UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

André Chagas Wander

Interface gráfica para projeto de controladores pelo método VRFT para conversores CC-CC

Porto Alegre

2022

UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA

André Chagas Wander

Interface gráfica para projeto de controladores pelo método VRFT para conversores CC-CC

Projeto de Diplomação II, apresentado ao Departamento de Engenharia Elétrica da Escola de Engenharia da Universidade Federal do Rio Grande do Sul, como requisito parcial para a obtenção do grau de Engenheiro Eletricista

UFRGS

Orientador: Prof. Dr. Jeferson Vieira Flores

Porto Alegre
2022

Resumo

A operação eficiente de conversores CC-CC passa pelo projeto de diferentes malhas de controle, conferindo ao sistema características importantes como regulação da tensão de saída e robustez à inserção ou retirada de cargas. Os métodos de projeto de controladores baseados em dados (*data driven*) se mostram como uma opção viável, visto que são capazes de projetar um controlador a partir de conjuntos de dados coletados em ensaios na planta, sem a necessidade de modelagem do sistema. Dentre esses métodos, o *Virtual Reference Feedback Tunning* - VRFT - se destaca pois necessita apenas de um único conjunto de dados e otimiza o desempenho do sistema em malha fechada para o seguimento de referência. Publicações recentes evidenciam o potencial de aplicação do VRFT em conversores CC-CC, como (REMES, 2021). Visando difundir a aplicação desse método nestes conversores, este trabalho apresenta o desenvolvimento de uma interface gráfica em código aberto que auxilia o projetista nas etapas de pré-projeto, projeto via VRFT e análises de desempenho. Para tal, foi utilizada a biblioteca *pyvrft*, desenvolvida por (ECKHARD; BOEIRA, 2019), na linguagem de programação Python, e o framework Qt, que permite a criação e operação de interfaces gráficas através da biblioteca *pyqt6*, nesta mesma linguagem de programação.

Palavras-chave: Método VRFT, Python, PyQt, GUI, Conversores CC-CC.

Abstract

The efficient operation of DC-DC converters requires the design of diferent control loops, giving the system important characteristics, such as output voltage regulation and robustness against load variances. Data-Driven controller design methods are a very interesting option given that they can design a controller based on a system's input-output dataset, without the need of a mathematical model for the system. Considering these methods, the Virtual Reference Feedback Tunning - VRFT - stands out among others because it only needs a single dataset and optimizes a performance criterion associated to the closed-loop reference tracking response. Recent publications have put forth evidence of the potential that the VRFT method has when applied to DC-DC converters, such as in (REMES, 2021). Aiming to disseminate the application of the method on these converters, this paper presents the development of an open source graphical interface that helps the user in the pre-project, design via VRFT and performance analysis steps. For such, the Python package pyvrft, developed by (ECKHARD; BOEIRA, 2019), and the Qt framework, that provides methods for creating and operating graphical interfaces through the Python package pyqt6, were used

Keywords: VRFT Method, Python, PyQt, GUI, DC-DC Converters.

Lista de abreviaturas

UFRGS	Universidade Federal do Rio Grande do Sul
GUI	*Graphical User Interface*
VRFT	*Virtual Reference Feedback Tunning*
P	Proporcional
PI	Proporcional Integral
PD	Proporcional Derivativo
PID	Proporcional Integral Derivativo
IDE	*Integrated Development Enviroment*
CC	Corrente Contínua
DD	*Data Driven*
MSE	*Mean Squared Error*

Sumário

1 Introdução

Conversores estáticos são dispositivos elétricos essenciais para o funcionamento da vida moderna, considerando que estes são amplamente usados em diversas aplicações que necessitam de uma fonte de energia elétrica para funcionar ou que forneçam energia elétrica para outros dispositivos. Dentre eles, há os conversores CC-CC, que geram um sinal de saída contínuo com valor médio maior ou menor que a entrada, também de nível contínuo (KANZIAN; AGOSTINELLI; HUEMER, 2019).

Conversores CC-CC são de grande interesse prático já que estão associados a um grande número de aplicações. Esses conversores podem ser encontrados, por exemplo: na geração fotovoltaica, em que é necessário fazer uma adequação dos níveis de tensão gerados pelos painéis (GOPI; SREEJITH, 2018); no acionamento de motores contínuos, para fazer o controle de parâmetros como conjugado, velocidade e o acionamento com partida suave (SILVA-ORTIGOZA *et al.*, 2015); no carregamento de baterias que necessitem de um controle preciso de corrente na carga (REMES, 2021).

Tratando de conversores em geral, é muito importante haver um sistema de controle implementado para que o sinal de saída do conversor não se altere de forma indesejada quando houver alguma variação na tensão de alimentação ou na carga aplicada. Usualmente, esses controladores são projetados a partir de técnicas clássicas de projeto, como Lugar Geométrico das Raízes ou a partir da resposta em frequência do processo (ERICKSON; MAKSIMOVIC, 2001). Um ponto em comum a essas técnicas clássicas é a suposição do conhecimento de um modelo que descreve o comportamento do sistema. No caso de conversores de potência, a obtenção de um modelo é uma tarefa complexa, considerando que estes são formados por elementos chaveados e não-lineares. Com o objetivo de obter um modelo linear e invariante no tempo, geralmente são realizadas simplificações e aproximações que restringem a utilização do controlador projetado a uma faixa restrita de operação. Além disso, todo o processo de modelagem matemática deve ser refeito caso ocorram alterações significativas no circuito do conversor (SALATI, 2021).

Uma forma de contornar estas dificuldades são as metodologias de projeto de controladores baseados em dados (ou DD, do inglês *Data Driven*). Estas técnicas fazem uso de dados de entrada e saída coletados a partir de ensaios na planta para projetar um controlador que minimiza um dado critério de desempenho, sem a necessidade da modelagem e identificação prévia do sistema (BAZANELLA; CAMPESTRINI; ECKHARD, 2011).

Dentre os métodos DD no domínio do tempo, o método *Virtual Reference Feedback Tuning* (VRFT) é um que se destaca por ser um método que exige apenas um único

experimento (*one-shot*) para fazer o projeto do controlador do sistema (BAZANELLA; CAMPESTRINI; ECKHARD, 2011). Esse método possibilita o projeto de um controlador linearmente parametrizado (PI ou PID, por exemplo) que garante um comportamento desejado definido a partir de um dado modelo de referência. Além disso, a literatura apresenta extensões deste método para lidar com sistemas contendo zeros de fase não-mínima e/ou ruído (CAMPESTRINI *et al.*, 2011). Ainda, o problema de otimização associado ao seguimento de referências pode ser resolvido eficientemente a partir do método de mínimos quadrados, tanto via Matlab quanto via Python (ECKHARD; BOEIRA, 2019).

Em (REMES, 2021), é estudado o desempenho de controladores projetados com esta metodologia DD aplicada a conversores CC-CC, em especial nos conversores *buck*, *boost*, *buck-boost* e derivados, através dos métodos *Virtual Reference Feedback Tuning* (VRFT) e *Virtual Disturbance Feedback Tuning* (VDFT). A metodologia baseada em dados traz algumas vantagens no contexto de conversores como: supressão do modelo da planta e do dilema entre complexidade e representatividade deste; atributos da planta como a resistência série equivalente de elementos armazenadores, atrasos vinculados à modulação por largura de pulso, entre outros, são captados implicitamente pelos dados de funcionamento coletados da planta; desempenho ótimo do ponto de vista de critérios associados ao seguimento de referência ou rejeição a distúrbio (CAMPI; LECCHINI; SAVARESI, 2000).

Dentro desse contexto, o objetivo principal deste trabalho é o desenvolvimento de uma ferramenta computacional (uma *Graphical User Interface*, ou GUI) que auxilie o projetista na aplicação do método VRFT para conversores de potência. Serão contempladas as três principais etapas de controle baseado em dados: o experimento para coleta de dados, a aplicação do método de projeto e a avaliação do desempenho do controlador projetado quando comparado à resposta desejada. O desenvolvimento da solução foi centrado na biblioteca *pyvrft*, que traz uma implementação do método VRFT na linguagem Python, e do *framework* PyQt, que traz ferramentas para o desenvolvimento e operação de interfaces gráficas nessa mesma linguagem. A partir desse objetivo geral, são propostos os seguintes objetivos específicos:

- Desenvolvimento de uma aplicação gráfica, a partir do *framework* PyQt, em linguagem de programação Python que faça a interação com o projetista e a análise e validação dos dados fornecidos;
- Implementação do método VRFT para projeto de controladores com a biblioteca *pyvrft* que faz a manipulação de dados, informados através da GUI desenvolvida, para calcular os parâmetros de uma estrutura de controlador escolhido pelo usuário;

- Analisar o desempenho do controlador projetado pela GUI por meio de diferentes métricas, como o erro médio quadrático e geração de gráficos comparativos entre a resposta em malha fechada desejada e a resposta obtida na planta.

Este trabalho segue a seguinte organização: o Capítulo 2 traz conceitos básicos sobre conversores CC-CC, com o intuito de familiarizar o leitor com os processos relevantes a cerca deste assunto, bem como uma revisão sobre conceitos do controle desses conversores. Também há uma contextualização quanto a abordagem DD para o projeto de controladores no domínio do tempo, da implementação da biblioteca do método VRFT desenvolvida e uma breve explicação sobre o projeto de interfaces gráficas com a linguagem Python. No Capítulo 3, é apresentado o desenvolvimento da GUI, desde a contextualização quanto as ferramentas escolhidas até o desenvolvimento da interface. No Capítulo 4, são apresentadas análises realizadas para comparar os resultados obtidos com os resultados esperados para os controladores projetados. Por fim, o Capítulo 5 traz as conclusões deste trabalho.

2 Fundamentação Teórica e Revisão Bibliográfica

A seguir, são abordados os pontos referentes a fundamentação teórica utilizado no desenvolvimento deste trabalho. Inicialmente é apresentada uma breve explicação do funcionamento das topologias de conversores CC-CC dos tipos estudados em (REMES *et al.*, 2019), sendo estes os conversores *Buck*, *Boost*, *Buck-Boost* e SEPIC. Na sequência, é apresentado o método VRFT para projeto de controladores, conforme (BAZANELLA; CAMPESTRINI; ECKHARD, 2011). Finalmente, é abordada a implementação do VRFT na linguagem Python, proposta em (ECKHARD; BOEIRA, 2019) e a utilização do Python no desenvolvimento de interfaces gráficas.

2.1 Conversores CC-CC

Conversores CC-CC são dispositivos eletroeletrônicos que transformam um nível de tensão elétrica contínua na sua entrada para outro em sua saída, podendo este ter um valor médio maior ou menor que a tensão de entrada. Estes dispositivos são, tipicamente, controlados por semicondutores com um funcionamento similar ao de uma chave. Esses semicondutores são acionados por sinais com modulação por largura de pulso (*pulse width modulation* - PWM) onde a variável a ser controlada é o chamado Ciclo de Trabalho. Este parâmetro corresponde ao percentual de tempo dentro de um período do sinal PWM em que a chave está na posição ligada ou desligada.

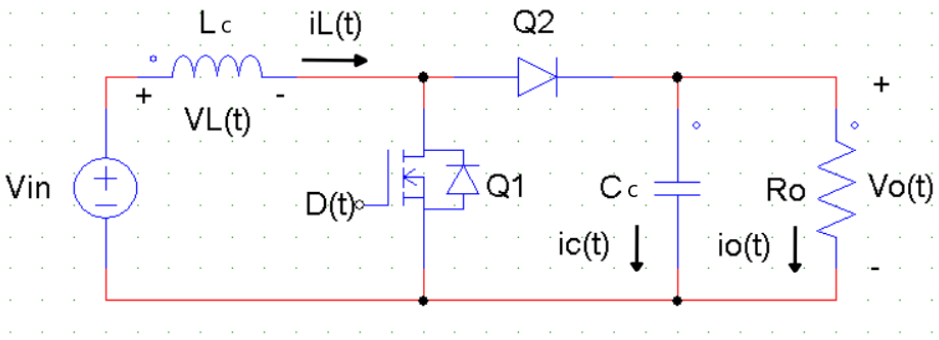
A literatura de conversores de potência apresenta diversas topologias de conversores CC-CC (YI *et al.*, 2022). Considerando as topologias clássicas de conversores, o conversor *Buck* tem a característica de ser um rebaixador de tensão, enquanto que o *Buck-Boost* pode tanto rebaixar quanto elevar a tensão média na saída dependendo do *ciclo de trabalho* em que opera. Ambos os conversores *Boost* e SEPIC tem a função de elevação do nível médio da tensão elétrica na saída do sistema. A grande diferença entre estas duas topologias é a presença de um transformador no conversor SEPIC, trazendo a característica de isolação galvânica entre a fonte de alimentação e a saída junto de uma maior proteção contra possíveis curto-circuitos.

2.1.1 Conversor *Boost*

O circuito simplificado de um conversor CC-CC do tipo *Boost* é representado na Figura 1, onde V_{in} corresponde a tensão elétrica da alimentação e $V_o(t)$ a tensão elétrica

na saída. L_c e C_c representam o indutor e o capacitor, respectivamente. Ainda, assume-se que o transistor Q_1 e o diodo Q_2 atuam como interruptores ideais. R_o corresponde a carga conectada à saída do conversor.

Figura 1 – Circuito Esquemático Conversor Boost



Fonte: o Autor, 2022.

Um conversor *boost* opera pela comutação periódica do semicondutor Q_1 e controlado pelo sinal PWM D(t). Por definição, Q_1 está conduzindo quando D(t) = 1 e está em corte quando D(t) = 0. O ciclo de trabalho d do conversor pode então ser definido como a parcela do período de chaveamento T_s em que D(t) = 1, ou seja, a cada período D(t) = 1 no intervalo $0 \leq t \leq dT_s$ e D(t) = 0 em $dT_s < t < T_s$.

É possível descrever a relação entre a tensão elétrica de entrada e a tensão elétrica de saída em função da razão cíclica de trabalho, como:

$$\frac{V_o}{V_{in}} = \frac{1}{1-d}, \ 0 \leq d < 1 \tag{1}$$

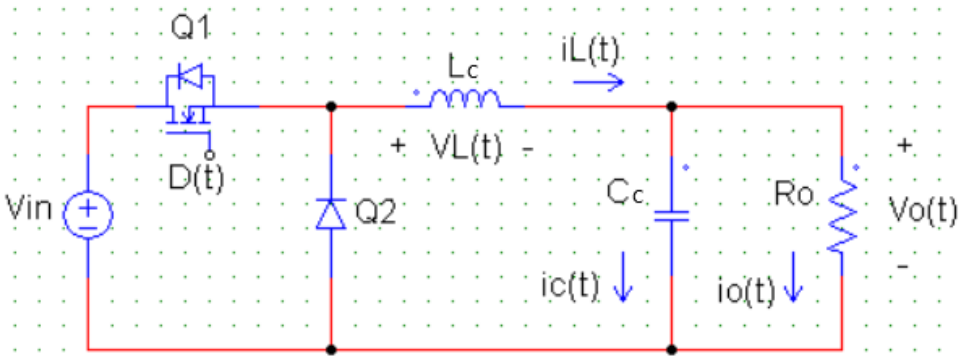
Percebe-se que, para valores de $0 \leq d < 1$, a V_o em regime permanente é sempre maior ou igual a V_{in}, evidenciando a característica elevadora de tensão do conversor. Note ainda, que no caso limite de d = 1, a relação entrada-saída seria nula, e não infinita. Isso se deve ao fato de que neste caso Q_1 estará sempre conduzindo e Q_2 estará sempre bloqueado, cortando a transferência de energia da entrada para a saída. Por estar em regime permanente, toda a tensão elétrica armazenada no capacitor já terá sido dissipada pela carga, resultando em $V_o = 0V$. Perceba também, que esta situação implica em um curto na fonte de alimentação após a saturação do indutor. Devido a esses pontos, é comum na prática limitar o valor máximo de d, utilizando-se tipicamente valores abaixo de 0,85 (REMES, 2021).

2.1.2 Conversor *Buck*

A topologia do conversor CC-CC *Buck* não difere muito da topologia do conversor *Boost*, porém este conversor tem característica de rebaixador de tensão. A Figura 2 mostra

essa topologia, onde cada elemento corresponde àqueles componentes do conversor *Boost*, mas é trocada a posição do indutor L_c e dos semicondutores Q_1 e Q_2.

Figura 2 – Circuito Esquemático Conversor Buck



Fonte: o Autor, 2022.

É possível descrever a relação entrada-saída entre a tensão elétrica da fonte de alimentação e a tensão na saída em função do ciclo de trabalho d pela relação:

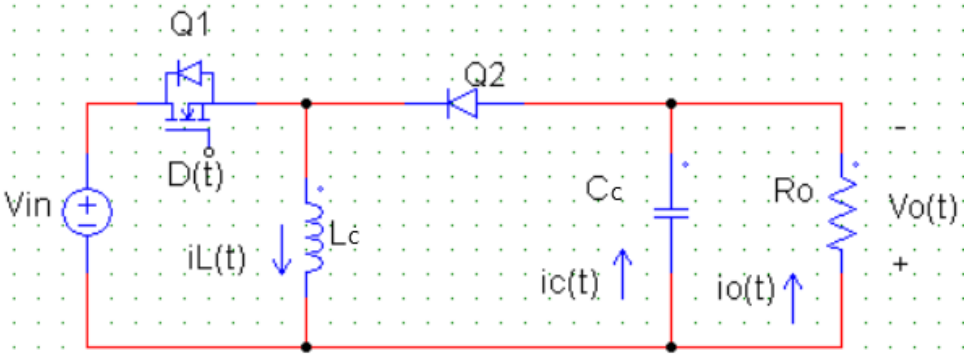
$$\frac{V_o}{V_{in}} = d \tag{2}$$

Observe que, por definição, $0 < d < 1$, logo V_o será sempre menor que V_{in}.

2.1.3 Conversor *Buck-Boost*

A topologia de um conversor CC-CC *Buck-Boost* modifica, em relação a topologia do conversor *Buck*, apenas na posição entre o indutor e o semicondutor Q_2. Um ponto relevante sobre este conversor é que ele pode tanto rebaixar quanto elevar a tensão na saída em relação a tensão da entrada, porém, a polarização entre elas sempre será invertida. O que define se o conversor é um rebaixador ou elevador é a razão de trabalho que para valores de $d < 0.5$, o conversor tem a dinâmica de ser um rebaixador e para $d > 0.5$ a dinâmica é de um elevador.A Figura 3 mostra a topologia deste conversor.

Figura 3 – Circuito Esquemático Conversor Buck-Boost



Fonte: o Autor, 2022.

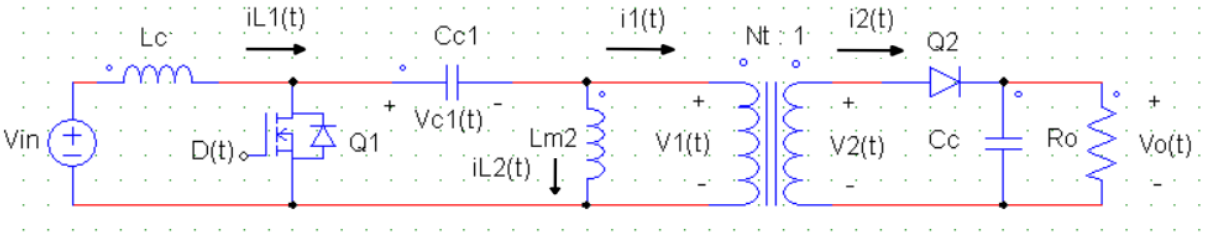
É possível descrever a relação entrada-saída deste conversor entre a tensão elétrica da fonte de alimentação e a tensão na saída em função da razão cíclica de trabalho d. A Equação (3) mostra este equacionamento:

$$\frac{V_o}{V_{in}} = \frac{d}{1-d} \tag{3}$$

2.1.4 Conversor SEPIC

A topologia de conversor CC-CC *single-ended primary-inductor converter* (SEPIC), seria uma versão um pouco mais complexa do conversor *boost*. O circuito simplificado da forma isolada deste conversor é representado pela Figura 4. L_c corresponde ao indutor de entrada por onde passa a corrente $i_{L1}(t)$, L_{m2} corresponde a indutância magnetizante do transformador, N_t:1 corresponde a relação do número de voltas entre os enrolamentos do transformador considerado como um transformador ideal, C_{c1} e C_c correspondem a capacitores com suas respectivas tensões V_{c1} e V_{c2}, Q_1 corresponde ao semicondutor interruptor e Q_2 corresponde ao diodo de saída. Assim como nos outros conversores, o sinal D(t) controla a tensão elétrica aplicada a carga R_o na saída.

Figura 4 – Circuito Esquemático Conversor SEPIC



Fonte: o Autor, 2022.

É possível descrever a relação entre a tensão elétrica da fonte de alimentação e a tensão na saída em função da razão cíclica de trabalho d e da relação de transformação do transformador como sendo:

$$\frac{V_s}{V_{in}} = \frac{1}{N_t}\frac{d}{1-d} \tag{4}$$

2.1.5 Característica Dinâmica de Conversores

Supondo que os conversores apresentados nesta seção operam no Modo de Condução Contínua e aplicando a técnica do modelo médio em um período de chaveamento, (REMES, 2021) mostra que as topologias clássicas de conversores apresentam uma resposta dinâmica entre a tensão de saída e a variação do ciclo de trabalho que pode ser aproximada por uma

função de transferência de segunda ordem. Na Tabela 1 são apresentados os parâmetros que descrevem essa relação dinâmica baseados na seguinte função de transferência padrão:

$$G(s) = \frac{G_{d0}(1 - \frac{s}{w_z})}{\frac{s^2}{w_0^2} + \frac{s}{Q_{0w0}} + 1} \tag{5}$$

Desta tabela, destaca-se o ganho G_{d0}, que depende somente dos valores do ciclo de trabalho d e da tensão de saída V_o no ponto nominal de operação. Esse parâmetro tem uma relação direta com a coleta de dados do método VRFT.

Tabela 1 – Coeficientes para os modelos de conversores CC-CC em função dos parâmetros construtivos

Topologia	G_{d0}	ω_0	Q_0	ω_z	$\frac{\omega_z}{Q_0\omega_0}$
Buck	$\frac{V_o}{d}$	$\frac{1}{\sqrt{L_cC_c}}$	$R_o\sqrt{\frac{C_c}{L_c}}$	∞	∞
Boost	$\frac{V_o}{1-d}$	$\frac{1-d}{\sqrt{L_cC_c}}$	$R_o(1-d)\sqrt{\frac{C_c}{L_c}}$	$\frac{R_o(1-d)^2}{L_c}$	1
Buck-Boost/SEPIC	$\frac{V_o}{d(1-d)}$	$\frac{1-d}{\sqrt{L_cC_c}}$	$R_o(1-d)\sqrt{\frac{C_c}{L_c}}$	$\frac{R_o(1-d)^2}{dL_c}$	$\frac{1}{d}$
Flyback/SEPIC Isol.	$\frac{V_o}{d(1-d)}$	$\frac{N_t(1-d)}{\sqrt{L_cC_c}}$	$N_tR_o(1-d)\sqrt{\frac{C_c}{L_c}}$	$\frac{N_t^2R_o(1-d)^2}{dL_c}$	$\frac{1}{d}$

Fonte: (REMES, 2021)

2.2 Controle por modelo de referência para Conversores CC-CC

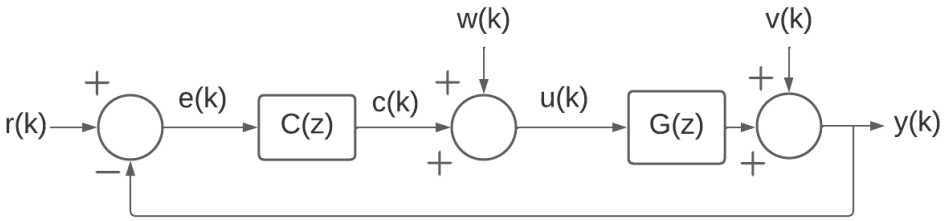
Do ponto de vista de controladores aplicados a conversores CC-CC, há alguns requisitos de desempenho em regime transitório e permanente que são buscados na hora de implementar um sistema de controle, dependendo da aplicação. No contexto de controle de conversores CC-CC, geralmente as malhas de controle são projetadas visando o atendimento de um ou mais dos seguintes requisitos:

- Erro nulo no regime permanente para seguimento de referências constantes;
- Rejeição de distúrbios do tipo degrau decorrentes de variações de carga ou da tensão de alimentação;
- Robustez frente a variações no ponto de operação do conversor(d, i_L, V_o);
- Tempo de acomodação condizente com a necessidade da aplicação do conversor.

Uma das maneiras de garantir o atendimento desses requisitos de desempenho é através do chamado Controle por Modelo de Referência. Nessa técnica, é escolhida uma função de transferência desejada para o sistema em malha fechada que atenda aos

requisitos desejados e é projetado um controlador que mais se aproxima dessa função.[1] A formulação desse método é apresentada a seguir.

Figura 5 – Diagrama de blocos em malha fechada de um conversor CC-CC



Fonte: Fonte: o Autor, 2022.

Considere um sistema operando em malha fechada conforme apresentado na Figura 5. Nesse diagrama de blocos, a planta em malha aberta (apenas o conversor) é representada pela função de transferência G(z), onde a saída de interesse é

$$y(k) = G(z)u(k) + v(k) \tag{6}$$

k corresponde a unidade temporal em tempo discreto, z é operador de deslocamento do tempo definido como *zx(t) = x(t + 1)*, u(k) é o sinal aplicado à entrada da planta e v(k) é um sinal correspondente ao ruído do processo, onde está incluso todas as características imperfeitas do sistema não descritas por G(z). Pode-se considerar que o sinal de entrada da planta é composto por um sinal $u_c(k)$ vindo do controlador somado a um sinal de perturbação de entrada *w(k)*, de forma que $u(k) = u_c(k) + w(k)$.

Suponha que o sinal de saída do controlador é definido por

$$u_c(k) = C(z, \rho)(r(k) - y(k)) \tag{7}$$

onde r(k) é o sinal de referência e C(z, ρ) é um controlador parametrizado por ρ. O objetivo da etapa de projeto de controlador é a determinação do vetor de parâmetros ρ que leve y(k) a ter o comportamento desejado. A estrutura C(z, ρ) pode representar uma grande variedade de controladores clássicos da literatura de controle. Em particular, o controlador PID (proporcional integral derivativo) podem ser descrito como:

$$C(z, k_p, k_i, k_d) = k_p + k_i \frac{z}{z-1} + k_d \frac{z-1}{z} \tag{8}$$

[1] É considerado que: uma dada função de transferência *H(z)*, que é uma função racional, possui polinômios *nH(z)* e *dH(z)* no numerador e no denominador, respectivamente, de forma que *H(z) = nH(z)/dH(z)*; o grau de um polinômio *p(z)* é descrito por *degp(z)*; o grau relativo de uma função de transferência é dado por *H(z) = degdH(z) - degnH(z)*.

onde k_p, k_i e k_d correspondem aos ganhos proporcional, integral e derivativo respectivamente. A expressão do controlador pode ser reescrita em uma forma vetorial por

$$C(z,\rho) = \rho^T \bar{C}(z) \tag{9}$$

onde o vetor de parâmetros ρ e o vetor de funções de transferências $\bar{C}(z)$ são definidos como

$$\rho = \begin{bmatrix} k_p \\ k_i \\ k_d \end{bmatrix} \quad \bar{C}(z) = \begin{bmatrix} 1 \\ \frac{z}{z-1} \\ \frac{z-1}{z} \end{bmatrix}. \tag{10}$$

A partir da inserção do controlador C(z, ρ) no diagrama de blocos da Figura 5, é possível determinar as seguintes relações de malha fechada:

$$\begin{cases} y(k,\rho) = T(z,\rho)r(k) + S(z,\rho)v(k) + Q(z,\rho)w(k) \\ S(z,\rho) = \frac{1}{1+C(z,\rho)G(z)} \\ T(z,\rho) = \frac{C(z,\rho)G(z)}{1+C(z,\rho)G(z)} = C(z,\rho)G(z)S(z,\rho) \\ Q(z,\rho) = \frac{G(z)}{1+C(z,\rho)G(z)} = \frac{T(z,\rho)}{C(z,\rho)} = S(z,\rho)G(z) \end{cases} \tag{11}$$

Em (11), S(z, ρ) corresponde a função de sensibilidade; T(z, ρ) corresponde a função de sensibilidade complementar, visto que T(z, ρ) = 1 - S(z, ρ); Q(z, ρ) corresponde a função de sensibilidade do sistema a distúrbios de entrada.

Suponha agora que o comportamento desejado para o sistema em malha fechada é definido pela função de transferência $T_d(z)$, ou seja,

$$y_d(k) = T_d(z)r(k). \tag{12}$$

Nesse caso, o problema de projeto seria encontrar um vetor de parâmetros ρ tal que y(k, ρ) chegue o mais próximo possível de y_d(k), ou seja, minimize a seguinte função objetivo (REMES, 2021):

$$J^{MR}(\rho) = ||[T(z,\rho) - T_d(z)]r(k)||_2^2 \tag{13}$$

Note que a minimização de (13) implica no comportamento em malha fechada T(z, ρ) ser o mais próximo possível do comportamento desejado $T_d(z)$. Este objetivo de minimização pode ser alcançado através do controlador ideal definido como

$$C_d^{MR}(z) = \frac{T_d(z)}{G(z)(1 - T_d(z))}. \tag{14}$$

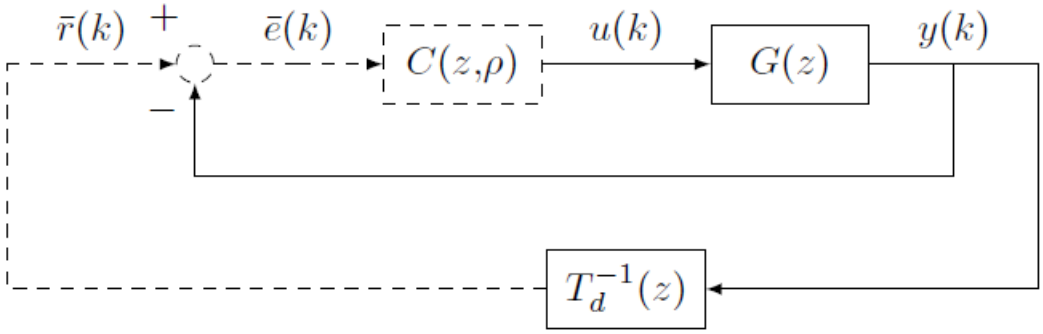
2.3 *Virtual Reference Feedback Tunning*

Na abordagem DD, G(z) é desconhecido, logo $C_d{}^{MR}(z)$ não pode ser obtido diretamente por (14). Dito isso, se um conjunto de dados entrada-saída da planta for obtido, é possível formular um novo problema de otimização, onde o objetivo de minimização é identificar o controlador C(z, ρ) cuja resposta do sistema em malha fechada seja próxima àquela obtida com o controlador ideal $C_d{}^{MR}(z)$. Como se deseja minimizar (13) sem o conhecimento de G(z) ou aproximações, tal objetivo se enquadra numa abordagem de projeto DD no domínio do tempo.

O método de projeto de controladores *Virtual Reference Feedback Tunning* (VRFT) é um método DD baseado em dados coletados no domínio do tempo. Este é desejável por ser do tipo *one-shot*, que significa que ele precisa de apenas um grupo de dados entrada-saída da planta para a determinação do controlador. Além disso, ele otimiza o desempenho do sistema em malha fechada para a resposta à referência. Estes pontos motivaram (REMES, 2021) a escolhe-lo como método para o projeto de controladores para os conversores CC-CC escolhidos e, assim, será o método adotado para a execução deste trabalho.

O VRFT precisa de um conjunto de dados entrada-saída coletados da planta, um controlador linearmente parametrizado como descrito em (9) e um modelo de referência a ser seguido($T_d(z)$). O método VRFT é baseado na construção dos sinais virtuais $\bar{r}(k)$ e $\bar{e}(k)$ ilustrados na Figura 6.

Figura 6 – Sistema em malha fechada com os sinais virtuais do método VRFT



Fonte: (REMES, 2021)

De forma simplificada, os dados necessários para o método são adquiridos, inicialmente, considerando um sistema sem ruído. Deve ser aplicado sobre a planta, um sinal de entrada *u(k)* e é adquirido o sinal de saída *y(k)*. Esses sinais devem ser armazenados para formar o conjunto de dados entrada-saída com um tamanho N de amostras coletadas. O sinal de referência virtual e de erro virtual, $\bar{r}(k)$ e $\bar{e}(k)$, respectivamente, podem ser reconstruídos a partir de $T_d(z)$ e do sinal de saída $y(k)$ armazenado pelas seguintes relações:

$$\bar{r}(k) = T_d^{-1}(z)y(k) \tag{15}$$

$$\bar{e}(k) = \bar{r}(k) - y(k) = (T_d^{-1}(z) - 1)y(k) \tag{16}$$

O sinal $\bar{r}(k)$ se trata do sinal virtual de referência do sistema, ou seja, este é o sinal que produziria a saída y(k) caso fosse aplicado a planta em malha fechada. Da mesma forma, $\bar{e}(k)$ é o sinal virtual de erro que, caso fosse aplicado ao controlador ideal, produziria a ação de controle u(k) do experimento.

Com os sinais $\bar{e}(k)$ e $u(k)$, o controlador C(z, ρ) pode ser projetado visando a minimização da seguinte função custo:

$$J^{VR}(\rho) = ||L(z)[u(k) - C(z,\rho)\bar{e}(k)]||_2^2 \tag{17}$$

onde L(z) é um filtro de ponderação. Quando o controlador C(z,ρ) pertence à chamada Classe de Modelos (BAZANELLA; CAMPESTRINI; ECKHARD, 2011), isto é, existe um valor ótimo de ρ_d tal que C(z, ρ_d) = $\mathrm{C_d}$(z), então $J^{VR} = 0$ implica em $J^{MR} = 0$. Por outro lado, caso o controlador escolhido não tenha graus de liberdade suficientes, se diz que o controlador está fora da classe de modelos e, consequentemente, o valor ótimo de $J^{VR} \neq 0$. Nesse caso, o filtro L(z) tem a função de aproximar os mínimos de J^{VR} e de J^{MR}. Em geral, a escolha de L(z) pode variar dependendo da aplicação e do C(z, ρ) escolhido, mas (BAZANELLA; CAMPESTRINI; ECKHARD, 2011) sugere a utilização de um filtro padrão dado por

$$L(z) = T_d(z)(1 - T_d(z)). \tag{18}$$

2.4 Implementação em Python do método VRFT

Para fazer a implementação do método VRFT para sua utilização numa interface gráfica seria necessário implementar todos os algoritmos do método de forma que um programa desenvolvido fosse capaz de fazer a identificação dos parâmetros do controlador. Para suprir esta necessidade há a biblioteca *pyvrft*, desenvolvida para a linguagem de programação Python por (ECKHARD; BOEIRA, 2019).

A biblioteca traz uma implementação das funcionalidades necessárias para aplicar o método VRFT, iniciando na etapa posterior a aquisição dos dados da planta do usuário, onde deve ser feito um pré-processamento destes para dar início ao projeto do controlador. A biblioteca tem funções que fazem o projeto do controlador, projetam o filtro caso necessário na aplicação e funções que fazem inversões estáveis de sistemas lineares. Vale

ressaltar que esta biblioteca não traz uma implementação do método VRFT com critério flexível. Esta biblioteca foi utilizada como a base para o desenvolvimento do trabalho.

Outro aspecto importante para a execução do trabalho é o desenvolvimento de uma janela gráfica que faça a interação com o usuário para obter as informações que este precisa fornecer à ferramenta. Como complemento a essa janela, algumas formas de validação do resultado obtido podem ser adicionadas, para que a aplicação disponibilize formas que permitam verificar a qualidade do modelo projetado.

Como a biblioteca utilizada para a implementação do método VRFT foi disponibilizada para a linguagem de programação Python, o caminho mais natural foi optar por desenvolver a interface gráfica também nesta linguagem. Assim, a biblioteca *PyQt6* foi escolhida para implementar as funcionalidades da janela.

2.5 Considerações Finais

Com esta revisão de alguns conceitos básicos de conversores CC-CC e de abordagens ao desenvolvimento de controladores baseados em dados, foi possível passar para a etapa de desenvolvimento das funcionalidades esperadas da aplicação. A abordagem do método VRFT ao problema de projeto do controlador ser do tipo *one-shot* se mostra bastante promissora por ser capaz de projetar uma estrutura de controlador sem a necessidade de identificação de informações muito complexas do sistema do usuário.

Neste capítulo também foi comentado sobre o uso de interfaces gráficas para auxiliar o usuário a tratar do problema de uma forma mais amigável, além da implementação disponibilizada em Python para a execução do método VRFT. Uma biblioteca que traga esta implementação reduz a barreira de conhecimento aprofundado necessário para aplicar o método VRFT, enquanto que, uma interface gráfica desenvolvida como se fosse uma ferramenta para aplicar o método agrega a este ponto reduzindo ainda mais a barreira de conhecimento necessária sobre a linguagem em que a biblioteca foi implementada, no caso deste trabalho, a linguagem Python.

3 Desenvolvimento da GUI

Neste capítulo serão abordadas as ferramentas escolhidas para o desenvolvimento do trabalho, os requisitos estipulados para cada funcionalidade da aplicação gerada e uma explicação de como cada estrutura da ferramenta foi implementada, tanto do ponto de vista do planejamento da interface gráfica quanto do ponto de vista da criação das relações relevantes entre cada objeto da interface. Também são expostos os formatos esperados dos dados que o usuário irá fornecer para a aplicação. Outro ponto é que a ferramenta desenvolvida foi criada para funcionar em sistemas operacionais do tipo Windows.

3.1 Ferramentas

As ferramentas utilizadas para o desenvolvimento das atividades relacionadas a este trabalho estão listadas abaixo, junto do racional para a escolha de cada uma.

- *JetBrains PyCharm Community* 2022.1 Build 221.5080.212: *Integrated Development Enviroment*(IDE) escolhida para o desenvolvimento do programa que será feito para implementar a GUI;

IDE desenvolvida pela JetBrains e distribuída para uso gratuito no desenvolvimento de aplicações de código aberto em Python. Foi utilizada para o desenvolvimento das funcionalidades da interface gráfica, desde a interpretação dos comandos feitos pelo usuário até a manipulação dos dados para gerar a saída estimada. Escolha feita devido a familiaridade previamente adquirida com a ferramenta.

- Python 3.10.2: Linguagem Python de programação;

Linguagem de programação desenvolvida na década de 90 para aplicações que sigam o paradigma de orientação a objetos. Parte central de todas as atividades relacionadas ao desenvolvimento deste trabalho. Escolha feita devido a familiaridade previamente adquirida com a linguagem e pela biblioteca *pyvrft* ter sido desenvolvida para esta linguagem. Esta linguagem serve como uma alternativa ao software Matlab, através de diversas bibliotecas que trazem implementações relevantes associadas a sistemas de controle.

- QT Designer 5.11.1: IDE para desenvolvimento de janelas gráficas através do *framework* Qt;

IDE desenvolvida pela Qt e distribuída de forma gratuita para o desenvolvimento de aplicações de *software* livre. Utilizada para desenvolver a interface que mostra ao usuário as opções de comunicação com a aplicação dispostas a ele para a entrada de dados e configurações necessárias para a implementação do método VRFT, assim como interface de visualização dos resultados gerados através do método. Escolha feita por ser uma das opções de desenvolvimento de interfaces gráficas mais consolidadas para a linguagem Python.

- Biblioteca *pyqt6* 6.3.0: Biblioteca que faz a interpretação das interfaces gráficas geradas com o Qt Designer. Está sob a licença GPL v3 que permite distribuição e uso pessoal, além de uso comercial se o código fonte for disponibilizado em conjunto com a aplicação;

Biblioteca que implementa objetos e métodos utilizados para criar e interpretar janelas gráficas, inclusive aquelas desenvolvidas com o Qt Designer. Nesta aplicação, foi utilizada para possibilitar a comunicação com o usuário da aplicação através das opções de entrada de dados e dos resultados obtidos pelas entradas escolhidas por este na GUI. Esta biblioteca fornece uma extensa documentação para ilustrar como cada um dos métodos de cada classe definida pela biblioteca deve ser utilizada (QTCOMPANY, 2022). Escolha feita por ser uma das opções mais consolidadas para o desenvolvimento de interfaces gráficas, além de funcionar bem em conjunto com o Qt Designer.

- Biblioteca *pyvrft* 1.1: Biblioteca que traz a implementação do método VRFT na linguagem Python. Está sob a licença MIT que permite modificação, distribuição e uso pessoal e comercial da biblioteca;

Biblioteca que implementa os algoritmos necessários para algumas manipulações de dados necessárias e para a implementação do método VRFT na linguagem Python. Escolha feita devido a ser uma das poucas implementações do método feita para a linguagem Python, além disso também foi desenvolvida no âmbito do PPGEE da UFRGS.

- Biblioteca *Numpy* 1.21.2: Biblioteca com a implementação de métodos e objetos matemáticos na linguagem Python. Necessária para o funcionamento da biblioteca *pyvrft*. Está sob a licença BSD *3-Clause* que permite modificação, distribuição e uso pessoal e comercial da biblioteca, mas que proíbe o uso sem permissão do nome do projeto ou dos seus contribuintes para promoção de produtos que a utilizem;

Biblioteca que implementa soluções para trabalhar com matrizes e arranjos multi-dimensionais, além de diversas funções matemáticas para operar sobre estas matrizes.

Escolha feita por ser uma biblioteca muito utilizada em aplicações desenvolvidas em Python e por ser necessária para a biblioteca *pyvrft*.

- Biblioteca *Matplotlib* 3.4.3: Biblioteca com a implementação de métodos de visualização de dados na liguagem Python. Necessária para o funcionamento da biblioteca *pyvrft*. Está sob a licença BSD *3-Clause*;

Biblioteca que implementa funções para visualização gráfica de dados na linguagem Python. Tem uma boa sinergia com a biblioteca *Numpy*. Escolha feita por ser uma biblioteca muito utilizada em aplicações desenvolvidas em Python e pela familiaridade previamente adquirida com a biblioteca.

- Biblioteca *Scipy* 1.7.1: Biblioteca usada para manipulação e visualização de operações matemáticas na linguagem Python. Necessária para o funcionamento da biblioteca *pyvrft*. Está sob a licença BSD *3-Clause*;

Biblioteca que expande as funcionalidades da biblioteca *Numpy*, com uma ênfase em aplicações para engenharia em geral. Escolha feita por ser uma biblioteca muito utilizada em aplicações desenvolvidas em Python e por ser necessária para a biblioteca *pyvrft*.

- Biblioteca *Control* 0.9.1 : Biblioteca que implementa um ambiente *Matlab-like* para trabalhar com sistemas de controle na linguagem Python. Necessária para manipulações e operações matemáticas necessárias sobre as funções de transferência. Está sob a licença BSD *3-Clause*;

Biblioteca que traz funções interessantes para sistemas de controle, implementando um ambiente de funcionalidades similar ao do software Matlab na linguagem Python. Nessa aplicação, foi utilizada principalmente para manipulações matemáticas de funções de transferência, do comportamento desejada do sistema ao cálculo da função de sensibilidade estimada do sistema completo. Escolha feita por possibilitar a manipulação matemática de alguns dados usados pela aplicação, funcionalidade não suportada pela biblioteca *Scipy* .

- Biblioteca *Pandas* 1.4.2: Biblioteca usada para criação e manipulação de *dataframes* na linguagem Python. Necessária para a etapa de leitura dos dados fornecidos pelo usuário. Está sob a licença BSD *3-Clause*;

Biblioteca que traz funções para manipulação de *dataframes* gerados a partir do conjunto de dados fornecido pelo usuário para usar na aplicação do método VRFT. Foi utilizada para fazer a leitura dos arquivos com os dados coletados da planta do usuário, tanto na etapa de projeto do controlador quanto na etapa de validação do comportamento

em malha fechada. Escolha feita por ser uma biblioteca muito utilizada em aplicações desenvolvidas em Python e pela familiaridade previamente adquirida com a biblioteca.

- Biblioteca *OpenPyXL* 3.0.9: Biblioteca que traz métodos para leitura de arquivos de planilhas de dados, como do Microsoft Excel. Necessária para o funcionamento da biblioteca *Pandas*. Está sob a licença MIT;

Biblioteca que traz métodos para leitura de dados a partir de arquivos em formatos padronizados de planilhas de dados, como .xlsx, .csv, etc. Dependência necessária para o funcionamento da biblioteca *Pandas*.

- Biblioteca *auto-py-to-exe* 2.18.2: Biblioteca que traz uma forma de gerar um arquivo executável para um programa desenvolvido na linguagem Python. Está sob a licença MIT;

Esta biblioteca traz uma aplicação que é capaz de gerar arquivos executáveis a partir de programas escritos em Python. Usada para gerar o executável da aplicação deste trabalho disponibilizado no repositório informado, removendo a necessidade de futuros usuários desta ferramenta precisarem configurar um sistema com todas as dependências necessárias, nas versões corretas, para utilizar esta aplicação.

3.2 Requisitos

Para atender as funcionalidades desejadas desta ferramenta, alguns requisitos foram estabelecidos de forma que, quando atendidos, permitam o seu correto funcionamento. Abaixo serão listadas as funcionalidades previstas para a janela gráfica, pensadas de forma que atendendo os requisitos de cada uma, se alcance os objetivos propostos para este trabalho.

A interface gráfica foi desenvolvida considerando que o projeto do controlador para a planta do projetista ocorrerá em três etapas definidas como: planejamento da coleta do conjunto de dados de entrada-saída da planta; aplicação do método VRFT a partir dos dados coletados, do modelo de referência desejado, da classe de controlador escolhida e do filtro L(z); análise quantitativa e visual do controlador projetado a partir de dados obtidos em malha fechada e da simulação da $T_d(z)$. A seguir, serão detalhadas cada uma dessas etapas.

3.2.1 Requisitos da Etapa de Pré-Projeto

O primeiro conjunto de funcionalidades da aplicação está relacionado a etapa de pré-projeto do controlador para a planta do usuário. Estas se relacionam com a etapa de

coleta dos dados que serão usados para a aplicação do método VRFT. Conversores CC-CC são sistemas com dinâmica aproximadamente linear em torno de seu ponto de operação. No entanto, variações na tensão de alimentação ou na carga podem alterar o seu ciclo de trabalho e, por consequência, fazer com que o conversor acabe operando fora do ponto de operação nominal. Em (REMES, 2021) é estabelecido o melhor ponto de operação para a coleta de dados do sistema, utilizando informações como o tipo de conversor, do seu ciclo de trabalho nominal e a tensão de saída desejada.

Este ponto de operação ideal, em conjunto com a topologia do conversor deve ser informado pelo usuário. Além do ponto de operação ideal, a coleta de dados pode ser realizada com o sistema operando em malha aberta ou em malha fechada com um dado controlador conhecido. Dependendo da aplicação, a coleta de dados em malha fechada pode apresentar uma melhor robustez numérica na implementação do VRFT. Nesse caso, fica a cargo do projetista a escolha de qual controlador utilizar, sendo que (REMES, 2021) sugere a utilização de um controlador proporcional com ganho $\mathrm{K_p}$ que depende da topologia do conversor e do ponto de operação. Pela análise da função de transferência (14), segue que o sistema em malha fechada será estável para todos conversores da Tabela 1 se

$$K_p < \frac{1}{G_{d0}} < \frac{1}{G_{d0}d} < \infty. \tag{19}$$

Considerando que d assume valores entre 0 e 1, a restrição anterior é atendida para qualquer

$$K_p < \frac{1}{G_{d0}}. \tag{20}$$

A etapa de pré projeto da interface a ser desenvolvida deve informar ao usuário qual o valor máximo de $\mathrm{K_p}$ que pode ser utilizado caso este opte pela coleta dos dados em malha fechada.

3.2.2 Requisitos da Etapa de Projeto

O segundo conjunto de funcionalidades estão relacionadas à implementação de fato do método VRFT no sistema. A primeira funcionalidade é o carregamento de dados de entrada e saída da planta. Além disto, caso ocorra a contaminação dos dados coletados devido a ruídos, o método pode apresentar uma estimativa polarizada do controlador ideal. Neste caso, é possível adaptar o método VRFT para considerar mais um conjunto de dados, este referente ao uso de variáveis instrumentais, que visam eliminar esta possível polarização (CAMPI; LECCHINI; SAVARESI, 2002; BAZANELLA; CAMPESTRINI; ECKHARD, 2011). O usuário também deverá informar qual a função de transferência desejada, podendo informar essa característica através de parâmetros como sobressinal (*overshoot*) e tempo

de acomodação desejados, ou então informando diretamente a função de transferência $T_d(z)$. Nesta parte também é onde o usuário define qual o tipo de controlador que ele gostaria de utilizar na sua planta, podendo ser do tipo Proporcional, Proporcional-Integral, Proporcional-Derivativo, Proporcional-Integral-Derivativo ou então informando o vetor de funções de transferência $\bar{C}(z)$ de outro controlador linearmente parametrizável que não corresponda a estes citados. Por fim, esta etapa também compreende a parte de definição do filtro L(z) utilizado durante o método VRFT para aproximar os mínimos de J^{MR} e J^{VR}. A partir do momento em que todas as informações necessárias forem informadas pelo usuário, o método pode ser aplicado para estimar os ganhos referentes ao tipo de controlador escolhido.

3.2.3 Requisitos da Etapa de Análise de Desempenho

Por fim, o último conjunto de funcionalidades planejadas para a aplicação está relacionado com a análise dos resultados quando o controlador projetado é adicionado a planta, além de estimativas de comportamento do sistema projetado. A partir de um conjunto de dados entrada-saída do sistema em malha fechada com o controlador projetado, é possível comparar o comportamento da saída frente ao comportamento da função de transferência desejada pelo projetista, informada na etapa de projeto do controlador. Havendo este conjunto de dados do sistema em malha fechada, será calculado o valor da função objetivo (13). Por fim, ainda é possível estimar a função de sensibilidade do sistema projetado considerando o equacionamento para S(z, ρ) em (11). Essas estimativas podem auxiliar o projetista a ter uma noção melhor da adequação do controlador calculado aos requisitos do projeto.

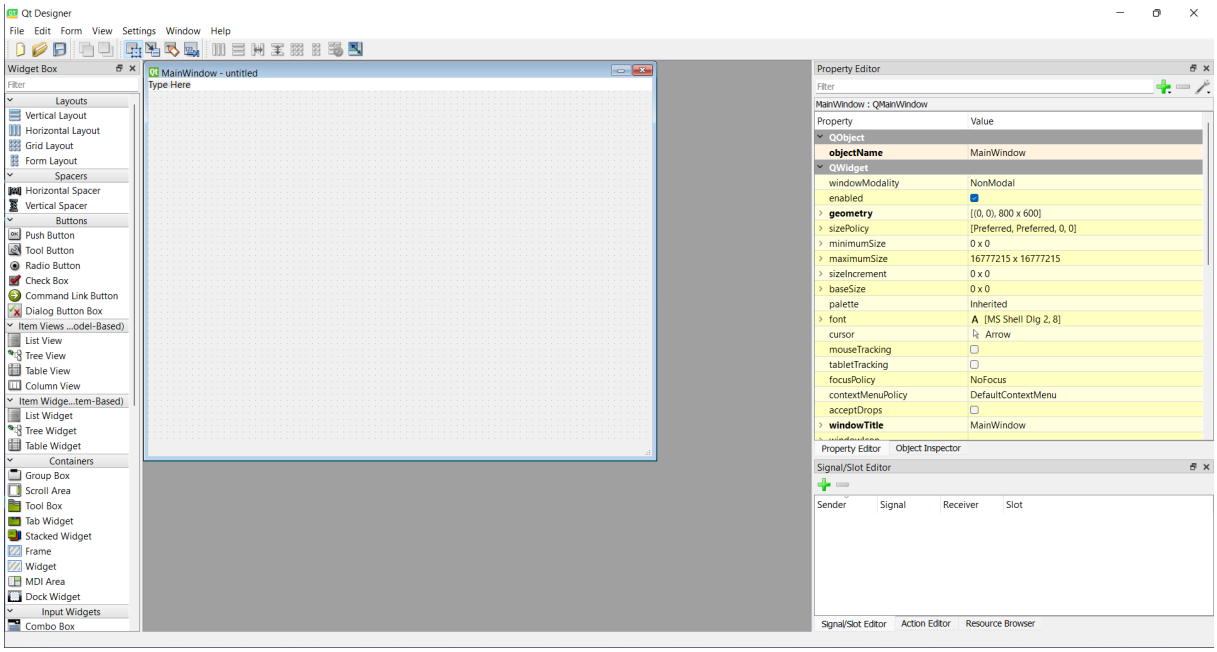
3.3 Desenvolvimento do Ambiente

Nesta seção será descrito como cada funcionalidade da ferramenta foi desenvolvida, tanto na parte de criação da interface gráfica pelo Qt Designer como no desenvolvimento do código em Python. A ferramenta desenvolvida funciona através da captura de eventos. Isso significa que, com a GUI aberta, a aplicação aguarda que o usuário faça alguma comunicação com o programa através de algum dos elementos dispostos para este fim, para então executar a ação definida para aquela opção selecionada. O chamado evento seria, por exemplo, o usuário selecionando o botão para indicar onde está o conjunto de dados entrada-saída da planta, que faria com que a aplicação captasse a seleção deste botão específico para então, neste caso, abrir uma nova janela com o explorador de arquivos do sistema operacional onde pode ser selecionado o arquivo com a planilha de dados.

Começando pelo desenvolvimento da interface gráfica, de modo simplificado existem duas formas para gerar uma GUI utilizável através da biblioteca *pyqt6*, com a primeira

sendo criar cada elemento da janela por linhas de código, onde cada característica de cada elemento precisa de um comando específico para ser configurado. Essas características podem ser o tipo do objeto, o nome dado a ele, o tamanho desejado, a posição na janela, etc. Apesar de funcional, esta opção de desenvolvimento é muito demorada, trabalhosa e pouco intuitiva. Em função disso, foi optado por utilizar o Qt Designer, que é uma ferramenta que permite a criação de GUIs de forma gráfica, onde há uma lista objetos e de propriedades que podem ser definidas pelo desenvolvedor. A Figura 7 mostra a tela inicial do Qt Designer onde pode ser vista uma janela sem nenhum elemento adicionado e as duas listas mencionadas, de objetos e de propriedades, uma de cada lado da janela.

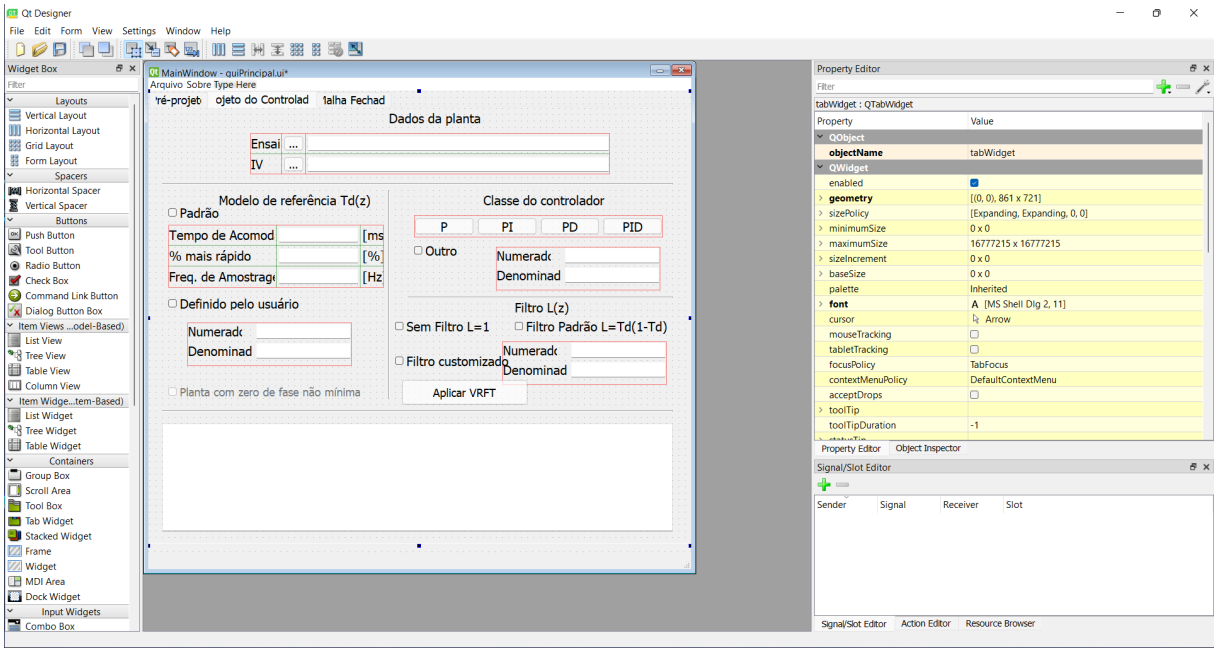
Figura 7 – Tela inicial de desenvolvimento do Qt Designer



Fonte: o Autor, 2022.

No lado esquerdo da tela está a lista com todos os elementos que podem ser utilizados para compor a GUI que se deseja criar, bastando arrastá-los da lista para dentro da janela central para posicionar aquele escolhido. No lado direito está a lista com todas as propriedades do objeto selecionado sendo que para editar, basta substituir a informação padrão que já vem configurada quando algum objeto é selecionado. A partir desta interface foi possível dispor os objetos necessários para montar as funcionalidades desejadas da aplicação. A Figura 8 mostra como ficou uma das abas da janela montada dentro do Qt Designer.

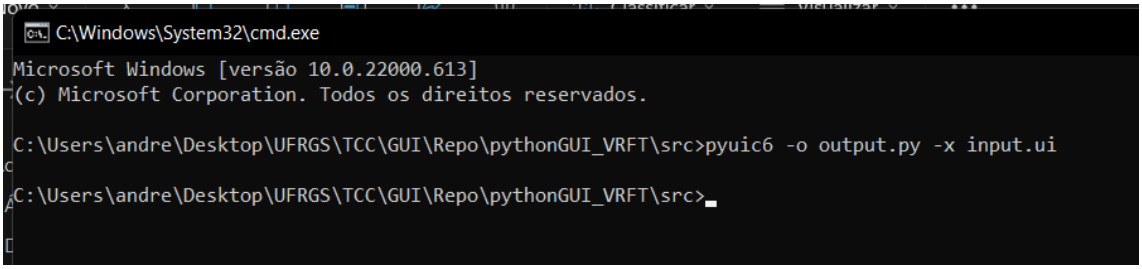
Figura 8 – Segunda aba da aplicação no Qt Designer



Fonte: o Autor, 2022.

No software Qt Designer, quando uma GUI em desenvolvimento é salva, o arquivo gerado é do tipo '.ui', onde ficam salvas todas as configurações definidas a partir do posicionamento e configuração das propriedades dos objetos que compõem a janela. Este tipo de arquivo por si só não é interpretado diretamente pelo Python, assim, antes de utilizar a janela criada, é necessário converter este arquivo em um arquivo padrão de código Python. Para tal, a biblioteca *pyqt6* fornece uma utilidade que faz a conversão automática do arquivo para código Python através de um comando específico. Abrindo o Prompt de Comando dentro da pasta em que se encontra o arquivo '.ui' criado, ao utilizar o comando exibido na Figura 9, é feita a conversão das informações da janela gráfica salvas em 'input.ui', ou o nome que for dado ao arquivo com a configuração da janela, numa classe em Python que implementa todos os elementos da janela criada e salvará essa classe em 'output.py', ou qualquer outro nome que seja definido para o arquivo Python. Caso não seja informado um caminho diferente para este arquivo 'output.py', ele será salvo na mesma pasta de 'input.ui'.

Figura 9 – Comando para converter a janela montada em código Python



Fonte: o Autor, 2022.

```
pyuic6 -o output.py -x input.ui
```

A partir desse arquivo 'output.py', é possível integrar os objetos dispostos com as funcionalidades planejadas para cada um. Este arquivo, após conversão, é entregue de forma que já possa ser executado, tanto que ao ser executado abrirá uma janela com todos os elementos conforme configurados no Qt Designer. Neste primeiro momento, o usuário já pode interagir com os elementos presentes nesta interface, no entanto, nenhum botão ou caixa de texto vai ter o seu comportamento definido, ou seja, nada irá acontecer ao clicar em um botão, por exemplo.

Por fim, para que fique tudo pronto para que as funcionalidades dos elementos da janela sejam programadas, é necessário fazer duas pequenas mudanças no código deste arquivo gerado automaticamente, o 'output.py' do exemplo. Abaixo é apresentado o início código exatamente conforme ele é gerado pela ferramenta 'pyuic6'.

```
# Form implementation generated from reading ui file 'guiPrincipal.ui'
#
# Created by: PyQt6 UI code generator 6.1.0
#
# WARNING: Any manual changes made to this file will be lost when
   pyuic6 is
# run again.  Do not edit this file unless you know what you are doing.


from PyQt6 import QtCore, QtGui, QtWidgets

class Ui_MainWindow(object):
```

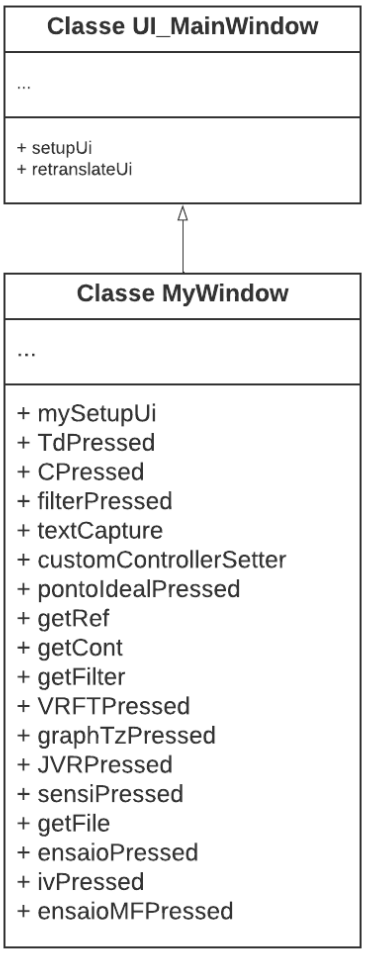
As duas modificações correspondem a incluir ao projeto uma classe diferente tal que a classe 'UI_MainWindow' herde as funcionalidades dessa nova classe ao invés de herdar da classe genérica 'object'. Abaixo pode ser visto como fica este mesmo início do código após as modificações.

```
1  # Form implementation generated from reading ui file 'guiPrincipal.ui'
2  #
3  # Created by: PyQt6 UI code generator 6.1.0
4  #
5  # WARNING: Any manual changes made to this file will be lost when
       pyuic6 is
6  # run again.  Do not edit this file unless you know what you are doing.
7
8
9  from PyQt6 import QtCore, QtGui, QtWidgets
10
11 from PyQt6.QtWidgets import QMainWindow
12
13 class Ui_MainWindow(QMainWindow):
```

Nas linhas 11 e 13 podem ser vistas as alterações feitas. Foi importado para o projeto a classe 'QMainWindow' e definido que a classe 'UI_MainWindow' irá herdar as funcionalidades implementadas por ela. Com estas alterações feitas, não é mais preciso fazer qualquer outra mudança neste arquivo para iniciar a parte de programação das funcionalidades da interface. Como pode ser visto acima, há um aviso no código gerado pela ferramenta 'pyuic6' alertando que modificações feitas manualmente ao código seriam perdidas quando o comando fosse executado novamente, ou seja, qualquer modificação feita à interface gráfica no Qt Designer implica em executar novamente o comando para rodar o 'pyuic6' e perder qualquer adição ou remoção feita diretamente ao arquivo gerado previamente. Em função disso, foi estabelecida a premissa de manter ao mínimo possível a quantidade de modificações feitas nesse arquivo. Isso levou ao próximo passo do andamento do trabalho, que foi a criação, em um arquivo separado, e a utilização de uma segunda classe que herde desta classe 'UI_MainWindow' as configurações que definem os elementos adicionados à GUI. Respeitar essa premissa implicou também no seguimento do princípio de herança do paradigma de orientação à objetos.

Neste segundo arquivo foi adicionada a classe que herda a configuração da janela gráfica e implementa as funções de cada botão, etiqueta e caixa de texto adicionado à ferramenta. A Figura 10 mostra uma diagrama de classes simplificado da aplicação desenvolvida.

Figura 10 – Diagrama de classes simplificado



Fonte: o Autor, 2022.

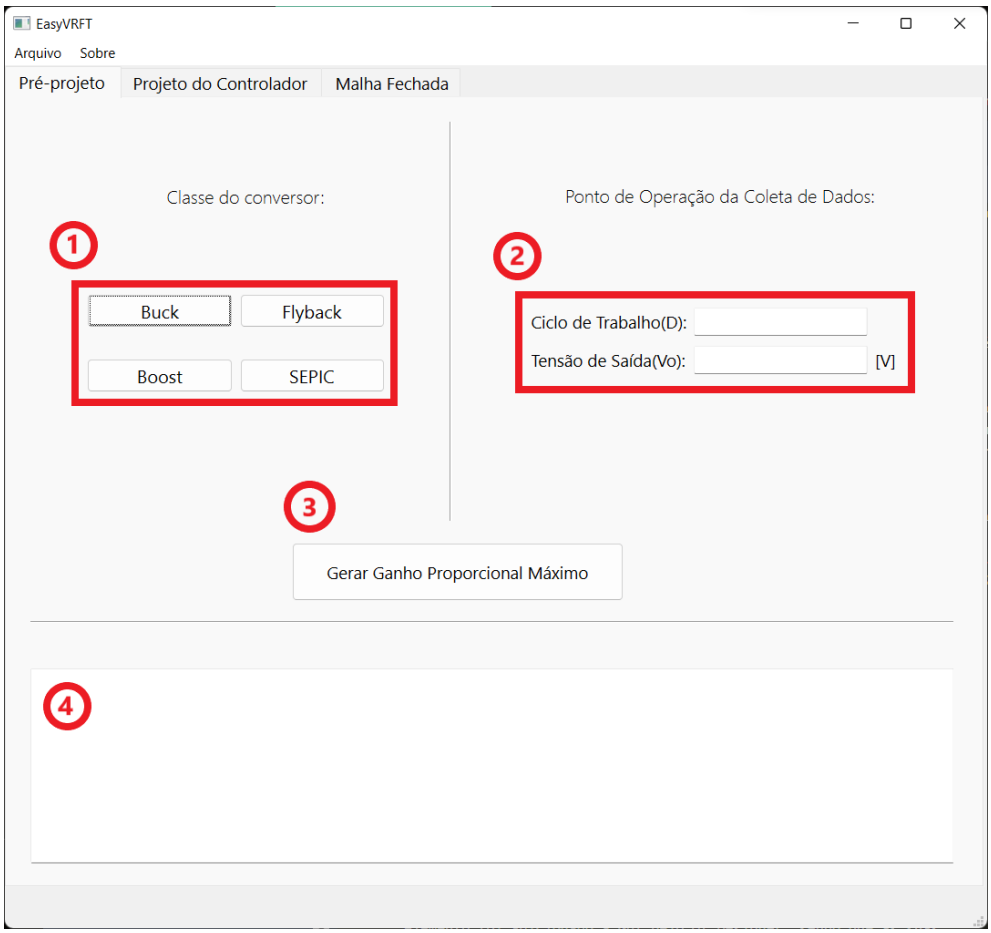
No diagrama de classes apresentado, são mostradas apenas as duas classes em que houveram modificações para a implementação da aplicação. Também, diferentemente de um diagrama de classes padrão, nenhum dos membros das duas classes foram listados, visto que a quantidade destes está na casa das dezenas e não caberiam na figura, além de não trazerem informações tão relevantes para a discussão deste trabalho. Agora, será abordado como foi feito o desenvolvimento das funções de cada uma das abas definidas para a aplicação. Todas as capturas de eventos são feitas através de métodos disponibilizados pela biblioteca *pyqt6*.

3.3.1 Primeira aba - Pré-Projeto

Esta primeira aba foi planejada para calcular o ganho máximo para a coleta de dados em malha fechada a partir dos dados do conversor quando este funciona no Modo de Condução Contínua. Em função desse objetivo, as informações que o usuário precisa fornecer são o tipo de conversor da planta, o ciclo de trabalho nominal e a tensão de saída

do sistema. A Figura 11 traz a tela vista nesta primeira aba com algumas indicações visuais, em vermelho, dos diferentes objetos configurados para formar a interface de Pré-Projeto.

Figura 11 – Primeira aba com indicações visuais dos diferentes elementos da janela



Fonte: o Autor, 2022.

Abaixo segue a relação, conforme numeração, de cada elemento de interação com o usuário com a sua função planejada.

3.3.1.1 Seleção do conversor

Esta etapa corresponde ao bloco 1 da Figura 11. Em um primeiro momento, foram implementados quatro botões diferentes, um para cada um dos grupos de conversores da Tabela 1. Novas topologias de conversores poderão ser adicionadas seguindo a metodologia detalhada a seguir. Cada um destes botões foram adicionados individualmente à GUI através do Qt Designer, para então serem definidos via Python como pertences a um grupo de botões referente a classe de conversor do usuário. Esta definição de grupo de botões possibilita o uso de métodos que auxiliam a verificar quando algum deles for selecionado sem que seja necessário testar todos os botões individualmente para verificar seu estado de seleção. A definição de um grupo de botões ainda traz mais uma funcionalidade interessante, visto que por padrão os botões de um grupo de botões são definidos como exclusivos, ou

seja, só pode haver um botão selecionado ao mesmo tempo e no momento em que o usuário seleciona algum botão, o outro que já estava selecionado é desselecionado automaticamente. Esta função é útil nesta aba por impedir que o usuário acabe selecionando mais de um tipo de conversor para que seja feita a estimativa do ponto de operação por vez. Utiliza a biblioteca *pyqt6* para gerar o grupo de botões.

3.3.1.2 Parâmetros nominais do ponto de operação

Esta etapa corresponde ao bloco 2 da Figura 11, sendo composta por caixas de texto para entrada dos valores de razão cíclica nominal e tensão de saída. Estes elementos da janela gráfica são dedicados a receber as informações referentes as características relevantes do conversor do usuário. Algumas etiquetas foram posicionadas para indicar para o projetista a informação esperada em cada campo de texto. Também foram posicionados os campos de texto mencionados que recebem as informações indicadas. Para estes campos, é esperado que seja informado apenas valores numéricos e com o separador decimal sendo do tipo ponto, que corresponde ao padrão usado na linguagem Python. Todos os outros campos de texto que tem a função de serem entradas de informações providas pelo usuário seguem o padrão em que o ponto é o separador decimal.

3.3.1.3 Cálculo do ganho máximo para coleta de dados

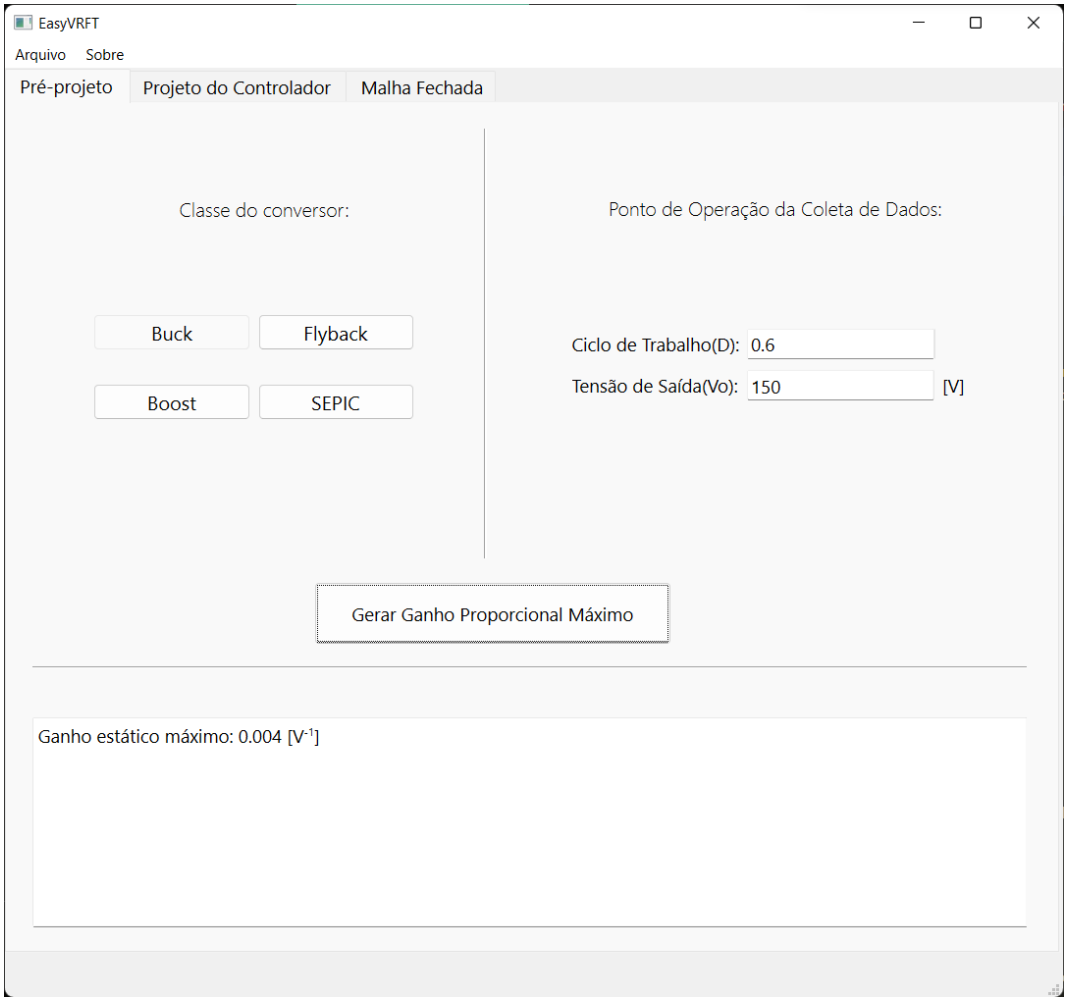
Esta etapa corresponde ao bloco 3 da Figura 11, que traz um botão para informar que os dados necessários para o cálculo do ganho estático máximo foram adicionadas aos seus respectivos campos. Quando o usuário seleciona este botão, a aplicação irá verificar o tipo de conversor selecionado, bem como os valores refentes ao ciclo de trabalho e tensão de saída, para então calcular a estimativa do ponto de operação ideal, conforme (20) e as relações definidas na Tabela 1. Utiliza a biblioteca *pyqt6* para fazer a leitura das informações adicionadas aos campos de texto e para escrever na caixa de texto enumerada como 4 o resultado da operação do cálculo do ponto ideal para coleta dos dados.

3.3.1.4 Caixa de texto de saída

Esta etapa corresponde ao bloco 4 da Figura 11, onde é exibido o valor estimado para K_p. A caixa de texto foi configurada como sendo do tipo somente leitura e utiliza-se a biblioteca *pyqt6* para escrever na caixa de texto o valor obtido.

A Figura 12 mostra uma simulação da funcionalidade desta aba, onde foi selecionado um conversor do tipo *Buck*, foi definido um ciclo de trabalho de 0.6 e uma tensão de saída de 150V. Na caixa de texto da aba pode ser visualizado o ponto ideal estimado para que seja feita a coleta do conjunto de dados para o projeto do controlador.

Figura 12 – Funcionamento da primeira aba da aplicação

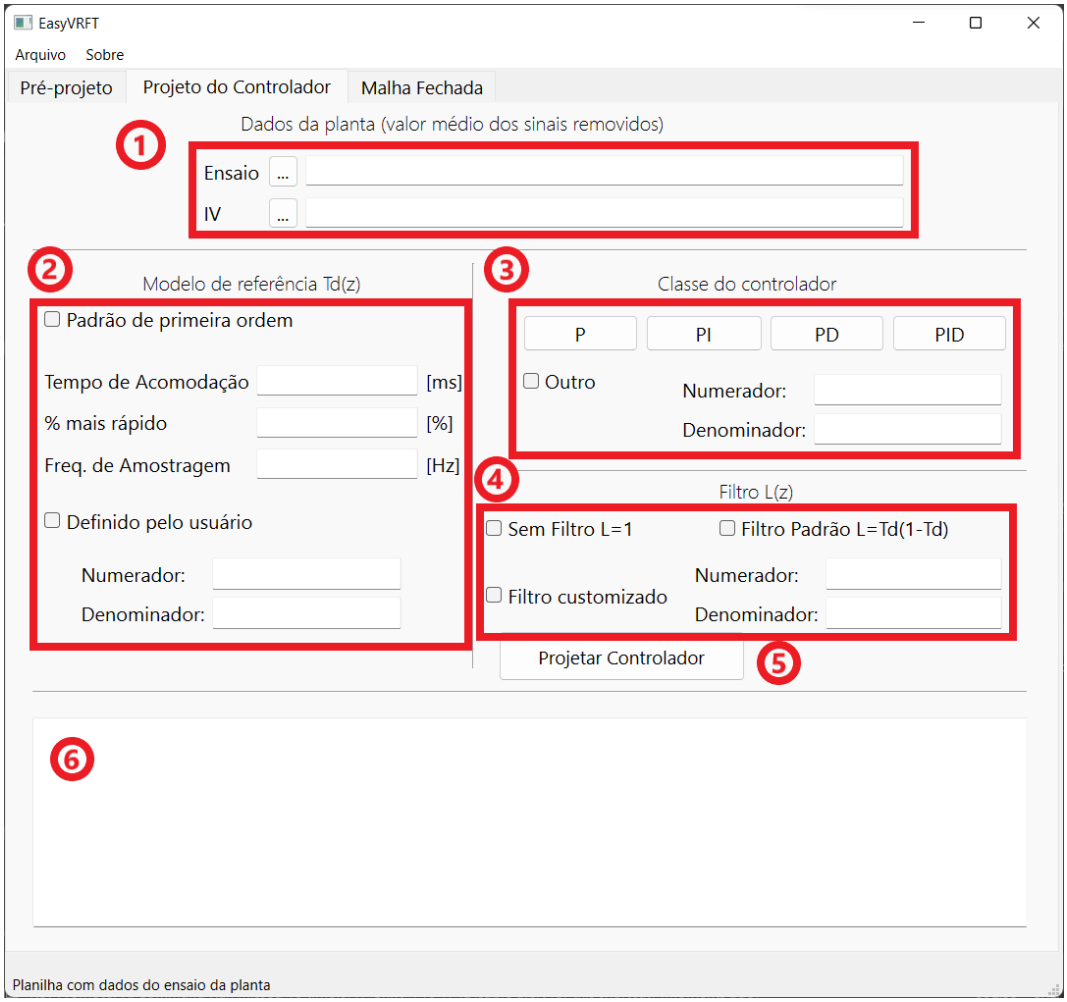


Fonte: o Autor, 2022.

3.3.2 Segunda aba - Projeto do Controlador

A segunda aba foi prevista para ser aquela responsável por agregar as funcionalidades referentes a aplicação do método VRFT. A Figura 13 traz a tela vista na segunda aba, com indicações visuais, em vermelho, dos diferentes objetos configurados para formar a interface de Projeto do Controlador.

Figura 13 – Segunda aba com indicações visuais dos diferentes elementos da janela



Fonte: o Autor, 2022.

Abaixo segue a relação, conforme numeração, de cada elemento de interação com o usuário da interface com a sua função planejada.

3.3.2.1 Entrada dos conjuntos de dados para projeto do controlador

Esta etapa corresponde ao bloco 1 da Figura 13. onde foram adicionadas etiquetas para informar a utilidade de cada campo para o usuário, botões e campos de texto do tipo somente leitura. Ao selecionar qualquer um dos botões com as reticências, uma nova janela é aberta onde pode ser selecionado o arquivo que contém os dados obtidos através de um ensaio que meça os dados de entrada e saída do sistema em malha aberta ou malha fechada. Este conjunto de dados será interpretado como os dados de ensaio padrão da planta ou como os dados de variável instrumental, dependendo de qual dos botões for selecionado. Caso o usuário não queira utilizar as variáveis instrumentais, tanto não informar nenhum arquivo quanto adicionar o mesmo arquivo de dados a esse campo manterá o correto funcionamento da aplicação. Nos campos de texto será exibido o caminho em que o arquivo se encontra quando este for selecionado. Os tipos de arquivo previstos para a aplicação

aceitar são '.xlsx', '.xls' e '.csv'. A ferramenta também espera que os dados informados sejam organizados dentro da planilha fornecida de uma forma específica para os arquivos '.xlsx' e '.xls' e de outra para os arquivos '.csv'. As Figuras 14 e 15 mostram como é esperada esta organização dos dados para cada tipo de arquivo.

Figura 14 – Organização esperada dos dados para arquivos '.xlsx' e '.xls'

	A	B	C
1	Input	Output	
2	0	-0,67843022	
3	1	-0,52970811	
4	1	0,85286324	
5	1	1,9532507	
6	1	2,47543617	
7	1	3,53257379	
8	1	3,99155208	
9	1	4,85231215	
10	1	5,54651261	
11	1	5,50649611	

Fonte: o Autor, 2022.

Figura 15 – Organização esperada dos dados para arquivos '.csv'

	A	B
1	Input,Output	
2	0,-0.67843022	
3	1,-0.52970811	
4	1,0.85286324	
5	1,1.9532507	
6	1,2.47543617	
7	1,3.53257379	
8	1,3.99155208	
9	1,4.85231215	
10	1,5.54651261	
11	1,5.50649611	
12	1,6.20004514	

Fonte: o Autor, 2022.

Este formato esperado para os conjuntos de dados se repete também na próxima aba, onde também é necessário que seja informado outro conjunto para as funcionalidades propostas. Outro ponto relevante sobre os dados esperados, é que, para a correta aplicação do VRFT, o valor médio dos sinais deve ser removido. Atualizações futuras da ferramenta poderão incorporar a funcionalidade de remoção automática desses valores médios. Na versão atual, isso fica a cargo do usuário. A biblioteca *Pandas* é utilizada para ler os dados do arquivo e para fazer as manipulações necessárias para que estes fiquem em um formato

utilizável pela aplicação e a biblioteca *pyqt6* é utilizada para integrar o funcionamento da janela de seleção do endereço do arquivo com o código em Python.

3.3.2.2 Etapa de entrada do modelo de referência desejado

Conforme representado no bloco 2 da Figura 13, essa etapa é composta por um grupo com dois botões, utilizados para informar qual é o tipo de função de transferência que será utilizado para formar o modelo de referência $T_d(z)$ desejado pelo usuário. Estes botões foram implementados de forma que, quando um for selecionado, os campos de texto atrelados ao outro botão são definidos como somente leitura, evitando, assim, que o usuário acabe passando informações não condizentes com a opção selecionada na interface gráfica. As duas opções de definição do modelo de referência foram denominadas como 'Padrão' e 'Definido pelo usuário'. Para a primeira opção, estão vinculados três campos de texto: tempo de acomodação em malha aberta da planta, informado em milissegundos; relação percentual desejada para o tempo de acomodação em malha fechada em relação ao tempo de acomodação em malha aberta[1]; frequência de amostragem da coleta de dados da planta, informada em Hz. Utilizando as considerações de (REMES, 2021), o modelo da $T_d(z)$ pode ser definido como

$$T_d(z) = \frac{1 - p_1}{z - p_1} \tag{21}$$

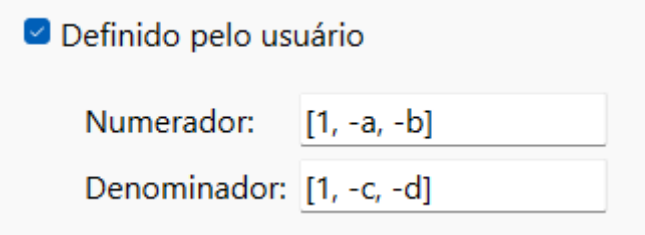
A função de transferência (21) foi escolhida de forma a apresentar |Td(1)|=1, garantindo assim o seguimento de referências constantes em regime permanente. Além disso, O tempo de acomodação em malha fechada pode ser definido através da escolha do polo p_1. Para uma resposta $x_{\%}$ mais rápida que o tempo de acomodação do conversor em malha aberta define-se, então, a seguinte expressão para definir o valor deste polo (REMES, 2021):

$$p_1 = exp\left(-\frac{4T_a}{t_{so}(1 - 0.01x_{\%})}\right) \tag{22}$$

onde T_a corresponde ao período de amostragem da planta e t_{so} corresponde ao tempo de acomodação em malha aberta. Para a opção do segundo botão, 'Definido pelo usuário', há dois campos de texto, que são utilizados para que o usuário informe diretamente a função de transferência para o modelo da $T_d(z)$ que este deseja utilizar. Isso pode ser feito através dos campos que recebem o numerador e o denominador dessa função de transferência, respectivamente. Estes campos foram implementados esperando o formato padrão de texto apresentado na Figura 16.

[1] Por exemplo, o projetista deseja um sistema em malha fechada 20% mais rápido que em malha aberta.

Figura 16 – Formato de entrada de dados esperada para $T_d(z)$



Fonte: o Autor, 2022.

onde 'a', 'b', 'c' e 'd' correspondem a valores numéricos. Do ponto de vista da entrada de dados implementada na ferramenta, para o exemplo dado, a expressão de $T_d(z)$ interpretada seria:

$$T_d(z) = \frac{z^2 - az - b}{z^2 - cz - d} \tag{23}$$

É importante destacar que os polinômios do numerador e denominador podem ser de qualquer grau, sendo os seus coeficientes informados na ordem decrescente de potências de z. A partir da seleção de uma das duas opções dispostas nesta parte da interface, obtém-se o modelo de referência necessário tanto para executar o método VRFT, quanto para as etapas de avaliação do sistema projetado pela aplicação, da aba seguinte. A biblioteca *pyqt6* é utilizada para criar o grupo de botões e atribuir os botões relevantes a este grupo, bem como modificar o estado de escrita-leitura dos campos de texto, conforme interações com o usuário.

3.3.2.3 Escolha da classe do controlador

Como ilustrado no bloco 3 da Figura 13, esta parte é composta por um grupo com cinco botões, um para cada tipo de controlador que pode ser usado na aplicação do método VRFT. Quando o método for aplicado, o resultado exposto ao usuário serão os ganhos para cada parcela da estrutura que forma o controlador selecionado. Novamente, aqui está presente a funcionalidade que impede que haja mais de um dos botões do grupo selecionado ao mesmo tempo. Destes cinco botões, quatro foram definidos para seleção das estruturas de controlador convencionais, o P, o PI, o PD e o PID. O último botão habilita dois campos de texto para que o usuário informe alguma outra estrutura de controlador linearmente parametrizável, mas que não corresponda necessariamente as estruturas citadas. Aqui novamente foi implementada a funcionalidade que faz com que estes campos de texto virem do tipo somente leitura quando qualquer botão referente as estruturas de controladores convencionais for selecionada e habilita a escrita quando o botão denominado "*Outro*" é escolhido.

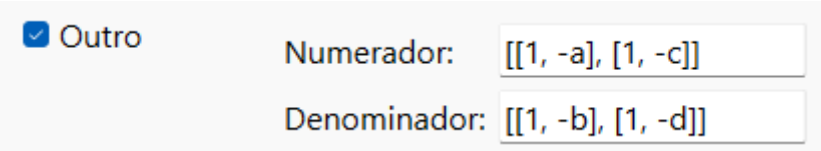
Pela estrutura apresentada em (9), tem-se que $\bar{C}(z)$ é um vetor onde cada um de seus elementos é uma função de transferência com numerador e denominador, ou seja,

$$\bar{C}(z) = \begin{bmatrix} \frac{n_1(z)}{d_1(z)} \\ \frac{n_2(z)}{d_2(z)} \\ \vdots \\ \frac{n_m(z)}{d_m(z)} \end{bmatrix}. \tag{24}$$

Para simplificar a entrada de dados, nos campos "Numerador"e "Denominador"o projetista deve inserir um vetor onde cada um dos elementos são os coeficientes de $n_i(z)$ e $d_i(z)$, i=1,...,m, em ordem decrescente de potências de z.

Para fins de exemplo, considere o caso ilustrado pela Figura 17,

Figura 17 – Formato de entrada de dados esperada para $\bar{C}(z)$



Fonte: o Autor, 2022.

onde 'a', 'b', 'c' e 'd' correspondem a valores numéricos. Do ponto de vista da implementação do VRFT, esses dados seriam interpretados como

$$\bar{C}(z) = K_1 \frac{z-a}{z-b} + K_2 \frac{z-c}{z-d}. \tag{25}$$

Com a estrutura de controlador definida, junto das informações anteriores, fica faltando apenas mais uma característica a ser definida para que o método VRFT possa ser executado. Novamente foi utilizada a biblioteca *pyqt6* para criar o grupo de botões e atribuir os botões relevantes a este grupo, bem como modificar o estado de escrita-leitura dos campos de texto, conforme interações com o usuário.

3.3.2.4 Etapa de seleção do filtro desejado

O bloco 4 da Figura 13 consiste em um grupo de botões para que o usuário faça a seleção do tipo de filtro que será utilizado durante a execução do método VRFT. De forma similar aos outros grupos de botões, este também garante apenas uma única seleção dentre os botões do grupo por vez e faz com que o estado dos campos de texto passem a ser somente leitura enquanto a opção "*Filtro Customizado*" não for selecionada. Os botões definidos neste grupo fazem as seleções entre diferentes filtros possíveis, sendo elas: não aplicar filtro algum nos dados (L(z) = 1); filtro padrão (L(z) = $T_d(z)$ (1 - $T_d(z)$)); filtro genérico com função de transferência definida pelo usuário. Para esta última opção,

a lógica de entrada dos dados é a mesma usada para a definição da função de transferência desejada Seção 3.3.2.2, ou seja, são informados os coeficientes do numerador e denominador de L(z) na ordem decrescente de potências de z.

A partir deste ponto, toda as informações necessárias para a aplicação do método VRFT foram adicionadas a ferramenta. Novamente foi utiliza a biblioteca *pyqt6* para criar o grupo de botões e atribuir os botões relevantes a este grupo, bem como modificar o estado de escrita-leitura dos campos de texto, conforme interações com o usuário.

3.3.2.5 Cálculo do controlador via VRFT

O botão marcado com o número 5 na Figura 13 dará início a execução do método VRFT, supondo que todas as outras informações mencionadas anteriormente para esta aba da aplicação tenham sido informadas pelo usuário. Selecionar este botão fará com que a aplicação utilize as informações dadas para o modelo de referência para definir $T_d(z)$, fará com que seja instanciado um objeto com a estrutura de controlador escolhida, utilizará a seleção de filtro do usuário para estabelecer qual filtro irá passar na execução do método e irá extrair do *dataframe* gerado a partir dos dados fornecidos os sinais de entrada e de saída do experimento feito na planta. A partir de todas estas informações, é utilizada a biblioteca *pyvrft* para finalmente executar o método e obter a parametrização do controlador selecionado. Os coeficientes do controlador são então exibidos na caixa de texto na parte inferior da janela. São utilizadas as seguintes bibliotecas: *Numpy*, para extrair informações necessárias para a manipulação dos dados de ensiao da planta; *pyqt6*, para ler as informações dos campos de texto de cada um dos pontos de entrada de informação e para escrever o resultado obtido no campo de texto; *Math*, para calcular o valor de p_1 do modelo de referência conforme (22), utiliza a biblioteca *Scipy* para gerar os objetos referentes as funções de transferência nos formatos que o método da biblioteca *pyvrft* espera que sejam informadas; *Pandas*, para fazer a leitura e manipulação dos dados referentes aos ensaios entrada-saída da planta, e a biblioteca *Control* para fazer manipulações algébricas nas funções de transferência relevantes.

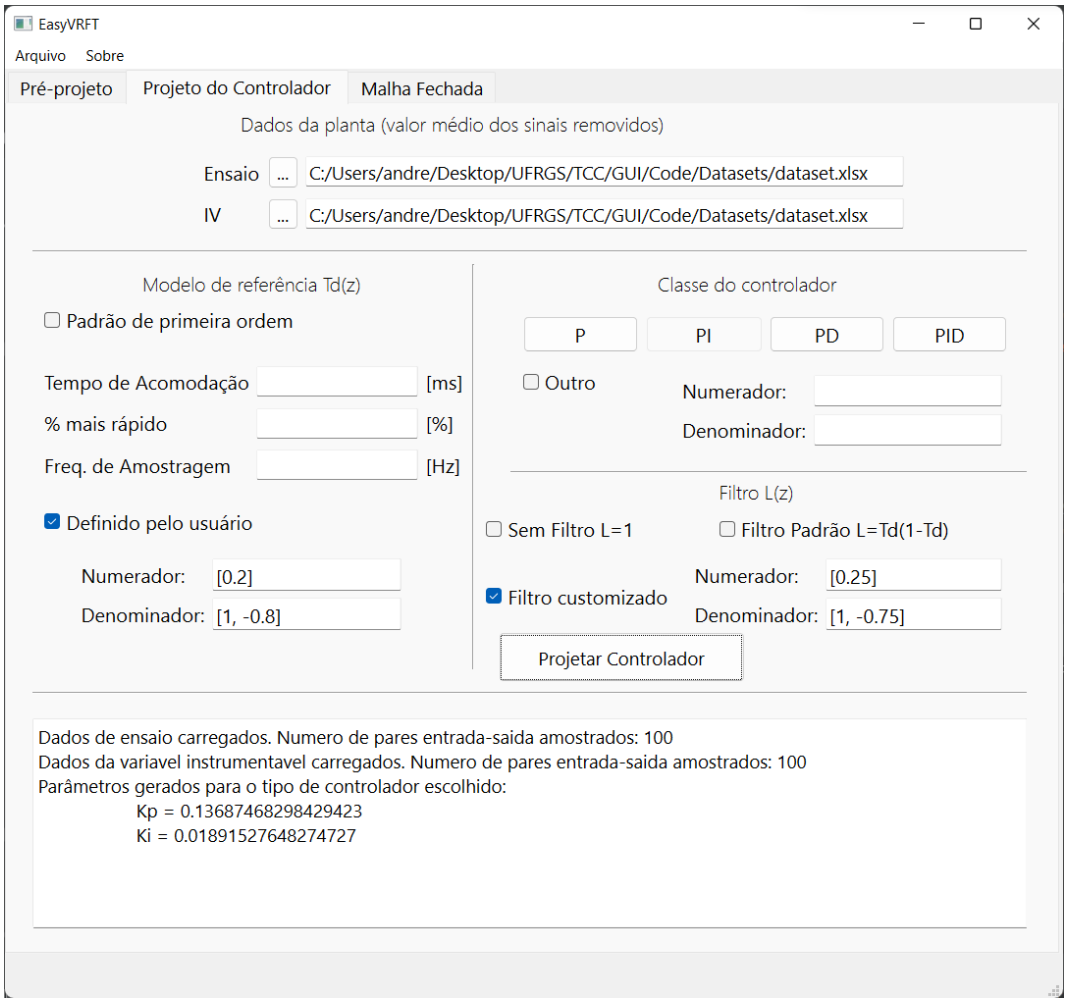
3.3.2.6 Caixa de texto de saída

Por fim, nesta caixa de texto marcada na Figura 13 com o número 6 é onde é exibido o resultado da aplicação do método VRFT a partir dos dados fornecidos pelo usuário. Também é utilizada como forma de comunicação com o usuário, mostrando o número de amostras identificadas nos conjuntos de dados fornecidos, por exemplo. A caixa de texto também foi configurada como sendo do tipo somente leitura. Utiliza a biblioteca *pyqt6* para escrever na caixa de texto as informações relevantes.

A Figura 18 mostra uma simulação da funcionalidade desta aba, onde foram informados conjuntos de dados de teste, foi definido um modelo de referência através da

opção definida pelo usuário, foi selecionado um controlador do tipo PI e foi informada uma função de transferência para o filtro L(z). Na caixa de texto podem ser vistas as mensagens referentes ao número de amostras identificadas de cada ensaio informado à aplicação, bem como os parâmetros calculados para o controlador PI selecionado.

Figura 18 – Funcionamento da segunda aba da aplicação

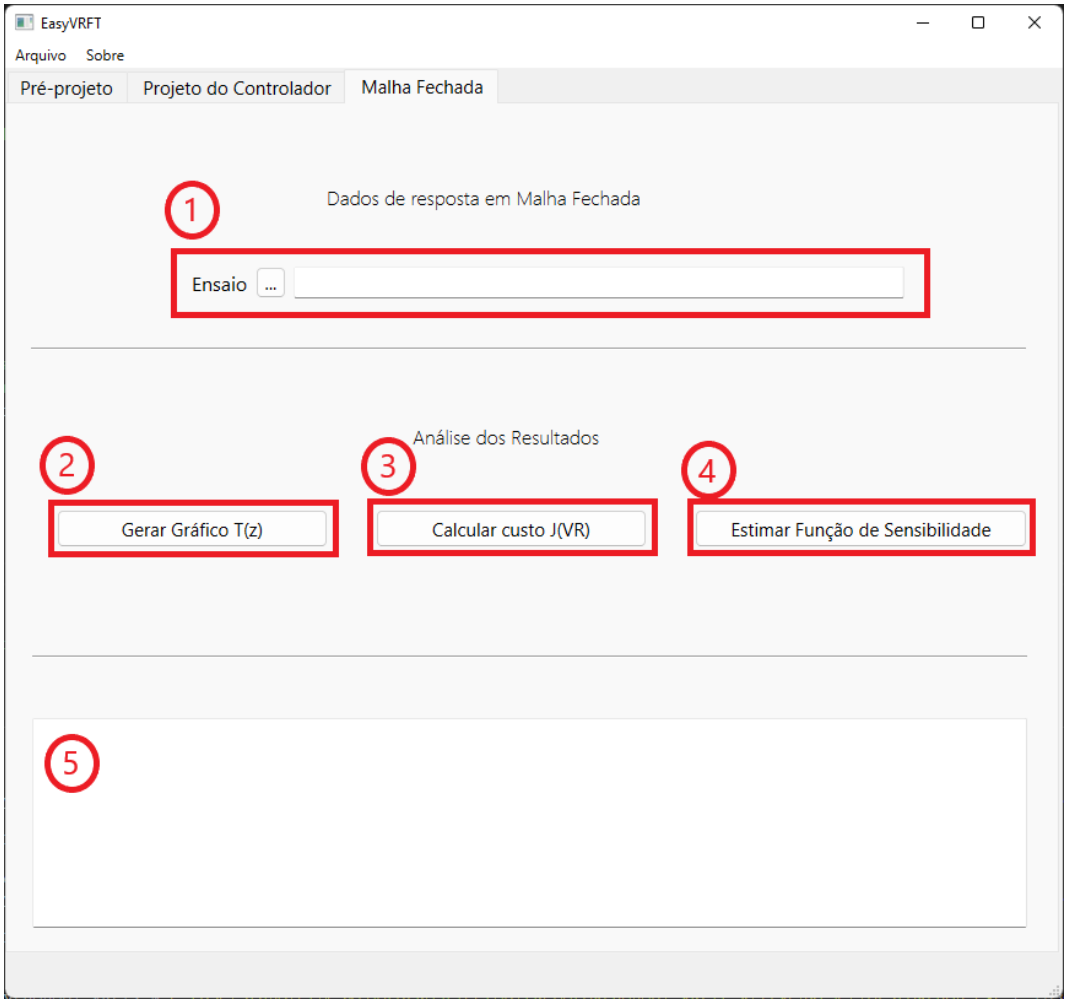


Fonte: o Autor, 2022.

3.3.3 Terceira aba - Malha Fechada

Na terceira aba é onde foram concentradas as funcionalidades de avaliação de desempenho transitório e em regime permanente do controlador projetado na etapa anterior. Para cumprir este objetivo, algumas opções foram dispostas para o usuário, sendo elas a comparação de funções de transferência, a estimativa da minimização de J^{MR} e a estimativa da função de sensibilidade do sistema em malha fechada. A Figura 19 traz a tela vista na terceira aba, novamente com indicações visuais, em vermelho, dos diferentes objetos configurados para formar a interface de análise de desempenho.

Figura 19 – Terceira aba com indicações visuais dos diferentes elementos da janela



Fonte: o Autor, 2022.

Abaixo segue a relação, conforme numeração, de cada elemento de interação com o usuário da interface com a sua função planejada.

3.3.3.1 Entrada do conjunto de dados da malha fechada com o controlador projetado

No bloco 1 da Figura 19, há uma etiqueta para informar a finalidade do campo para o usuário, um botão e um campo de texto de tipo somente leitura. Ao selecionar o botão com as reticências, uma nova janela é aberta onde pode ser selecionado o arquivo que contém os dados obtidos através de um ensaio que meça os dados de referência r(k) e da saída y(k) do sistema operando em malha fechada com o controlador calculado na aba anterior. No campo de texto será exibido o caminho em que o arquivo se encontra quando este for selecionado. O arquivo esperado segue o mesmo padrão de organização dos dados mostrado no objeto de entrada de dados da aba anterior e descrito na Seção 3.3.2.1. Novamente foi utilizada a biblioteca *Pandas* para ler os dados do arquivo e para fazer as manipulações necessárias para que estes fiquem em um formato utilizável pela aplicação

e a biblioteca *pyqt6* para integrar o funcionamento da janela de seleção do endereço do arquivo com o código em Python.

3.3.3.2 Geração de gráficos para análise

O botão indicado como 2 na Figura 19 gera os gráficos para comparação visual do comportamento dos dados informados pelo usuário. Este botão opera de duas formas distintas. A primeira forma corresponde a quando o usuário não executou a etapa de projeto do controlador, mas adicionou o conjunto de dados referente ao ensaio em malha fechada. Assim, quando este botão for selecionado, a aplicação irá abrir uma nova janela onde será exibido o gráfico da referência e da saída deste conjunto de dados. A segunda forma corresponde a quando o usuário executou a etapa de projeto do controlador e adicionou o conjunto de dados referente ao ensaio em malha fechada do sistema com controlador. Nesta situação, quando este botão for selecionado, a aplicação irá abrir uma nova janela onde serão exibidos dois gráficos. O primeiro gráfico corresponde ao sinal de referência informado no conjunto de dados adicionado à esta aba e a saída da função de transferência desejada $T_d(z)$ para esta referência. O segundo gráfico corresponde aos sinais de referência e saída informados através do conjunto de dados fornecido nesta aba. Foram utilizadas as bibliotecas *Pandas* e *Numpy* para manipular os dados obtidos dos conjuntos de dados, a biblioteca *Matplotlib* para exibir os gráficos gerados, a biblioteca *Scipy* para criar uma função de transferência referente a $T_d(z)$ e a biblioteca *pyvrft* para calcular a resposta dessa $T_d(z)$ para o sinal de referência contido no conjunto de dados informado na etapa de projeto do controlador;

3.3.3.3 Cálculo da função custo do método VRFT

O botão indicado por 3 na Figura 19 fará uma avaliação da função custo J^{MR} apresentada em (13). Para tal, é necessário que o usuário tenha executado a etapa de projeto do controlador e tenha adicionado o conjunto de dados referente ao ensaio em malha fechada. A aplicação utiliza o sinal de referência dos dados da malha fechada para simular a saída da função de transferência $T_d(z)$, definida pelo usuário na etapa anterior, a esta referência. Com os dados de saída do sistema em malha fechada e os dados de saída da $T_d(z)$ para o mesmo sinal de entrada, pode ser calculado o erro médio quadrático entre os dois conjuntos para se obter uma estimativa da função custo J^{MR}. A equação abaixo mostra como é feito o cálculo do erro médio quadrático entre dois conjuntos de valores com N amostras cada.

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(y_i - \bar{y_i})^2 \tag{26}$$

onde MSE corresponde ao erro médio quadrático(do inglês, *Mean Squared Error*), y_i corresponde ao valor de saída obtido do conjunto de dados adicionado nesta aba e $\bar{y}_i$ corresponde ao valor da saída gerada pela $T_d(z)$. Este cálculo resulta num valor numérico para MSE correspondente ao erro entre os dois sinais, onde conforme menor for este valor, mais a resposta do sistema em malha fechada com o controlador projetado se aproxima da resposta desejada. O valor obtido é então escrito na caixa de texto enumerada como 5 desta aba para visualização do usuário. Novamente, são utilizadas as bibliotecas *Pandas* e *Numpy* para manipular os dados obtidos dos conjuntos de dados, a biblioteca *Scipy* para criar uma função de transferência referente a $T_d(z)$ e a biblioteca *pyvrft* para calcular a resposta dessa $T_d(z)$ para o sinal de referência do conjunto de dados do sistema em malha fechada adicionado nesta aba.

3.3.3.4 Estimativa da função de sensibilidade

O botão indicado como 4 na Figura 19 utiliza os dados informados na etapa de projeto do controlador e os parâmetros calculados a partir da aplicação do método VRFT para estimar a função de sensibilidade do sistema completo. A função de sensibilidade pode ser calculada utilizando (11), no entanto, $G(z)$ não é conhecido. Para contornar este problema, em (REMES, 2021) é feita a suposição que $T(z, \rho) = T_d(z)$, uma vez que, no caso ideal, a diferença entre as duas funções de transferência é nula. Como geralmente não é possível alcançar este caso ideal, é possível calcular uma estimativa de $G(z)$, aqui denotada por $G(z, \rho)$ como:

$$G(z, \rho) = \frac{T_d(z, \rho)}{C(z, \rho) - T_d(z, \rho)C(z, \rho)} \tag{27}$$

A partir de $G(z, \rho)$ é possível então chegar na estimativa da função de sensibilidade seguindo a equação abaixo.

$$S(z, \rho) = \frac{1}{C(z, \rho) + (G(z, \rho)C(z, \rho))} \tag{28}$$

A função de sensibilidade é, após calculada, mostrada na caixa de texto da terceira aba. Foi utilizada a biblioteca *Control* para criar as funções de transferência, manipulá-las para chegar na função de sensibilidade e para obter esta função no formato de zeros, polos e ganho, a biblioteca *Numpy* para ter acesso a parâmetros dos dados utilizados para gerar a saída em formato de texto e a biblioteca *pyqt6* para escrever na caixa de texto da aba a função de sensibilidade estimada.

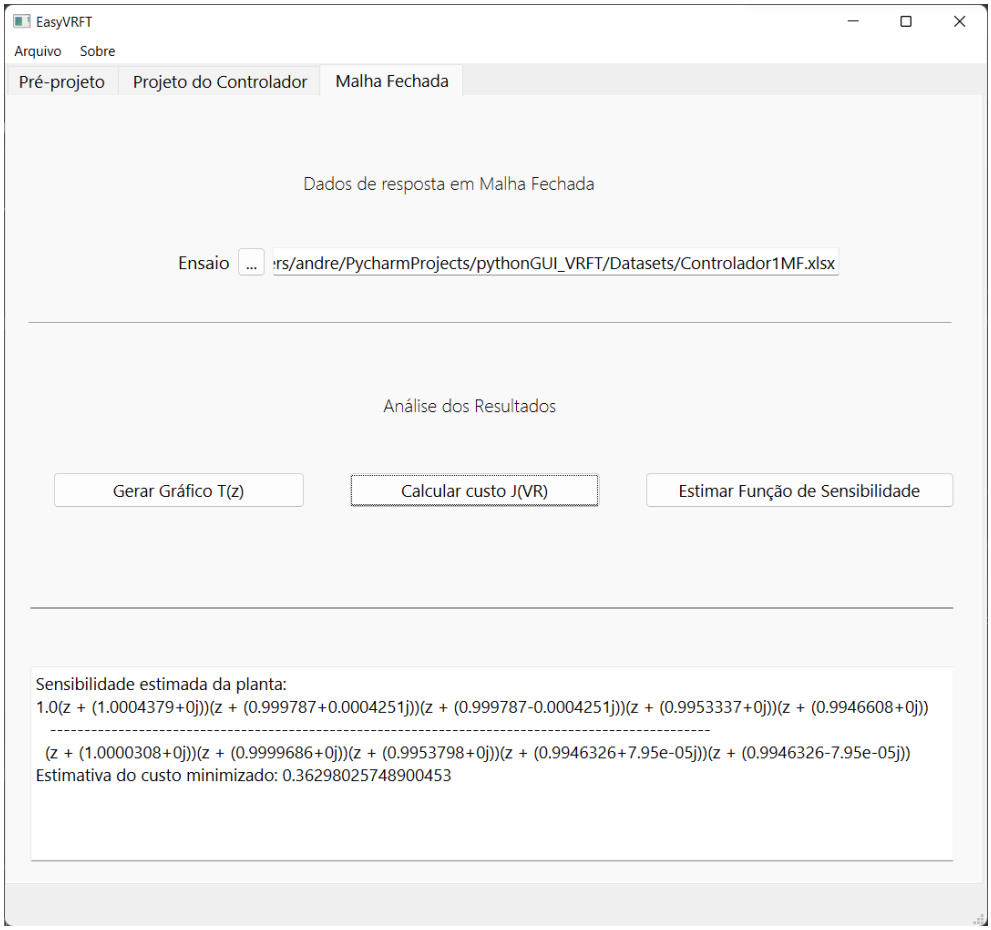
3.3.3.5 Caixa de texto de saída

Por fim, nesta caixa de texto é onde é exibido o valor calculado da função custo e a função de sensibilidade estimada. Também mostra a quantidade de amostras identificadas

quando o usuário adiciona um conjunto de dados de um ensaio em malha fechada da planta. A caixa de texto também foi configurada como sendo do tipo somente leitura. Utiliza a biblioteca *pyqt6* para escrever na caixa de texto os valores obtidos.

A Figura 20 mostra uma simulação da funcionalidade desta aba, onde foi informado um conjunto de dados de um ensaio em malha fechada. A partir do projeto de um controlador feito na aba anterior, pode ser visto na caixa de teste de saída a função de sensibilidade e o custo J^{VR} estimados para o sistema em malha fechada.

Figura 20 – Funcionamento da terceira aba da aplicação



Fonte: o Autor, 2022.

3.4 Considerações Finais

Neste capítulo, foram abordados os procedimentos adotados para o desenvolvimento da janela gráfica, assim como as suas funcionalidades. Cada elemento foi adicionado à GUI pelo Qt Designer, sendo que as suas configurações mais básicas são geradas automaticamente quando a janela é importada para o Python, mas a implementação da relação que cada objeto tem uns com os outros e também com o usuário da aplicação foi desenvolvida manualmente por comandos disponibilizados pela biblioteca *pyqt6*. Além

disso, algumas das principais implementações desenvolvidas durante a execução deste trabalho foram adicionadas ao Apêndice A.

4 Aplicação da Solução Proposta

Nesta seção será ilustrada a utilização da ferramenta em dois cenários distintos: planta totalmente teórica conforme (ECKHARD; BOEIRA, 2019), para avaliar se o método VRFT está sendo aplicado corretamente; conversor simulado DSRAC(*Dual Series-Resonant Active-Clamp*) apresentado em (SALATI, 2021). Apesar de não fazer parte das topologias clássicas discutidas anteriormente, esse conversor foi escolhido por já ter uma implementação em PSIM operando em malha fechada pronta para a coleta de dados.

Os códigos gerados para a implementação da ferramenta, junto de um exemplo de uso, e do arquivo executável, podem ser encontrados no seguinte repositório: <https://github.com/AndreCW/pythonGUI_VRFT.git>.

4.1 Resultados com o Exemplo da Biblioteca *pyvrft*

Este exemplo considerado define uma função de transferência para uma planta fictícia

$$G(z) = \frac{1}{z - 0.9} \tag{29}$$

e um sinal de entrada para coleta de dados do tipo degrau unitário aplicado em $k = 0$ com 100 amostras. Utilizando uma função da biblioteca *pyvrft* é possível obter o sinal de saída da planta para esse sinal de entrada. Esse conjunto de dados de entrada e saída, apresentado nas Figuras 21 e 22, será utilizado para o projeto do controlador. Além disso, foi adicionada uma parcela a cada amostra do sinal de saída que simula um ruído pseudo-aleatório para tentar aproximar o comportamento desta planta ao de uma planta real. Após isso, foram definidas as funções de transferência para a $T_d(z)$ e para o filtro $L(z)$, como segue

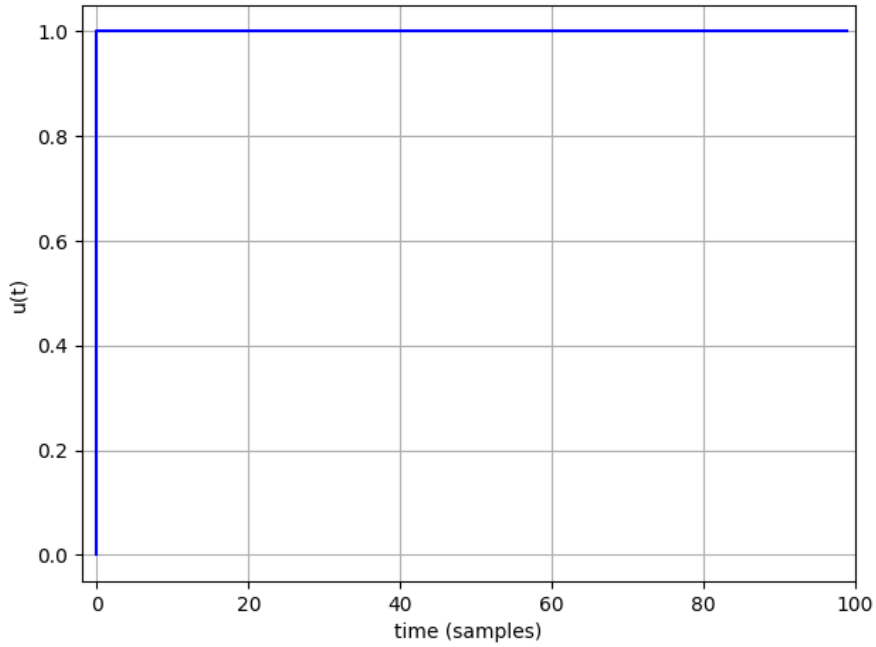
$$T_d(z) = \frac{0.2}{z - 0.8} \tag{30}$$

$$L(z) = \frac{0.25}{z - 0.75}. \tag{31}$$

Por fim, falta apenas definir a estrutura do controlador que será utilizado. Foi escolhido utilizar um controlador do tipo PI, descrito como:

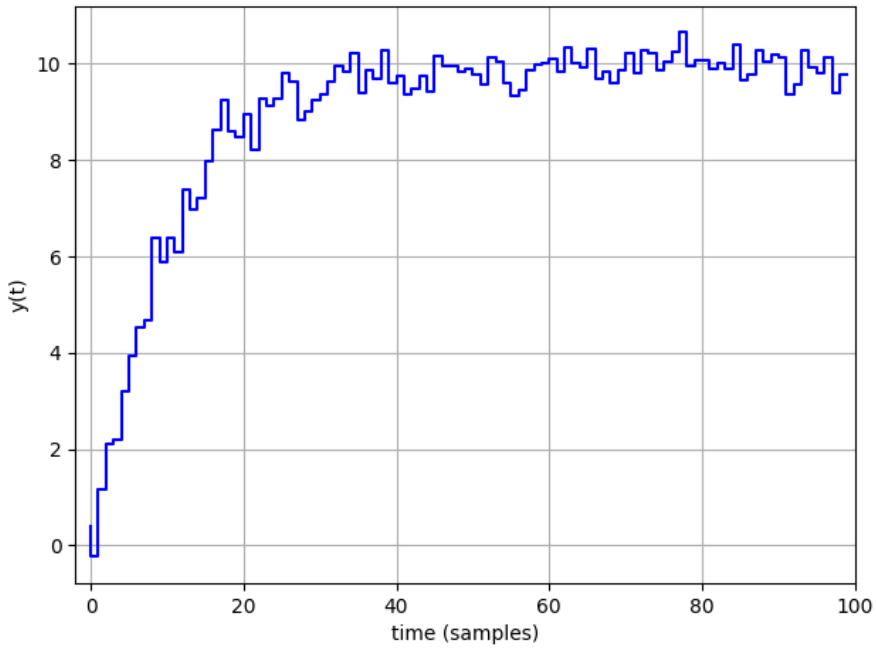
$$C(z) = K_p + K_i \frac{1}{z - 1} \tag{32}$$

Figura 21 – Sinal de entrada da G(z)



Fonte: o Autor, 2022.

Figura 22 – Sinal de saída da G(z)



Fonte: o Autor, 2022.

O primeiro passo foi adicionar ao código do exemplo uma parte que exportasse o conjunto de dados gerado pelo exemplo para um arquivo que pudesse ser passado para a aplicação desenvolvida. Após isso, foi verificado se os ganhos obtidos com a interface proposta correspondiam aos ganhos calculados através da execução direta do exemplo

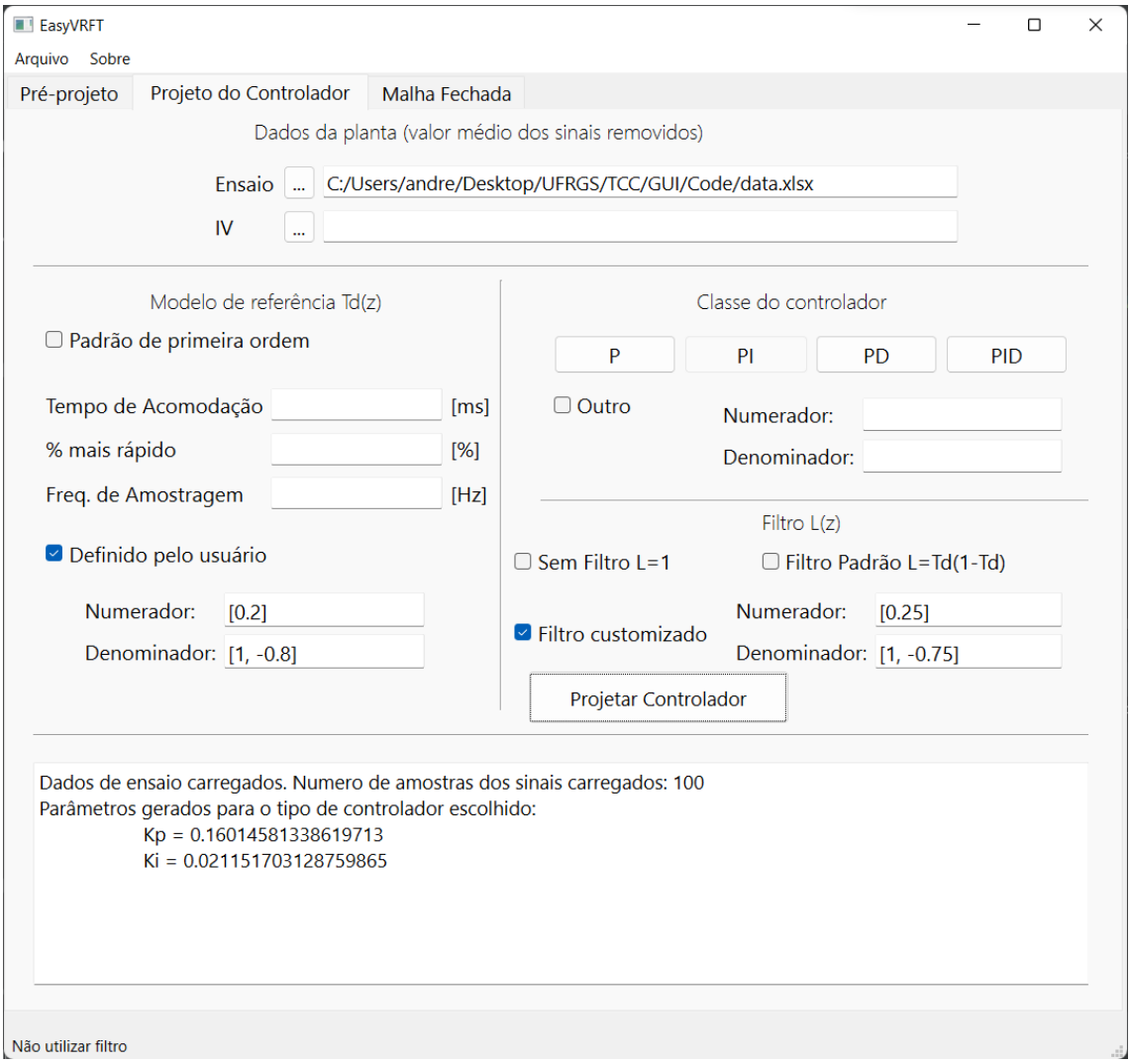
da biblioteca *pyvrft*. O código completo utilizado para este exemplo pode ser encontrado no Apêndice A.4. Com a execução do exemplo, os ganhos do controlador gerados são apresentado na Tabela 2 e as modificações feitas ao exemplo se encontram nas linhas 46 a 57 do código.

Tabela 2 – Parâmetros do controlador do exemplo da biblioteca *pyvrft*

$\mathbf{K_p}$	0.16014581
$\mathbf{K_i}$	0.0211517

Para fins de comparação, foi, então, adicionado à ferramenta desenvolvida o conjunto de dados entrada-saída e os outros parâmetros relevantes para realizar o projeto do controlador. A Figura 23 mostra a aba correspondente a etapa de projeto do controlador com todos os parâmetros inseridos, bem como, a exibição dos resultados gerados pela aplicação.

Figura 23 – Interface gráfica preenchida com os dados do exemplo da biblioteca



Fonte: o Autor, 2022.

Como pode ser observado, o resultado exibido na interface corresponde exatamente aos ganhos obtidos para o controlador com o exemplo da biblioteca. Este resultado permite assumir que a implementação do método VRFT está de acordo com o funcionamento da biblioteca *pyvrft*.

4.2 Resultados com Conversor DSRAC

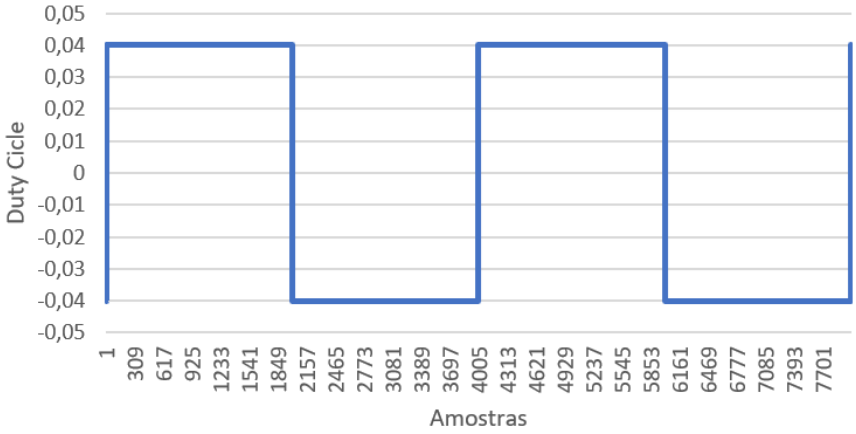
Como forma de validar o funcionamento da ferramenta desenvolvida em uma planta que melhor reflita os objetivos do trabalho, foi utilizado um conversor de potência DSRAC, estudado em (SALATI, 2021) e implementado em PSIM.

Para verificar o funcionamento da aplicação foram gerados dois conjuntos de dados deste conversor, um com a malha aberta e outro com a malha fechada. Para fechar a malha, foi utilizado um controlador proporcional cujo valor foi definido como:

$$K_p = \frac{0.2}{G_{d0}} < \frac{1}{G_{d0}}, \tag{33}$$

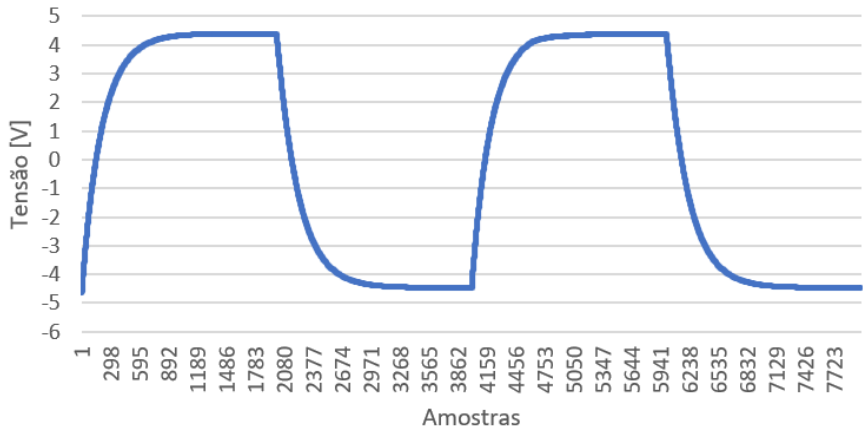
respeitando assim, o critério de ganho estático máximo para utilizar o ponto mais robusto para coleta de dados da planta, discutido anteriormente. As Figura 24 e 25 trazem, respectivamente os sinais de entrada e saída do conjunto de dados coletados considerando a operação em malha aberta do conversor.

Figura 24 – Sinal de entrada de malha aberta do conversor DSRAC



Fonte: o Autor, 2022.

Figura 25 – Sinal de saída de malha aberta do conversor DSRAC



Fonte: o Autor, 2022.

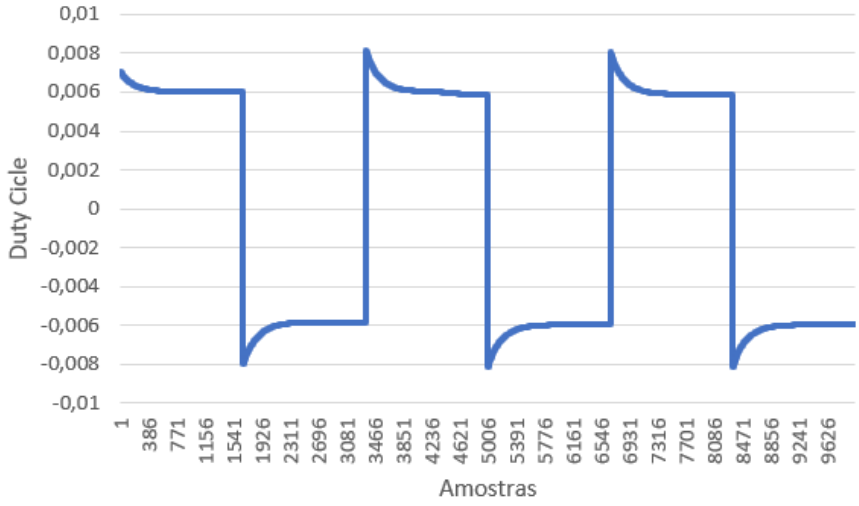
Pode ser observado que o sinal aplicado na entrada é uma onda quadrada, onde o conjunto de dados foi pré-processado para remover o nível médio do sinal. Isso pode ser observado também no conjunto de dados da malha fechada, a seguir, em que o nível médio dos sinais também foi removido.

A partir de uma análise gráfica do sinal de saída, foi estimado um número aproximado de 930 amostras partindo da borda de subida da onda quadrada do sinal de entrada até que o sinal de saída se acomodasse em um valor de regime permanente. Estas 930 amostras, à 50kHz implicam em um tempo de acomodação de 18.6ms. Esta frequência de amostragem foi definida em função da frequência utilizada tanto em (SALATI, 2021) quanto na simulação pelo PSIM.

$$t_{so} = \frac{n\ amostras}{frequencia} = \frac{930}{50000} = 18.6ms \tag{34}$$

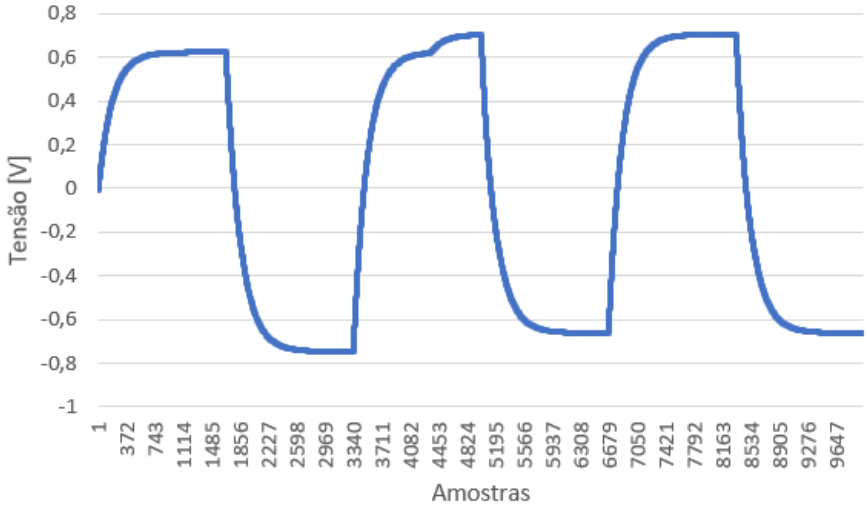
Para o conjunto de dados da malha fechada com o controlador proporcional, as Figuras 26 e 27 mostram os sinais de entrada e saída, respectivamente.

Figura 26 – Sinal de entrada de malha fechada do conversor DSRAC



Fonte: o Autor, 2022.

Figura 27 – Sinal de saída de malha fechada do conversor DSRAC



Fonte: o Autor, 2022.

Com os dois conjuntos de dados, foi feito o projeto de quatro controladores, sendo um PI e um PID para cada conjunto. Os parâmetros utilizados para o projeto destes controladores foram os seguintes:

- Tempo de acomodação em malha aberta: 18.6ms;
- Tempo de acomodação x% mais rápido em malha fechada: 20%
- Frequência de amostragem: 50kHz

O tempo de acomodação em malha fechada foi escolhido arbitrariamente dentro do intervalo [0, 100)% e o tempo de acomodação em malha aberta foi estimado em função de simulações em malha aberta no PSIM, conforme mostrado anteriormente.

Com os dados para o cálculo da $T_d(z)$ desejada estabelecidos, é possível aplicar as relações (22) e (21), resultando em:

$$T_d(z) = \frac{1 - p_1}{z - p_1} = \frac{0,004517214}{z - 0,995482786} \tag{35}$$

Quanto ao filtro $L(z)$, foi escolhido o filtro padrão, definido conforme (**??**). A função de transferência para o filtro utilizado é dado por:

$$L(z) = T_d(z)(1 - T_d(z)) = \frac{0.004517z + 0.004476}{z^2 + 1.991z + 0.991} \tag{36}$$

Foram projetados quatro controladores, definidos da seguinte forma:

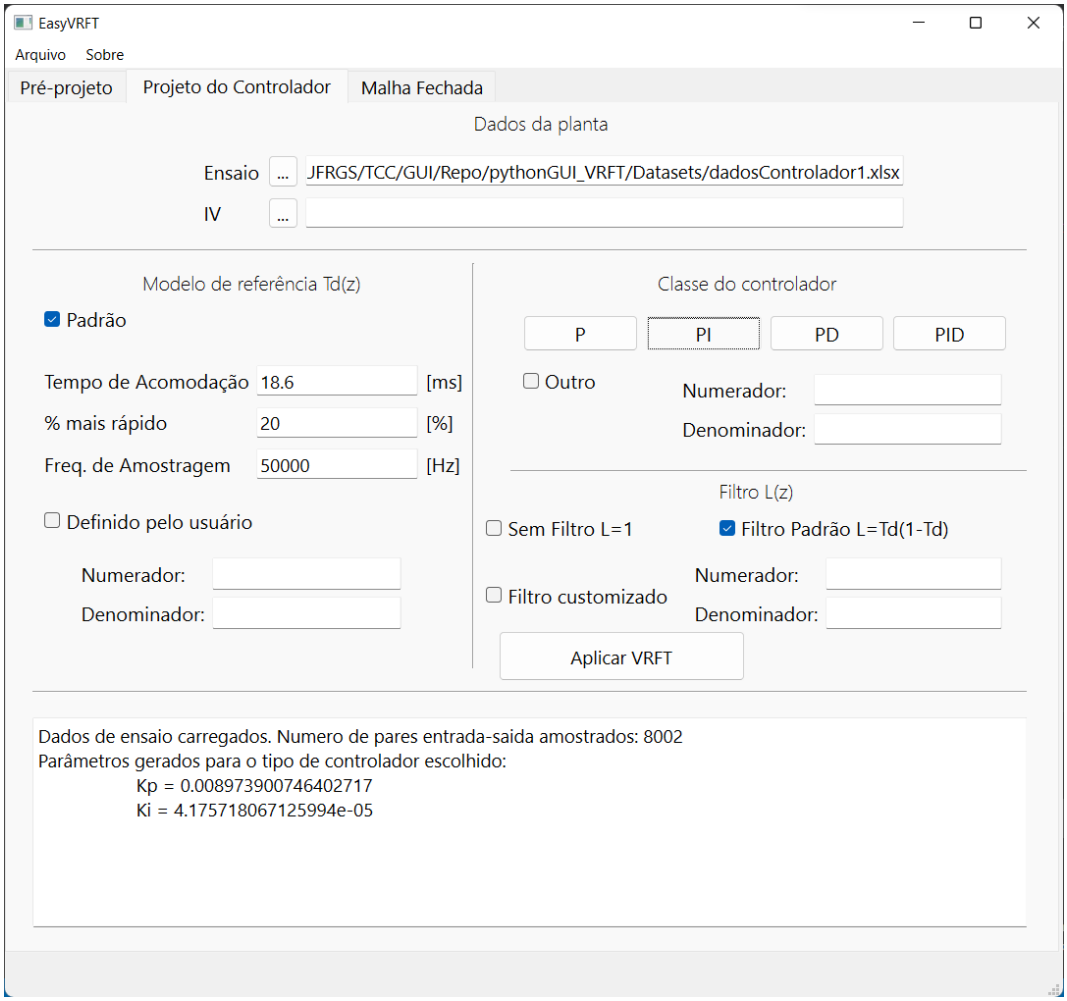
- Controlador 1: Controlador do tipo PI projetado com os dados em malha aberta;
- Controlador 2: Controlador do tipo PI projetado com os dados em malha fechada;
- Controlador 3: Controlador do tipo PID projetado com os dados em malha aberta;
- Controlador 4: Controlador do tipo PID projetado com os dados em malha fechada.

4.2.1 Resultados com Controlador 1

Nesta seção será exposto e analisado os resultados obtidos para o primeiro controlador projetado. Este controlador foi projetado conforme os parâmetros estabelecidos na Seção 4.2, utilizando os dados do ensaio em malha aberta fornecidos da planta, onde se deseja parametrizar um controlador do tipo PI.

A Figura 28 traz os parâmetros informados à aplicação, assim como os parâmetros gerados para o controlador selecionado.

Figura 28 – Interface preenchida com os dados para projetar o primeiro controlador



Fonte: o Autor, 2022.

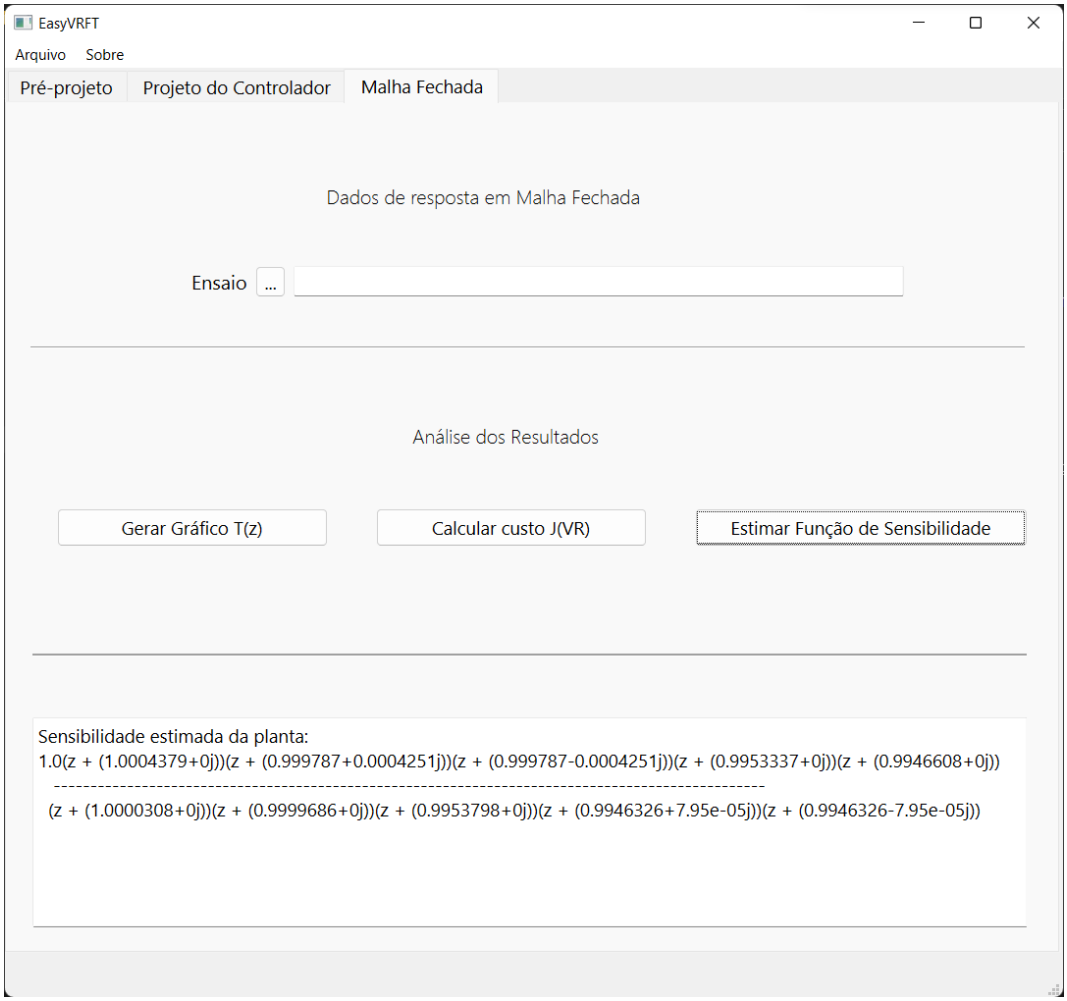
Como pode ser observado na Figura 28, os parâmetros para gerar o controlador foram informados conforme especificado. Para o controlador projetado os valores de ganho obtidos estão na Tabela 3.

Tabela 3 – Parâmetros do primeiro controlador

$\mathbf{K_p}$	0.00897390074
$\mathbf{K_i}$	4.17571806e-05

Dentre as funcionalidades implementas a partir do projeto do controlador, já pode ser feita a análise da função de sensibilidade estimada para o sistema projetado, conforme (28). A Figura 29 mostra a função de sensibilidade calculada pela ferramenta, no formato de zeros, polos e ganho, respectivamente.

Figura 29 – Função de sensibilidade estimada para o primeiro controlador

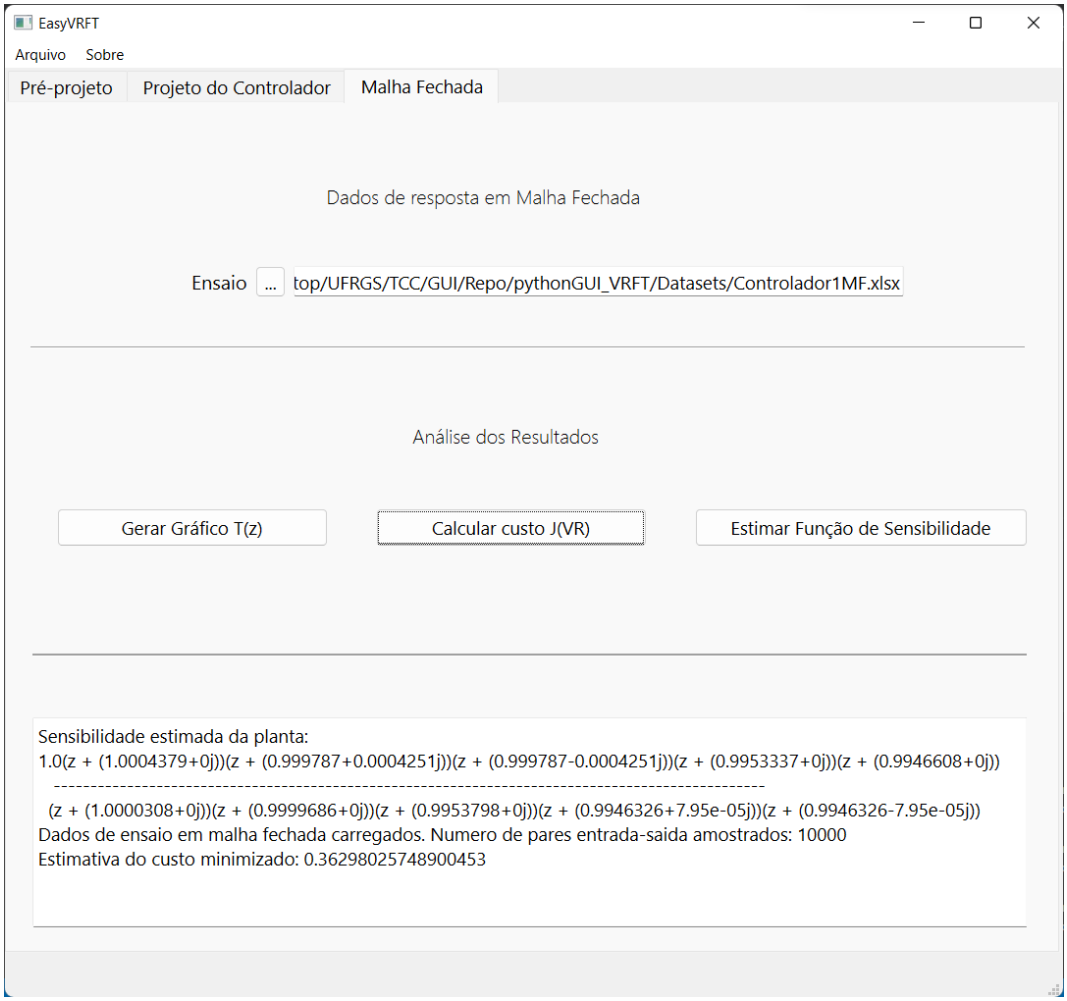


Fonte: o Autor, 2022.

O controlador PI projetado foi então simulado em PSIM, gerando um conjunto de dados de validação. Tendo o ensaio de malha fechada com o controlador projetado, este pode ser passado à ferramenta para que as últimas funcionalidades de avaliação do sistema projetado possam ser utilizadas. Informando mais este conjunto de dados, na terceira aba da GUI, pode ser avaliado o comportamento gráfico do sinal obtido da malha fechada com o controlador frente ao gráfico do sinal de saída da $T_d(z)$ obtido quando esta é excitada pelo sinal de referência deste ensaio em malha fechada. Também pode ser avaliado o valor da função custo feita pela implementação do método VRFT. As Figuras 30 e 31 mostram os resultados das duas funcionalidades mencionadas.

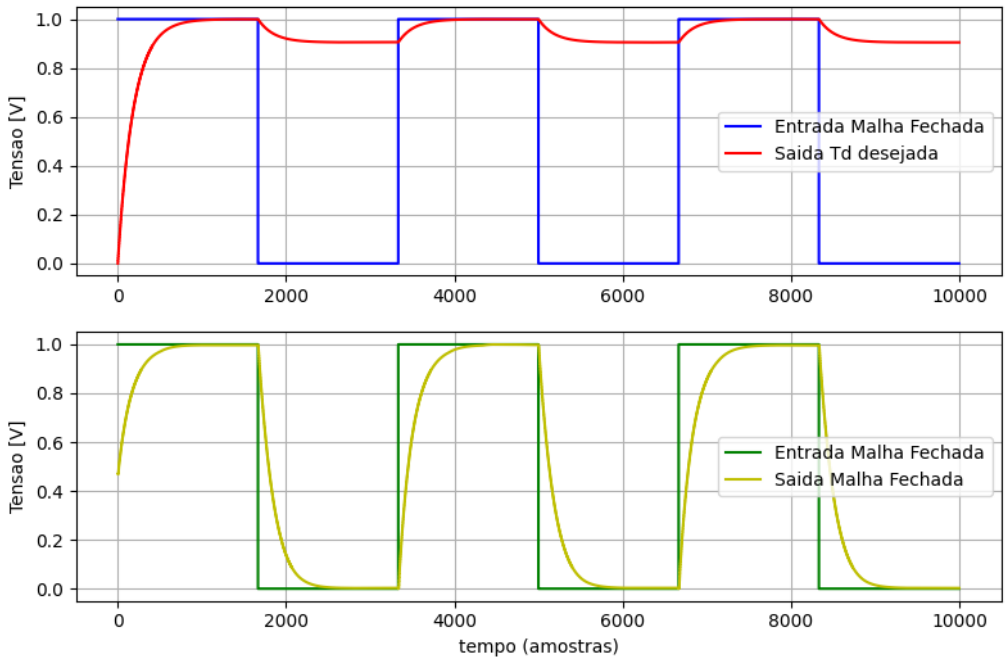
Conforme pode ser visto na imagem, o erro calculado entre os resultados teórico e prático foi de 0.36298. Este valor corresponde a um número suficientemente baixo para poder afirmar que os resultados obtidos estão consideravelmente próximos.

Figura 30 – Custo J^{VR} calculado para o sistema em malha fechada com o primeiro controlador



Fonte: o Autor, 2022.

Figura 31 – Gráfico das saídas da $T_d(z)$ e do sistema em malha fechada com o controlador projetado



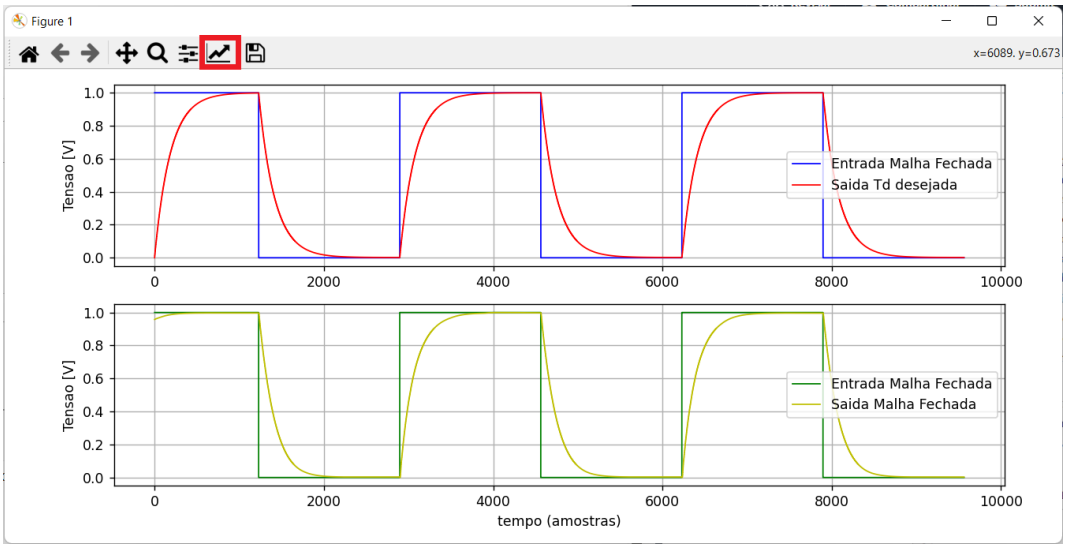
Fonte: o Autor, 2022.

Aqui é ilustrado um problema na comparação gráfica entre a simulação e os dados de validação. Pode ser observado que o sinal de saída da $T_d(z)$ está com uma escala muito diferente dos outros sinais gerados, de forma que, não seja possível fazer alguma comparação entre os dois sinais de saída. Isto ocorre pois os sinais de referência e saída em malha fechada do sistema foram adquiridos quando o sistema já estava em regime permanente, o que difere da simulação de $T_d(z)$, que precisa de um tempo para atingir o seu regime permanente. A solução adotada para este problema foi desconsiderar a parte transitória da simulação de $T_d(z)$ tal que, todos os valores ficam dentro dos limites do regime permanente do sinal, possibilitando a comparação gráfica entre as respostas teóricas e práticas.

Após ser constatado este problema, foi modificada a função da GUI que exibe os gráficos, fazendo com que o número de amostras considerados para gerar eles, após calcular a estimativa da resposta da $T_d(z)$, seja reduzido. Essa quantidade de amostras são removidas do início do conjunto de dados que é exibido no gráfico e foi definida a partir do índice do primeiro valor da saída simulada que assume um valor maior ou igual ao menor valor da saída armazenada nos dados de validação.

Essa modificação nos dados utilizados para calcular a normalização dos conjuntos faz com que a parcela transitória da saída simulada não interfira na escala entre os sinais, permitindo, assim, que uma comparação gráfica possa ser feita entre os sinais de saída. A Figura 32 mostra o gráfico gerado a partir dessa modificação na ferramenta desenvolvida.

Figura 32 – Gráfico das saídas de $T_d(z)$ e do sistema em malha fechada com o controlador projetado



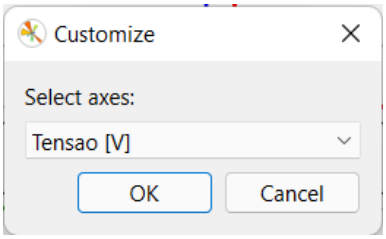
Fonte: o Autor, 2022.

Complementando a análise feita para a minimização da função custo, visualmente os dois sinais apresentam comportamentos similares, levando a crer que o erro calculado de fato reflete numa diferença pequena entre os resultados teórico e prático. A última

análise que pode ser feita é validar se o controlador projetado atingiu o objetivo proposto de resultar em um sistema em malha fechada que tenha um tempo de acomodação 20% mais rápido que o sistema em malha aberta. Para isso, foi utilizada uma funcionalidade da biblioteca *Matplotlib* que permite modificar os valores dos eixos do gráfico para deixar mais visível uma faixa específica de valores sendo que, neste caso, deseja-se redimensionar o eixo das abscissas. Foi escolhido aproximar a visualização do gráfico ao redor da primeira borda de subida visível do sinal de entrada, assim, os limites de exibição para esta análise foram definidos com 2800 amostras no limite inferior e 3900 amostras no limite superior.

Na Figura 32 tem uma indicação visual em vermelho de um botão na barra superior da janela e este botão traz a possibilidade de fazer esta modificação nos eixos do gráfico. Selecionar o botão indicado abre outra janela que permite escolher qual dos dois gráficos, neste caso, se deseja modificar, mostrada na Figura 33.

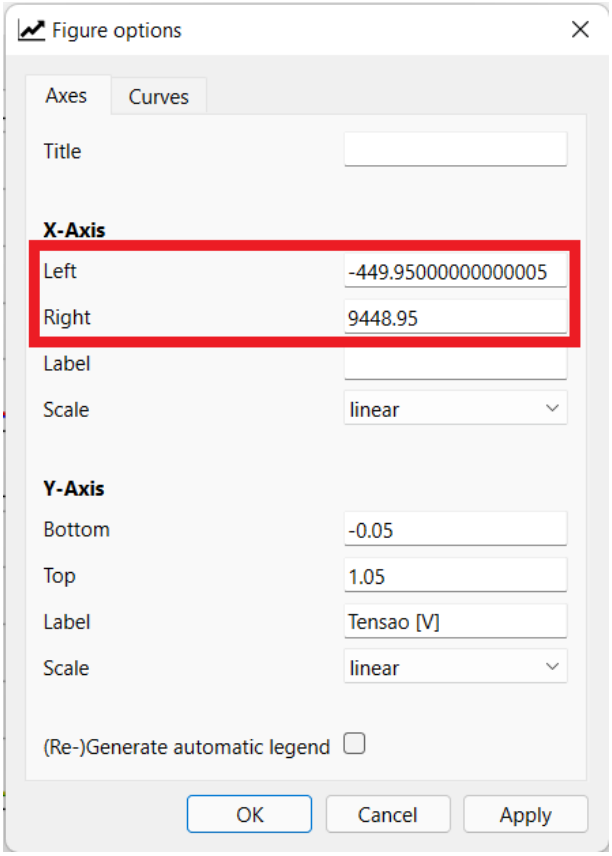
Figura 33 – Janela para selecionar o gráfico que será modificado



Fonte: o Autor, 2022.

Nesta janela, a seta para baixo permite a seleção entre os dois gráficos que foram gerados, sendo que o nome que aparece corresponde aos rótulos dados ao eixo x e eixo y de cada gráfico. Neste exemplo, não foi dado um rótulo ao eixo x do primeiro gráfico, mas foi dado ao eixo y, enquanto que o segundo gráfico possui rótulo em ambos os eixos. Clicar no botão 'OK' abre a janela que permite, de fato, editar os eixos de exibição do gráfico, mostrada na Figura 34.

Figura 34 – Janela para editar os eixos dos gráficos



Fonte: o Autor, 2022.

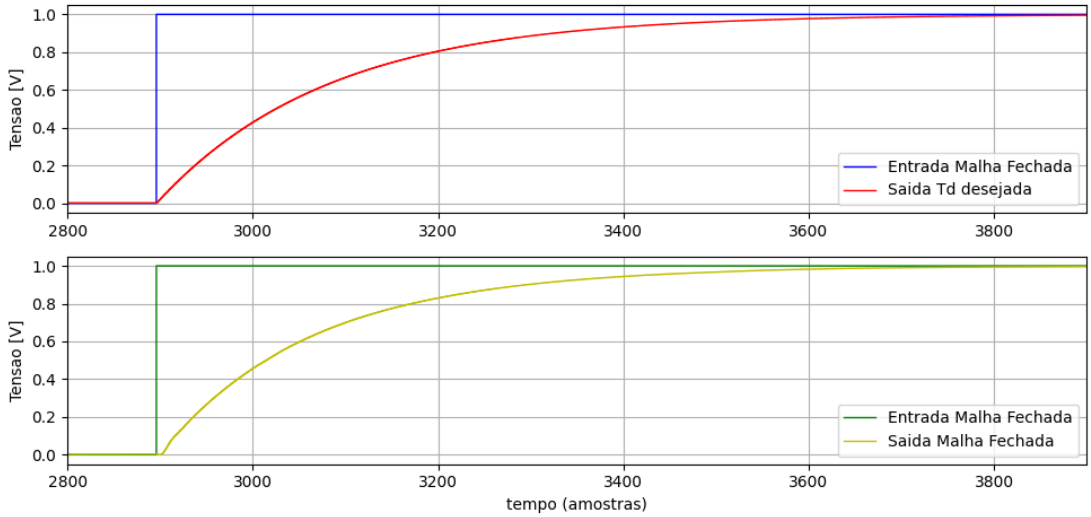
O retângulo vermelho indica onde pode ser alterado os limites de exibição do eixo x. Para a análise do caso do primeiro controlador, como procura-se modificar estes limites de forma a se exibir somente as amostras necessárias para identificar o tempo de acomodação dos sinais de saída, nestes campos foram inseridos os valores mencionados para os limites inferior e superior. A Figura 35 mostra como ficou o gráfico após reduzir os limites de exibição para os dois gráficos.

Para calcular o tempo de acomodação em malha fechada, foi considerado que o degrau da onda quadrada ocorre aproximadamente na amostra 2900 e que os sinais de saída chegaram ao regime permanente aproximadamente na amostra 3650. O tempo de acomodação pode ser encontrado da mesma forma feita para estimar o tempo de acomodação em malha aberta, em (34). Segue na Equação (37) o tempo de acomodação aproximado do sistema com a malha fechada com o primeiro controlador.

$$t_{sc} = \frac{n\ amostras}{frequencia} = \frac{3650 - 2900}{50000} = 15ms \tag{37}$$

$$t_{sc} = t_{so}(1 - x\%) = 18,6(1 - 0.01 * 20) = 14.88ms \tag{38}$$

Figura 35 – Gráfico aproximado para os sinais do sistema em malha fechada com o primeiro controlador



Fonte: o Autor, 2022.

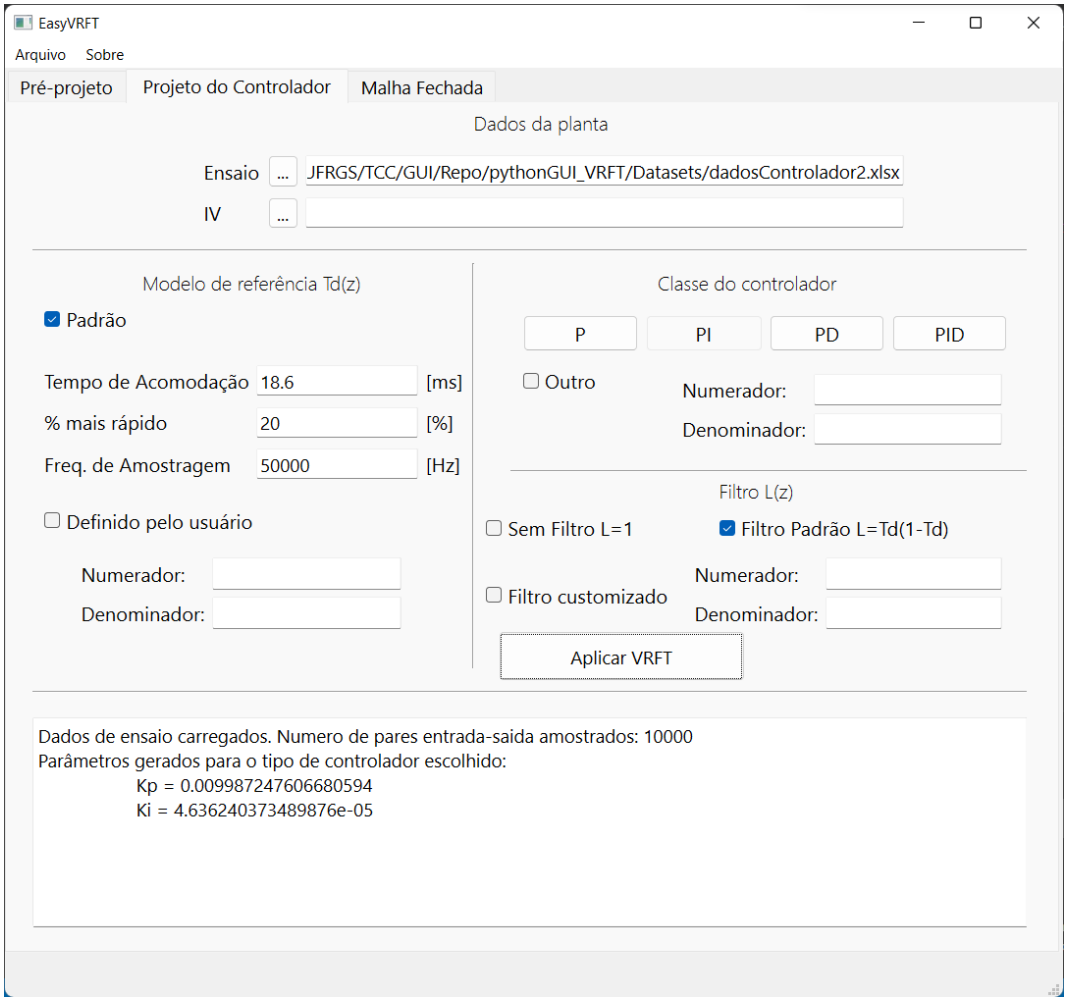
onde t_{sc} corresponde ao tempo de acomodação em malha fechada. A estimativa feita implica num tempo de acomodação da malha fechada de aproximadamente 15ms. Considerando que há um erro intrínseco nesta medição, em função das seleções feitas para a amostra em que ocorre a subida da onda quadrada e para amostra em que é considerado que o sinal atinge o regime permanente não serem precisas, este resultado está suficientemente próximo do valor almejado para o tempo de acomodação da malha fechada definido como sendo 20% mais rápido que o tempo da malha aberta, que seria 14.88ms.

4.2.2 Resultados com Controlador 2

A seguir será exposto e analisado os resultados obtidos para o segundo controlador projetado. Este controlador foi projetado conforme os parâmetros estabelecidos na Seção 4.2, utilizando o conjunto de dados em malha fechada com um controlador proporcional fornecido da planta, onde se deseja parametrizar um controlador do tipo PI.

A Figura 36 traz os parâmetros informados à aplicação, assim como os parâmetros gerados para o controlador selecionado.

Figura 36 – Interface preenchida com os dados para projetar o segundo controlador



Fonte: o Autor, 2022.

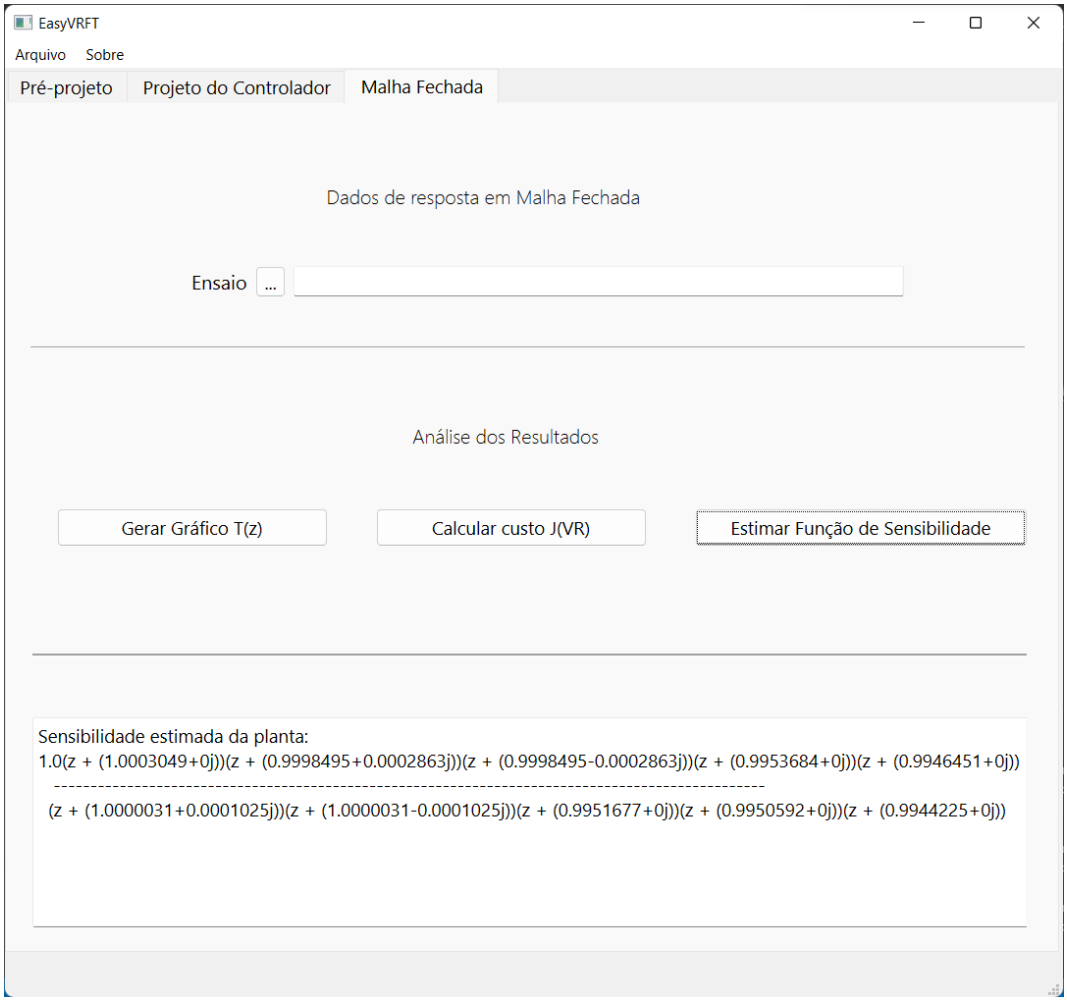
Como pode ser observado na imagem, os parâmetros para gerar o controlador foram informados conforme especificado. Para o controlador projetado os valores de ganho obtidos estão na Tabela 4.

Tabela 4 – Parâmetros do segundo controlador

$\mathbf{K_p}$	0.00998724760
$\mathbf{K_i}$	4.63624037e-05

Com estes resultados, pode ser feita a análise gráfica inicial e pode ser verificada a função de sensibilidade estimada para o sistema projetado, conforme (28). A Figura 37 mostra a função de sensibilidade calculada pela ferramenta, no formato de zeros, polos e ganho.

Figura 37 – Função de sensibilidade estimada para o segundo controlador

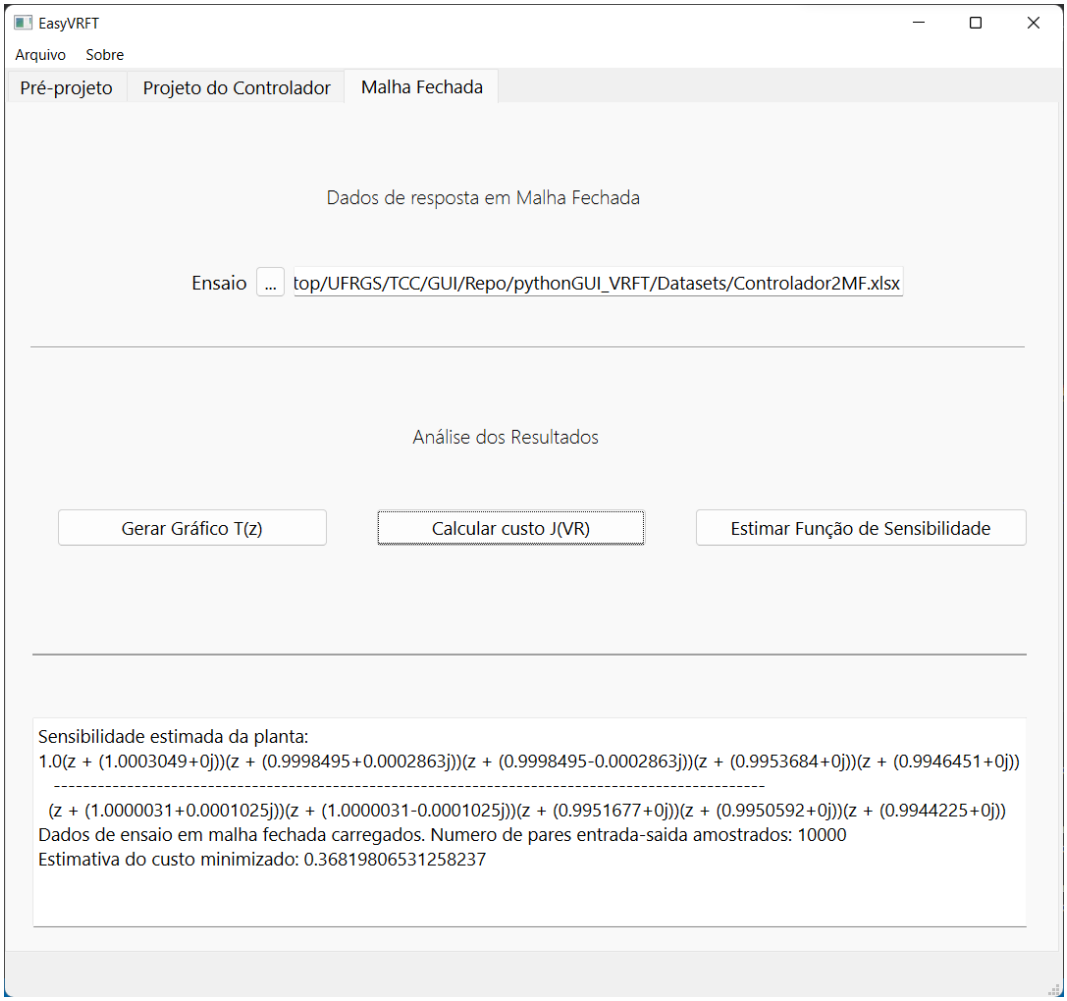


Fonte: o Autor, 2022.

Agora, com o controlador em mãos, este foi implementado na simulação em PSIM do conversor DSRAC para realizar o ensaio do sistema completo em malha fechada. Tendo o ensaio de malha fechada com o controlador projetado, este pode ser passado à ferramenta para que as últimas funcionalidades de avaliação do sistema projetado possam ser utilizadas. Informando mais este conjunto de dados, na terceira aba da GUI, pode ser avaliado o comportamento gráfico do sinal obtido da malha fechada com o controlador frente ao gráfico do sinal de saída da $T_d(z)$ obtido quando esta é excitada pelo sinal de entrada deste ensaio em malha fechada. Também pode ser avaliada a minimização da função custo feita pela implementação do método VRFT. As Figuras 38 e 39 mostram os resultados das duas funcionalidades mencionadas.

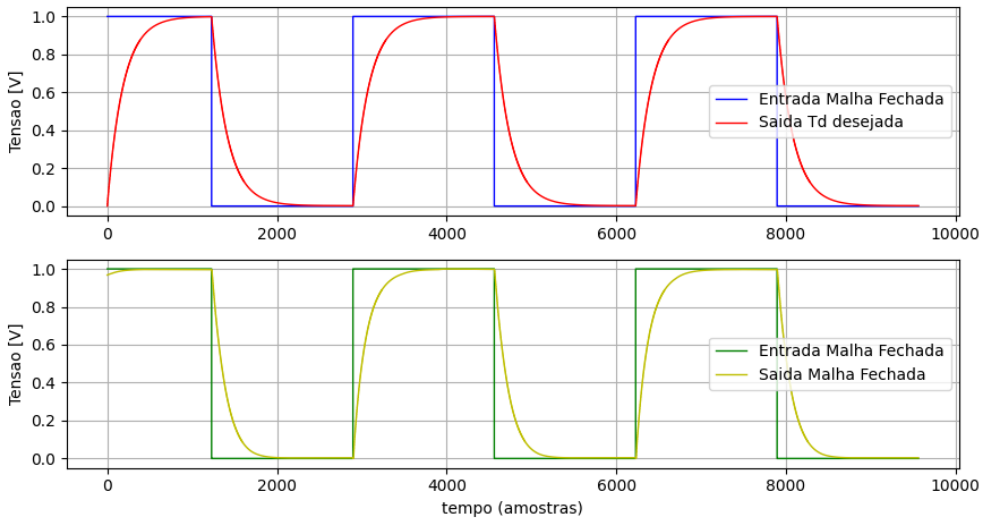
Conforme pode ser visto na imagem, o erro calculado entre os resultados teórico e prático foi de 0.36819. Este valor corresponde a um número suficientemente baixo para poder afirmar que os resultados obtidos estão consideravelmente próximos. Esta análise é corroborada pela comparação gráfica dos dois sinais, vista na Figura 39.

Figura 38 – Custo J^{VR} calculado para o sistema em malha fechada com o segundo controlador



Fonte: o Autor, 2022.

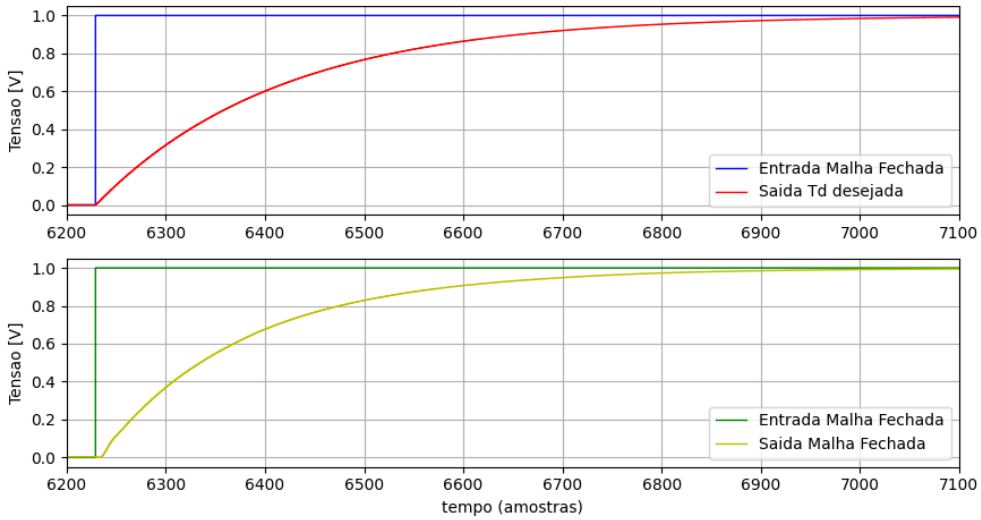
Figura 39 – Gráfico das saídas de $T_d(z)$ e do sistema em malha fechada com o segundo controlador projetado



Fonte: o Autor, 2022.

A última análise que pode ser feita para validar se o controlador projetado atingiu o objetivo proposto de compor um sistema que tenha um tempo de acomodação 20% mais rápido que o sistema em malha aberta é novamente estimar esse tempo de acomodação a partir dos dados fornecidos. Para isso, foi utilizada a funcionalidade de restringir os limites do eixo do gráfico, escolhendo a amostra 6200 como limite inferior e a amostra 7100 como limite superior. A Figura 40 mostra como ficou o gráfico após reduzir os limites de exibição.

Figura 40 – Gráfico aproximado para os sinais do sistema em malha fechada com o segundo controlador



Fonte: o Autor, 2022.

Para calcular o tempo de acomodação em malha fechada, foi considerado que o degrau da onda quadrada ocorre aproximadamente na amostra 6250 e que os sinais de saída chegaram no patamar de estabilização aproximadamente na amostra 7000. Aplicando o cálculo do tempo de acomodação, o resultado obtido foi novamente de 15ms:

$$t_{sc} = \frac{n\ amostras}{frequencia} = \frac{7000 - 6250}{50000} = 15ms \tag{39}$$

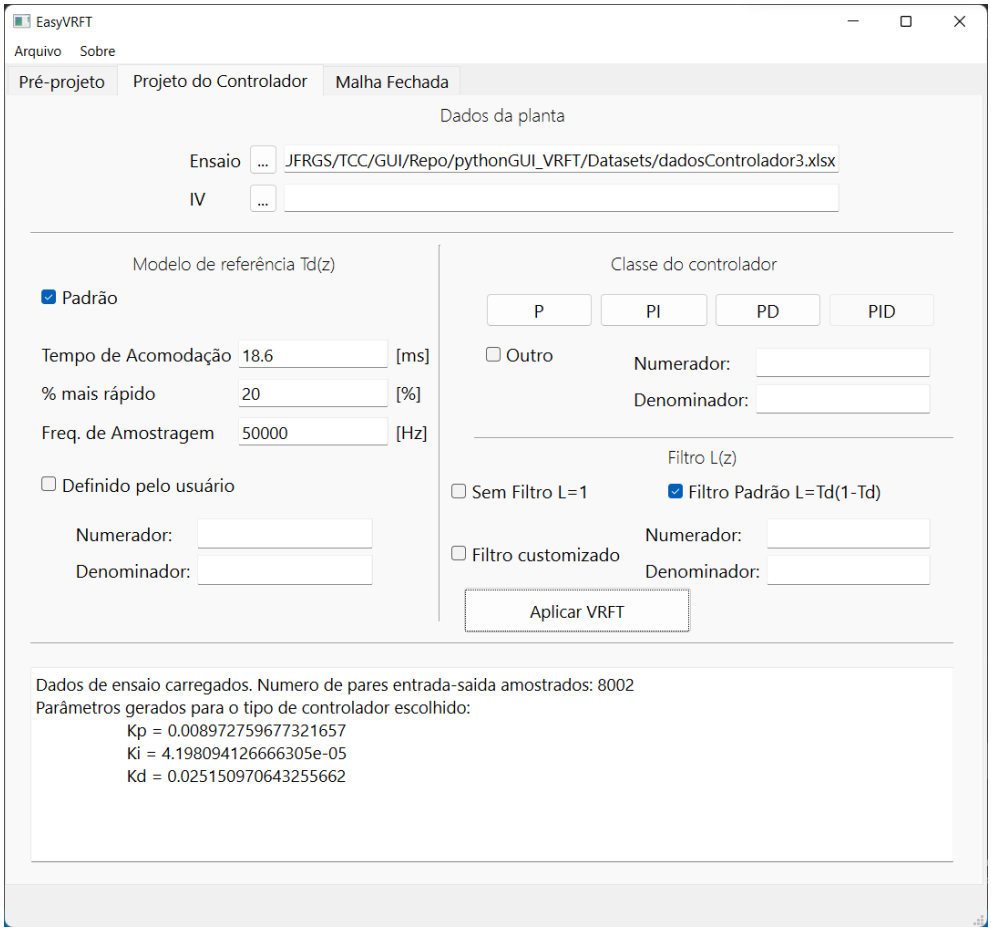
O mesmo erro discutido para estimar o tempo de acomodação da malha fechada com o primeiro controlador se repete aqui. Estes resultados podem ser considerados suficientemente próximos do valor almejado para o tempo de acomodação da malha fechada, que seria 14.88ms. Por isso, pode ser afirmado, assim, que os resultados obtidos da aplicação do método VRFT pela GUI são condizentes com os resultados esperados de seguimento de referência e ganho no tempo de acomodação do sistema para o segundo controlador projetado.

4.2.3 Resultados com Controlador 3

A seguir será exposto e analisado os resultados obtidos para o terceiro controlador projetado. Este controlador foi projetado conforme os parâmetros estabelecidos na Seção 4.2, utilizando o conjunto de dados em malha aberta fornecido da planta, onde se deseja parametrizar um controlador do tipo PID.

A Figura 41 traz os parâmetros informados à aplicação, assim como os parâmetros gerados para o controlador selecionado.

Figura 41 – Interface preenchida com os dados para projetar o terceiro controlador



Fonte: o Autor, 2022.

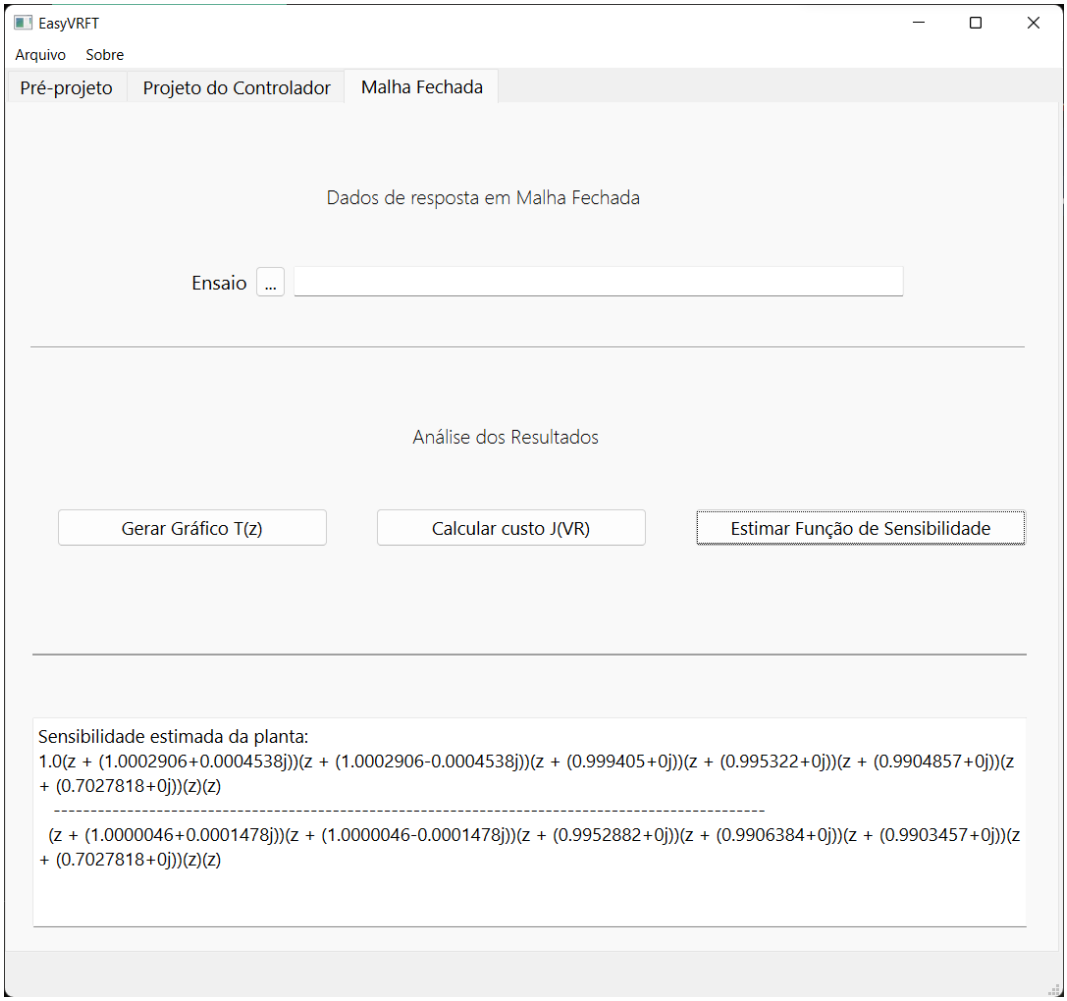
Como pode ser observado na imagem, os parâmetros para gerar o controlador foram informados conforme especificado. Para o controlador projetado os valores de ganho obtidos estão na Tabela 5.

Tabela 5 – Parâmetros do terceiro controlador

$\mathbf{K_p}$	0.00897275967
$\mathbf{K_i}$	4.19809412e-05
$\mathbf{K_d}$	0.02515097064

Com estes resultados, pode ser feita a análise da função de sensibilidade estimada para o sistema projetado, conforme (28). A Figura 42 mostra a função de sensibilidade calculada pela ferramenta, no formato de zeros, polos e ganho.

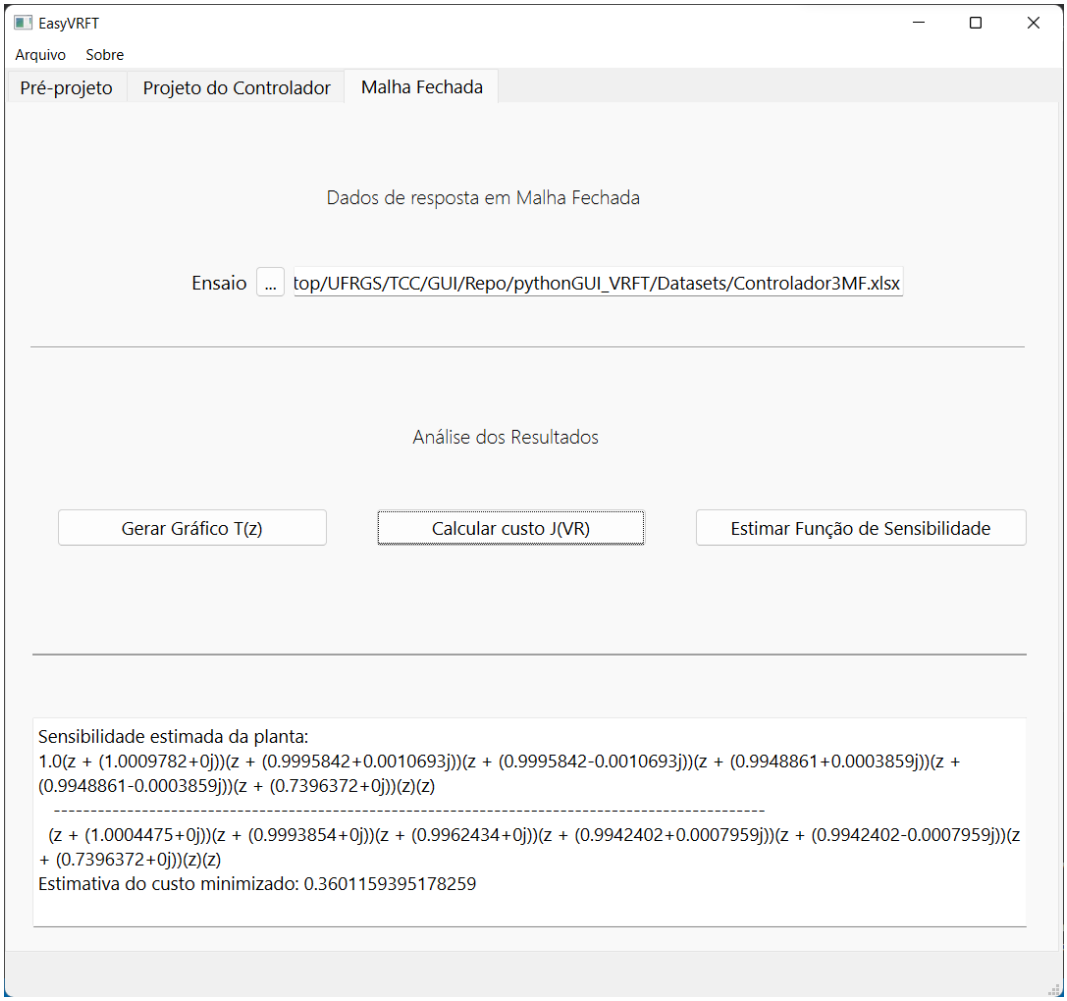
Figura 42 – Função de sensibilidade estimada para o terceiro controlador



Fonte: o Autor, 2022.

Agora, com o controlador em mãos, este foi simulado no conversor DSRAC para realizar o ensaio do sistema completo em malha fechada. Tendo o ensaio de malha fechada com o controlador projetado, este pode ser passado à ferramenta para que as últimas funcionalidades de avaliação do sistema projetado possam ser utilizadas. Informando mais este conjunto de dados, na terceira aba da GUI, pode ser avaliado o comportamento gráfico do sinal obtido da malha fechada com o controlador frente ao gráfico do sinal de saída da $T_d(z)$ obtido quando esta é excitada pelo sinal de referência deste ensaio em malha fechada. Também pode ser avaliada a minimização da função custo feita pela implementação do método VRFT. As Figuras 43 e 44 mostram os resultados das duas funcionalidades mencionadas.

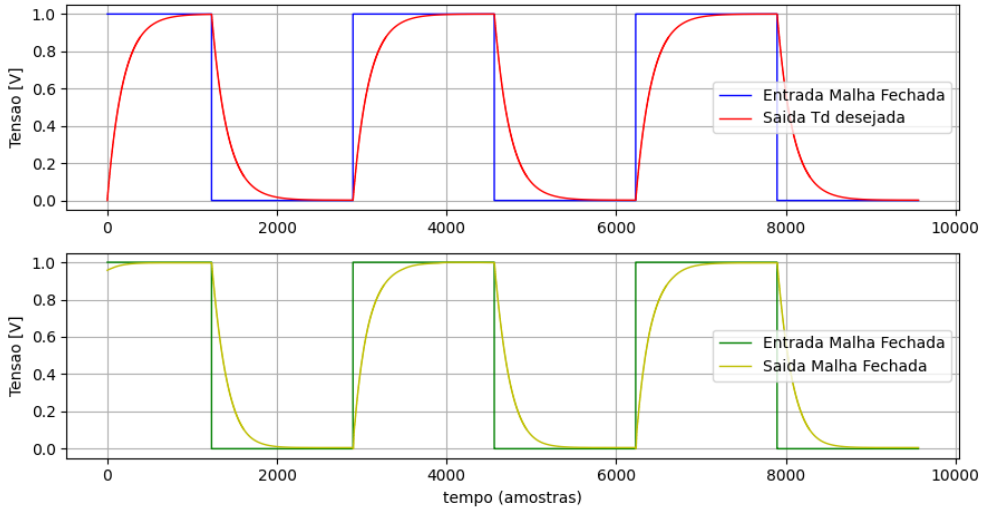
Figura 43 – Custo J^{VR} calculado para o sistema em malha fechada com o terceiro controlador



Fonte: o Autor, 2022.

Conforme pode ser visto na imagem, o erro calculado entre os resultados teórico e prático foi de 0.36011. Este valor corresponde a um número suficientemente baixo para poder afirmar que os resultados obtidos estão consideravelmente próximos. Esta análise é corroborada pela comparação gráfica dos dois sinais, vista na Figura 44.

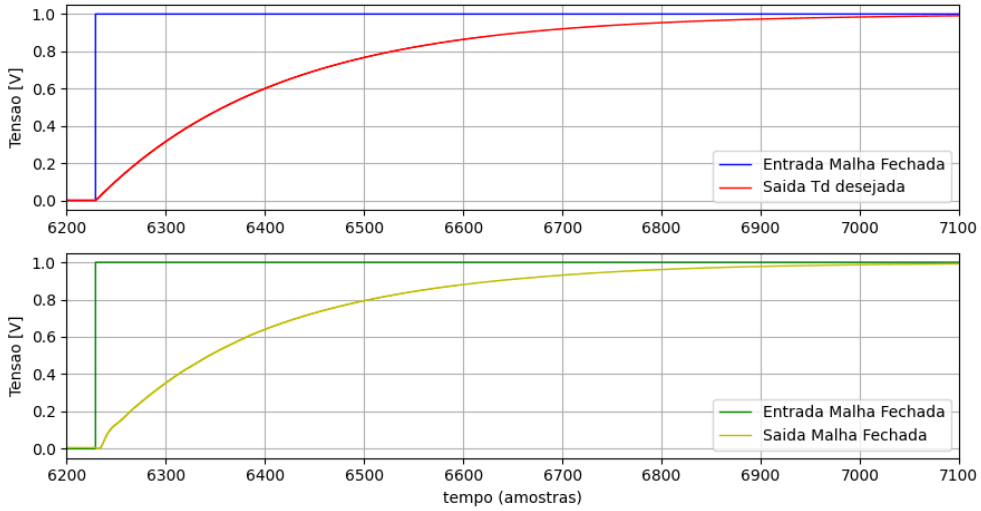
Figura 44 – Gráfico das saídas de $T_d(z)$ e do sistema em malha fechada com o terceiro controlador projetado



Fonte: o Autor, 2022.

Pode ser visto que o sinal gerado pela $T_d(z)$ está seguindo uma tendência similar ao sinal obtido via PSIM. E complementando a análise feita para a minimização da função custo, visualmente os dois sinais apresentam comportamentos parecidos, levando a crer que o erro calculado de fato reflete numa diferença pequena entre os resultados teórico e prático. A última análise que pode ser feita é validar se o controlador projetado atingiu o objetivo proposto de resultar em um sistema que tenha um tempo de acomodação 20% mais rápido que o sistema em malha aberta. Para isso, foi utilizada a funcionalidade de restringir os limites do eixo do gráfico, escolhendo a amostra 6200 como limite inferior e a amostra 7100 como limite superior. A Figura 45 mostra como ficou o gráfico após reduzir os limites de exibição.

Figura 45 – Gráfico aproximado para os sinais do sistema em malha fechada com o terceiro controlador



Fonte: o Autor, 2022.

Para calcular o tempo de acomodação em malha fechada, foi considerado que o degrau da onda quadrada ocorre aproximadamente na amostra 6250 e que os sinais de saída chegaram no patamar de estabilização aproximadamente na amostra 7000. Aplicando o cálculo do tempo de acomodação, o resultado obtido foi novamente de 15ms:

$$t_{sc} = \frac{n\ amostras}{frequencia} = \frac{7000 - 6250}{50000} = 15ms \tag{40}$$

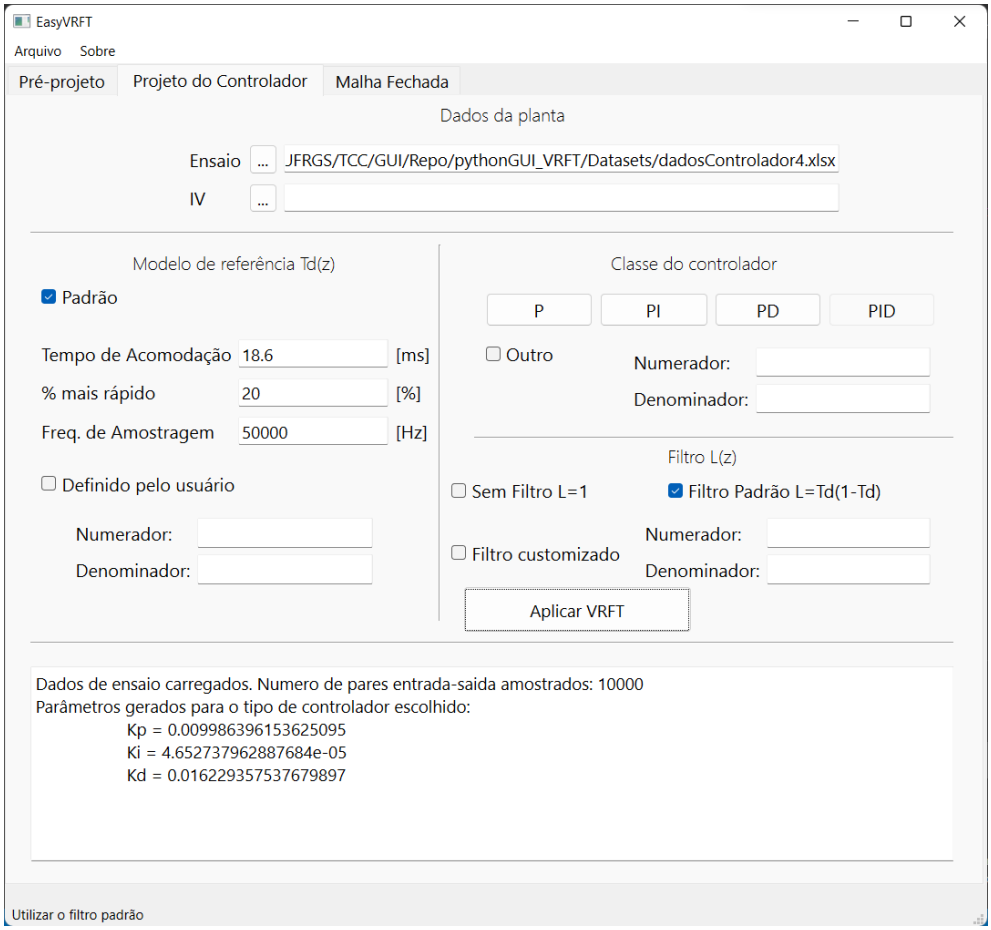
O mesmo erro discutido para estimar o tempo de acomodação da malha fechada com o primeiro controlador se repete aqui. Estes resultados podem ser considerados suficientemente próximos do valor almejado para o tempo de acomodação da malha fechada, que seria 14.88ms. Por isso, pode ser afirmado, assim, que os resultados obtidos da aplicação do método VRFT pela GUI estão condizentes com os resultados esperados de seguimento de referência e ganho no tempo de acomodação do sistema para o terceiro controlador projetado.

4.2.4 Resultados com Controlador 4

Nesta próxima seção será exposto e analisado os resultados obtidos para o quarto controlador projetado. Este controlador foi projetado conforme os parâmetros estabelecidos na Seção 4.2, utilizando o conjunto de dados em malha fechada com um controlador proporcional fornecido da planta, onde se deseja parametrizar um controlador do tipo PID.

A Figura 46 traz os parâmetros informados à aplicação, assim como os parâmetros gerados para o controlador selecionado.

Figura 46 – Interface preenchida com os dados para projetar o quarto controlador



Fonte: o Autor, 2022.

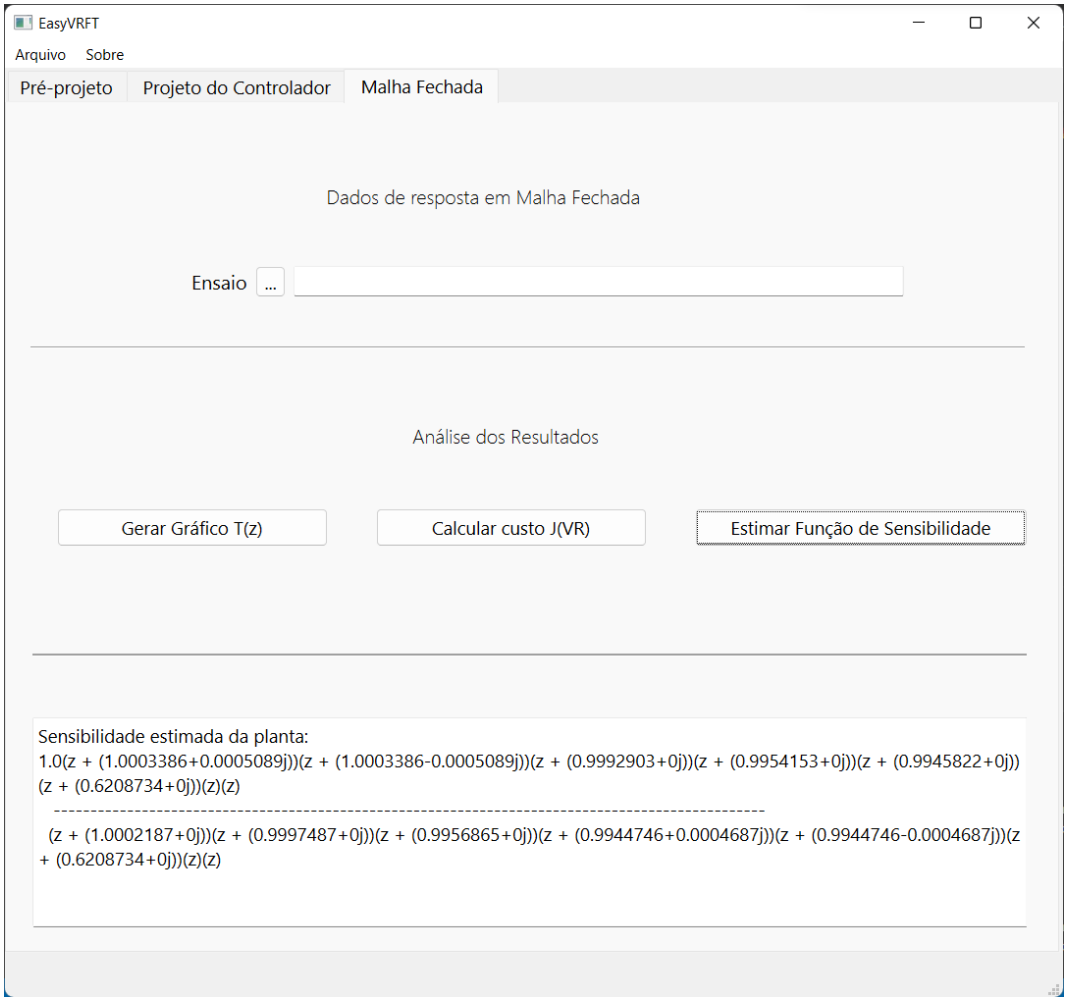
Como pode ser observado na imagem, os parâmetros para gerar o controlador foram informados conforme especificado. Para o controlador projetado os valores de ganho obtidos estão na Tabela 6.

Tabela 6 – Parâmetros do quarto controlador

$\mathbf{K_p}$	0.00998639615
$\mathbf{K_i}$	4.65273796e-05
$\mathbf{K_d}$	0.01622935753

Com estes resultados, pode ser feita a análise da função de sensibilidade estimada para o sistema projetado, conforme (28). A Figura 47 mostra a função de sensibilidade calculada pela ferramenta, no formato de zeros, polos e ganho.

Figura 47 – Função de sensibilidade estimada para o quarto controlador

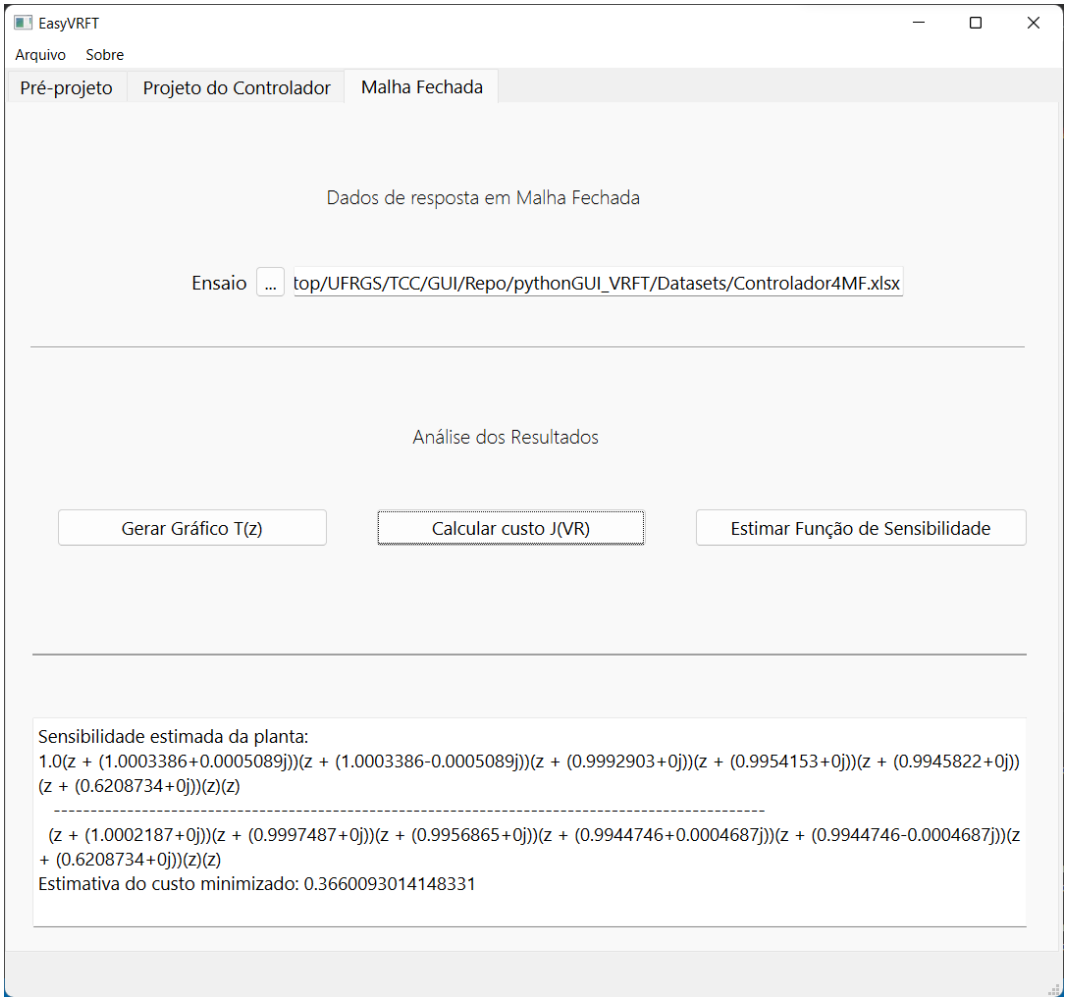


Fonte: o Autor, 2022.

Agora, com o controlador em mãos, este foi simulado no conversor DSRAC para realizar o ensaio do sistema completo em malha fechada. Tendo o ensaio de malha fechada com o controlador projetado, este pode ser passado à ferramenta para que as últimas funcionalidades de avaliação do sistema projetado possam ser utilizadas. Informando mais este conjunto de dados, na terceira aba da GUI, pode ser avaliado o comportamento gráfico do sinal obtido da malha fechada com o controlador frente ao gráfico do sinal de saída da $T_d(z)$ obtido quando esta é excitada pelo sinal de entrada deste ensaio em malha fechada. Também pode ser avaliada a minimização da função custo feita pela implementação do método VRFT. As Figuras 48 e 49 mostram os resultados das duas funcionalidades mencionadas.

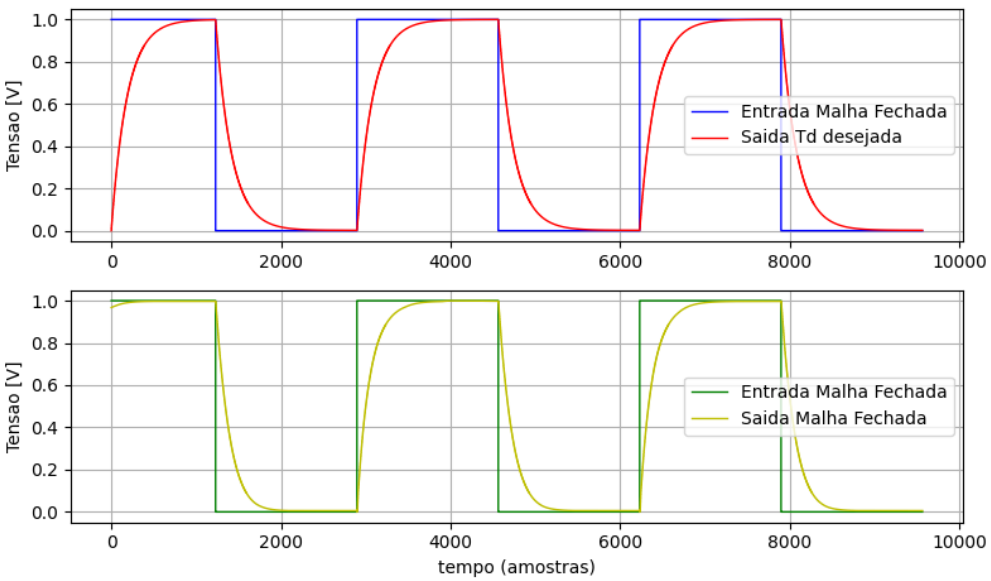
Conforme pode ser visto na imagem, o erro calculado entre os resultados teórico e prático foi de 0.366009. Este valor corresponde a um número suficientemente baixo para poder afirmar que os resultados obtidos estão consideravelmente próximos. Esta análise é corroborada pela comparação gráfica dos dois sinais, vista na Figura 49.

Figura 48 – Custo J^{VR} calculado para o sistema em malha fechada com o quarto controlador



Fonte: o Autor, 2022.

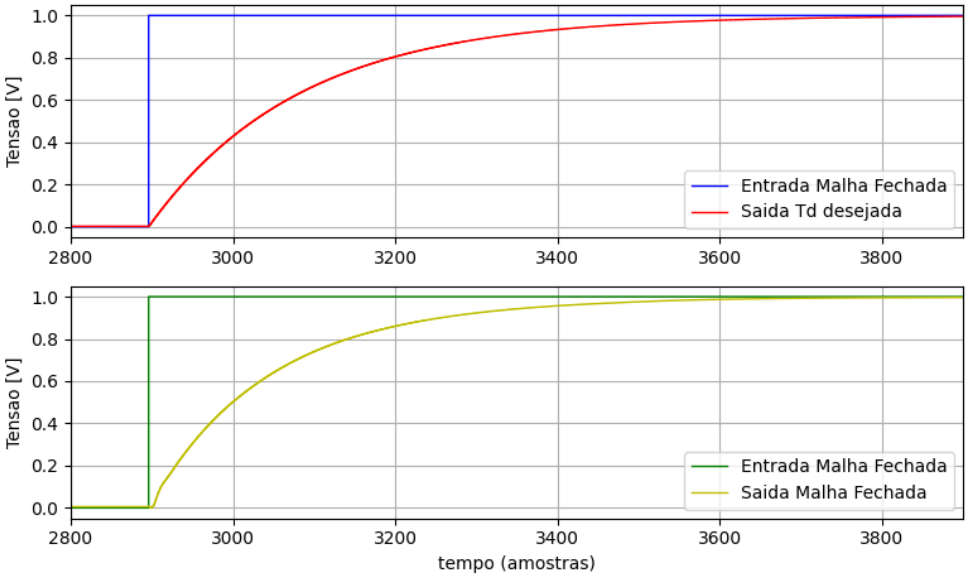
Figura 49 – Gráfico das saídas de $T_d(z)$ e do sistema em malha fechada com o quarto controlador projetado



Fonte: o Autor, 2022.

Pode ser visto que o sinal simulado está seguindo uma tendência similar ao sinal experimental. E complementando a análise feita para a minimização da função custo, visualmente os dois sinais apresentam comportamentos parecidos, levando a crer que o erro calculado de fato reflete numa diferença pequena entre os resultados teórico e prático. A última análise que pode ser feita para validar se o controlador projetado atingiu o objetivo proposto de compor um sistema que tenha um tempo de acomodação 20% mais rápido que o sistema em malha aberta é novamente estimar esse tempo de acomodação a partir dos dados fornecidos. Para isso, foi utilizada a funcionalidade de restringir os limites do eixo do gráfico, escolhendo a amostra 2800 como limite inferior e a amostra 3900 como limite superior. A Figura 50 mostra como ficou o gráfico após reduzir os limites de exibição.

Figura 50 – Gráfico aproximado para os sinais do sistema em malha fechada com o quarto controlador



Fonte: o Autor, 2022.

Para calcular o tempo de acomodação em malha fechada, foi considerado que o degrau da onda quadrada ocorre aproximadamente na amostra 2900 e que os sinais de saída chegaram no patamar de estabilização aproximadamente na amostra 3650. Aplicando o cálculo do tempo de acomodação, o resultado obtido foi novamente de 15ms:

$$t_{sc} = \frac{n\ amostras}{frequencia} = \frac{3650 - 2900}{50000} = 15ms \tag{41}$$

O mesmo erro discutido para estimar o tempo de acomodação da malha fechada com o primeiro controlador se repete aqui. Estes resultados podem ser considerados suficientemente próximos do valor almejado para o tempo de acomodação da malha fechada, que seria 14.88ms. Por isso, pode ser afirmado, assim, que os resultados obtidos da aplicação do método VRFT pela GUI são condizentes com os resultados esperados

de seguimento de referência e ganho no tempo de acomodação do sistema para o quarto controlador projetado.

4.3 Considerações Finais

Neste capítulo foram expostos os resultados obtidos a partir da GUI implementada. Os parâmetros calculados com o exemplo disponibilizado pela biblioteca *pyvrft* apresentaram os valores esperados, assim como, ao aplicar o controlador projetado no conversor DSRAC, foi observado uma redução no tempo de acomodação do sistema em relação ao tempo de acomodação em malha fechada, além de medidas de erro entre os resultados práticos e simulados baixas. Vale ressaltar que o cálculo da minimização da função custo pelo erro médio quadrático entre os sinais experimentais e simulados forneceu um grau de confiança mais elevado de forma que permitiu uma comparação gráfica entre estes dois sinais, sem que seja necessária uma rigorosidade maior na hora de estimar os valores para o tempo de acomodação.

5 Conclusões

Esse trabalho apresenta uma ferramenta gráfica para a aplicação do método VRFT no projeto de controladores no contexto de conversores CC-CC. Além disso, essa ferramenta possui funções que permitem uma análise gráfica e quantitativa dos resultados obtidos com o controlador calculado.

Para chegar no objetivo proposto do trabalho foi elaborada uma interface gráfica com o software Qt Designer, onde os elementos que fazem a interação com o usuário da aplicação foram adicionados. Esses elementos tem a função de receber as informações que o usuário precisa passar para que a aplicação funcione, além de servirem como forma de expor alguns dos resultados obtidos. Ao integrar a funcionalidade dos elementos dispostos com as funcionalidades das bibliotecas *pyqt6* e *pyvrft*, foi possível criar uma GUI que utilize as informações entradas pelo usuário para executar o método VRFT e encontrar os parâmetros do controlador que o usuário deseja parametrizar.

Simulações preliminares mostraram que o resultado obtido com a utilização da ferramenta estava condizente com a aplicação da biblioteca *pyvrft* diretamente por linha de código. O resultado obtido através da aplicação criada para o exemplo disponibilizado no repositório da própria biblioteca foi o mesmo que aquele encontrado quando o exemplo é executado sem o uso da aplicação desenvolvida.

Os resultados obtidos quando a aplicação foi usada para projetar o controlador para uma planta real também foram satisfatórios, visto que em todos os casos:

- O requisito proposto de alcançar uma resposta em malha fechada mais rápida que em malha aberta foi identificado, melhorando o tempo de acomodação de aproximadamente 18.6ms em malha aberta para aproximadamente 15ms em malha fechada, que corresponde aos 20% estabelecido como requisito de projeto;
- O comportamento do sistema em malha fechada correspondeu com o comportamento do sistema simulado, conforme evidenciado pelo erro médio quadrático obtido, que em todos os casos, ficou perto do valor de 0.36;
- A análise gráfica também permitiu observar tanto o sinal experimental quanto o sinal teórico seguindo o critério de projeto de seguimento de referência.

Em contra partida, certos fatores acabam impondo limitações às conclusões que podem ser tiradas dos resultados obtidos. Como só foi possível realizar o teste do controlador projetado em uma única planta real e sempre para o mesmo conjunto de condições de comportamento esperado, conclusões generalizadas para diferentes tipos de conversores

perdem um pouco a sua força. Dito isso, é fato que o método VRFT já foi estudado e validado por diversos outros trabalhos da literatura, trazendo assim um grau de confiança considerável aos resultados obtidos a partir da sua utilização.

Por fim, conclui-se que a primeira versão da ferramenta desenvolvida é de fato capaz de simplificar a aplicação do método VRFT em conversores CC-CC.

5.1 Trabalhos Futuros

O primeiro e mais evidente ponto a ser desenvolvido em trabalhos futuros é a implementação do método VRFT com critério flexível, para que a aplicação possa ser utilizada com plantas de conversores CC-CC que contenham zeros de fase não-mínima que influenciem no comportamento do sistema.

Maiores refinamentos na aplicação desenvolvida, principalmente, se tratando do tratamento de informações que forem passadas fora dos padrões dos tipos esperados pela aplicação, podem deixar a ferramenta mais robusta e mais amigável ao usuário. Como esta foi desenvolvida almejando garantir o funcionamento para quando as informações passadas pelo usuário estão corretas, não foram implementadas funcionalidades para tratar todos os casos em que as informações passadas produzem algum tipo de erro interno. Aperfeiçoamentos na forma como os resultados são expostos ao usuário também são um caminho possível de ser seguido, especialmente para a função de sensibilidade calculada.

Expansão das funcionalidades da aplicação, tanto para os tipos de controladores previstos na etapa de pré-projeto, quanto nos métodos disponibilizados para o projeto do controlador. O VRFT não é o único método para projeto de controladores baseado em dados, então seria interessante implementar outros métodos, de forma que a ferramenta funcione como um centralizador de métodos DD para esse tipo de aplicação.

Também é fundamental a criação de um manual de uso de todas as funcionalidades da ferramenta, interno a própria ferramenta, de forma que fique mais prática a consulta ao funcionamento de um determinado botão ou ao formato de dado aceito por algum campo de texto específico, por exemplo. Uma opção que permita que o usuário salve em um arquivo as informações passadas à aplicação e às geradas por ela para que possa retomar o projeto de onde ele parou é outra funcionalidade interessante de ser implementada.

Referências Bibliográficas

BAZANELLA, A.; CAMPESTRINI, L.; ECKHARD, D. *Data-Driven Controller Design*: The h2 approach. [S.l.]: Springer, 2011. 221 p. ISBN 9789400723009.

CAMPESTRINI, L.; ECKHARD, D.; GEVERSB, M.; BAZANELLA, A. S. Virtual reference feedback tuning for non-minimum phase plants. *Elsevier B.V.*, v. 47, p. 1778–1784, Aug. 2011.

CAMPI, M.; LECCHINI, A.; SAVARESI, S. Virtual reference feedback tuning (vrft): A new direct approach to the design of feedback controllers. *Proceedings of the 39th IEEE Conference on Decision and Control*, p. 623–629, Dez. 2000.

CAMPI, M.; LECCHINI, A.; SAVARESI, S. Virtual reference feedback tuning: a direct method for the design of feedback controllers. *Automatica*, v. 38, n. 8, p. 1337–1346, 2002. ISSN 0005-1098. Disponível em: <https://www.sciencedirect.com/science/article/pii/S0005109802000328>.

ECKHARD, D.; BOEIRA, E. pyvrft: A python package for the virtual reference feedback tuning, a direct data-driven control method. *Elsevier B.V.*, p. 6, Dez. 2019.

ERICKSON, R. W.; MAKSIMOVIC, D. Fundamentals of power electronics. 2nd. *New York: Kluwer Academic Publishers*, 2001.

GOPI, R. R.; SREEJITH, S. Converter topologies in photovoltaic applications - a review. *Elsevier B.V.*, v. 94, p. 1–14, Oct. 2018.

KANZIAN, M.; AGOSTINELLI, M.; HUEMER, M. Digital hysteresis sliding mode control for interleaved dc–dc converters. *Elsevier B.V.*, v. 90, p. 148–159, Sep. 2019.

QTCOMPANY. *PySide6 QtWidgets*. 2022. Disponível em: <https://doc.qt.io/qtforpython/PySide6/QtWidgets/index.html#>.

REMES, C. Aplicação de metodologias de controle baseado em dados em conversores cc-cc. p. 159, 2021.

REMES, C.; BINOTTO, R.; FLORES, J.; LÍBANO, F.; CAMPESTRINI, L. Virtual reference feedback tuning applied to dc-dc converters. p. 9, 2019.

SALATI, G. Modelagem e controle de um conversor cc-cc duplo serie-ressonante com grampeamento ativo. p. 97, 2021.

SILVA-ORTIGOZA, R.; HERNÁNDEZ-GUZMÁN, V. M.; ANTONIO-CRUZ, M.; MUÑOZ-CARRILLO, D. Dc-dc buck power converter as a smooth starter for a dc motor based on a hierarchical control. *IEEE Transactions on Power Electronics*, v. 30, p. 1076–1084, Feb. 2015.

YI, W.; MA, H.; PENG, S.; LIU, D.; ALI, Z. M.; DAMPAGE, U.; HAJJIAH, A. Analysis and implementation of multi-port bidirectional converter for hybrid energy systems. *Energy Reports*, v. 8, p. 1538–1549, 2022. ISSN 2352-4847. Disponível em: <https://www.sciencedirect.com/science/article/pii/S2352484721015079>.

APÊNDICE A – Códigos das principais implementações

Neste apêndice foram adicionadas as implementações dos principais métodos desenvolvidos para alcançar o funcionamento previsto para a interface criada. Os códigos completos, junto de um exemplo de uso, podem ser encontrados no repositório informado na Seção 4.

A.1 Primeira Aba

Método que faz o cálculo do ganho estático máximo para a coleta dos dados da planta.

```
## Botao para gerar o ponto ideal para coleta de dados
def pontoIdealPressed(self):
    duty = self.__textCapture(8)
    v0 = self.__textCapture(9)

    controllerType = self.gPre.checkedId()

    # Não faz nada enquanto todas as opções não forem selecionadas
    if controllerType == -1 or duty is None or v0 is None:
        pass
    elif controllerType == 15:
        ideal = v0[0] / duty[0]
    elif controllerType == 16:
        ideal = v0[0] / (1 - duty[0])
    elif controllerType == 17:
        ideal = v0[0] / (duty[0] * (1 - duty[0]))
    elif controllerType == 18:
        ideal = v0[0] / (duty[0] * (1 - duty[0]))

    if controllerType != -1 and duty is not None and v0 is not None:
        ideal = 1 / ideal

        texto = "Ganho estático máximo: " + str(ideal) + "
   [V<sup>-1</sup>]"

        self.preOutput.append(texto)
```

A.2 Segunda Aba

Método que interpreta a seleção feita para o modelo de referência.

```
def getRef(self):
    if self.ensaioText.text() != '':

        modelFunc = self.gTd.checkedId()

        if modelFunc == 5:
            tso = self.textCapture(1)
            speed = self.textCapture(2)
            freq = self.textCapture(3)

            p1 =
   math.exp(-(4/freq[0])/(tso[0]*10**-3*(1-0.01*speed[0])))

            self.modelNum = [1 - p1]
            self.modelDen = [1, -p1]

        elif modelFunc == 6:
            self.modelNum = self.textCapture(4)
            self.modelDen = self.textCapture(5)

        elif modelFunc == -1:
            # TODO: Error flag
            texto = "Modelo de referência invalido"
            self.controllerOutput.appendPlainText(texto)

        objectiveFunction = signal.TransferFunction(self.modelNum,
   self.modelDen, dt=1)

        return objectiveFunction
```

Método que interpreta a seleção feita para o tipo de controlador.

```
def getCont(self):
    # Conforme selecao do usuario, seta o tipo de controlador
    controlClass = self.gControlador.checkedId()

    pNum = [1]
    pDen = [1]

    iNum = [1, 0]
    iDen = [1, -1]

```

```
11      dNum = [1, -1]
12      dDen = [1, 0]
13
14      if controlClass == 7:  # Proporcional
15          controllerModel = [[signal.TransferFunction(pNum, pDen, dt=1)]]
        # Kp
16          k = "p"
17
18      elif controlClass == 8:  # Proporcional Integral
19          controllerModel = [[signal.TransferFunction(pNum, pDen, dt=1)],
        # Kp
20                             [signal.TransferFunction(iNum, iDen, dt=1)],
    ]  # Ki
21          k = "pi"
22
23      elif controlClass == 9:  # Proporcional Derivativo
24          controllerModel = [[signal.TransferFunction(pNum, pDen, dt=1)],
        # Kp
25                             [signal.TransferFunction(dNum, dDen, dt=1)],
    ]  # Kd
26          k = "pd"
27
28      elif controlClass == 10:  # Proporcional Integral Derivativo
29          controllerModel = [[signal.TransferFunction(pNum, pDen, dt=1)],
        # Kp
30                             [signal.TransferFunction(iNum, iDen, dt=1)],
        # Ki
31                             [signal.TransferFunction(dNum, dDen, dt=1)],
    ]  # Kd
32          k = "pid"
33
34      elif controlClass == 11:  # Custom
35          controllerNumTemp, self.numNum = self.customControllerSetter(0)
36          controllerDenTemp, numDen = self.customControllerSetter(1)
37          # TODO: flag que impede sequencia se o len de cada for diferente
38
39          contNum = len(controllerNumTemp)
40          contDen = len(controllerDenTemp)
41          temp = contNum + contDen
42
43          contador, i, j = 0, 0, 0
44          controllerTemp = []
45
46          # TODO:
47          while contador < temp:
48              if (contador % 2) == 0:
49                  controllerTemp.append(controllerDenTemp[i])
```

```
                i += 1
            else:
                controllerTemp.append(controllerNumTemp[j])
                j += 1
            contador += 1

        controllerModel = []
        contador = 1

        while contador < len(controllerTemp):

    controllerModel.append([signal.TransferFunction(controllerTemp[contador],
    controllerTemp[contador - 1], dt=1)])
            contador += 2

        k = "custom"

    elif controlClass == -1:
        # TODO: Error flag
        texto = "Modelo do controlador invalido"
        self.controllerOutput.appendPlainText(texto)
        controllerModel = 0
        k = 0

    return controllerModel, k
```

Método que interpreta a seleção feita para o filtro L(z).

```
def getFilter(self):
    # Conforme selecao do usuario, seta o tipo de filtro
    filterType = self.gFiltro.checkedId()

    if filterType == 12:  # Sem Filtro
        filterNum = [1]
        filterDen = [1]

    elif filterType == 13:  # Filtro Padrão
        # L = Td * (1 - Td)
        um = control.tf([1], [1], dt=1)
        tTemp = control.tf(self.modelNum, self.modelDen, dt=1)
        tDesejada = (um - tTemp)
        tDesejada = tDesejada * tTemp

        numL = tDesejada.num
        filterNum = numL[0][0]
        denL = tDesejada.den
        filterDen = denL[0][0]
```

```
20
21      elif filterType == 14:  # Filtro Custom
22          filterNum = self.textCapture(6)
23          filterDen = self.textCapture(7)
24
25      elif filterType == -1:
26          # TODO: Error flag
27          texto = "Modelo de filtro invalido"
28          self.controllerOutput.appendPlainText(texto)
29
30      filterL = signal.TransferFunction(filterNum, filterDen, dt=1)
31
32      return filterL
```

Método que agrega as informações relevantes ao VRFT e executa o método a partir delas.

```
1  def VRFTPressed(self):
2      self.flag = False
3
4      if self.ensaioText.text() != '':
5
6          # Gera o modelo de referência da planta
7          objectiveFunction = self.getRef()
8
9          pNum = [1]
10         pDen = [1]
11
12         iNum = [1, 0]
13         iDen = [1, -1]
14
15         dNum = [1, -1]
16         dDen = [1, 0]
17
18         # Gera a função do controlador
19         controllerModel, k = self.getCont()
20
21         # Gera a função do filtro
22         filterL = self.getFilter()
23
24         num = self.dfEnsaio.shape[0]
25
26         dfEnsaioCopia = self.dfEnsaio.copy()
27
28         entradaTemp =
    dfEnsaioCopia.pop(dfEnsaioCopia.columns[0]).to_numpy()
```

```
29          saidaTemp =
    dfEnsaioCopia.pop(dfEnsaioCopia.columns[0]).to_numpy()
30
31          entradaEnsaio = np.ones((num, 1))
32          saidaEnsaio = np.ones((num, 1))
33
34          for x in range(0, num):
35              entradaEnsaio[x] = entradaTemp[x]
36              saidaEnsaio[x] = saidaTemp[x]
37
38          iv = self.ivText.text()
39          if iv == "":  # Nao tem ensaio com variavel instrumentavel
40              saidaIV = saidaEnsaio
41          else:
42              numIV = self.dfIV.shape[0]
43              dfIVCopia = self.dfIV.copy()
44              saidaIVTemp = dfIVCopia.pop(dfIVCopia.columns[1]).to_numpy()
45              saidaIV = np.ones((numIV, 1))
46              for x in range(0, numIV):
47                  saidaIV[x] = saidaIVTemp[x]
48
49          '''
50          resultado = vrft.design(dfEnsaio.dados de entrada,
    dfEnsaio.dados de saida, dfIV.dados de saida com variavel
    instrumentavel, funcao de transferencia objetivo, modelo de
    controle, filtro)
51          '''
52
53          resultado = vrft.design(entradaEnsaio, saidaEnsaio, saidaIV,
    objectiveFunction, controllerModel, filterL)
54
55          textoaux = "Parâmetros gerados para o tipo de controlador
    escolhido: \n\tKp = "
56
57          # Exibe resultados de um controlador P
58          if k == "p":
59              ganhos = [resultado[0].item()]
60              texto = textoaux + str(ganhos[0])
61              cNum = [i * ganhos[0] for i in pNum]
62              self.contNum = cNum
63              self.contDen = pDen
64              self.flag = True
65
66          # Exibe resultados de um controlador PI
67          elif k == "pi":
68              ganhos = [resultado[0].item(), resultado[1].item()]
```

```
69              texto = textoaux + str(ganhos[0]) + "\n\tKi = " +
    str(ganhos[1])
70              cNum1 = [i * ganhos[0] for i in pNum]
71              cNum2 = [j * ganhos[1] for j in iNum]
72              cFinal = control.tf(cNum1, pDen, dt=1) + control.tf(cNum2,
    iDen, dt=1)
73              cAux = cFinal.num
74              self.contNum = cAux
75              cAux = cFinal.den
76              self.contDen = cAux
77              self.flag = True
78
79          # Exibe resultados de um controlador PD
80          elif k == "pd":
81              ganhos = [resultado[0].item(), resultado[1].item()]
82              texto = textoaux + str(ganhos[0]) + "\n\tKd = " +
    str(ganhos[1])
83              cNum1 = [i * ganhos[0] for i in pNum]
84              cNum2 = [i * ganhos[1] for i in dNum]
85              cFinal = control.tf(cNum1, pDen, dt=1) + control.tf(cNum2,
    dDen, dt=1)
86              cAux = cFinal.num
87              self.contNum = cAux
88              cAux = cFinal.den
89              self.contDen = cAux
90              self.flag = True
91
92          # Exibe resultados de um controlador PID
93          elif k == "pid":
94              ganhos = [resultado[0].item(), resultado[1].item(),
    resultado[2].item()]
95              texto = textoaux + str(ganhos[0]) + "\n\tKi = " +
    str(ganhos[1]) + "\n\tKd = " + str(ganhos[2])
96              cNum1 = [i * ganhos[0] for i in pNum]
97              cNum2 = [i * ganhos[1] for i in iNum]
98              cNum3 = [i * ganhos[2] for i in dNum]
99              cFinal = control.tf(cNum1, pDen, dt=1) + control.tf(cNum2,
    iDen, dt=1) + control.tf(cNum3, dDen, dt=1)
100             cAux = cFinal.num
101             self.contNum = cAux
102             cAux = cFinal.den
103             self.contDen = cAux
104             self.flag = True
105
106         # Exibe resultados de um controlador customizado
107         elif k == "custom":
108             ganhos = [resultado[x].item() for x in range(self.numNum)]
```

```
109             texto = "Parâmetros gerados para o tipo de controlador
    escolhido: \n\t"
110             textoaux = ""
111             for y in range(self.numNum):
112                 textoaux = textoaux + "K" + str(y + 1) + " = " +
    str(ganhos[y]) + "\n\t"
113             texto = texto + textoaux
114             self.flag = True
115 
116         self.controllerOutput.appendPlainText(texto)
117 
118     else:
119         texto = "Adicione um conjunto de dados."
120         self.controllerOutput.setPlainText(texto)
```

A.3 Terceira Aba

Método que exibe os gráficos da terceira aba.

```
1  def graphTzPressed(self):
2      lw = 1  # linewidth
3      case = 0
4      count1 = False
5      count2 = False
6  
7      if self.flag:
8          count1 = True
9      if self.ensaioMFText.text() != '':
10         count2 = True
11 
12     if count1 or count2:
13         if not count2:
14             case = 1
15         elif not count1:
16             case = 2
17         else:
18             case = 3
19 
20     # Gráfico dados da malha fechada
21     if case == 2:
22         num = self.dfEnsaioMF.shape[0]
23 
24         dfEnsaioMFCopia = self.dfEnsaioMF.copy()
25 
```

```
26          entradaMFTemp =
    dfEnsaioMFCopia.pop(dfEnsaioMFCopia.columns[0]).to_numpy()
27          saidaMFTemp =
    dfEnsaioMFCopia.pop(dfEnsaioMFCopia.columns[0]).to_numpy()
28
29          entradaMF = np.ones((num, 1))
30          saidaMF = np.ones((num, 1))
31
32          for x in range(0, num):
33              entradaMF[x] = entradaMFTemp[x]
34              saidaMF[x] = saidaMFTemp[x]
35
36          plt.plot(entradaMF, "r", drawstyle="steps", linewidth=lw,
    label="Entrada")
37          plt.plot(saidaMF, "b", drawstyle="steps", linewidth=lw,
    label="Saida")
38          plt.grid(True)
39          plt.xlabel("tempo (amostras)")
40          plt.ylabel("Tensao [V]")
41          plt.xlim(left=-2, right=num)
42          plt.legend()
43          plt.tight_layout()
44
45      # Gráfico Td e malha fechada juntas
46      elif case == 3:
47          num = self.dfEnsaioMF.shape[0]
48
49          Td = signal.TransferFunction(self.modelNum, self.modelDen, dt=1)
50
51          # Dados Ensaio Malha Fechada
52          dfEnsaioMFCopia = self.dfEnsaioMF.copy()
53
54          entradaMFTemp =
    dfEnsaioMFCopia.pop(dfEnsaioMFCopia.columns[0]).to_numpy()
55          saidaMFTemp =
    dfEnsaioMFCopia.pop(dfEnsaioMFCopia.columns[0]).to_numpy()
56
57          entradaMF = np.ones((num, 1))
58          saidaMF = np.ones((num, 1))
59
60          for x in range(0, num):
61              entradaMF[x] = entradaMFTemp[x]
62              saidaMF[x] = saidaMFTemp[x]
63
64          yTd = vrft.filter(Td, entradaMF)
65
66          # Remove os dados que influenciam na escala dos sianis
```

```
67          minimo = np.amin(entradaMF) if np.amin(entradaMF) <=
    np.amin(saidaMF) else np.amin(saidaMF)
68
69          where = np.where(yTd >= minimo)
70          index = where[0][0]
71
72          yTd = yTd[index: num]
73          saidaMF = saidaMF[index: num]
74          entradaMF = entradaMF[index: num]
75
76          # plot 1:
77          plt.subplot(2, 1, 1)
78          plt.plot(entradaMF, "b", drawstyle="steps", linewidth=lw,
    label="Entrada Malha Fechada")
79          plt.plot(yTd, "r", drawstyle="steps", linewidth=lw,
    label="Saida Td desejada")
80          plt.grid(True)
81          plt.ylabel("Tensao [V]")
82          plt.legend()
83
84          # plot 2:
85          plt.subplot(2, 1, 2)
86          plt.plot(entradaMF, "g", drawstyle="steps", linewidth=lw,
    label="Entrada Malha Fechada")
87          plt.plot(saidaMF, "y", drawstyle="steps", linewidth=lw,
    label="Saida Malha Fechada")
88          plt.grid(True)
89          plt.legend()
90          plt.xlabel("tempo (amostras)")
91          plt.ylabel("Tensao [V]")
92          plt.tight_layout()
93
94      elif case <= 1:
95          texto = "Adicione um ensaio de malha fechada e aplique o método
    VRFT primeiro."
96          self.MFOutput.setPlainText(texto)
97
98      if case > 0:
99          plt.show()
```

Método que calcula a função custo J^{VR}.

```
1 def JVRPressed(self):
2
3     count1 = False
4     count2 = False
5
```

```
6       if self.flag:
7           count1 = True
8       if self.ensaioMFText.text() != '':
9           count2 = True
10
11      if count1 == True and count2 == True:
12
13          num = self.dfEnsaioMF.shape[0]
14
15          dfEnsaioMFCopia = self.dfEnsaioMF.copy()
16
17          entradaMFTemp =
    dfEnsaioMFCopia.pop(dfEnsaioMFCopia.columns[0]).to_numpy()
18          saidaMFTemp =
    dfEnsaioMFCopia.pop(dfEnsaioMFCopia.columns[0]).to_numpy()
19
20          entradaMF = np.ones((num, 1))
21          saidaMF = np.ones((num, 1))
22
23          for x in range(0, num):
24              entradaMF[x] = entradaMFTemp[x]
25              saidaMF[x] = saidaMFTemp[x]
26
27          Td = signal.TransferFunction(self.modelNum, self.modelDen, dt=1)
28
29          yTd = vrft.filter(Td, entradaMF)
30
31          maxEnsaio = np.amax(yTd)
32          minEnsaio = np.amin(yTd)
33
34          for y in range(0, num):
35              yTd[y] = (yTd[y] - minEnsaio) / (maxEnsaio - minEnsaio)
36
37          maxMF = np.amax(saidaMF)
38          minMF = np.amin(saidaMF)
39
40          for y in range(0, num):
41              saidaMF[y] = (saidaMF[y] - minMF) / (maxMF - minMF)
42
43          soma = 0
44
45          for i in range(num):
46              diff = saidaMF[i][0] - yTd[i][0]
47              squareDiff = diff ** 2
48              soma += squareDiff
49
50          MSE = soma / num
```

```
        texto = "Estimativa do custo minimizado: " + str(MSE)
        self.MFOutput.appendPlainText(texto)

    else:
        texto = "Aplicar o método VRFT primeiro e adicionar os dados da
    malha fechada."
        self.MFOutput.setPlainText(texto)
```

Método que calcula a função de sensibilidade do sistema simulado.

```
def sensiPressed(self):

    if self.flag:
        '''
        T = C * G / (1 + C * G)
        G = T / (C - T * C)
        '''

        tdEsperado = control.tf(self.modelNum, self.modelDen, dt=1)

        cProj = control.tf(self.contNum, self.contDen, dt=1)

        G = tdEsperado / (cProj - (tdEsperado * cProj))

        sensi = 1 / (1 + G * cProj)

        x = sensi.num
        x = list(x[0][0])
        y = sensi.den
        y = list(y[0][0])

        z, p, k = control.tf2zpk(x, y)

        tam1 = z.shape[0]

        texto = "Sensibilidade estimada da planta: "
        self.MFOutput.setPlainText(texto)

        texto = [str(k)]
        auxtext = "(z + ("

        for i in range(tam1):
            redondo = np.round(z[i], 7)
            if z[i] == 0:
                aux = "(z)"
            elif "j" in str(redondo):
                aux = auxtext + str(redondo)[1:-1] + "))"
```

```
            else:
                aux = auxtext + str(redondo) + "))"
            texto.append(aux)

        texto = ''.join(texto)

        length = len(texto)

        self.MFOutput.appendPlainText(texto)

        linha = []
        for i in range(length):
            if i < 100:
                if i < len(str(k)):
                    linha.append(" ")
                else:
                    linha.append("-")

        linha = ''.join(linha)

        self.MFOutput.appendPlainText(linha)

        tam2 = p.shape[0]

        texto = ["  "]

        for j in range(tam2):
            redondo = np.round(p[j], 7)
            if p[j] == 0:
                aux = "(z)"
            elif "j" in str(redondo):
                aux = auxtext + str(redondo)[1:-1] + "))"
            else:
                aux = auxtext + str(redondo) + "))"
            texto.append(aux)

        texto = ''.join(texto)

        self.MFOutput.appendPlainText(texto)

    else:
        texto = "Aplicar o método VRFT primeiro."
        self.MFOutput.setPlainText(texto)
```

A.4 Exemplo da biblioteca *pyvrft*

Código utilizado para executar o exemplo disponibilizado junto à biblioteca *pyvrft*.

```
# -*- coding: utf-8 -*-
"""
Created on Thu Mar 7 14:37:39 2019
@authors: Diego Eckhard and Emerson Boeira
"""
"""
Testing the vrft on a SISO example
"""
#%% Header: importing python libraries

import numpy as np  # important package for scientific computing
from scipy import signal  # signal processing library
import matplotlib.pyplot as plt  # library to plot graphics
import vrft  # vrft package

#%% Simulating the open loop system to obtain the data for the VRFT

# declaration of the SISO transfer fuction of the process G(z)
G = signal.TransferFunction([1], [1, -0.9], dt=1)
# IMPORTANT: if the numerator of the transfer function is 1, for
    example, define it as num=[1], instead of num=[0,1]
# num=[0,1] produces a warning!

# number of samples
N = 100

# step signal
u = np.ones((N, 1))
u[0] = 0
# IMPORTANT: in our package, we decided to organize the input and
    output signals as a matrix (N,n)
# N=number of data samples, n=number of inputs and outputs

# calculating the output of the system
yu = vrft.filter(G, u)

# add noise to the output
# variance of the whie noise signal
sigma2_e1 = 0.1
# creating noise vector
w = np.random.normal(0, np.sqrt(sigma2_e1), N)
# pushing the dimensions to match our signals
```

```
41 w.shape = (N, 1)
42
43 # real (measured) output
44 y = yu + w
45
46 u2 = []
47 y2 = []
48
49 for x in range(N):
50     u2.append(u[x][0])
51     y2.append(y[x][0])
52
53 data = {'Input': u2, 'Output': y2}
54
55 df = pd.DataFrame(data)
56
57 df.to_excel('data.xlsx')
58
59 #%% Graphics
60
61 lw=1.5 # linewidth
62
63 # plot input signal
64 plt.figure()
65 plt.plot(u, "b", drawstyle="steps", linewidth=lw, label="u(t)")
66 plt.grid(True)
67 plt.xlabel("time (samples)")
68 plt.ylabel("u(t)")
69 plt.xlim(left=-2, right=N)
70 plt.show()
71
72 # plot output signal
73 plt.figure()
74 plt.plot(y, "b", drawstyle="steps", linewidth=lw, label="u(t)")
75 plt.grid(True)
76 plt.xlabel("time (samples)")
77 plt.ylabel("y(t)")
78 plt.xlim(left=-2, right=N)
79 plt.show()
80
81 #%% Control - VRFT parameters: reference model Td(z), filter L(z), and
       controller structure
82
83 # declaration of the transfer fuction of the reference model Td(z)
84 Td = signal.TransferFunction([0.2], [1, -0.8], dt=1)
85
86 # choosing the VRFT method filter
```

```
87 L = signal.TransferFunction([0.25], [1, -0.75], dt=1)
88
89 # defining the controller structure that will be used in the method
90 C = [
91     [signal.TransferFunction([1], [1], dt=1)],
92     [signal.TransferFunction([1, 0], [1, -1], dt=1)],
93 ]  # PI controller structure
94
95 #%% Design the controller using the VRFT method
96
97 # VRFT with least squares
98 p = vrft.design(u, y, y, Td, C, L)
99 print("p=", p)
```