.12)$$

Substituindo os estados estacionários da tabela 4.2 e os parâmetros da tabela 4.1, tem se:

$$\bar{h}_1(s) = \frac{0,046}{s + 0,063}\bar{h}_2 + \frac{0,937}{s + 0,063}\bar{u}_1 \quad (4.13)$$

$$\bar{h}_1(s) = \frac{0,937}{s + 0,063}\bar{u}_2 \quad (4.14)$$

Em formato de espaço de estados o sistema final linearizado será:

$$\begin{pmatrix} \dot{\bar{h}}_1 \\ \dot{\bar{h}}_2 \end{pmatrix} = \begin{pmatrix} -0,063 & 0,046 \\ 0 & -0,063 \end{pmatrix} \begin{pmatrix} \bar{h}_1 \\ \bar{h}_2 \end{pmatrix} + \begin{pmatrix} 0,937 & 0 \\ 0 & 0,937 \end{pmatrix} \begin{pmatrix} \bar{u}_1 \\ \bar{u}_2 \end{pmatrix} \quad (4.15)$$

### 4.2.2 Sintonia do PI

Para a sintonia do PI pelo método síntese, toma-se uma entrada para cada saída, pelo critério de maior impacto. Pela equação 4.15, nota-se que a entrada  $u_1$  tem maior impacto em  $h_1$  e  $u_2$  em  $h_2$ . Logo, os controladores, projetados para um tempo de resposta  $\tau_c$  de 10 segundos, de acordo com a equação 2.6, serão:

$$C_1(s) = C_2(s) = \frac{1}{10s} \frac{s + 0,063}{0,937} = 0,1067 + 0,0067 \frac{1}{s} \quad (4.16)$$

### 4.2.3 Sintonia do LQR

Para o presente caso de teste, partindo da equação 4.15, tem-se como matrizes  $A$  e  $B$ :

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

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

$$Q = \begin{pmatrix} 0,5 & 0 \\ 0 & 0,5 \end{pmatrix}, R = \begin{pmatrix} 0,5 & 0 \\ 0 & 0,5 \end{pmatrix} \quad (4.18)$$

, que geram a matriz de ganho  $K$ :

$$K = \begin{pmatrix} 0,935 & 0,023 \\ 0,023 & 0,936 \end{pmatrix} \quad (4.19)$$

De acordo com a equação 2.9, as entradas do sistema serão então regidas pelas equações:

$$\bar{u}_1 = 0,935\bar{h}_1 + 0,023\bar{h}_2 \quad (4.20)$$

$$\bar{u}_2 = 0,023\bar{h}_1 + 0,936\bar{h}_2 \quad (4.21)$$

A equação do LQR foi construída de acordo com o sistema linearizado. Assim, elas devem ser passadas para o controlador em formato de desvio.

## 4.3 Configuração do supervisor didático

Como já mencionado, o supervisor didático implementa duas funções que podem ser editadas pelo usuário, de forma que seja possível uma resposta ao dispositivo conectado, simulando um controlador. O usuário tem liberdade de escrever suas próprias funções e alterar os objetos do código-fonte como preferir, para atender às suas necessidades. As seguintes seções apresentam o código utilizado nas funções *setup\_control()* e *loop\_control()* para cada controlador escolhido.

### 4.3.1 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 em sua documentação, e tem sua função de controle descrita como a equação 2.3. 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()* lê a parte decimal do valor numérico presente na porta. Assim, foi preciso normalizar o sinal de controle, transformando um intervalo de valores que contenha qualquer valor possível seu (no caso,  $[0, 10]$  foi suficiente) em outro que possa ser lido pelo controlador ( $[0, 1]$ ).



Implementa-se, então, na função *loop\_control()*, o 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
5                 enumerate(input_data)]
6     for signal in signals:
7         self.porta.write('{:.2f}'.format(signal/10).encode('UTF-8'))
8
9     return
```

---

### 4.3.2 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 2.9

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, que serão empregados no cálculo de cada sinal de controle.

#### Código 3

---

```
1 A = [[-0.063, 0.046], [0, -0.063]]
2 B = [[0.937, 0], [0, 0.937]]
3 Q = [[0.5, 0], [0, 0.5]]
4 R = [[0.5, 0], [0, 0.5]]
5 K, P, V = control.lqr(A, B, Q, R)
6
7 rho = 1000
8 g = 9.8
9 k = 0.001
10
```

---

```

11 self.K = K
12 self.h1ss = 1
13 self.h2ss = 1
14 self.u1ss = 0
15 self.u2ss = k*math.sqrt(rho*g)
16
17 return

```

---

Na função *loop\_control()*, aplica-se a equação 4.21 para o cálculo dos sinais de controle a cada iteração.

#### Código 4

---

```

1 if len(input_data) == 0:
2     print('No Read')
3     return
4
5 #Transformacao em desvio
6 input_data[0] = input_data[0] - self.h1ss
7 input_data[1] = input_data[1] - self.h2ss
8
9 signals = [-(input_data[0]*self.K[0][0] + input_data[1]*self.K[0][1]) +
10            self.u1ss,
11            -(input_data[0]*self.K[1][0] + input_data[1]*self.K[1][1])
12            + self.u2ss]
13
14 for signal in signals:
15     resp = '{:.3f}'.format(float(signal)/10).encode('UTF-8')
16     self.porta.write(resp)
17
18 return

```

---

## 4.4 Validação dos dados

Todos os casos de teste foram executados, e as séries foram salvas no programa, como mostrado na Figura 4.3.


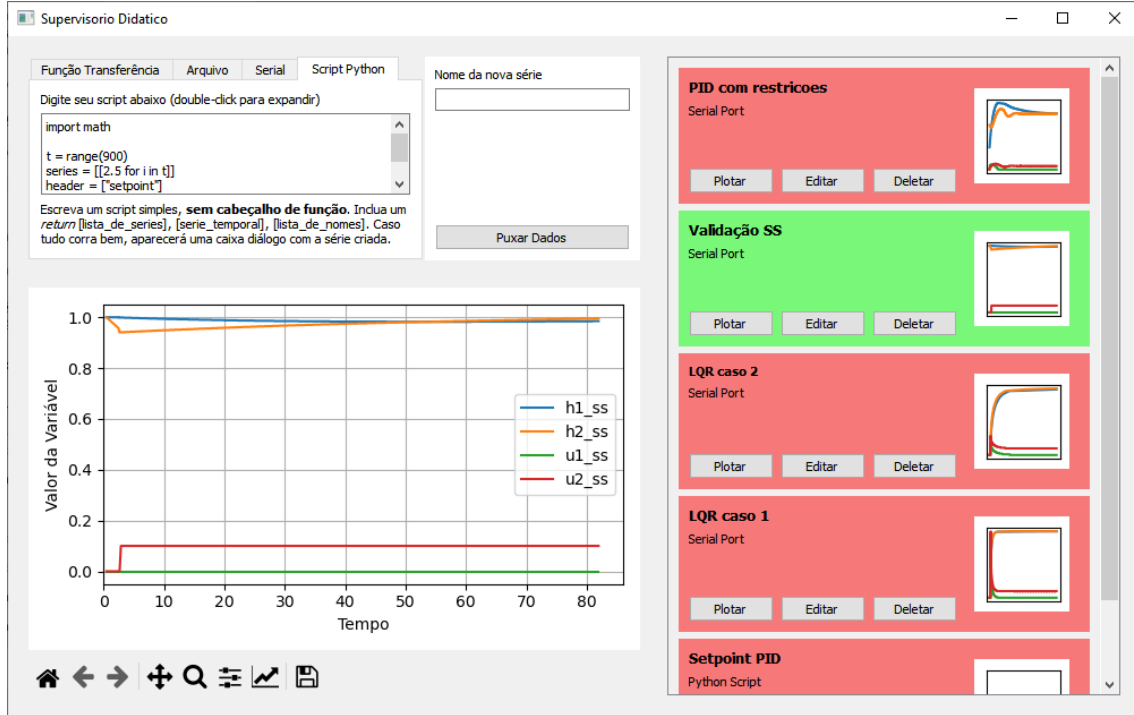
Com isto, a os gráficos foram exportandos clicando em  e copiados para este documento.

Figura 4.3: Interface com as séries salvas



O código do Arduino encontra-se no Apêndice A. Por simular o processo em si, foi incluído nele um trecho de código para limitar os estados e entradas nos seguintes critérios:

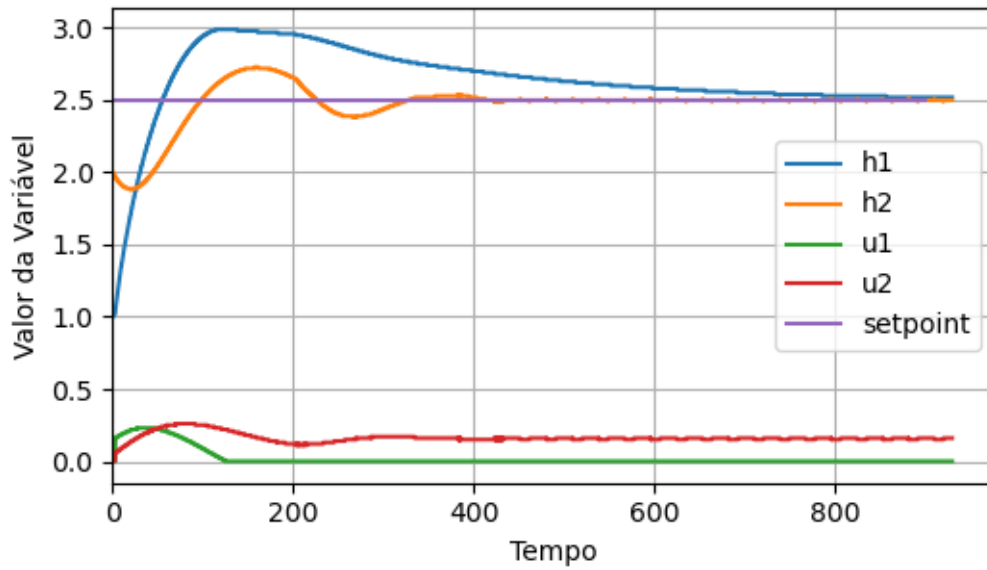
1. As alturas  $h_{1ss}$  e  $h_{2ss}$  devem estar compreendidas no intervalo  $[0, 4]m$
2. As entradas do sistema assumem valores somente entre 0 e  $1 m^3/s$

Com o script contendo os códigos 1 e 2, a comunicação com o arduino foi testada e retornou sucesso. A caixa *SCADA*Dialog apareceu sem grandes problemas. Foi constatado, em tempo real, como os estados do sistema adquiriram um comportamento oscilatório, para depois suavemente atingirem os setpoints.

Se tratando do controlador PI, entretanto, o controle não foi perfeito, como constatado na Figura 4.4. Não somente o sistema levou cerca de 10 minutos para estabilizar, como também tem comportamento dinâmico oscilatório, o que não corresponde com o sistema de primeira ordem *a priori* projetado.

Nota-se que o sinal de controle foi corretamente saturado em 0, validando o limitador do sistema. O gráfico também é contínuo, sem falhas visíveis de comunicação ou leituras discrepantes. Através de um curto script Python foi criada uma linha

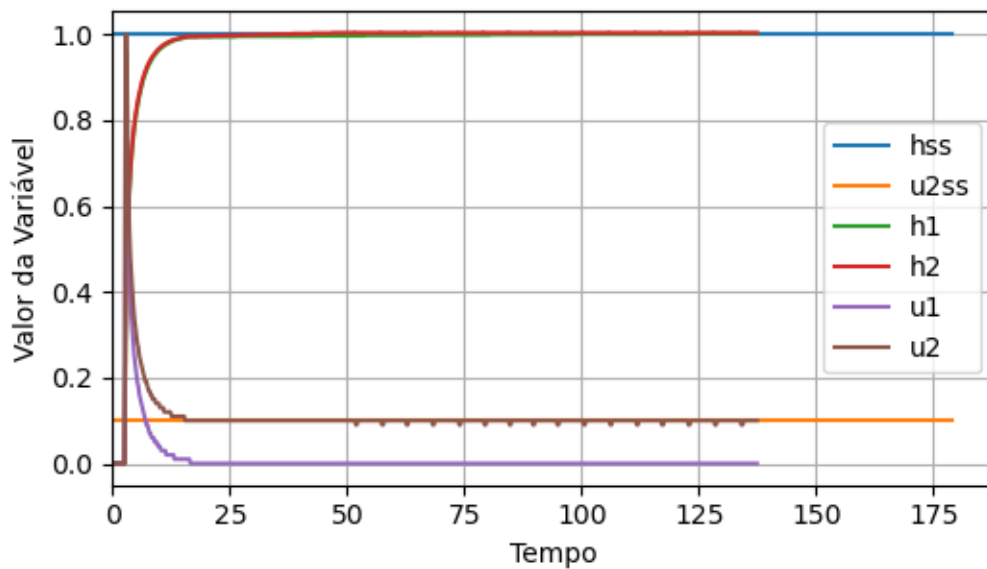
Figura 4.4: Resposta do processo para controlador PI, incluindo restrições de processo



reta "setpoint" no gráfico plotada em conjunto com os resultados.

Também para o caso do LQR não houveram interrupções ou leituras inesperadas durante o monitoramento. Percebeu-se que o sistema respondeu bem mais rápido que o do caso anterior, a custo de um maior esforço de controle.

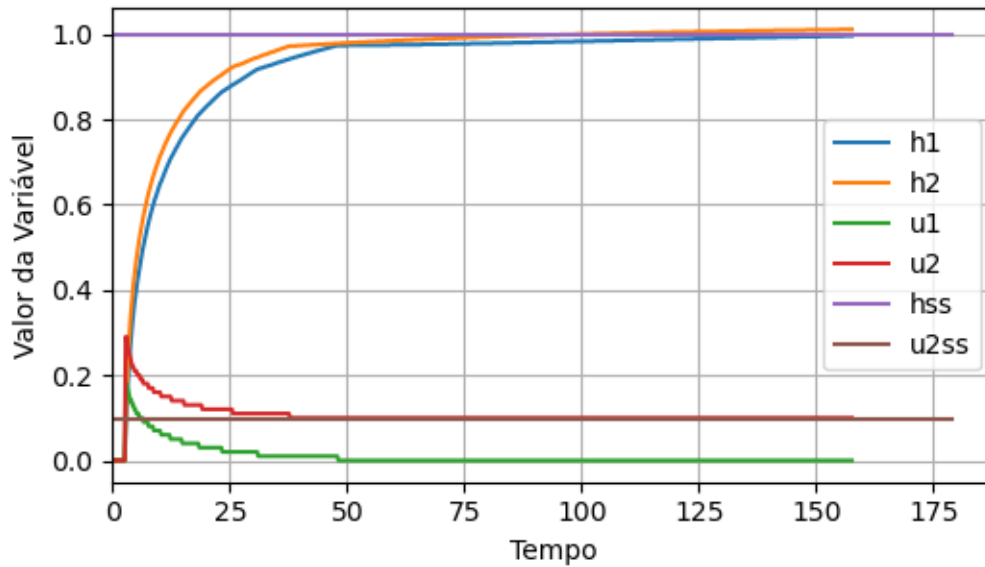
Figura 4.5: Resposta do LQR no caso 1



Como o de se esperar de um LQR, ocorre um pico no início da execução, que foi corretamente saturado. Após meio minuto de simulação, o sistema já se encontrava em estado estacionário.

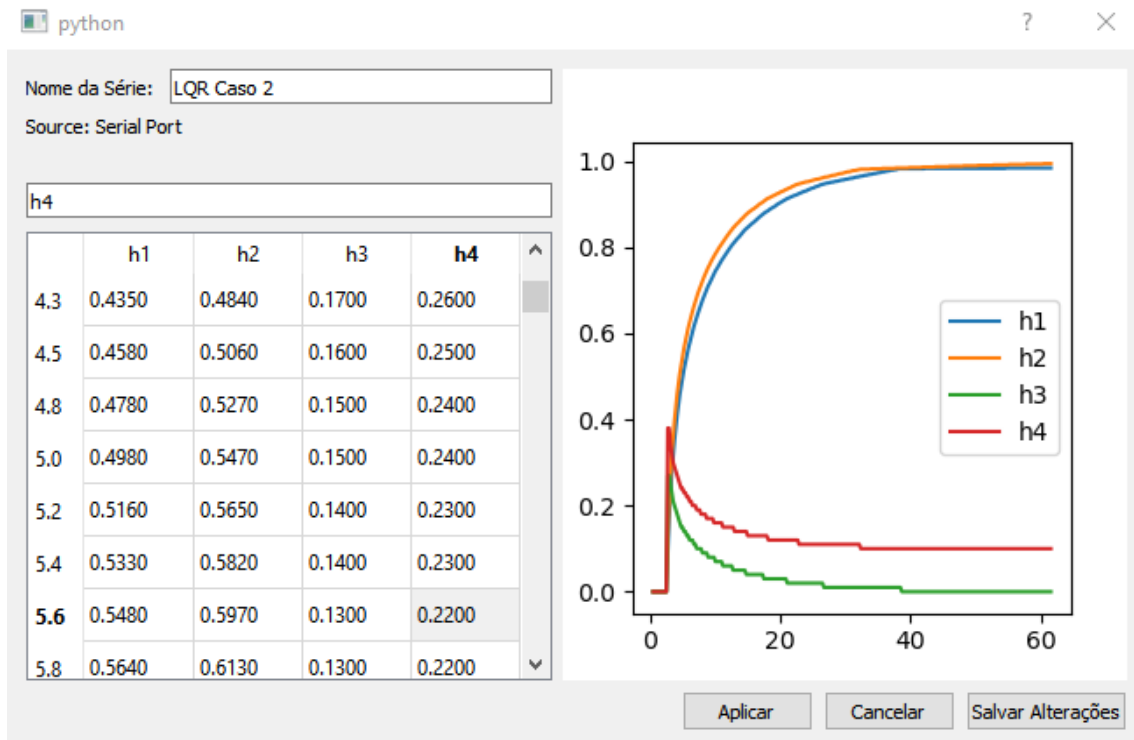
Para diminuir o esforço de controle, o peso de cada variável controlada foi aumentado de 0.5 para 5. Como constatado na Figura 4.6, este objetivo foi, de fato, alcançado, e o sistema levou mais tempo para se acomodar, a troco de menor esforço das bombas.

Figura 4.6: Resposta do LQR no caso 2



Ao salvar as séries de dados (Figura 4.7), observou-se no eixo de tempo que o período dos dados é de cerca de 0,2s, tempo satisfatório para simulação de sistemas químicos, que tendem a ser lentos. Porém, em aplicações reais, isto é um ponto a se avaliar, se se tornará um problema.

Figura 4.7: Resposta do LQR no caso 2



## 4.5 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  $\tau$  do sistema final modelado para o PID.

No geral, pode-se afirmar que o supervisor 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 supervisor didático construído em Python, que permitisse aos usuários monitorar 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 Python, que vem ganhando adeptos 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 ferramenta 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 biblioteca 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 Trabalhos futuros

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 supervisor 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. Criar, separadamente do supervisor, um objeto que emule um CLP;
4. Incluir novos meios de comunicação com periféricos, como por bluetooth ou rede Wi-Fi;
5. 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<sup>®</sup> e o SciLab;
6. 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.

BAYER, M. Sqlalchemy. In: BROWN, A.; WILSON, G. (Ed.). *The Architecture of Open Source Applications Volume II: Structure, Scale, and a Few More Fearless Hacks*. aosabook.org, 2012. Disponível em: <<http://aosabook.org/en/sqlalchemy.html>>.

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

CONTROL python. Python control systems library. 2019. Disponível em: <<https://python-control.readthedocs.io/en/0.8.3/>>.

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

DEVELOPER Survey Results 2019. Disponível em: <<https://insights.stackoverflow.com/survey/2019>>.

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

HARRIS, C. R.; MILLMAN, K. J.; WALT, S. J. van der; GOMMERS, R.; VIRTANEN, P.; COURNAPEAU, D.; WIESER, E.; TAYLOR, J.; BERG, S.; SMITH, N. J.; KERN, R.; PICUS, M.; HOYER, S.; KERKWIJK, M. H. van; BRETT, M.; HALDANE, A.; RÍO, J. F. del; WIEBE, M.; PETERSON, P.; GÉRARD-MARCHANT, P.; SHEPPARD, K.; REDDY, T.; WECKESSER, W.; ABBASI, H.; GOHLKE, C.; OLIPHANT, T. E. Array programming with NumPy. *Nature*, Springer Science and Business Media LLC, v. 585, n. 7825, p. 357–362, setembro 2020. Disponível em: <<https://doi.org/10.1038/s41586-020-2649-2>>.

HUNTER, J. D. Matplotlib: A 2d graphics environment. *Computing in science & engineering*, IEEE, v. 9, n. 3, p. 90–95, 2007.

Júnior, 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.

LATHI, B. P. *Sinais e sistemas lineares*. [S.l.: s.n.], 2007.

LIECHTI, C. pyserial 3.0 documentation. 2020. Disponível em: <<https://pythonhosted.org/pyserial/#>>.

- LOURENÇO, J. *Sintonia de Controladores*. 1997. Disponível em: <<http://ltodi.est.ips.pt/smarques/CS/Pid.pdf>>.
- MANUAL do Usuário do E3. [S.l.].
- 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.
- PYQT. Pyqt reference guide. 2020. Disponível em: <<https://www.riverbankcomputing.com/static/Docs/PyQt5/>>.
- ROGGIA, L.; FUENTES, R. C. Apostila, *Automação Industrial*. 2016.
- ROSSUM, G. van. The making of python. 2003.
- SANTANA, D. D. Notas de aula. 2019.
- SIEMENS. *SIMATIC HMI WinCC Manual*. [S.l.], 1999.
- SMITH, C. A.; CORRIPIO, A. *Princípios e prática do contrle automático de processos*. [S.l.: s.n.], 2006.

---

# Apêndice A

## Códigos

---

### 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 = 3.1415926535897932384626433832795;
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 milliseconds
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 serial 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*
78             pow(h2,0.5)-c*pow(h1,0.5)));
79
80         dh2dt = (1./(pi*pow(gamma,2)*pow(h2+(B/2)/gamma,2))*(u2-c*
81             pow(h2,0.5)));
82
83         h1 += dh1dt*tstep/1000;
84         h2 += dh2dt*tstep/1000;
85
86         //Limitacao dos estados
87         if (h2<0){
88             h2 = 0;
89         } else if(h2>hM){
90             h2 = hM;
91         }
92
93         if (h1<0){
94             h1 = 0;
95         } else if(h1>hM){
96             h1 = hM;
97         }
98     }
99
100     Serial.print(t0/1000);
101     Serial.print('\t');
102     Serial.print(h1, 2); //+(float) random(-1,1)/80.,2);
103     Serial.print('\t');
104     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 }
```

---

---

## Apêndice B

# Tutorial de utilização do supervisorio didático

---

### B.1 Apresentação

Esta seção contém um guia de utilização do supervisorio didático, organizado em tópicos. Cada tópico descreve um passo-a-passo sobre como realizar as operações que o programa oferta.

### B.2 Primeira utilização

O supervisorio didático foi programado em linguagem Python e dividido em alguns arquivos, a fins de organização. Estes arquivos são agrupados e referenciados no script principal Supervisorio.py, sendo imprescindível que a organização dentro da pasta do programa não seja alterada. Caso seja, o script principal não conseguira encontrar os arquivos que contém as declarações de classes e será mostrado um erro no terminal.

Ao utilizar o programa, supõe-se que a máquina operante já possui Python instalado. O compilador pode ser baixado gratuitamente em seu site oficial. Os testes foram feitos com a versão 3.8.5 da linguagem, não sendo claros os efeitos que versões anteriores a esta causarão na operação.

Para intalação das bibliotecas utilizadas, roda-se o arquivo `install_packages.bat`, presente na pasta `utils`. Talvez seja necessário permissão de administrador para executá-lo. Alternativamente, através do comando 1, as bibliotecas listadas em `requirements.txt` será instaladas.

---

```
1 pip install -r requirements.txt
```

---

## B.3 Iniciando o programa

O script principal pode ser executado através de vários métodos, pois trata-se apenas de um script python. Para uma maneira rápida de fazê-lo, basta executar Supervisorio.bat. Ele não deve ser movido da pasta, mas pode ser criado um atalho para ele.

Quando o script é executado, a tela principal do supervisorio aparecerá. Caso esteja presente um arquivo de salvamento automático autosave.dat no diretório raiz do programa, ele tentará restaurar as séries da sessão anterior. Caso apareçam erros no trecho que carrega estes dados, o usuário pode tentar deletar este arquivo.

## B.4 Importando séries estáticas no programa

Existem diversos métodos de entrada de dados no supervisorio. Primeiro, alguns parâmetros são configurados, de acordo com cada método, descritos nas subseções posteriores, e depois clica-se no botão "Puxar Dados".

### B.4.1 Função de Transferência

A simulação de funções de transferência no programa requer o numerador e denominador de cada função, multiplicadas entre si, bem como o estado inicial da saída do sistema e o ganho do degrau aplicado. Para este método, o sistema responde somente a uma entrada degrau. A entrada de dados por função de transferência se dá pelo objeto ilustrado na Figura B.1.

Com o campo num e den preenchidos com uma sequência de números separados por espaço, clica-se no botão "Adicionar Função" para incluí-la na lista à direita. Cada número é atribuído em ordem de posição a uma potência de  $s$ , sendo o número mais à direita atribuído a  $s^0$ . Como exemplo, um numerador "1 3" e denominador "2 0 1" representam a função de transferência  $\frac{s + 3}{2s^2 + 1}$ .



Figura B.1: Objeto *TransferFunctionConfig* (à esquerda)

**OBS:** o numerador não pode ter grau maior que o denominador, por restrições da biblioteca empregada neste módulo.

A ordem das funções não impacta a resposta final, mas por questão de organização, foram incluídos botões à direita da tabela que mudam a posição de cada função na lista. O botão "-" remove a função selecionada, se houver, da mesma.

Terminada a entrada de funções de transferência, configuram-se o estado inicial e o ganho do degrau. Após isto, clica-se no botão "Puxar Dados"

## B.4.2 Entrada por arquivo

O programa aceita 4 formatos de arquivos para inserção de séries, que são .csv, .tsv, .xls e .xlsx. Todos estes formatos armazenam dados em tabela, logo compartilham seus parâmetros de importação:

- **Cabeçalho:** Caso a opção "Considerar cabeçalho" seja marcada, o programa transformará a primeira linha de dados lida como a lista de nomes de cada série.
- **Eixo de tempo:** Com a opção "1ª coluna como eixo de tempo", o programa considerará a coluna mais à esquerda como eixo de tempo, compartilhado pelas outras séries contidas no arquivo. Caso a opção marcada seja "Gerar eixo de tempo autom.", o programa criará este eixo pela quantidade de linhas lidas, começando em 1, e em incrementos unitários a cada leitura.

Na parte superior do objeto *FileConfig*, são listados cada um dos formatos compatíveis com o programa. O formato selecionado configura o filtro de extensão do

explorador de arquivos que aparece quando o botão "... "é clicado. Ao selecionar um arquivo por este explorador, o programa preencherá automaticamente os campos "Diretório" e "Arquivo".

### B.4.3 Serial

A importação por porta serial é configurada pelos seguintes parâmetros:

- ***baud rate***: velocidade de recebimento e transmissão dos dados na porta;
- **Porta**: nome da porta;
- ***timeout***: tempo máximo de resposta do dispositivo conectado, quando a comunicação for iniciada.
- **N<sup>#</sup> colunas** quantas séries de dados recebidas devem ser esperadas pelo programa, incluindo a série de tempo. Após cada leitura da porta serial, se a quantidade de valores lidos diferir deste número, a leitura será desconsiderada.

Abaixo de algumas configurações existe uma lista com os valores mais populares de parâmetros. Ao clicar nos itens da lista, a caixa de texto será atualizada. O botão "Testar" abre e fecha uma conexão na porta informada, a fim de verificar se não houve problemas.

Para este caso, ao iniciar a importação, a janela de monitoramento irá aparecer, como descrito na seção B.5.

### B.4.4 Script Python

Esta alternativa busca oferecer aos usuários maneiras próprias de criação de dados. Na caixa de texto pode ser digitado um script python que retorne, nesta ordem, uma lista de séries, um eixo de tempo e uma lista de nomes. O programa criará um objeto *SeriesObject* a partir do retorno deste script. Ainda é possível clicar duplamente na caixa de texto, momento no qual uma caixa maior aparece, garantindo maior visibilidade. Nela há também alguns exemplo de códigos, que podem ser modificados livremente pelo usuário.

A puxar dados de um script python, o programa escreve o texto em um arquivo denominado script.py, em seu diretório raiz, cuidando de toda a sintaxe adicional.

Em seguida, envia um comando ao sistema operacional para executar o script final, que armazena o resultado em um objeto serializado. Por fim, este objeto é decodificado e, caso não hajam erros, uma nova série é criada.

## B.5 Monitoramento em tempo real

Ao clicar em "Puxar Dados" por porta serial, o programa tenta estabelecer uma conexão com os parâmetros informados, na porta definida. O nome da porta pode variar com o sistema operacional.

Após a conexão ser estabelecida, o programa limpará qualquer informação escrita no buffer da porta e, em intervalos de 2 segundos nela escreverá "go", até que alguma informação de retorno seja recebida. Isso se repetirá até 100 vezes. Este procedimento por ser alterado na função *setup\_connection()* do objeto *SCADADialog*, presente no script *realtime\_objects.py* na pasta *objects*.

Após esta rotina de sincronização, a função personalizada *setup\_control()* é chamada. Caso o usuário deseje enviar algum valor para o controlador (setpoints, parâmetros do sistema, entre outros), deve fazê-lo por aqui. O programa abre também a janela de monitoramento *SCADADialog*, com um gráfico que plota cada série de dados recebida em tempo real.

Finalmente, iniciam-se as rotinas de leitura da porta e atualização do gráfico, respectivamente a cada 0,2 e 1 segundo. Como mencionado, a rotina de leitura lê toda uma linha de dados da porta, divide-a pelas tabulações contidas, e atribui cada trecho a uma série de dados interna, sendo a primeira sempre o eixo de tempo. Este formato deve ser seguido pelo emissor, ou controlador, acoplado, pois caso não coincida com a quantidade de séries esperada passado como parâmetro, toda a linha será ignorada.

Contida na rotina de leitura está a função *loop\_control()*. Idealmente, ela tem como objetivo mandar valores para o controlador em tempo de execução.

Durante o monitoramento, o usuário pode interromper a execução ao clicar em "Parar" ou "Cancelar". O segundo fecha a janela de monitoramento e descarta a série armazenada até então. O primeiro somente interrompe o fluxo de dados, e possibilita que o usuário salve as séries lidas, clicando em "Criar Série".

## B.6 Salvando e editando séries de dados

Qualquer que tenha sido o método de importação, o usuário terá uma visualização dos dados importados sempre antes de efetivamente salvá-los no programa. Isto é ilustrado na Figura 3.2.

Nesta caixa diálogo, o usuário pode editar o título do conjunto e cabeçalho de cada série sendo importada. Para editar o cabeçalho, clica-se no nome da série contido na tabela à esquerda e altera-se a caixa de texto imediatamente acima dela. Cada alteração deve ser confirmada com Enter.

Após realizadas as edições, o usuário pode clicar em "Criar Série", e ela aparecerá na lista de séries à direita da aplicação principal.







Para editar uma série já criada, clica-se no respectivo botão "Editar", na lista de série, e esta mesma janela aparecerá. Para este caso, o botão "Criar Série" será substituído por "Salvar Alterações".

## B.7 Plotando séries

A plotagem de séries na área principal se dá clicando em "Plotar", momento no qual o gráfico no canto inferior esquerdo da aplicação desenhará a série, sem perder outras já plotadas. A legenda também será atualizada.

## B.8 Editando e exportando gráficos

Abaixo da área de plotagem, existe uma barra com algumas opções de configurações, como:

-  Permite aplicar zoom no gráfico
-  Permite mover o gráfico
-  Retorna o gráfico à escala original
-  Configura o espaçamento do gráfico e sua moldura
-  Configura os eixos do gráfico, editando seu rótulo, limites e escala
-  Salva o gráfico em formato de imagem

## B.9 Deletando séries

Ao clicar em "Deletar", a série é apagada. Esta operação é irreversível.



---

# Apêndice C

## Códigos

---

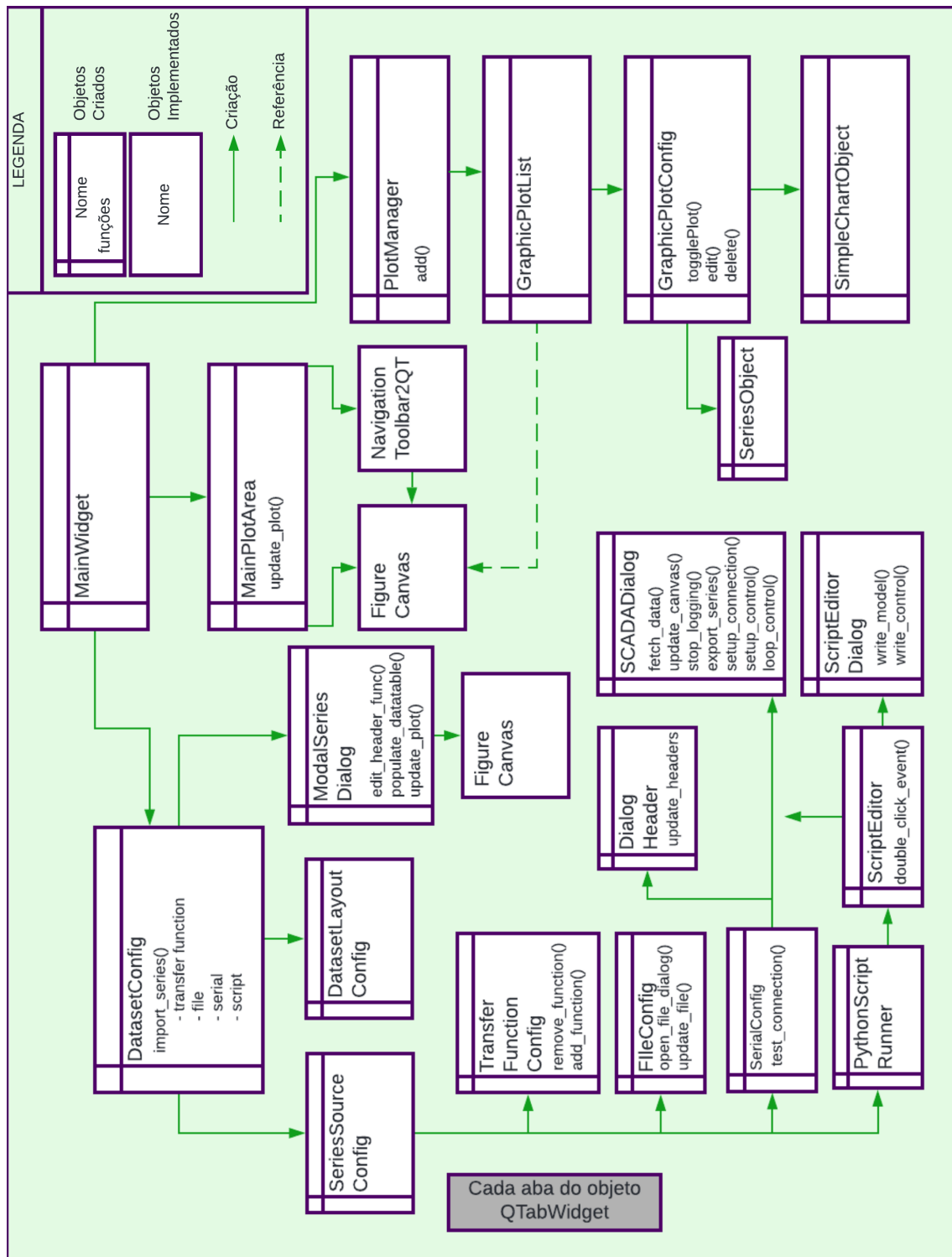


Figura C.1: Diagrama de relações entre os objetos empregados e funções principais