

Ana Paula Flores de Melo

Contribuições a uma plataforma web para projetos de sistemas de controle lineares

Campina Grande, Brasil

22 de outubro de 2021

Ana Paula Flores de Melo

Contribuições a uma plataforma web para projetos de sistemas de controle lineares

Relatório de trabalho de conclusão de curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharelado em Ciências no Domínio da Engenharia Elétrica.

Universidade Federal de Campina Grande - UFCG
Centro de Engenharia Elétrica e Informática - CEEI
Departamento de Engenharia Elétrica - DEE

Orientador: Saulo Oliveira Dornellas Luiz, D. Sc.

Campina Grande, Brasil

22 de outubro de 2021

Ana Paula Flores de Melo

Contribuições a uma plataforma web para projetos de sistemas de controle lineares

Relatório de trabalho de conclusão de curso submetido à Unidade Acadêmica de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharelado em Ciências no Domínio da Engenharia Elétrica.

Trabalho aprovado em: / /

Saulo Oliveira Dornellas Luiz, D. Sc.
Orientador

João Batista Moraes dos Santos, D. Sc.
Convidado

Campina Grande, Brasil
22 de outubro de 2021

Agradecimentos

Agradeço aos meus pais por terem me criado e por sempre cuidarem de mim. A minha mãe Maria Leodilia por me prover risadas, conforto, dengos e ao meu pai Paulo por me incentivar desde criança a brincar com jogos de lógica, enigmas, quebra-cabeças, além de ter deixado um pedaço dele presente em mim para sempre. Aos dois por terem me dado a oportunidade de crescimento pessoal e profissional mesmo que isso seja longe de casa.

Agradeço a minha irmã Ana Carolina por todo o apoio que sempre tem me dado. Além de sempre me ouvir e me distrair, fazendo com que eu queira experimentar diversos hobbies diferentes.

Agradeço ao meu namorado Pedro por sempre me apoiar, ajudar, aconselhar, distrair, animar e brigar comigo quando necessário, além de sempre estar presente em todos os momentos. Também agradeço a família dele por ter me acolhido.

Agradeço a Sara que topou a loucura de ir fazer Engenharia Elétrica comigo em Campina Grande, PB, onde não conhecíamos ninguém, e foi o lugar que conquistou nossos corações. Também agradeço por ela sempre estar ao meu lado e pelos momentos de alegria e de frustrações compartilhados. Agradeço também a Túlio por ter entrado na nossa vida e me dar a minha afilhada que tanto amo.

Agradeço a todos os meus amigos pelas risadas, conselhos, ajudas, pois sem a força que eles me deram eu não teria conseguido chegar nessa etapa da minha vida. Em especial aos que fazem parte dos grupos: BBB; Caminho p sucesso; Fofoca; e, Não Cola, obrigada por me acompanharem nessa jornada.

Agradeço ao meu orientador Saulo por todas as reuniões, pelos auxílios e por sempre estar disponível para sanar as minhas dúvidas.

Resumo

Os engenheiros procuram auxílio de ferramentas computacionais para facilitar e agilizar trabalhos. Além disso, essas ferramentas tornam possível o desenvolvimento de projetos que, de outra forma, não seriam implementados. Contudo, deve-se considerar a acessibilidade em relação às plataformas existentes. Muitas são caras, computacionalmente custosas ou simplesmente de difícil utilização. O trabalho proposto tem como objetivo adicionar contribuições à plataforma *web*, desenvolvida por Ximenes (2021), no contexto de projetos de sistemas de controle. O desenvolvimento foi realizado em três etapas: 1) *back end*, utilizando Python para incrementar novas funcionalidades; 2) *front end*, com Figma, JavaScript e React para melhorar a interface do usuário; e 3) testes de integração, feitos com o framework Cypress em JavaScript. As contribuições foram realizadas para corrigir as falhas da ferramenta desenvolvida por Ximenes (2021), além de acrescentar novas funcionalidades. Desse modo, ela é capaz de fornecer ao usuário informações úteis para projetos de sistemas de controle: resposta ao degrau do processo e do sistema de controle em malha fechada; verificação dos parâmetros do sistema em malha fechada em relação às especificações do projetista; diagramas de Bode e lugar das raízes. Foi apresentada a fundamentação teórica tanto dos conceitos de Controle Analógico quanto das tecnologias utilizadas e foi descrito todo o processo de desenvolvimento da plataforma. Também, foram apresentadas discussões sobre o resultado do projeto. Por fim, percebe-se que o trabalho atingiu o objetivo de melhorar a experiência de aprendizado do aluno na plataforma *web* para análise e projetos de sistemas de controle.

Palavras-chave: ferramentas computacionais, plataforma *web*, sistemas de controle, desenvolvimento *web*

Abstract

Engineers use computational tools to facilitate and streamline their work. Furthermore, these tools make possible the development of projects that otherwise would not be developed. However, accessibility should be considered in relation to existing platforms. Some are expensive, hardware consuming, or simply difficult to use. The work presented aims to add contributions to the web platform developed by Ximenes (2021), in the Control Systems context. It was developed in three phases: 1) back end, using Python to add new features; 2) front end, with Figma, JavaScript and React to improve the user interface experience; and 3) integration tests, built with Cypress framework in JavaScript. Contributions were made to correct the flaws of the tool developed by Ximenes (2021), and new features were added. Thus, it is capable of returning important information for control projects: Open-Loop Process and Closed-Loop System Step Response; Closed-Loop System parameters verification against the designer's specifications; Bode Diagrams and Root Locus. The theoretical foundation of both the Analog Control concepts and the technologies used was presented, and the entire platform development process was described. Also, discussions about the project results were presented. Finally, it is clear that the work has improved the student's learning experience on the control systems analysis and design by means of the web platform.

Key-words: computational tools, web platform, control system, web development

Lista de ilustrações

Figura 1 – Diagrama de blocos de um sistema de controle em malha fechada. . . .	12
Figura 2 – Diagrama de blocos simplificado.	12
Figura 3 – Resposta ao degrau sub-amortecida de um sistema de segunda ordem. .	14
Figura 4 – Protótipo do projeto <i>Achilles Control</i> utilizando a ferramenta <i>Figma</i> . .	20
Figura 5 – Relação entre ω_n , ζ e o ponto do lugar das raízes	29
Figura 6 – Resposta ao Degrau do Sistema em Malha Fechada gerada pela plata- forma.	36
Figura 7 – Diagramas de Bode gerados pela plataforma.	37
Figura 8 – Lugar das raízes gerado pela plataforma.	38
Figura 9 – Funções de Transferência geradas pela plataforma.	39
Figura 10 – Comparação gráfica entre as especificações temporais e a resposta ao degrau em malha fechada.	40
Figura 11 – Verificação dos parâmetros da resposta ao degrau em malha fechada. .	41
Figura 12 – Botão estilizado por meio de <i>styled-components</i>	42
Figura 13 – Página "Sobre" do site <i>Achilles Control</i>	44
Figura 14 – Página "Analysis" do site <i>Achilles Control</i> , parte 1.	45
Figura 15 – Página "Analysis" do site <i>Achilles Control</i> , parte 2.	46
Figura 16 – Página "Help" do site <i>Achilles Control</i>	47
Figura 17 – Página "Home" do site <i>Achilles Control</i>	48
Figura 18 – Página "Inputs" do site <i>Achilles Control</i>	49
Figura 19 – Página "Specifications" do site <i>Achilles Control</i>	52
Figura 20 – Resposta ao Degrau do sistema de controle em malha fechada gerado por Ximenes (2021).	60
Figura 21 – Resposta ao Degrau do sistema de controle em malha fechada gerado pelo <i>Achilles Control</i>	60
Figura 22 – Verificação dos parâmetros do sistema de controle gerado pelo <i>Achilles Control</i>	61
Figura 23 – Diagrama de Bode gerado por Ximenes (2021).	62
Figura 24 – Diagrama de Bode gerado pelo <i>Achilles Control</i>	63
Figura 25 – Lugar das Raízes gerado por Ximenes (2021).	64
Figura 26 – Lugar das Raízes gerado pelo <i>Achilles Control</i>	64
Figura 27 – Informações adicionais no Lugar das Raízes gerado pelo <i>Achilles Control</i> . .	65

Lista de abreviaturas e siglas

API	<i>Application Programming Interface</i>
CSS	<i>Cascading Style Sheets</i>
HTTP	<i>Hypertext Transfer Protocol</i>
JSON	<i>JavaScript Object Notation</i>
UI	<i>User Interface</i>
URL	<i>Uniform Resource Locator</i>

Sumário

1	INTRODUÇÃO	10
2	FUNDAMENTAÇÃO TEÓRICA	11
2.1	Sistemas de Controle	11
2.1.1	Especificações no domínio do tempo	13
2.1.1.1	Variação permitida do regime estacionário	14
2.1.1.2	Sobre-sinal	14
2.1.1.3	Tempo de subida	15
2.1.1.4	Tempo de pico	15
2.1.1.5	Tempo de acomodação	15
2.1.2	Lugar das Raízes	16
2.1.2.1	Projeto do controlador	16
2.1.3	Diagramas de Bode	17
2.1.3.1	Projeto do controlador	17
2.1.4	Teorema do Valor Final	18
2.2	Tecnologias Utilizadas	18
2.2.1	Linguagens de Programação	19
2.2.2	UI Design	19
2.2.3	React	20
2.2.4	Testes Unitários	22
2.2.5	Testes de Integração	24
3	DESENVOLVIMENTO	26
3.1	Servidor	26
3.1.1	Funções modificadas	26
3.1.2	Funções novas	30
3.2	Cliente	32
3.2.1	<i>App.js</i> e <i>index.css</i>	33
3.2.2	Diretório <i>components</i>	35
3.2.3	Diretórios: <i>config</i> e <i>imgs-ac</i>	43
3.2.4	Diretório <i>screens</i>	43
3.2.5	Diretório <i>services</i>	52
3.2.6	Testes de Integração	53
4	RESULTADOS E DISCUSSÕES	59
4.1	Comparação entre as plataformas	59

4.2	Resultados e Melhorias Futuras	66
5	CONSIDERAÇÕES FINAIS	67
	REFERÊNCIAS	68

1 Introdução

Cada vez mais os engenheiros procuram auxílio de ferramentas computacionais para facilitar e agilizar o trabalho, uma vez que o tempo para realizar cálculos e simulações pode ser diminuído drasticamente ao utilizar a ferramenta adequada para o objetivo específico. Atualmente, existem diversas aplicações no âmbito computacional com funcionalidades características, além de um alto crescimento no desenvolvimento de novas. Entretanto, deve-se considerar a acessibilidade em relação a essas plataformas existentes. Muitas são caras, computacionalmente custosas ou simplesmente de difícil utilização.

Ximenes (2021), desenvolveu uma plataforma *web* gratuita e de fácil utilização para auxiliar projetos de controle analógico. Nela, o usuário insere as funções de transferência do processo e do controlador e recebe os gráficos de resposta ao degrau para malha aberta e fechada, o lugar das raízes e o diagrama de Bode. O intuito da aplicação é ser uma alternativa ao Control System Designer (MathWorks, 2021), que pode ser encontrada como *Toolbox* no *Matlab*, pois este é um *software* financeiramente caro, que exige poder computacional e que pode ser de difícil utilização para muitas pessoas.

No entanto, há problemas a serem corrigidos e melhorias a serem implementadas na plataforma desenvolvida por Ximenes (2021) antes de disponibilizá-la para testes com o público alvo. Desse modo, a proposta desse trabalho é adicionar contribuições à plataforma, para melhorar a experiência do usuário, além de incrementar novas funcionalidades ao *software*.

O trabalho está estruturado em 5 capítulos, incluindo este introdutório, conforme será descrito a seguir. Nesse capítulo foi apresentada uma breve introdução e os objetivos do trabalho de conclusão de curso, bem como a estrutura de organização do relatório. No capítulo 2 serão apresentados os principais conceitos da Teoria de Controle Analógico e as tecnologias utilizadas para o desenvolvimento do projeto. No capítulo 3 serão apresentadas as etapas de desenvolvimento realizadas para melhorar a plataforma *web* desenvolvida por Ximenes (2021). No capítulo 4 serão expostos os resultados finais do projeto, além das possíveis futuras melhorias. Por fim, no capítulo 5, é apresentada a conclusão sobre o trabalho.

2 Fundamentação Teórica

Neste capítulo são apresentados os princípios da Teoria de Controle Analógico necessários para o desenvolvimento do projeto em questão, além das tecnologias utilizadas para o desenvolvimento da plataforma web.

2.1 Sistemas de Controle

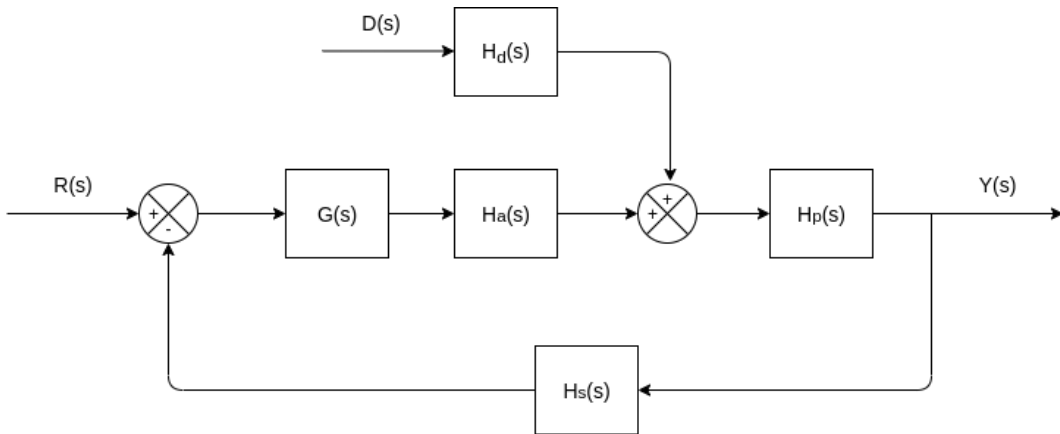
Um sistema de controle pode possuir diversos subsistemas que serão utilizados para fazer com que a saída do processo permaneça igual à referência escolhida, por exemplo, o ar condicionado possui um controlador que regula a temperatura de saída de acordo com a temperatura de referência indicada pelo usuário.

Existem dois tipos de sistema de controle: malha aberta e malha fechada. Em malha aberta, o controlador age de maneira independente da saída do sistema, pois não há retroalimentação. Já em malha fechada, o sinal de entrada do controlador sofre influência do erro entre o sinal de saída e o sinal de referência, por causa da retroalimentação.

A representação gráfica comumente utilizada para análise de sistemas de controle é o diagrama de blocos. Neste trabalho, os sinais e os sistemas serão representados no domínio da frequência, por meio da transformada de Laplace. Na Fig. 1 é apresentado o diagrama de um sistema de controle em malha fechada, composto pelos sinais de entrada $R(s)$, de perturbação $D(s)$ e de saída $Y(s)$, além das funções de transferência dos componentes:

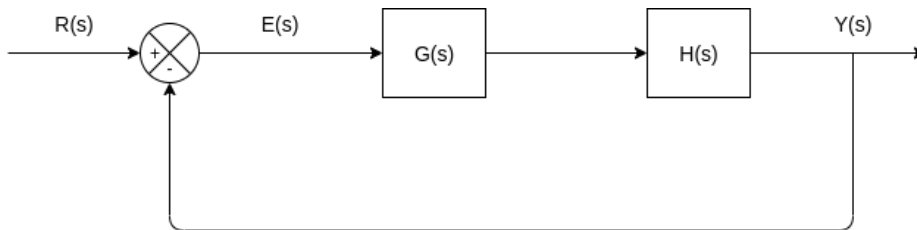
- $G(s)$: controlador
- $H_a(s)$: atuador
- $H_p(s)$: processo a ser controlado
- $H_s(s)$: sensor
- $H_d(s)$: perturbação

Figura 1 – Diagrama de blocos de um sistema de controle em malha fechada.



No entanto, nesse trabalho não será considerada a influência de perturbações externas, logo, é possível considerar $D(s) = 0$ no diagrama de blocos. Ainda é possível simplificar mais o diagrama, ao considerar que a associação entre planta e atuador é reduzida a um único processo ($H_p(s)H_a(s) = H(s)$). Também foi considerado que o sensor não apresenta influência no sinal de retroalimentação, ou seja, $H_s(s) = 1$. Dessa maneira, o diagrama resultante é apresentado na Fig. 2, em que $E(s)$ representa o sinal de erro.

Figura 2 – Diagrama de blocos simplificado.



Ao utilizar o diagrama simplificado, é possível montar uma função de transferência de malha fechada, ou seja, a relação entre a transformada de Laplace da variável de saída e a transformada de Laplace da variável de entrada, com todas as condições iniciais supostas iguais a zero (BISHOP, DORF, 2009). Dessa forma, é possível desenvolver a equação a partir da relação entre os três sinais apresentados:

$$E(s) = R(s) - Y(s) \tag{2.1}$$

$$Y(s) = E(s)G(s)H(s)$$

$$E(s) = \frac{Y(s)}{G(s)H(s)} \tag{2.2}$$

Substituindo (2.2) em (2.1):

$$R(s) - Y(s) = \frac{Y(s)}{G(s)H(s)}$$

$$\frac{Y(s)}{R(s)} = \frac{G(s)H(s)}{1 + G(s)H(s)} \quad (2.3)$$

Portanto, a função de transferência de malha fechada do diagrama genérico simplificado apresentado pode ser representada como mostrado em (2.3).

Após obter o modelo matemático do sistema de controle, é possível analisar o desempenho do sistema a partir dos vários métodos disponíveis (OGATA, 2010) que serão explorados nas seções seguintes.

2.1.1 Especificações no domínio do tempo

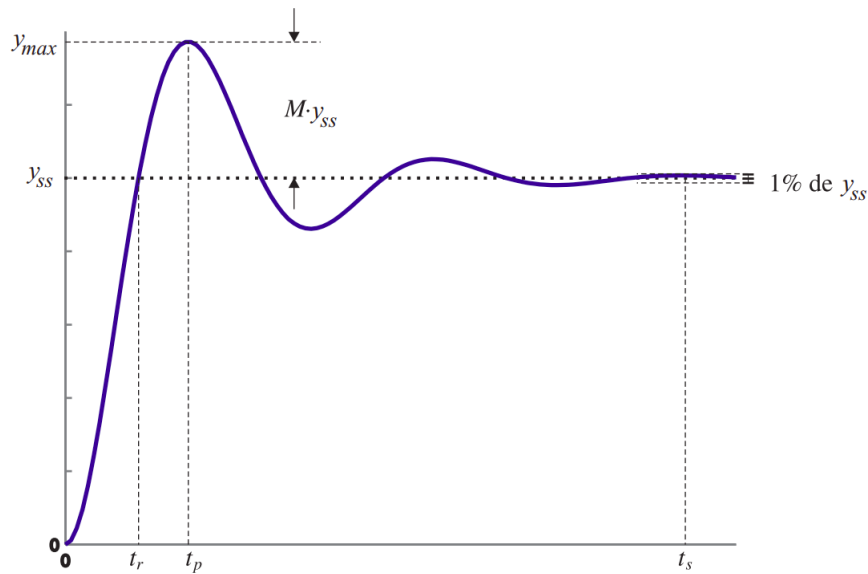
A resposta ao degrau é um gráfico que serve para analisar o comportamento do sistema de controle ao decorrer do tempo. A partir dela, é possível analisar o desempenho em que o sistema atua. Neste trabalho, será utilizada a resposta ao degrau de um sistema padrão de segunda ordem, em que $G_{MF}(s)$ é a função de transferência da malha fechada (2.4).

$$G_{MF}(s) = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2} \quad (2.4)$$

Para projetar um controlador para sistemas lineares, é preciso definir as especificações do sistema de controle para que este atue com o desempenho esperado. É desejável que o controlador produza uma resposta rápida com a maior precisão possível. Na Fig. 3 é apresentada a resposta ao degrau sub-amortecida do sistema de segunda ordem (2.4) com os índices de desempenho indicados:

- y_{ss} : valor de referência;
- y_{max} : valor de pico máximo;
- M : sobre-sinal;
- t_r : tempo de subida;
- t_p : tempo de pico;
- t_s : tempo de acomodação.

Figura 3 – Resposta ao degrau sub-amortecida de um sistema de segunda ordem.



Também, sabe-se que a dinâmica do sistema pode ser especificada por meio da frequência natural ω_n e do coeficiente de amortecimento ζ . Desse modo, como o sistema possui comportamento sub-amortecido, o coeficiente de amortecimento está contido no intervalo: $0 < \zeta < 1$.

Então, será verificado se a resposta ao degrau satisfaz as especificações requisitadas pelo projetista, que serão exploradas nas seções seguintes.

2.1.1.1 Variação permitida do regime estacionário

O projetista decide qual é a variação máxima que o sinal de saída y pode ter em relação ao valor de referência y_{ss} para que seja possível considerar que o sistema atingiu o regime permanente. Na Fig. 3 a variação máxima permitida é de 1% de y_{ss} .

2.1.1.2 Sobre-sinal

O sobre-sinal M é definido como a porcentagem máxima em que a saída ultrapassa y_{ss} . Dessa maneira, o sobre-sinal máximo é calculado a partir do maior valor de pico atingido pelo sinal de saída y_{max} em relação a y_{ss} , como mostrado em (2.5).

$$M = \frac{y_{max} - y_{ss}}{y_{ss}} \times 100\% \quad (2.5)$$

O projetista indica o valor máximo que a resposta do sistema de controle poderá atingir acima do valor de referência y_{ss} ao especificar o sobre-sinal máximo M_{max} . Assim, para verificar se o controlador obedece a essa especificação, pode-se calcular o sobre-sinal

M do sistema por meio da equação (2.6).

$$M = \exp\left(-\frac{\zeta\pi}{\sqrt{1-\zeta^2}}\right) \quad (2.6)$$

Em (2.6) quanto maior for ζ , menor será o sobre-sinal atingido pela saída do sistema. Desse modo, caso $M > M_{max}$, o projetista do controlador pode utilizar um coeficiente de amortecimento tal que $\zeta > \zeta_{min}$, onde $M_{max} = \exp(-\zeta|min\pi/...)$, para garantir que $M < M_{max}$.

2.1.1.3 Tempo de subida

O tempo de subida t_r é o intervalo de tempo em que a resposta do sistema se torna pela primeira vez igual ao valor de referência y_{ss} . É desejável que esse valor seja o menor possível para que a saída do sistema de controle chegue rapidamente na referência estabelecida. Considera-se neste trabalho um tempo de subida máximo t_{rmax} . Para verificar se t_r obedece à especificação do projetista $t_r < t_{rmax}$, é possível calcular o tempo de subida pela aproximação apresentada em (2.7).

$$t_r \approx \frac{1.8}{\omega_n} \quad (2.7)$$

Desse modo, caso $t_r > t_{rmax}$, deve-se aumentar a frequência natural do sistema ω_n para garantir que o sistema obedeça a especificação previamente projetada.

2.1.1.4 Tempo de pico

O tempo de pico t_p é o tempo em que ocorre o valor máximo y_{max} . Considera-se neste trabalho um tempo de pico máximo t_{pmax} . Para garantir que obedeça à especificação do projetista $t_p < t_{pmax}$, ele pode ser calculado por (2.8), em que $\omega_d = \omega_n\sqrt{1-\zeta^2}$. Vale ressaltar que a frequência amortecida ω_d corresponde à parte imaginária do polo de malha fechada do sistema.

$$t_p = \frac{\pi}{\omega_d} \quad (2.8)$$

Desse modo, caso $t_p > t_{pmax}$, deve-se aumentar a frequência natural do sistema ω_n ou reduzir o coeficiente de amortecimento ζ para garantir que o sistema obedeça à especificação.

2.1.1.5 Tempo de acomodação

O tempo de acomodação t_s é definido como o intervalo de tempo em que a resposta do sistema precisa para sair do tempo zero t_0 e atingir o valor de referência y_{ss} variando entre a variação máxima permitida estabelecida pelo projetista. Quando a saída atinge esse ponto, é considerado que o sistema está em regime permanente. Para garantir que $t_s < t_{smax}$, em que t_{smax} é o tempo de acomodação máximo definido pelo projetista,

pode-se calcular t_s por (2.9), em que $\sigma = \zeta\omega_n$. Vale ressaltar que $-\sigma$ corresponde a parte real do polo de malha fechada do sistema, ou seja, é a magnitude da parte real do polo.

$$t_s = \frac{4.6}{\sigma} \quad (2.9)$$

Desse modo, caso $t_s > t_{smax}$, deve-se aumentar a frequência natural do sistema ω_n ou o coeficiente de amortecimento ζ para garantir que o sistema obedeça a especificação previamente projetada.

2.1.2 Lugar das Raízes

Uma das análises realizadas no trabalho se trata do lugar das raízes, técnica específica que mostra como mudanças em um dos parâmetros do sistema irá modificar as raízes da equação característica, ou seja, os polos da malha fechada. O lugar das raízes é usado normalmente para o estudo do efeito da variação de um ganho de malha aberta (FRANKLIN, et al., 2013).

Utilizando (2.3), pode-se definir sua equação característica como:

$$1 + G(s)H(s) = 0 \quad (2.10)$$

Segundo Franklin, assume-se que a equação característica pode ser definida por meio de componentes polinomiais $a(s)$ e $b(s)$ e, assim, defini-se a função de transferência $L(s) = \frac{b(s)}{a(s)}$. Desse modo (2.10) pode ser reescrita como (2.11).

$$1 + KL(s) = 0 \quad (2.11)$$

O lugar das raízes é, portanto, o gráfico de todas as possíveis raízes da equação (2.11) relativas ao parâmetro K (FRANKLIN, et al., 2013).

2.1.2.1 Projeto do controlador

É possível projetar um controlador utilizando o lugar das raízes, uma vez que ele é útil para alocar os polos dominantes de malha fechada de acordo com as especificações do sistema definidas pelo projetista. Sabe-se que é possível determinar os polos de acordo com o coeficiente de amortecimento ζ e com a frequência natural ω_n . Desse modo, o polo dominante desejado s_0 que garante que o sistema obedeça as especificações é definido em (2.12).

$$s_0 = -\zeta\omega_n + j\omega_n\sqrt{1 - \zeta^2} \quad (2.12)$$

Deve-se verificar se s_0 obedece a condição de ângulo do lugar das raízes definida em (2.13), em que o ângulo de $G(s)H(s)$ é sempre um múltiplo ímpar de $\pm 180^\circ$. Para

projetar o controlador, é necessário manipular (2.13) para isolar $\arg(G(s_0))$, pois ele é o responsável por alocar o polo desejado no lugar das raízes.

$$\arg(G(s_0)H(s_0)) = \pm 180^\circ(2l + 1), l \in \mathbb{N} \quad (2.13)$$

$$\arg(G(s_0)) + \arg(H(s_0)) = \pm 180^\circ(2l + 1)$$

$$\arg(G(s_0)) = -\arg(H(s_0)) \pm 180^\circ(2l + 1) \quad (2.14)$$

Desse modo, com o resultado de (2.14), deve-se analisar o ângulo que falta para que a condição de fase seja satisfeita e projetar o controlador para que seja possível incluir o polo desejado s_0 no lugar das raízes do sistema de controle.

2.1.3 Diagramas de Bode

Até agora, foi realizada a análise do sistema de controle no domínio do tempo, mas também é possível analisá-lo no domínio da frequência. Dessa maneira, o sistema será analisado a partir da resposta em frequência: resposta em regime permanente de um sistema para uma entrada senoidal.

Nos métodos de resposta em frequência, a frequência do sinal de entrada é variada dentro de certo intervalo e a resposta resultante é estudada. No projeto de um sistema de malha fechada, as características da resposta em frequência da malha aberta são ajustadas para que sejam obtidas características aceitáveis da resposta transitória do sistema em malha fechada (OGATA, 2010).

Assim, é possível construir os diagramas de Bode que, segundo Ogata, são constituídos por dois gráficos: o gráfico em decibéis do módulo de uma função de transferência senoidal e o gráfico do ângulo de fase. O eixo da frequência é traçado em escala logarítmica.

2.1.3.1 Projeto do controlador

Existem dois tipos de especificações utilizadas para analisar a resposta do sistema por meio do diagrama de Bode: a margem de fase (PM) e a margem de ganho (GM). Enquanto a margem de fase é responsável por garantir que a resposta temporal do sistema possua as características temporais esperadas pelo projetista, a margem de ganho corresponde ao valor do ganho de malha aberta para o qual o sistema de controle em malha fechada é marginalmente estável. Desse modo, para projetar um controlador, deve-se garantir a margem de fase desejada (PM) na frequência de cruzamento de ganho especificada (ω_{PM}).

Em (2.15) é mostrada a equação para projetar o controlador no domínio da frequência, em que $\omega = \omega_{PM}$, na qual $PM > 0$.

$$G(j\omega_{PM})H(j\omega_{PM}) = e^{j(-180^\circ + PM)} \quad (2.15)$$

A escolha da margem de fase no projeto está relacionada com o tempo de acomodação do sistema (t_s) estabelecido pelo projetista. Em (2.16) é apresentada a relação utilizando a frequência da margem de fase (ω_{PM}). Já em (2.17) é mostrada a relação entre a margem de fase e o coeficiente de amortecimento (ζ).

$$\tan(PM) = \frac{8}{t_s \omega_{PM}} \quad (2.16)$$

$$PM \approx 100\zeta, \text{ para } 0^\circ \leq PM \leq 60^\circ \quad (2.17)$$

2.1.4 Teorema do Valor Final

Uma das medidas de desempenho de um sistema de controle, é analisar se o sistema segue o sinal de referência (y_{ss}) em regime permanente. A ferramenta matemática utilizada para essa análise é o Teorema do Valor Final. Nesse trabalho, considere que a função de transferência de malha fechada é do tipo $\frac{N(s)}{D(s)}$, em que $N(s)$ e $D(s)$ são polinômios. O valor em regime permanente da resposta do sistema de controle, sendo a referência igual ao degrau unitário, é dado em (2.18).

$$\lim_{t \rightarrow \infty} y(t) = \lim_{s \rightarrow 0} s \frac{N(s)}{D(s)} \frac{1}{s} = \lim_{s \rightarrow 0} \frac{N(s)}{D(s)} = \frac{N(0)}{D(0)} \quad (2.18)$$

Sendo $N(s) = b_m s^m + \dots + b_1 s + b_0$ e $D(s) = a_n s^n + \dots + a_1 s + a_0$ polinômios, e $n \geq m$. Portanto, é possível calcular o valor final da resposta ao degrau desse sistema de controle por meio da divisão entre os coeficientes constantes do numerador e do denominador (2.19).

$$\lim_{t \rightarrow \infty} y(t) = \frac{b_0}{a_0} \quad (2.19)$$

Na próxima seção serão apresentadas as tecnologias usadas no desenvolvimento de uma plataforma web para análise e projeto de sistemas de controle.

2.2 Tecnologias Utilizadas

Este trabalho consistiu em aprimorar uma aplicação *web* aberta ao público geral desenvolvida por Ximenes (2021) capaz de gerar gráficos e informações relevantes ao

projeto de um sistema de controle. A aplicação possui *back end* desenvolvido em Python com auxílio do *framework Flask* que é encarregado do servidor e das suas requisições compatíveis. Desse modo, foi utilizada essa linguagem para adicionar funcionalidades ao sistema.

A plataforma web possui um *front end*, que se comunica com o servidor em uma relação cliente-servidor. O *front end* é a interface entre usuário e servidor que permite uma utilização do *software* de forma simples. Foi desenvolvido uma nova interface para melhorar a experiência do usuário utilizando os conceitos de *UI Design*. Desse modo, para o design da interface foi utilizada a ferramenta *Figma*¹, e para implementação do *front end* foi utilizado *JavaScript* com auxílio do *framework React*.

Para garantir que a lógica envolvida no processo de receber a entrada do usuário e transformar essa informação em dados para o projeto do sistema de controle atue corretamente, foram implementados dois tipos de testes na aplicação. No lado do servidor, foram implementados testes unitários, em que cada função utilizada no *back end* é testada individualmente. Já no lado do cliente, foram realizados testes de integração para garantir que as duas aplicações: *back end* e *front end* funcionem em conjunto. Para implementação dos testes de unidade, foi utilizada a *framework unittest*² em Python, enquanto para os testes de integração foi utilizado o *framework Cypress*³ em *JavaScript*.

Nessa seção, o foco será explicar cada tecnologia utilizada e o motivo de sua escolha no projeto.

2.2.1 Linguagens de Programação

Para o desenvolvimento desse trabalho foram utilizadas duas linguagens de programação: *Python* e *JavaScript*. A primeira foi utilizada no desenvolvimento do *back end*, escolhida por possuir o pacote *control*, necessário para geração dos gráficos obtidos e a segunda no desenvolvimento do *front end*.

2.2.2 UI Design

A sigla UI significa *User Interface*, em português, Interface do Usuário. A interface permite que o usuário interaja com a aplicação de forma simples. Desse modo, *UI Design* é uma área que consiste em criar interfaces mais fáceis e amigáveis, ao pensar na forma em que o usuário irá interagir com o *software*. O principal objetivo dessa área é guiar o usuário pela aplicação por meio das interfaces visuais, ou seja, existe um caminho previamente

¹ Ferramenta de UI Design. Figma. Disponível em: <<https://www.figma.com/br/ui-design-tool/>>. Acesso em: 02 ago. 2021

² unittest — Unit testing framework. Python Docs, c2001-2021. Disponível em: <<https://docs.python.org/3/library/unittest.html>>. Acesso em: 20 set. 2021

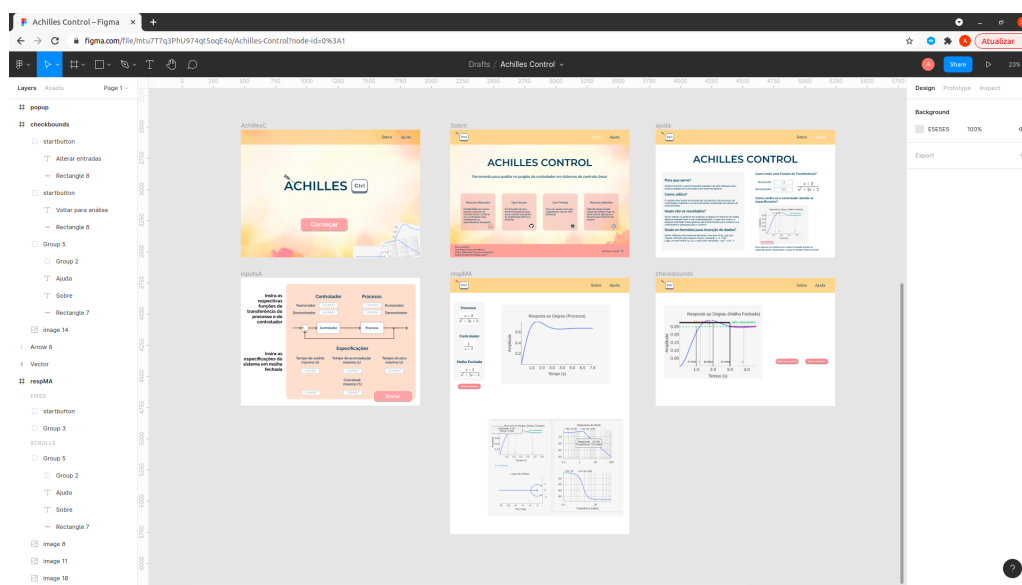
³ Why Cypress?. Cypress Docs, c2021. Disponível em: <<https://docs.cypress.io/guides/overview/why-cypress>>. Acesso em: 26 set. 2021

especificado no site que é desejável que o usuário siga e a missão do *UI Design* é conduzir os usuários pela navegação, levando-os a tomar ações de forma natural.

Para realizar o design da nova interface do usuário, foi utilizada a ferramenta online e gratuita *Figma*, que é um editor online de design gráfico e de prototipagem de projetos baseado no navegador web. A ferramenta é colaborativa e permite que uma equipe trabalhe simultaneamente no mesmo projeto.

Na Fig. 4 é apresentada a ferramenta *Figma* com o protótipo do projeto desenvolvido neste trabalho denominado *Achilles Control*.

Figura 4 – Protótipo do projeto *Achilles Control* utilizando a ferramenta *Figma*.



2.2.3 React

Sendo uma das principais ferramentas atuais para o desenvolvimento *web*, o *React* é uma biblioteca da linguagem de programação *JavaScript* para criação de interfaces de usuário em projetos *web*. Desenvolvida e mantida por engenheiros do *Facebook*, a biblioteca se tornou *open-source* em 2013 e vem crescendo cada vez mais ao longo dos anos, sendo bastante utilizada em diversas aplicações, desde simples interfaces para pequenos projetos até interfaces complexas de grandes empresas multinacionais como o próprio *Facebook*, *Instagram*, *Netflix* e *Microsoft* (em muitos dos seus recursos).

O *React* serve para realizar a criação da interface, ou seja, desenvolvimento dos componentes funcionais ou não, e telas. Além disso, existem os *Hooks* que são funções que facilitam a programação e a reutilização dos componentes funcionais, uma vez que permitem a utilização de recursos do *React* sem usar estrutura de classe. Os *Hooks* a seguir foram utilizados no projeto:

- ***useState***: controla uma variável de estado dentro de um componente funcional. Portanto, o *useState()* retorna tanto a variável de estado que é preservada pelo *React*, quanto a função *setVariavelDeEstado* que permite que a variável seja alterada.
- ***useEffect***: recebe uma função como primeiro parâmetro que é executada após inicialização e atualização do componente, já o segundo parâmetro é opcional e indica quando essa função deve ser executada, ou seja, ela só é executada se a variável de estado sofrer alteração em seu valor;
- ***useContext***: compartilha dados que podem ser considerados “globais” para a árvore de componentes do *React*.

Na parte da estilização dos componentes e das telas, foi utilizada a biblioteca *styled-components* que permite escrever CSS⁴ por meio de *JavaScript*. Por meio dessa funcionalidade, o código fica mais organizado e com melhor manutenção, pois existe um arquivo específico para estilização de cada componente. Isso também permite que o componente seja mais reutilizável.

Também existe o servidor no *front end* cuja finalidade é realizar requisições para o servidor *back end* em nome do cliente, assim como receber as respostas desse último servidor e montar uma nova resposta para o usuário. No caso, o *React* é utilizado para a criação da tela interativa e criação das rotas do *front end*. No projeto, a criação das rotas foi feita utilizando a biblioteca *react router dom* por meio dos componentes:

- ***BrowserRouter***: sincroniza as rotas no *browser* com a interface;
- ***Switch***: seleciona uma das rotas filhas de acordo com a aplicação;
- ***Route***: define o que será renderizado em cada rota.

O Trecho de código 2.1 apresenta um exemplo de utilização desses componentes da biblioteca *react router dom*.

Trecho de código 2.1 – Utilização dos componentes da biblioteca *react router dom*

```
1 import React from "react";
2 import { BrowserRouter, Route, Switch } from "react-router-dom";
3 import "./App.css";
4
5 import { Home, Inputs } from "./screens";
6
7 function App() {
8   return (
```

⁴ CSS Tutoriais. MDN Web Docs, c2005-2021. Disponível em: <<https://developer.mozilla.org/pt-BR/docs/Web/CSS>>. Acesso em: 02 ago. 2021

```
9     <BrowserRouter >
10         <Switch >
11             <Route component={Home} path="/" exact render />
12             <Route component={Inputs} path="/inputs" exact render />
13         </Switch >
14     </BrowserRouter >
15 );
16 }
17
18 export default App;
```

No exemplo acima, o componente **Route** recebe as propriedades *component* e *path*, em que o *path* indica o caminho da URL onde o componente será renderizado. Desse modo, o componente **Switch** é pai de todos os componentes **Route**, pois o **Switch** é responsável por selecionar qual rota será renderizada por meio da propriedade *children*. Por fim, o componente **BrowserRouter** sincroniza a rota escolhida pelo **Switch** com o *browser*.

Para transitar entre as rotas definidas dentro do escopo do **Switch**, foi utilizado o Hook **useHistory** da biblioteca *react router dom* que permite acesso à rota atual para que seja possível navegar para a próxima rota.

Além disso, quando a aplicação recebe a resposta da requisição enviada, é utilizado o *Victory*, um conjunto de módulos e componentes para *React*, para a criação dos gráficos apresentados.

2.2.4 Testes Unitários

Testes unitários são responsáveis por assegurar que uma unidade do sistema ou uma função se comporte da maneira esperada. No *software*, seu principal objetivo é verificar se uma determinada função continua funcionando corretamente mesmo que haja uma refatoração na aplicação, ou seja, os testes unitários possuem como objetivo avaliar as funções para garantir que estejam corretas. O ideal é que sejam escritos todos os cenários possíveis em relação ao que pode ocorrer dada uma determinada entrada na função para preparar o código para situações fora do "ótimo".

O Trecho de código 2.2 apresenta um exemplo de uma função que será testada por meio do *framework unittest* em Python, utilizado neste trabalho. Essa função serve para separar a parte real da parte imaginária de um vetor, e o papel do teste unitário é garantir a saída da função para toda entrada que esteja corretamente tipada.

Trecho de código 2.2 – Função que separa a parte real da parte imaginária de um vetor

```
1 def separateRealImag(array):
2     part_real = []
```

```
3     part_imag = []
4     for i in array:
5         part_imag.append(np.imag(i))
6         part_real.append(np.real(i))
7     return part_real, part_imag
```

O Trecho de código 2.3 apresenta um exemplo de teste unitário que pode ser escrito para garantir o funcionamento da função `separateRealImag(array)`.

Trecho de código 2.3 – Teste unitário para função `separateRealImag(array)` utilizando o *framework unittest* em Python

```
1 import unittest
2
3 class TestSeparateRealImag(unittest.TestCase):
4
5     def test_splitOk(self):
6         #given
7         array = [3+8j]
8
9         #when
10        part_real, part_imag = separateRealImag(array)
11
12        #then
13        self.assertEqual(part_real, [3])
14        self.assertEqual(part_imag, [8])
15
16 if __name__ == '__main__':
17     unittest.main()
```

No exemplo acima, primeiramente, foi necessário importar o *framework unittest* que fornece uma classe base para testes (`TestCase`), que pode ser utilizada para criar novos cenários de teste. Em seguida, foi criada a classe `TestSeparateRealImag` que é herdeira da classe `TestCase`, onde os testes foram realizados. Desse modo, cada método/função dessa classe corresponde a um cenário de teste diferente. O exemplo possui um cenário de teste "`splitOk`", que tem como objetivo validar se a função `separateRealImag(array)` retorna a parte real e a parte imaginária como o esperado. Esse teste é feito por meio da função assertiva "`assertEqual(resultado, valor_esperado)`" do *framework unittest*, que verifica se o resultado é igual ao valor esperado.

Os cenários de testes foram escritos de acordo com o padrão:

Scenario: especifica o que será testado, nesse caso é o nome do método;

Given: dada uma entrada para função;

When: quando ocorrer um evento;

Then: resultado esperado.

Vale ressaltar que é obrigatório que o nome dos métodos criados seja no formato "*test_NomeDoCenário*" para que a execução dos testes seja possível. Além disso, para que os testes sejam executados de forma simples, a função *unittest.main()* fornece uma interface de linha de comando para o *script* de teste. Logo, para executá-los, é só escrever no terminal o comando: "*python3 NomeDoArquivo.test.py*"

2.2.5 Testes de Integração

Os testes de integração servem para verificar os requisitos funcionais, de desempenho e de confiabilidade na modelagem do sistema. Com eles é possível descobrir erros de interface entre os componentes do sistema. Como era necessário fazer testes automatizados para testar várias entradas diferentes na aplicação, foi escolhido utilizar o *framework Cypress* em *JavaScript*.

O *Cypress* opera executando os testes automatizados no próprio navegador no mesmo ciclo de execução da aplicação. Isso faz com que ele tenha acesso tanto ao *back end* quanto ao *front end* do site. Ele também opera na camada de rede, onde lê e altera o tráfego da web em tempo real, o que permite que altere o código que possa interferir em sua capacidade de automatizar o navegador. Finalmente, como o *Cypress* controla todo o processo de automação dos testes do sistema, ele é capaz de fornecer resultados mais consistentes do que outra ferramenta de testes. Além disso, como ele é instalado localmente na máquina, isso lhe dá permissão de acesso ao próprio sistema operacional para realizar as tarefas de automação. Isso permite a execução de tarefas como tirar *screenshots* e gravar vídeos dos testes automatizados.

Para escrever os testes de integração por meio do *Cypress*, é necessário seguir regras. Logo, os testes são escritos dentro da pasta *cypress/integration* e o nome de cada arquivo de teste deve seguir o formato: "*NomeDoArquivo.spec.js*". Além disso, dentro de cada arquivo a primeira linha deve ser: */// <reference types="cypress"/>*, para que seja possível executar os arquivos de teste por meio dos comandos do *framework*.

Os cenários de teste são escritos de acordo com o padrão:

describe: especifica o que será testado;

beforeEach: antes de cada teste ser realizado, o *Cypress* executa as ações do que estiver dentro do escopo da função *beforeEach()*;

it: escopo onde cada teste é escrito;

data-cy: identificador do componente do *front end*;

fixtures: pasta que contém os arquivos de entrada para teste massivos.

O Trecho de código 2.4 mostra o código de um arquivo de teste escrito utilizando os conceitos do *Cypress* apresentados acima.

Trecho de código 2.4 – Teste de integração utilizando o *framework Cypress*

```
1  /// <reference types="cypress" />
2  import { apiURL } from "../src/config";
3
4  describe("test home endpoints", () => {
5    beforeEach(() => {
6      cy.visit(`${apiURL}/`);
7    });
8
9    it("press start button", () => {
10     cy.get("[data-cy=start]").click();
11     cy.location("pathname").should("include", "inputs");
12   });
13 });
```

No exemplo, é testado um *endpoint* da tela "home". Logo, antes que o teste seja inicializado, é necessário que o browser esteja localizado na tela principal. Então, dentro da função *beforeEach*, é comandado que o *Cypress* visite o *endpoint* em que a *home* esteja localizada. Dessa maneira, dentro do escopo da função *it* é testado o que ocorre quando é apertado o botão de *Começar* da interface. O *Cypress* identifica esse botão, por meio do *data-cy*. Logo, após apertar o botão, é esperado que o site redirecione o usuário para a página de *inputs*, em que será pedido para o usuário fornecer as entradas para a plataforma *Achilles Control*.

Por fim, existem dois modos de executar os testes. O primeiro é por meio da linha de comando "*npm run ./node_modules/.bin/cypress run*" para qualquer projeto; ou por meio do comando "*npm run cypress:run*" num projeto específico em que haja a definição do *script* de teste. Em ambos os casos, todos os testes são executados no terminal. Já o segundo é por meio da linha de comando "*npm run ./node_modules/.bin/cypress open*" para qualquer projeto; ou por meio do comando "*npm run cypress:open*" num projeto específico em que haja a definição do *script* de teste. Em ambos os casos, a interface do *Cypress* será aberta e, assim, é possível visualizar a execução de cada arquivo de teste separadamente no navegador.

No próximo capítulo será apresentado todo o processo de desenvolvimento do projeto. Cada etapa será detalhada e os trechos de código principais serão explicados.

3 Desenvolvimento

Nesse capítulo, é descrito o processo de desenvolvimento do projeto. Serão abordados detalhes sobre cada etapa (*back end* e *front end*), explicando as principais partes dos códigos e como os componentes se conectam para formar a solução proposta nesse trabalho. O código fonte do projeto está disponível em um repositório público no *Github* e pode ser encontrado no endereço eletrônico <https://github.com/PedroXimenes/project-achilles>.

3.1 Servidor

Como já explicado no capítulo anterior, o desenvolvimento do *back end* foi realizado por Ximenes (2021) por meio da linguagem de programação Python e com auxílio do micro *framework* Flask. A aplicação criada tem como objetivo determinar os resultados que serão mostrados ao usuário na interface gráfica. Para isso, são utilizados, principalmente, os seguintes pacotes do Python:

- control 0.8.4
- numpy 1.19.5
- scipy 1.5.4

Nessa parte do projeto, existem dois arquivos principais: o *app.py* e o *controller.py*. O primeiro é responsável por definir as rotas e o que cada rota retorna de acordo com a requisição recebida. O segundo tem por objetivo calcular e retornar as informações via construção de um objeto *json*. Nesse projeto, foram adicionadas contribuições somente ao arquivo *controller.py*. O arquivo *app.py* foi mantido igual ao de Ximenes (2021).

O arquivo *controller.py* apresenta oito funções principais e cinco auxiliares. Das funções principais, duas foram mantidas igual a Ximenes (2021): *stepResponse* e *bodeDiagram*; duas foram modificadas: *calculate* e *rootLocus*; e, quatro foram acrescentadas: *clSystem*, *stepInfo*, *finalValue* e *bodeInfo*. Nessa seção, serão apresentadas as funções principais modificadas ou novas do arquivo *controller.py*.

3.1.1 Funções modificadas

A função *calculate* (Trecho de código 3.1) é responsável por organizar tanto a chamada das funções quanto as informações resultantes em um objeto *json* e retorná-las para que a rota devolva uma resposta à requisição. Nesse *json* foram acrescentadas

informações referentes à malha fechada do sistema em relação aos gráficos de resposta ao degrau, diagrama de Bode e lugar das raízes que serão explicados mais adiante nessa mesma seção.

Trecho de código 3.1 – *calculate - controller.py*

```
1 def calculate(data=None):
2     hnum, hden, gnum, gden = separateSystem01(data)
3     clNum, clDen = clSystem(hnum, hden, gnum, gden)
4
5     h_t, h_y, g_t, g_y, series, MF = stepResponse(hnum, hden, gnum, gden
6         )
7     S = stepInfo(MF)
8     yss = finalValue(MF)
9
10    mag, phase, omega = bodeDiagram(series)
11    bode_info = bodeInfo(series)
12
13    real, imag, klist, wn, zeta = rootLocus(series)
14
15    poles = control.pole(series)
16    zeros = control.zero(series)
17    zero_real, zero_imag = separateRealImag(zeros)
18
19    num_poles = len(poles)
20    num_zeros = len(zeros)
21
22    fb_data = {
23        "x_axis_ol": h_t.tolist(),
24        "y_axis_ol": h_y.tolist(),
25        "hnum": hnum,
26        "hden": hden,
27        "gnum": gnum,
28        "gden": gden,
29        "x_axis_cl": g_t.tolist(),
30        "y_axis_cl": g_y.tolist(),
31        "omega": omega.tolist(),
32        "magnitude": mag.tolist(),
33        "phase": phase.tolist(),
34        "root_real": real.tolist(),
35        "root_imag": imag.tolist(),
36        "root_gain": klist.tolist(),
37        "num_poles": num_poles,
38        "num_zeros": num_zeros,
39        "zero_real": zero_real,
40        "zero_imag": zero_imag,
41        "step_info": S,
42        "bode_info": bode_info,
```

```

42         "yss": yss,
43         "cl_num": clNum,
44         "cl_den": clDen,
45         "wn": wn,
46         "zeta": zeta,
47     }
48     data = json.dumps(fb_data)
49
50     return data

```

Como pode ser visto na função acima, são chamadas duas funções auxiliares: *separateSystemOl* e *separateRealImag*. A primeira serve para separar a lista recebida no *body* da requisição POST, enviada pelo cliente, em numerador e denominador tanto do processo quanto do controlador, como pode ser visto no Trecho de código 3.2.

Trecho de código 3.2 – *separateSystemOl* - *controller.py*

```

1 def separateSystemOl(data):
2     hnum = list(map(float, data['hnum'].split(',')))
3     hden = list(map(float, data['hden'].split(',')))
4     gnum = list(map(float, data['gnum'].split(',')))
5     gden = list(map(float, data['gden'].split(',')))
6     return hnum, hden, gnum, gden

```

Já a segunda função auxiliar (Trecho de código 3.3) serve para separar a parte real da parte imaginária de um vetor. Nesse caso, ela foi utilizada para separar a parte real da parte imaginária dos zeros de malha aberta para que seja possível mapeá-los no lugar das raízes.

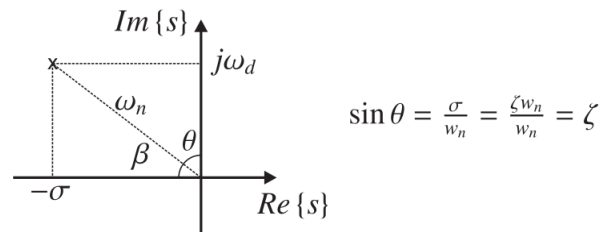
Trecho de código 3.3 – *separateRealImag* - *controller.py*

```

1 def separateRealImag(array):
2     part_real = []
3     part_imag = []
4     for i in array:
5         part_imag.append(np.imag(i))
6         part_real.append(np.real(i))
7     return part_real, part_imag

```

Por fim, a função *rootLocus* calcula os vetores contendo as partes real e imaginária dos pontos do Lugar das Raízes, além do ganho correspondente. Ainda, é necessário aplicar uma transformação nos dados para que sejam utilizados no *front end* de forma mais eficiente e, por isso, dois novos vetores são formados antes de serem retornados. Também, foram calculados o coeficiente de amortecimento (ζ) e a frequência natural (ω_n) de acordo com o apresentado na Fig. 5.

Figura 5 – Relação entre ω_n , ζ e o ponto do lugar das raízes

De acordo com a Fig. 5, pode-se deduzir que: $\omega_n = \sqrt{(-\sigma)^2 + \omega_d^2}$, em que $-\sigma$ corresponde à parte real do ponto no lugar das raízes e ω_d à parte imaginária. Também, é possível observar que $\zeta = \frac{\sigma}{\omega_n}$. Desse modo, o Trecho de código 3.4 apresenta a função *rootLocus* com o acréscimo do cálculo do coeficiente de amortecimento (ζ) e da frequência natural (ω_n).

Trecho de código 3.4 – *rootLocus - controller.py*

```

1 def rootLocus(series=None):
2     try:
3         rlist, klist = control.root_locus(series)
4
5         total, shape = rlist.shape
6         index = 0
7         j = 0
8         real = np.zeros((total, shape))
9         imag = np.zeros((total, shape))
10        wn = []
11        zeta = []
12        for x in rlist:
13            for i in x:
14                if j == shape:
15                    j = 0
16                    real[index][j] = i.real
17                    imag[index][j] = i.imag
18                    if(j == 0):
19                        w_n = math.sqrt(i.real**2 + i.imag**2)
20                        wn.append(np.round(w_n,3))
21                        zeta.append(np.round(-i.real/w_n,3))
22
23                    j = j + 1
24                    index = index + 1
25
26        return real, imag, np.round(klist,3), wn, zeta
27    except ValueError:
28        print('error:', ValueError)

```

3.1.2 Funções novas

Primeiramente, foi implementada a função *clSystem* que é responsável por calcular a função de transferência do sistema em malha fechada a partir das entradas inseridas pelo usuário na plataforma *Achilles Control*. Desse modo, o Trecho de código 3.5 utiliza as funções *polyadd* e *polymul* da biblioteca *numpy* para somar e multiplicar polinômios para que seja possível calcular o numerador e denominador da função de transferência do sistema em malha fechada. Por fim, é feita uma refatoração para que a resposta da função seja no formato desejável para ser recebida pelo *front end*.

Trecho de código 3.5 – *clSystem* - *controller.py*

```

1 def clSystem(hnum, hden, gnum, gden):
2     num = polymul(hnum, gnum)
3     den = polymul(hden, gden)
4     clNum = num
5     clDen = polyadd(num, den)
6     clNumstr = str(clNum).replace('.', ',').replace('.', ',').replace('[',
7         ',').replace(']', ',')
8     clDenstr = str(clDen).replace('.', ',').replace('.', ',').replace('[',
9         ',').replace(']', ',')
10
11     return clNumstr, clDenstr

```

Em seguida, foi implementada a função *stepInfo* (Trecho de código 3.6), responsável por calcular os parâmetros da resposta ao degrau do sistema em malha fechada abordados na Seção 2.1.1.: valor de pico, sobre-sinal, tempo de subida, tempo de pico e tempo de acomodação. Para isso, foi utilizada a função *step_info* do pacote *control*. Entretanto, para que seja possível calcular esses valores, o sistema em malha fechada precisa atingir o regime estacionário. Logo, foi necessário implementar a função *verifyPolesPositive*.

Trecho de código 3.6 – *stepInfo* - *controller.py*

```

1 def stepInfo(TF):
2     polesPos = verifyPolesPositive(TF)
3     yss = finalValue(TF)
4     if not polesPos:
5         return control.timeresp.step_info(TF, finalValue=yss,
6             RiseTimeLimits=(0,1))
7     return None

```

A função *verifyPolesPositive* (Trecho de código 3.7) serve para verificar a estabilidade do sistema, uma vez que polos da malha fechada localizados no semiplano direito causam instabilidade no sistema de controle. Já a função *finalValue* (explicada a seguir nessa mesma Seção) foi utilizada para a determinação da resposta ao degrau unitário em

regime permanente. Desse modo, essa função serve para configurar os limites relativos à resposta ao degrau para o cálculo do tempo de subida, uma vez que o tempo de subida corresponde ao primeiro tempo em que a resposta atinge o valor de referência.

Trecho de código 3.7 – *verifyPolesPositive* - *controller.py*

```

1 def verifyPolesPositive(series):
2     poles_MF = control.pole(series)
3     poles_real, _ = separateRealImag(poles_MF)
4     for i in poles_real:
5         if i > 0:
6             return True
7     return False

```

A função *finalValue* serve para encontrar qual o valor da resposta ao degrau em regime permanente por meio da função de transferência em malha fechada do sistema. Esse cálculo é baseado no teorema do valor final para a resposta ao degrau, que só é possível quando o sistema de controle é estável. Desse modo, foi utilizada a função *verifyPolesPositive* para verificar a estabilidade do sistema. Também, foi necessário implementar a função auxiliar *separateTF* que serve para separar o numerador e o denominador de uma função de transferência. Por fim, para calcular o valor final, foi realizada a divisão entre os coeficientes constantes dos polinômios no numerador e no denominador, como foi explicado na Seção 2.1.4. Os Trechos de código 3.8 e 3.9 são referentes a essas duas funções respectivamente.

Trecho de código 3.8 – *finalValue* - *controller.py*

```

1 def finalValue(TF):
2     polesPos = verifyPolesPositive(TF)
3     if not polesPos:
4         num, den = separateTF(TF)
5         return num[-1]/den[-1]
6     return None

```

Trecho de código 3.9 – *separateTF* - *controller.py*

```

1 def separateTF(series):
2     numerador = series.num[0][0]
3     denominador = series.den[0][0]
4     return numerador, denominador

```

Por fim, a última função acrescentada foi *bodeInfo* (Trecho de código 3.10), necessária para obter informações sobre o diagrama de Bode. Para isso, foi utilizada a função *margin* do pacote *control*. A partir da função de transferência em malha aberta, é possível obter os parâmetros: margem de ganho, margem de fase, frequência de cruzamento de ganho e frequência crítica. Desse modo, esses resultados foram inseridos dentro do objeto *bode_info* para serem enviados para o *front end*.

Trecho de código 3.10 – *bodeInfo* - *controller.py*

```
1 def bodeInfo(series):
2     gm, pm, wg, wp = control.margin(series)
3     bode_info = {
4         'gainMargin': str(round(gm,2)),
5         'phaseMargin': str(round(pm,2)),
6         'criticFreq': str(round(wg,2)),
7         'gainFreq': str(round(wp,2)),
8     }
9     return bode_info
```

3.2 Cliente

Como foi explicado no capítulo anterior, a plataforma web *Achilles Control* possui um *front end*, que é a interface entre usuário e servidor, e que permite uma utilização do *software* de forma simples. Para melhorar a experiência do usuário, foi desenvolvida uma nova interface gráfica. No desenvolvimento dessa interface, foi utilizada a técnica de gamificação¹ que consiste em inserir características que aparecem em jogos para motivar o aluno a utilizar a plataforma. Primeiramente, o protótipo foi feito por meio da ferramenta web *Figma*, como pode ser visto na Fig. 4. Em seguida, a implementação foi realizada por meio de *JavaScript* com o *framework React*.

No diretório principal, é possível encontrar a pasta *front*, em que tudo relacionado a essa etapa do desenvolvimento se encontra, tendo a estrutura de diretórios descrita a seguir:

- project-achilles
 - front
 - * cypress
 - * nginx
 - * node_modules
 - * public
 - * src

Desse modo, no diretório *cypress* há arquivos que são usados para os testes de integração realizados na aplicação, como descrito no capítulo anterior. Já o *nginx* serve para configurar o *Docker*, feito por Ximenes (2021). O *node_modules* é onde todas as dependências do projeto ficam instaladas. A pasta *public* é onde o *React App* foi criado.

¹ Gamificação: o que é, vantagens e como implementar. FIA: Fundação Instituto de Administração, 2020. Disponível em: <<https://fia.com.br/blog/gamificacao/>>. Acesso em: 08 out. 2021.

Por fim, o diretório *src* é o principal do projeto, onde toda a lógica do *front end* está inserida. Desse modo, a estrutura do *src* é apresentada a seguir:

- *src*
 - *App.js*
 - *index.css*
 - *components*
 - *config*
 - *imgs-ac*
 - *screens*
 - *services*

3.2.1 *App.js* e *index.css*

Nessa seção será explicado o desenvolvimento dos arquivos *App.js* e *index.css*. O primeiro é responsável pela criação das rotas da aplicação por meio da biblioteca *react router dom*. O funcionamento dos componentes *BrowserRouter*, *Route*, *Switch* foi explicado no capítulo anterior, na Seção 2.2.3. O Trecho de código 3.11 apresenta a função *App*.

Trecho de código 3.11 – *App.js*

```
1 import React from "react";
2 import { BrowserRouter as Router, Route, Switch } from "react-router-dom";
3 import { Home, Inputs, Analysis, Help, About, Specifications } from "./screens";
4 import DataProvider from "./components/DataContext";
5
6 function App() {
7   return (
8     <DataProvider>
9       <Router>
10        <>
11          <Switch>
12            <Route component={Home} path="/" exact render />
13            <Route component={Inputs} path="/inputs" exact render />
14            <Route component={Analysis} path="/analysis" exact render />
15            <Route component={Help} path="/help" exact render />
16            <Route component={About} path="/about" exact render />
17            <Route component={Specifications} path="/check" exact render />
18          </Switch>

```

```
19     </>
20     </Router>
21   </DataProvider>
22 );
23 }
24
25 export default App;
```

Outra função importante exercida nesse arquivo é feita pelo componente *DataProvider*, responsável por compartilhar dados entre as telas. O Trecho de código 3.12 apresenta o arquivo *DataContext.js*, em que esse componente fora criado. Primeiramente, foi necessário "criar um contexto", onde as variáveis globais iriam ser inseridas. Em seguida, foi utilizado o *hook useContext*, explicado na Seção 2.2.3, para usar o contexto criado. Por fim, para criar o componente *DataProvider*, foram definidas as variáveis de estado que seriam utilizadas globalmente: *input* e *dataAnalysis*. Desse modo, todas as variáveis que estiverem dentro da propriedade *value* do componente *DataContext.Provider* poderão ser utilizadas por todos os filhos do componente *DataProvider*.

Trecho de código 3.12 – *DataContext.js*

```
1 import React, { createContext, useContext, useState } from "react";
2
3 const DataContext = createContext();
4
5 export const useDataContext = () => useContext(DataContext);
6
7 const DataProvider = ({ children }) => {
8   const [input, setInput] = useState({});
9   const [dataAnalysis, setDataAnalysis] = useState({});
10
11   return (
12     <DataContext.Provider
13       value={{ dataAnalysis, setDataAnalysis, input, setInput }}
14     >
15       {children}
16     </DataContext.Provider>
17   );
18 };
19
20 export default DataProvider;
```

Já o arquivo *index.css* é responsável pela estilização predefinida de todos os componentes que foram criados. Ele também poderia ser utilizado para estilizar toda a aplicação. No entanto, para que o projeto tenha maior organização e manutenibilidade foi optado por utilizar *styled-components*, que é uma biblioteca que permite a escrita de CSS em *JavaScript*.

3.2.2 Diretório *components*

Todos os componentes criados estão inseridos no diretório *components*, apresentado a seguir:

- components
 - styled-components
 - BodeDiagram.js
 - CheckBounds.js
 - CheckStepCLT.js
 - DataContext.js
 - Ocomponent.js
 - RootLocus.js
 - ShowInput.js
 - StepCLTemplate.js
 - StepResponse.js
 - StepResponseOL.js
 - Xcomponent.js

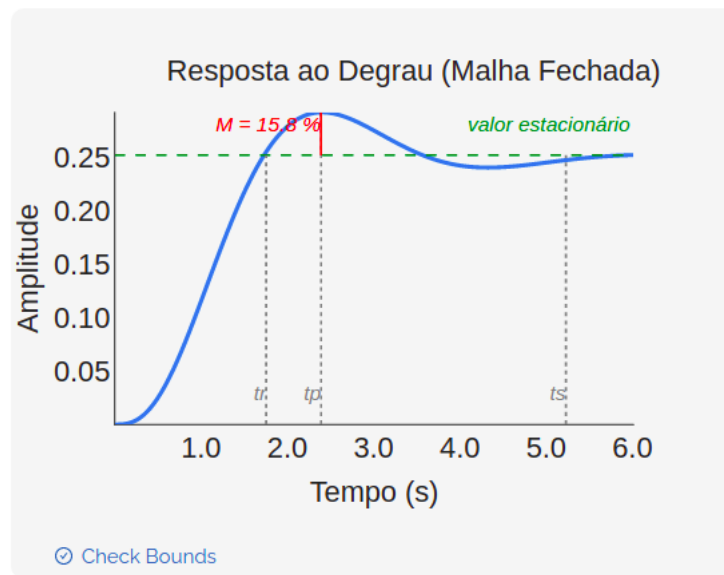
Os componentes principais, responsáveis pelos gráficos e que serão utilizados apenas uma vez na aplicação estão localizados na raiz do diretório *components*. Já os componentes auxiliares que serão utilizados várias vezes no site estão localizados dentro da pasta *styled-components*.

Os gráficos foram plotados por meio da biblioteca *Victory* que possui um conjunto de módulos e componentes para *React*, como foi feito por Ximenes (2021). Nesse trabalho, foram realizadas algumas modificações para consertar erros presentes nesses gráficos, além de acrescentar novas funcionalidades.

StepResponse

Primeiramente, as respostas ao degrau do sistema em malha aberta e do sistema em malha fechada foram implementadas em dois arquivos: *StepResponse.js* referente à malha fechada e *StepResponseOL.js* referente à malha aberta. O gráfico da resposta em malha aberta continuou igual ao de Ximenes (2021), enquanto o gráfico da malha fechada foi modificado para incluir os parâmetros do sistema, calculados pela função *stepInfo* no servidor *back end* do projeto. Na Fig 6 é apresentada a resposta ao degrau em malha fechada gerada pela plataforma.

Figura 6 – Resposta ao Degrau do Sistema em Malha Fechada gerada pela plataforma.



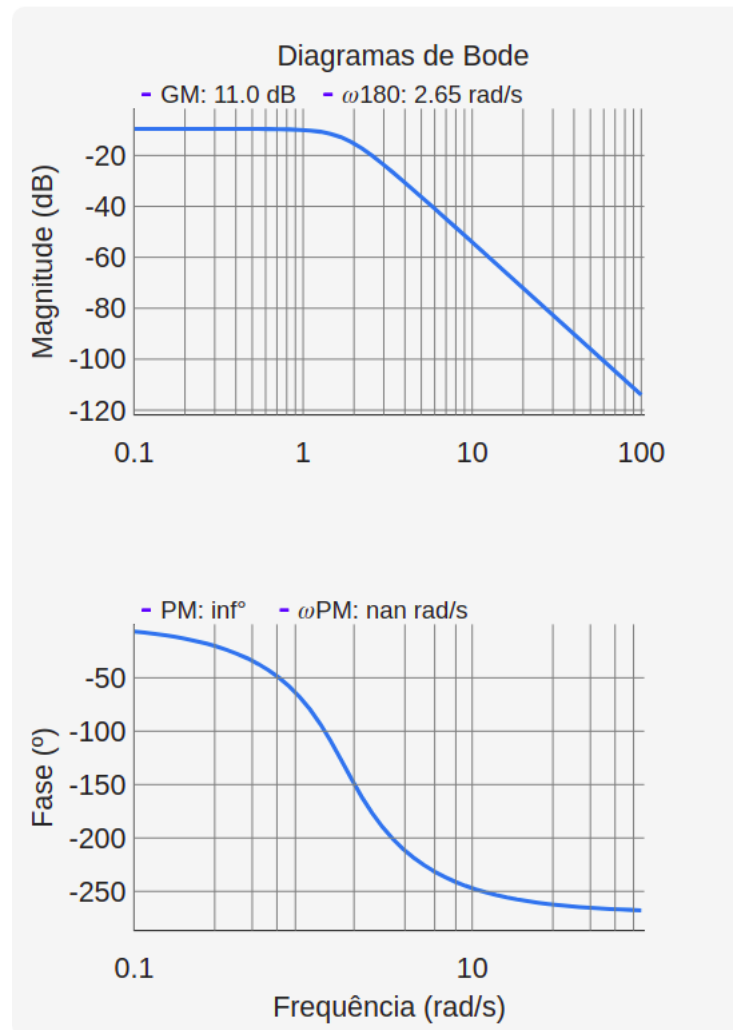
Fonte: autoria própria, 2021

Vale ressaltar que o botão "*Check Bounds*" serve para verificar se o sistema em malha fechada atende as especificações inseridas no projeto pelo usuário e que será explicado mais adiante nesta mesma seção.

BodeDiagram

No diagrama de Bode foram acrescentados os valores da margem de fase (PM) e margem de ganho (GM), assim como a frequência de cruzamento de ganho (ω_{PM}) e a frequência crítica (ω_{180}), além dos nomes dos eixos, como pode ser visto na Fig. 7.

Figura 7 – Diagramas de Bode gerados pela plataforma.

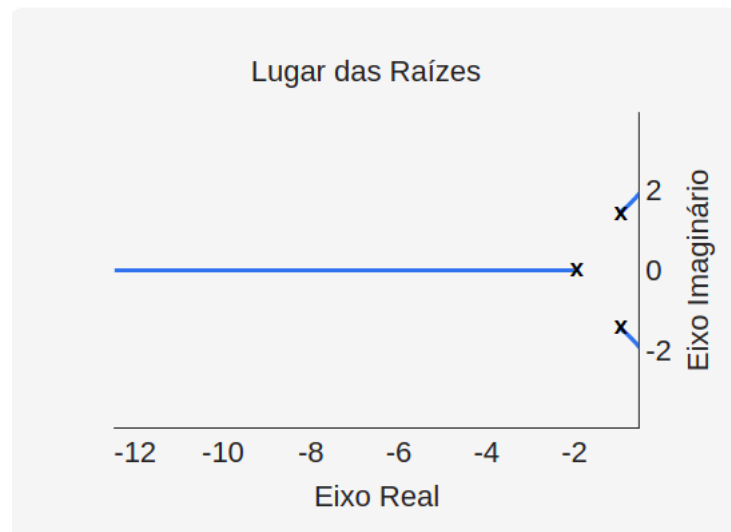


Fonte: autoria própria, 2021

RootLocus

No lugar das raízes foi trocada a representação dos polos para simbologia padrão, pois em Ximenes (2021) eram representados por um '+'. Para isso, foi necessário criar um componente auxiliar *Xcomponent*, responsável pela representação do polo. Também, foram inseridos os zeros correspondentes ao lugar das raízes, que são comumente representados por 'o'. Então, foi necessário criar o componente *Ocomponent*. Além disso, foi acrescentado o nome do eixo imaginário, como pode ser visto na Fig. 8.

Figura 8 – Lugar das raízes gerado pela plataforma.



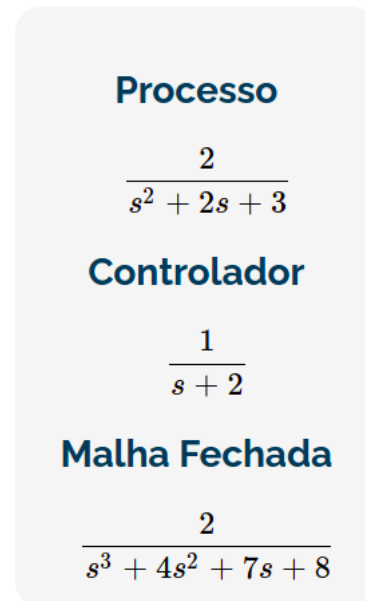
Fonte: autoria própria, 2021

Em Ximenes (2021), o gráfico do lugar das raízes possuía problema de descontinuidade. Para resolver essa questão, foi analisado como os vetores que correspondem as partes reais e imaginárias dos polos do sistema de controle estavam plotados. Então, percebeu-se que o plot estava ordenado em relação ao eixo x, correspondente as raízes reais do sistema. Essa ordenação causava descontinuidade no eixo y, uma vez que quando há trechos circulares no lugar da raízes, ele não seguia a ordem de acordo com o eixo y. Para resolver esse problema, foi adicionado propriedade `sortKey=0` no gráfico do lugar das raízes referente ao pacote *Victory*, utilizado nesse trabalho.

ShowInput.js

O componente *ShowInput* é responsável por mostrar as funções de transferência que estão em análise para o usuário. Desse modo, ele apresenta a função de transferência do processo, do controlador e da malha fechada para que o usuário possa verificar se inseriu as entradas certas na plataforma, além de ter a função de transferência do sistema em malha fechada calculada, que possibilita uma melhor análise da localização dos polos e zeros dominantes do sistema de controle. Na Fig. 9 é apresentada as funções de transferência do sistema de controle em análise.

Figura 9 – Funções de Transferência geradas pela plataforma.



Processo

$$\frac{2}{s^2 + 2s + 3}$$

Controlador

$$\frac{1}{s + 2}$$

Malha Fechada

$$\frac{2}{s^3 + 4s^2 + 7s + 8}$$

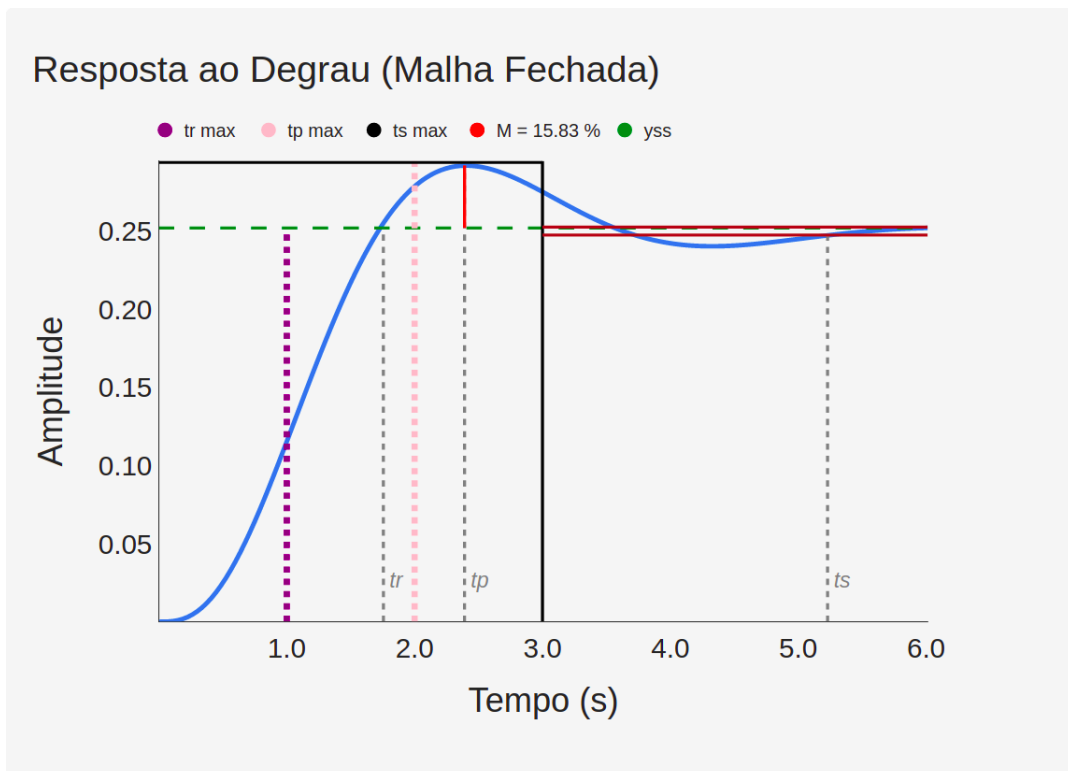
Fonte: autoria própria, 2021

Primeiramente, foi necessário importar a biblioteca *react-mathjax-preview*, responsável pela escrita das expressões matemáticas no *front end*. Além disso, foi necessário obter a resposta do servidor *back end* com o numerador e denominador da função de transferência do sistema em malha fechada. Isso foi possível de ser obtido por meio do *useDataContext* (Trecho de código 3.12). Os polinômios dos numeradores e denominadores foram formatados por meio do arquivo *ShowInput.js* para serem exibidos como mostrado na Fig. 9.

StepCLTemplate.js e CheckStepCLT.js

O componente *StepCLTemplate* possui uma construção parecida com o componente *StepResponse*, uma vez que utiliza a biblioteca *Victory* para criação e formatação do gráfico. Também, as informações inseridas pelo usuário em relação às especificações do sistema de controle são adicionadas ao gráfico da resposta ao degrau em malha fechada, para que seja possível realizar uma comparação entre a requisição do projetista e o controlador projetado para o sistema de controle em análise. Na Fig. 10 é apresentada a comparação gráfica entre os parâmetros do sistema em malha fechada.

Figura 10 – Comparação gráfica entre as especificações temporais e a resposta ao degrau em malha fechada.



Fonte: autoria própria, 2021

O gráfico apresentado na Fig. 10 pode ser interpretado da seguinte maneira:

- **Linhas pretas:** correspondem à janela de tempo em que a resposta ao degrau deve atingir o regime estacionário. A linha superior delimita o sobre-sinal máximo, enquanto que a linha vertical delimita o tempo de acomodação máximo;
- **Linha roxa:** delimita o tempo de subida máximo do projetista inserido pelo usuário;
- **Linha rosa:** delimita o tempo de pico máximo do projetista inserido pelo usuário;
- **Linhas marrons:** delimitam a variação máxima em que a resposta pode ter para ser considerado que o sistema de controle atingiu o regime estacionário.

Para facilitar a verificação se os parâmetros do sistema de controle obedecem às especificações do projetista, foi implementado o componente *CheckStepCLT* que realiza a comparação dos parâmetros automaticamente. Na Fig. 11 é apresentada a verificação gerada pela plataforma *Achilles Control* para o exemplo anterior apresentado na Fig. 10.

Figura 11 – Verificação dos parâmetros da resposta ao degrau em malha fechada.

❗	Tempo de subida
❗	Tempo de pico
❗	Tempo de acomodação
✅	Sobre-sinal
✅	Desvio da saída do seu valor de regime permanente

Fonte: autoria própria, 2021

Desse modo, é possível observar tanto pelo gráfico quanto pela verificação automática dos parâmetros que os tempos de subida, de pico e de acomodação não estão de acordo com as especificações solicitadas pelo projetista do sistema de controle. Assim, espera-se que o usuário recalcule a função de transferência do controlador ao modificar os polos dominantes do sistema de controle para que os parâmetros do sistema obedeçam às especificações.

Diretório *styled-components*

A pasta *styled-components* possui componentes que são essenciais para uma boa experiência do usuário no site, como botão, caixa de texto e tipografias estilizadas. O objetivo desses componentes é serem genéricos, reutilizáveis e padronizados. Todos os componentes apresentados a seguir foram estilizados por meio da biblioteca *styled-components*, além disso, todos eles são exportados por meio do arquivo *index.js*.

- styled-components
 - BackImg.js
 - Button.js
 - index.js
 - Input.js
 - Loading.js
 - Logo.jsx
 - MenuBar.js
 - Text.js
 - Title.js

Todos os componentes seguem o formato representado no Trecho de código 3.13 referente ao componente *Button*. Então, é possível observar que dentro das crases, depois da interface a ser estilizada ser definida, a escrita do código é estilizada por meio de CSS (e.g., *styled.button* ‘código em CSS’).

Trecho de código 3.13 – *Button.js*

```
1 import styled from "styled-components";
2 import { Theme } from "../../config";
3
4 export const Button = styled.button `
5   outline: none;
6   position: absolute;
7   width: 24rem;
8   height: 6.25rem;
9   background: ${Theme.pink};
10  border: none;
11  border-radius: 1.5rem;
12  box-shadow: 0px 4px 4px rgba(0, 0, 0, 0.4);
13  cursor: pointer;
14
15  display: flex;
16  justify-content: center;
17  align-items: center;
18  font-size: 3rem;
19  font-weight: bold;
20  color: white;
21
22  a {
23    color: white;
24    text-decoration: none;
25  }
26
27  &:focus,
28  &:hover {
29    background: ${Theme.buttonHover};
30  }
31
32  &:disabled {
33    background: #cccccc;
34  }
35 `;
```

Na Fig. 12 é apresentado o botão estilizado pelo Trecho de código 3.13.

Figura 12 – Botão estilizado por meio de *styled-components*.



Fonte: autoria própria, 2021

Outro componente, interessante de ser comentado, é o *MenuBar* responsável por transitar o usuário entre as páginas do site. Isso é possível por meio do *hook useHistory*, explicado na Seção 2.2.3.

3.2.3 Diretórios: *config* e *imgs-ac*

O diretório *config* possui a estrutura a seguir:

- *config*
 - *index.js*
 - *Theme.js*
 - *useScroll.js*

Esse diretório serve para definir configurações gerais que serão utilizadas por todo o projeto. O arquivo *index.js* serve para exportar os componentes da raiz do diretório, além de configurar a URL do servidor *back end* e a URL da aplicação *front end*, como pode ser visto no Trecho de código 3.14.

Trecho de código 3.14 – *index.js*

```
1 export const apiBaseUrl = "http://localhost:5000";
2 export const apiUrl = "http://localhost:3000";
3 export { useScroll } from "./useScroll";
4 export { Theme, width, height } from "./Theme";
```

Já o arquivo *Theme.js* é responsável por configurar o tema de cores utilizado pela aplicação, além do tamanho das telas do *front end*. E por fim, o arquivo *useScroll.js* possui um componente que serve para bloquear ou ativar o *scroll* nativo do browser.

Já o diretório *imgs-ac* é onde todos os arquivos de imagem utilizados nas telas da plataforma *Achilles Control* estão localizados.

3.2.4 Diretório *screens*

O diretório *screens* é onde todas as telas da plataforma estão localizadas e a sua estrutura é mostrada a seguir:

- *screens*
 - *About.jsx*
 - *Analysis.jsx*
 - *Help.jsx*

- Home.jsx
- index.js
- Inputs.jsx
- Loading.jsx
- Specifications.jsx

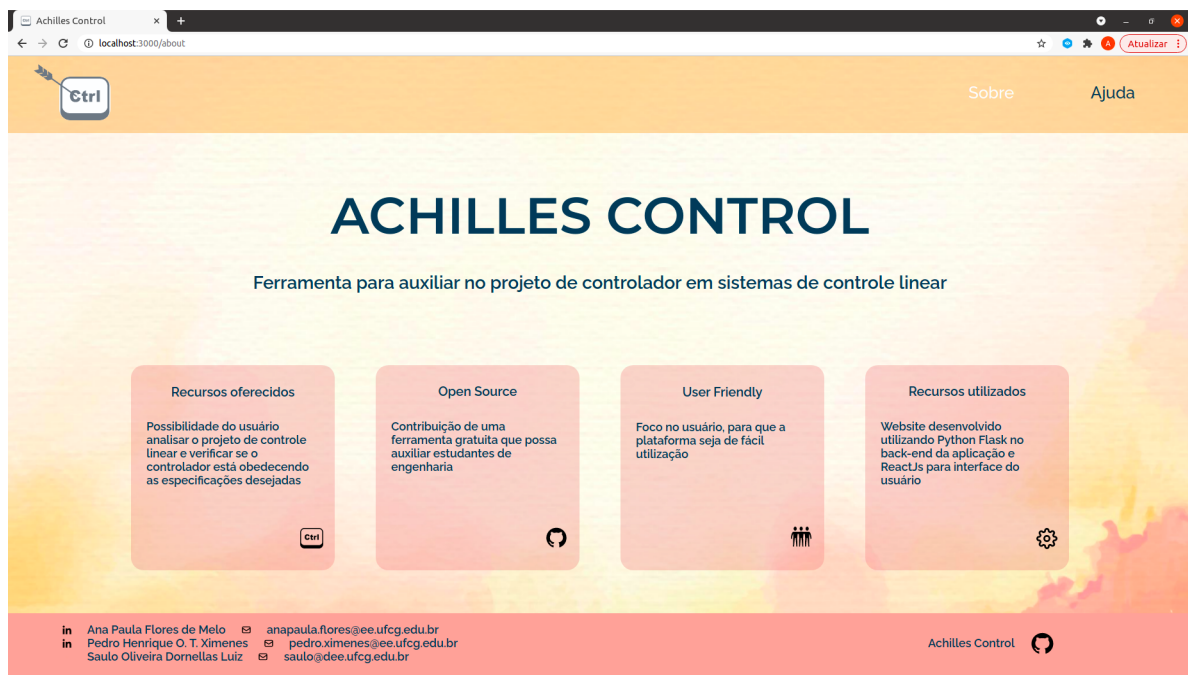
O arquivo *index.js* é onde todas as páginas do *front end* são exportadas para que possam ser utilizadas pela função *App*, explicada na Seção 3.2.1.

A tela de *Loading* serve como transição entre as telas de *Inputs* para *Analysis*, para que o usuário saiba que é o momento de esperar pela resposta da requisição feita na tela de *Inputs*.

About

Na plataforma *Achilles Control* foi implementada uma página de **Sobre**. Ela serve para situar o usuário sobre a utilidade do site, além de abordar características importantes em relação à construção da aplicação, por exemplo, ser um projeto de código aberto. Também, nessa página é fornecido o contato das pessoas envolvidas com o site. Na Fig. 13 é apresentada a tela de Sobre da plataforma *Achilles Control*.

Figura 13 – Página "Sobre" do site *Achilles Control*.



Fonte: autoria própria, 2021

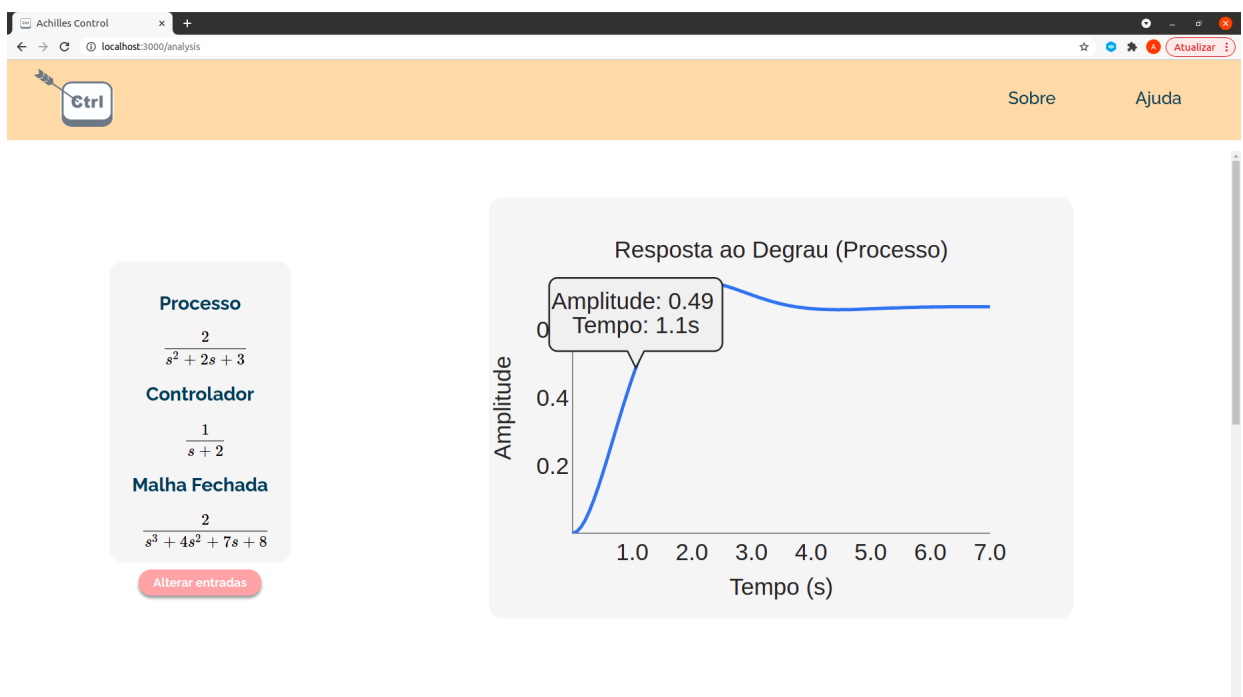
Essa página foi implementada, ao utilizar alguns componentes já existentes: *MenuBar* e *BackImg*, além das tipografias previamente estilizadas dentro do componente *Text*. Também, foram importados ícones do pacote *react-icons*. Outros componentes foram feitos especialmente para compor essa página, como o *foot*, onde os contatos estão localizados e os *cards*, onde as características da plataforma estão localizadas.

Outro aspecto, é que foi utilizada a função *window.location.replace* para redirecionar o site para um link externo. Por exemplo, se o usuário clicar no ícone do *LinkedIn*, ele será redirecionado para o *LinkedIn* da pessoa selecionada. Também, é possível acessar o projeto no *Github* se o usuário clicar no ícone do *Github* posicionado na parte inferior esquerda do site.

Analysis

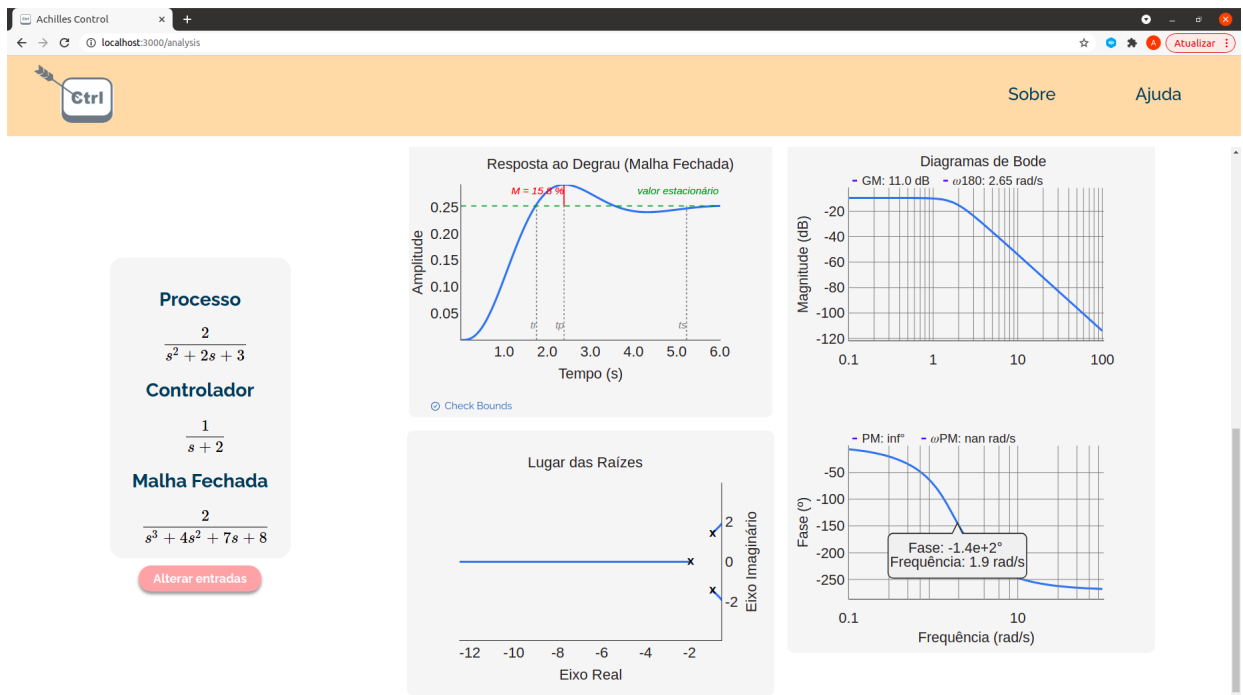
A página *Analysis* é onde os gráficos para análise do sistema de controle estão localizados. Ou seja, nessa tela são importados os componentes: *BodeDiagram*, *RootLocus*, *Step*, *StepResponse* e *ShowInput*. Além disso, todos esses componentes esperam receber como propriedade a resposta da requisição feita para o *back end*. Portanto, por meio da função *useDataContext* do arquivo *DataContext.js*, explicado na Seção 3.2.1, é possível obter a variável de estado *dataAnalysis* que recebe a resposta da requisição feita pela página *Inputs*. Nas Figs. 14 e 15 são apresentadas a tela de *Analysis* da plataforma *Achilles Control*.

Figura 14 – Página "Analysis" do site *Achilles Control*, parte 1.



Fonte: autoria própria, 2021

Figura 15 – Página "Analysis" do site *Achilles Control*, parte 2.



Fonte: autoria própria, 2021

Help

A página de **Ajuda** é necessária para que o usuário saiba como utilizar a plataforma *Achilles Control*. Nela, estão inseridas as respostas para possíveis dúvidas que o usuário possa ter, além de exemplos de como utilizar o site. Na Fig. 16 é apresentada a página *Help* desenvolvida para a plataforma.

Figura 16 – Página "Help" do site *Achilles Control*.

Para que serve?
Achilles Control é uma ferramenta gratuita e de fácil utilização para auxiliar projetos de controlador para sistemas lineares.

Como utilizo?
O usuário deve inserir as funções de transferência do processo, do controlador projetado e as especificações requisitadas do sistema de controle linear.

Quais são os resultados?
Serão obtidos os gráficos de resposta ao degrau do sistema em malha aberta e fechada (sem e com realimentação), o lugar das raízes e o diagrama de Bode. Esses gráficos são fundamentais para verificar se o controlador é adequado para o sistema.

Quais os formatos para inserção de dados?
Ponto: utilizado para números decimais, exemplo: 10.25, 5.55, 12.1
Vírgula: utilizada para separar termos, exemplo: $s + 2 + 1.2$
Logo, se você inserir 1.5, 2.3, 1, terá como resultado $1.5s^2 + 2.3s + 1$

Legenda das especificações

- O sistema em malha fechada corresponde a especificação
- O sistema em malha fechada não corresponde a especificação

Como insiro uma Função de Transferência?
 Numerador:
 Denominador:

$$\frac{s - 2}{s^2 + 2s + 3}$$

Como confiro se o sistema atende as especificações?

 Para checar se o sistema em malha fechada atende as especificações requisitadas, clique no botão 'Check Bounds'.

Fonte: autoria própria, 2021

Nessa tela, foi utilizado o componente *MenuBar*, além das tipografias estilizadas do componente *Text*. Também, foram adicionadas outras estilizações específicas à página.

Home

A página de *Home* é a que aparece quando o usuário acessa o site. Ela pode ser acessada por todas as outras páginas por meio da logo do site localizada no componente *MenuBar*. Nessa tela, está localizado o botão de início da plataforma, além do menu contendo acesso às páginas de Sobre e Ajuda. Na Fig. 17 é apresentada a tela da Home da plataforma *Achilles Control*.

Figura 17 – Página "Home" do site *Achilles Control*.

Fonte: autoria própria, 2021

Nessa tela, há a utilização dos componentes: *MenuBar*, *Button*, *BackImg* e *Title*. Os dois primeiros são componentes funcionais, que dão acesso a outras páginas. Já os dois últimos são para estilização dessa página. O botão "Começar" redireciona o usuário para a página *Inputs*, onde as funções de transferência do sistema de controle serão pedidas.

Inputs

A página *Inputs* é onde o usuário irá inserir as funções de transferência para o sistema de controle ser analisado. Para que todas as funcionalidades da plataforma possam ser utilizadas, é necessário que o usuário forneça as entradas que serão descritas a seguir:

- **Numerador e Denominador do Processo:** necessários para montar a função de transferência $H(s)$ do processo;
- **Numerador e Denominador do Controlador:** necessários para montar a função de transferência $G(s)$ do controlador;
- **Especificações:** necessárias para realizar a verificação dos parâmetros do sistema em malha fechada por meio da funcionalidade *Check Bounds*.

Na Fig. 18 é apresentada a página *Inputs* da plataforma *Achilles Control* com os campos preenchidos pelo exemplo mostrado por toda Seção 3.2. Essa tela é diferente das outras pois é a única que não possui o componente *MenuBar*. No entanto, é possível navegar para as outras páginas contidas no menu: *Home* por meio da seta de voltar; *About* por meio do ícone 'i'; e, *Help* por meio do ícone '?'.

Figura 18 – Página "Inputs" do site *Achilles Control*.

Fonte: autoria própria, 2021

Após todos os campos serem preenchidos, e o usuário clicar no botão "Enviar", será feita a requisição para o servidor *back end* da aplicação. O Trecho de código 3.15 apresenta como funciona o procedimento de recuperar as entradas do usuário no *front end* para que seja possível enviá-las para o *back end*.

Trecho de código 3.15 – *Exemplo de como recuperar a entrada inserida pelo usuário*

```

1 import React, { useState, useEffect } from "react";
2 import { useContext } from "../components/DataContext";
3 import { useHistory } from "react-router-dom";
4 import api from "../services/api";
5
6
7 const Inputs = () => {
8   const history = useHistory();
9   const [gnum, setGnum] = useState("");
10  const [load, setLoad] = useState(false);

```

```
11   const [showLoading, setShowLoading] = useState(false);
12   const [sendInfo, setSendInfo] = useState("");
13   const { input, setInput } = useContext();
14   const { setDataAnalysis } = useContext();
15
16   const onSubmit = (e) => {
17     e.preventDefault();
18     setSendInfo({gnum, load});
19     setInput({gnum});
20   };
21
22   useEffect(() => {
23     (async () => {
24       if (sendInfo) {
25         if (sendInfo.load === true) {
26           setShowLoading(true);
27         }
28         try {
29           const { data } = await api.post("/analysis", {
30             ...sendInfo,
31           });
32           setShowLoading(false);
33           setDataAnalysis(data);
34           history.push("/analysis");
35         } catch (error) {
36           console.error(error);
37         }
38       }
39     })();
40   }, [sendInfo]);
41
42   return (
43     <>
44       <form onSubmit={onSubmit}>
45         <Input
46           type="text"
47           placeholder="Exemplo: 1"
48           value={gnum}
49           data-test="gnum"
50           onChange={(e) => setGnum(e.target.value)}
51         />
52         <Button
53           type="submit"
54           data-test="submit"
55           onClick={() => {
56             setLoad(true);
57           }}
58         />
59     </>
60   );
61 }
```

```
58         >
59         Enviar
60     </Button>
61 </form>
62     {showLoading && <LoadingPage />}
63 </>
64 );
65 };
```

Primeiramente, foram importados os *hooks* *useState* e *useEffect* do React, além do *hook* *useHistory* do pacote *react-router-dom*, explicados na Seção 2.2.3. Também, foi feita a importação da função *useDataContext* do arquivo *DataContext.js*, explicado na Seção 3.2.1. Finalmente, foi importada a *api*, responsável pelas requisições ao *back end*. Em seguida, foi declarada a variável de estado *gnum*, referente ao **Numerador do Controlador**. Também, foram declaradas as variáveis de estado que indicam que a requisição está em processamento: *load*, *showLoading*. Por fim, foram declaradas: *sendInfo*, onde todas as informações referentes à requisição ao *back end* serão armazenadas; *input*, onde todas as entradas do usuário serão armazenadas; e, *dataAnalysis*, onde a resposta da requisição será armazenada.

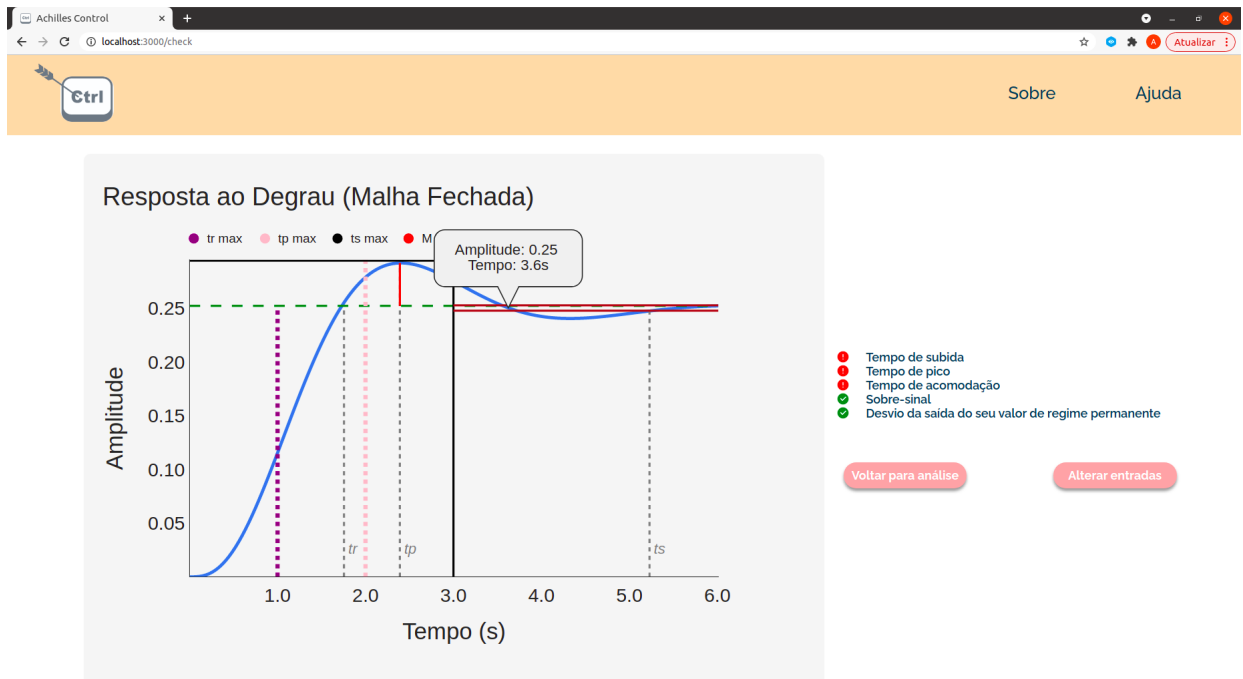
A lógica do Trecho de código 3.15 inicia no *return*. Primeiramente, a propriedade *onChange* do componente *Input* armazena os valores que o usuário digitar no campo de texto Numerador do Controlador. Então, quando o usuário clicar no botão de "Enviar", a propriedade *onClick* do botão é acionada, que configura o *load* como verdadeiro. Assim, o formulário realiza a chamada da função *onSubmit*, que configura quais as informações serão enviadas para requisição ao *back end* e armazena todas as entradas do usuário na variável global *input*, para que seja possível utilizá-la em outras páginas. Desse modo, sempre que a variável de estado *sendInfo* for atualizada, o *hook* *useEffect* será acionado. Nele, é realizada a requisição para o *back end*, por meio da *api* configurada no projeto, explicada na Seção 3.2.5. Após a resposta da requisição ser armazenada na variável *data*, ela é enviada para a variável global *dataAnalysis*, para que as outras telas tenham acesso ao objeto *json* retornado pelo servidor *back end*, necessário para plotar os gráficos da aplicação. Por fim, o usuário será redirecionado para a página *Analysis*.

Specifications

A página *Specifications* é onde tanto a comparação gráfica entre os parâmetros da resposta ao degraú do sistema em malha fechada quanto a verificação desses parâmetros em relação as especificações do projetista estão localizadas. Ou seja, nessa tela são importados os componentes: *StepCLTemplate* e *CheckStepCLT*. Além disso, todos esses componentes esperam receber como propriedade a resposta da requisição feita para o *back end* e as especificações do projeto inseridas pelo usuário na página *Inputs*. Portanto, por

meio da função `useDataContext` do arquivo `DataContext.js`, explicado na Seção 3.2.1, é possível obter as variáveis globais de estado: `input`, que recebe todas as entradas inseridas pelo usuário; e, `dataAnalysis` que recebe a resposta da requisição ao *back end*. Na Fig. 19 é apresentada a tela de *Specifications* da plataforma *Achilles Control*.

Figura 19 – Página "Specifications" do site *Achilles Control*.



Fonte: autoria própria, 2021

Também, após o usuário verificar se os requisitos do sistema estão de acordo com o projetado, ele pode optar por retornar para a página *Analysis* ou ir para tela *Inputs*, para alterar as entradas da plataforma *Achilles Control*.

3.2.5 Diretório *services*

O diretório *services* é onde o arquivo de configuração da API está localizado, como pode ser visto na estrutura a seguir:

- services
 - api.js

O Trecho de código 3.16 apresenta a configuração da API para que seja possível realizar as requisições ao servidor *back end* da plataforma *Achilles Control*.

Trecho de código 3.16 – *api.js*

```
1 import axios from "axios";
2 import { apiBaseUrl } from '../config/index';
3
4 const params = {
5   baseUrl: apiBaseUrl,
6 };
7
8 const api = axios.create(params);
9
10 api.interceptors.request.use(
11   (config) => config,
12   (error) => Promise.reject(error)
13 );
14
15 api.interceptors.response.use(
16   (response) => response,
17   (error) => Promise.reject(error)
18 );
19
20 export default api;
```

Desse modo, foi utilizado *axios* que é um cliente HTTP, responsável por fazer as requisições do sistema. Logo, na linha 4 foi feita a criação do cliente, ao passar como parâmetro a URL do servidor *back end*. Em seguida, foram configuradas, como a requisição e a resposta serão retornadas. Então, a requisição pode ser feita como mostra o Trecho de código 3.17.

Trecho de código 3.17 – Exemplo de requisição POST por meio da api criada com *axios*

```
1 import api from "../services/api";
2
3 try {
4   const { data } = await api.post("/analysis", {
5     ...sendInfo,
6   });
7 } catch (error) {
8   console.error(error);
9 }
```

3.2.6 Testes de Integração

Como explicado no capítulo anterior, foi utilizado o *framework Cypress* para realizar os testes de integração da plataforma *Achilles Control*. A estrutura do diretório *cypress* é apresentado a seguir:

- cypress
 - fixtures
 - * inputsTestNotOk.json
 - * inputsTestOk.json
 - integration
 - * analysis.spec.js
 - * check.spec.js
 - * home.spec.js
 - * input.spec.js

Na pasta *fixtures* estão localizados os arquivos para testes massivos. Nesse caso, exemplos de entradas corretas e incorretas que o usuário pode inserir na página *Inputs*. Já na pasta *integration* estão localizados os arquivos de teste referentes às principais telas do site.

Em todas as páginas foi realizado o teste de *endpoint*, para verificar se o clique do botão redireciona o usuário para a página correta. O Trecho de código 3.18 apresenta o teste realizado para a página *Home* que possui esse teste genérico que também foi implementado para as outras páginas.

Trecho de código 3.18 – *home.spec.js*

```
1  /// <reference types="cypress" />
2  import { apiURL } from "../../src/config";
3
4  describe("test home endpoints", () => {
5    beforeEach(() => {
6      cy.visit(`${apiURL}/`);
7    });
8
9    it("press start button", () => {
10     cy.get("[data-cy=start]").click();
11     cy.location("pathname").should("include", "inputs");
12   });
13
14   it("press about button", () => {
15     cy.get("[data-cy=about]").click();
16     cy.location("pathname").should("include", "about");
17   });
18
19   it("press help button", () => {
20     cy.get("[data-cy=help]").click();
21     cy.location("pathname").should("include", "help");
```

```
22   });  
23 }
```

O teste feito para a página *Specifications* no arquivo *check.spec.js* foi parecido com o Trecho de código 3.18, com acréscimo dos *endpoints* das páginas de *Inputs* e *Analysis*.

Já na página *Inputs* foram realizados testes no formulário de entradas. O Trecho de código 3.19 apresenta os testes realizados para algumas entradas, como exemplo. As demais entradas foram testadas igualmente.

Trecho de código 3.19 – *input.spec.js*: testes do formulário

```
1 describe("test inputs form", () => {  
2   beforeEach(() => {  
3     cy.visit(`${apiURL}/inputs`);  
4     cy.fixture("inputsTestOk.json").then((item) => {  
5       inputTestOk = item;  
6     });  
7     cy.fixture("inputsTestNotOk.json").then((item) => {  
8       inputTestNotOk = item;  
9     });  
10  });  
11  it("inputs be numbers", () => {  
12    const pattern = /[0-9]+/;  
13  
14    inputTestOk.map((item) => {  
15      cy.expect(item.hnum).to.match(pattern);  
16      cy.expect(item.hden).to.match(pattern);  
17    });  
18    it("ok if hden length > hnum length", () => {  
19      const hnum = "1,2";  
20      const hden = "1,2,3";  
21  
22      cy.expect(hden.length).to.be.greaterThan(hnum.length);  
23    });  
24  });  
25  it("is Ok when fill inputs", () => {  
26    inputTestOk.map((item) => {  
27      cy.visit(`${apiURL}/inputs`);  
28  
29      cy.get("[data-test=gnum]").type(`${item.gnum}`);  
30      cy.get("[data-test=hnum]").type(`${item.hnum}`, { force: true });  
31      cy.get("[data-test=gden]").type(`${item.gden}`);  
32      cy.get("[data-test=hden]").type(`${item.hden}`, { force: true });  
33  
34      cy.get("[data-test=submit]").click();  
35  
36      cy.request("POST", `${apiBaseURL}/analysis`, {
```



```
37     gden: item.gden,
38     gnum: item.gnum,
39     hden: item.hden,
40     hnum: item.hnum,
41   }).then((response) => {
42     expect(response).property("status").to.equal(200);
43     expect(response.body).to.not.be.empty;
44   });
45
46   cy.request(`${apiBaseURL}/analysis`).then((response) => {
47     expect(response).property("status").to.equal(200);
48     expect(response).property("body").to.not.be.empty;
49   });
50
51   cy.location().should((location) => {
52     expect(location.pathname).to.eq("/analysis");
53   });
54 });
55 });
56 it("alert must appear cus hnum > hden", () => {
57   inputTestNotOk.map((item) => {
58     cy.visit(`${apiURL}/inputs`);
59
60     cy.get("[data-test=gnum]").type(`${item.gnum}`);
61     cy.get("[data-test=hnum]").type(`${item.hnum}`, { force: true });
62     cy.get("[data-test=gden]").type(`${item.gden}`);
63     cy.get("[data-test=hden]").type(`${item.hden}`, { force: true });
64
65     cy.get("[data-test=submit]").click();
66
67     cy.location().should((location) => {
68       expect(location.pathname).to.eq("/inputs");
69     });
70
71     cy.on("window:alert", (text) => {
72       expect(text).to.contains(
73         "0 denominador do processo tem que ter grau maior ou igual do
74           que o do numerador"
75       );
76     });
77   });
78 });
```

Primeiramente, foram declaradas as ações que o *Cypress* deve realizar antes de cada teste, por meio da função *beforeEach*. Nesse caso, ele deve estar na tela de *Inputs* e os dois objetos json que contem várias entradas de teste devem ser carregados nas variáveis

inputTestOk e *inputTestNotOk*. Em seguida, foram realizados testes para saber se as entradas estão dentro do padrão especificado, por meio da assertiva *to.match*. Também, foi verificado se o grau do denominador é maior ou igual que o grau do numerador do processo, por meio da assertiva *to.be.greaterThan*. Por fim, foram feitos testes para verificar se o *front end* fará requisição para o servidor *back end*.

O primeiro caso de teste é positivo. Então, após o *Cypress* digitar as entradas disponíveis no vetor *inputTestOk* em suas respectivas caixas de texto, ele aciona o botão "Enviar" e realiza uma requisição ao servidor *back end*. É esperado que o código de *status* da requisição seja 200 (significa que a requisição foi bem sucedida) e que volte algo no corpo da resposta. Por fim, é esperado que o usuário seja redirecionado para a página *Analysis*.

Já o segundo caso de teste, é negativo. Logo, o *Cypress* digita as entradas disponíveis no vetor *inputTestNotOk* em suas respectivas caixas de texto. Em seguida, ele aciona o botão "Enviar", mas a requisição não será realizada, pois as entradas não estão de acordo com o padrão esperado. Portanto, é esperado que o usuário continue na tela *Inputs* e que apareça um alerta para avisar ao usuário que há um erro nas entradas inseridas.

Vale ressaltar que nos testes foi utilizada a função *map()*, que funciona como um *for*, para mapear todas as entradas disponíveis nos arquivos json.

Outro teste importante a ser comentado é o que verifica o que acontece quando o usuário clicar no botão 'Check Bounds' na página *Analysis*. No entanto, para que seja possível realizar esse teste, o usuário precisa digitar as informações de entrada na página *Inputs* para que a aplicação envie a requisição ao servidor *back end*, para gerar os gráficos localizados na tela *Analysis*. Portanto, o Trecho de código 3.20 apresenta esse procedimento de teste.

Trecho de código 3.20 – *analysis.spec.js*: verifica o caminho até a página *Specifications*

```
1 it("press check bounds button", () => {
2   cy.visit(`${apiURL}/inputs`);
3   let gnum = "1";
4   let hnum = "1,2";
5   let gden = "1,2";
6   let hden = "1,2,3";
7   let rT = "1.2";
8   let sT = "3.4";
9   let pT = "2";
10  let vSS = "1";
11  let ovst = "17";
12  cy.get("[data-test=gnum]").type(`${gnum}`);
13  cy.get("[data-test=hnum]").type(`${hnum}`, { force: true });
14  cy.get("[data-test=gden]").type(`${gden}`);
15  cy.get("[data-test=hden]").type(`${hden}`, { force: true });
```

```
16 cy.get("[data-test=riseTime]").type(`${rT}`);
17 cy.get("[data-test=settlingTime]").type(`${sT}`);
18 cy.get("[data-test=peakTime]").type(`${pT}`);
19 cy.get("[data-test=varSS]").type(`${vSS}`);
20 cy.get("[data-test=overshoot]").type(`${ovst}`);
21 cy.get("[data-test=submit]").click();
22
23 cy.request("POST", `${apiBaseURL}/analysis`, {
24   gden: gden,
25   gnum: gnum,
26   hden: hden,
27   hnum: hnum,
28 }).then((response) => {
29   expect(response).property("status").to.equal(200);
30   expect(response.body).to.not.be.empty;
31 });
32
33 cy.request(`${apiBaseURL}/analysis`).then((response) => {
34   expect(response).property("status").to.equal(200);
35   expect(response).property("body").to.not.be.empty;
36 });
37
38 cy.location().should((location) => {
39   expect(location.pathname).to.eq("/analysis");
40 });
41
42 cy.get("[data-cy=checkBounds]").click({ force: true });
43 cy.location("pathname").should("include", "check");
44 });
```

No próximo capítulo serão apresentados os resultados e discussões do projeto, além das futuras melhorias que poderão contribuir ainda mais para a plataforma *Achilles Control*.

4 Resultados e Discussões

Esse capítulo é dedicado à apresentação dos resultados finais obtidos no projeto. Os resultados serão analisados criticamente e serão expostas as possíveis futuras melhorias. Como esse trabalho é uma contribuição à plataforma desenvolvida por Ximenes (2021), para fins de validação, será feita uma comparação entre os resultados obtidos por este trabalho (*Achilles Control*) e os resultados obtidos por Ximenes (2021). Os seguintes gráficos serão analisados: resposta ao degrau do sistema de controle em malha fechada, diagramas de Bode e lugar das raízes.

4.1 Comparação entre as plataformas

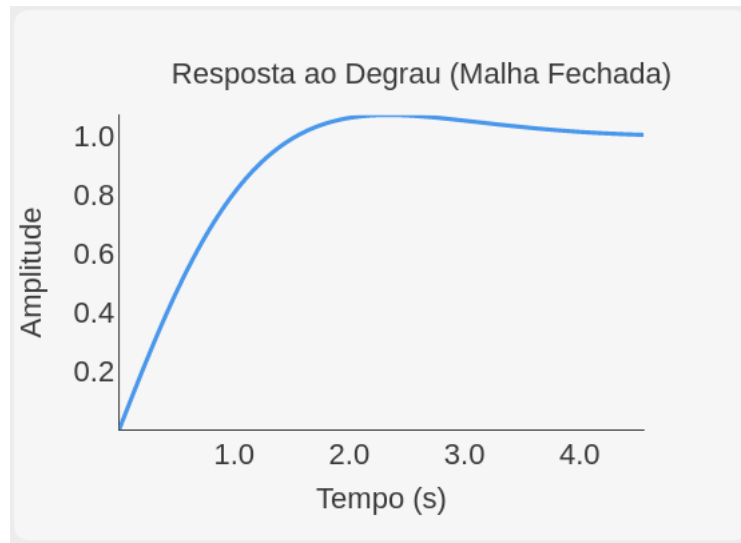
Foi selecionado o primeiro exemplo do Capítulo 4 de Ximenes (2021) para realizar a comparação e a análise dos resultados. Portanto, os dados inseridos são os seguintes:

- Processo: $\frac{1}{s+1}$
- Controlador: $\frac{s+2.02}{s}$

Resposta ao Degrau - Sistema de controle em Malha Fechada

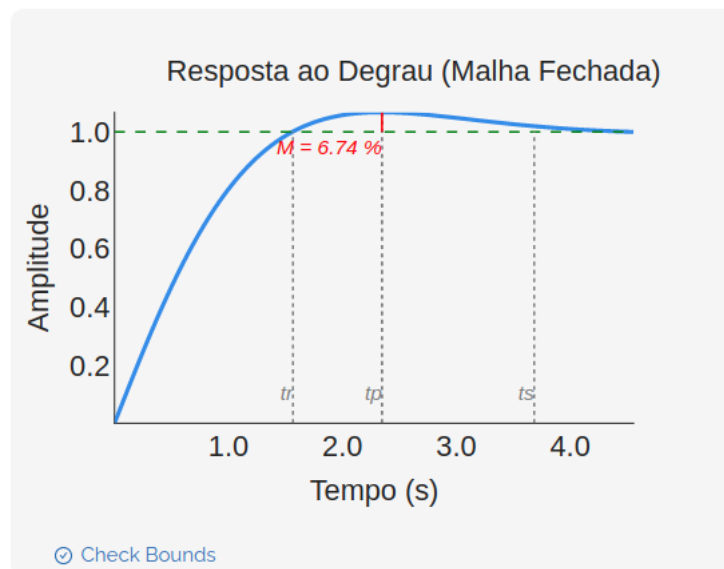
A resposta ao degrau do sistema de controle em malha fechada é apresentada nas Figs. 20 e 21. Percebe-se as contribuições adicionadas neste trabalho, como os parâmetros do sistema de controle, explicados na Seção 2.1.1, sendo mostrados graficamente.

Figura 20 – Resposta ao Degrau do sistema de controle em malha fechada gerado por Ximenes (2021).



Fonte: Ximenes (2021)

Figura 21 – Resposta ao Degrau do sistema de controle em malha fechada gerado pelo Achilles Control.



Fonte: autoria própria, 2021

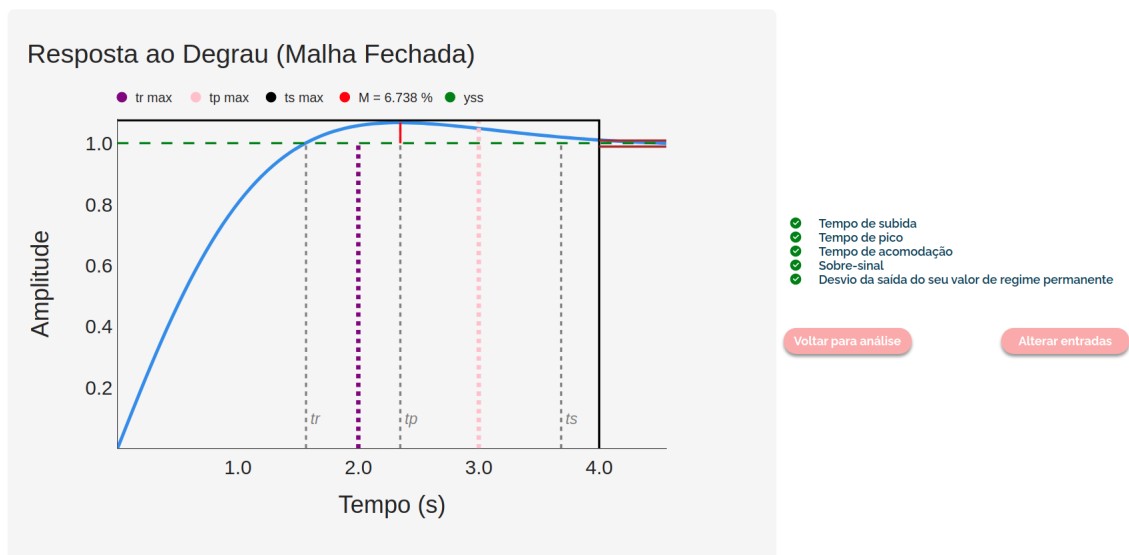
Vale ressaltar que há o acréscimo da funcionalidade "*Check Bounds*" que verifica se os parâmetros do sistema estão de acordo com as especificações do projetista. Então, supõe-se que o projeto precisa ter as seguintes especificações:

- Tempo de subida (t_r) igual a 2s

- Tempo de pico (t_p) igual a 3s
- Tempo de acomodação (t_s) igual a 4s
- Sobre-sinal (M) igual a 8%
- Variação da saída em regime permanente igual a 1%

Portanto, na Fig. 22 é apresentada a comparação gráfica e a verificação automática entre os parâmetros do sistema de controle e as especificações do projetista.

Figura 22 – Verificação dos parâmetros do sistema de controle gerado pelo *Achilles Control*.



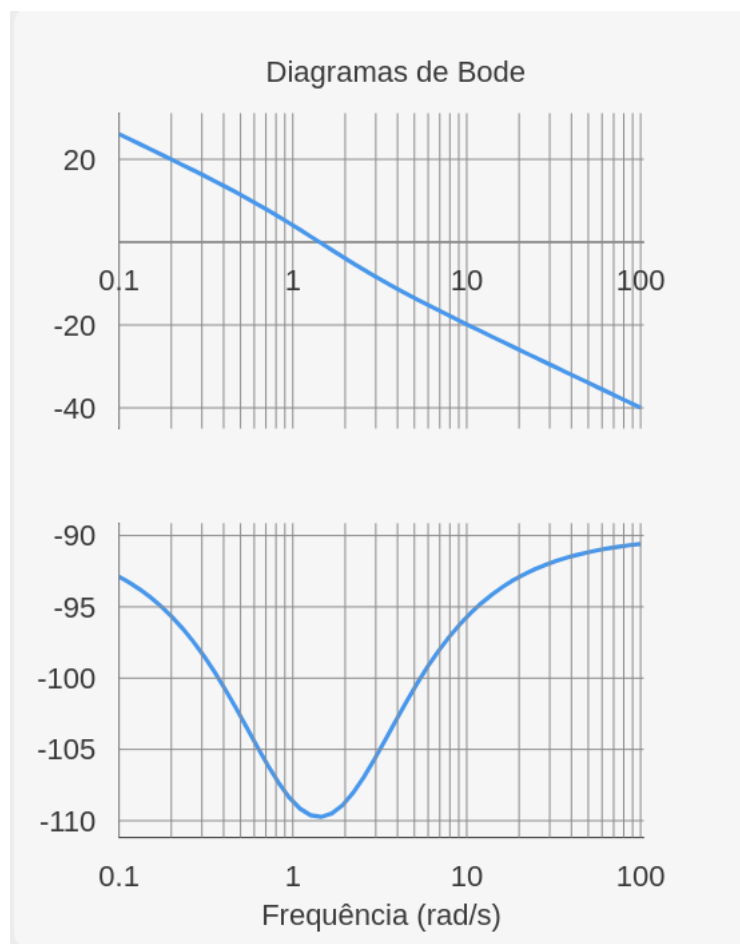
Fonte: autoria própria, 2021

É possível analisar que o controlador projetado satisfaz as especificações no domínio do tempo para o processo modelado. Caso algum parâmetro estivesse fora dos limites estabelecidos, o usuário saberia que é necessário reprojeter o controlador.

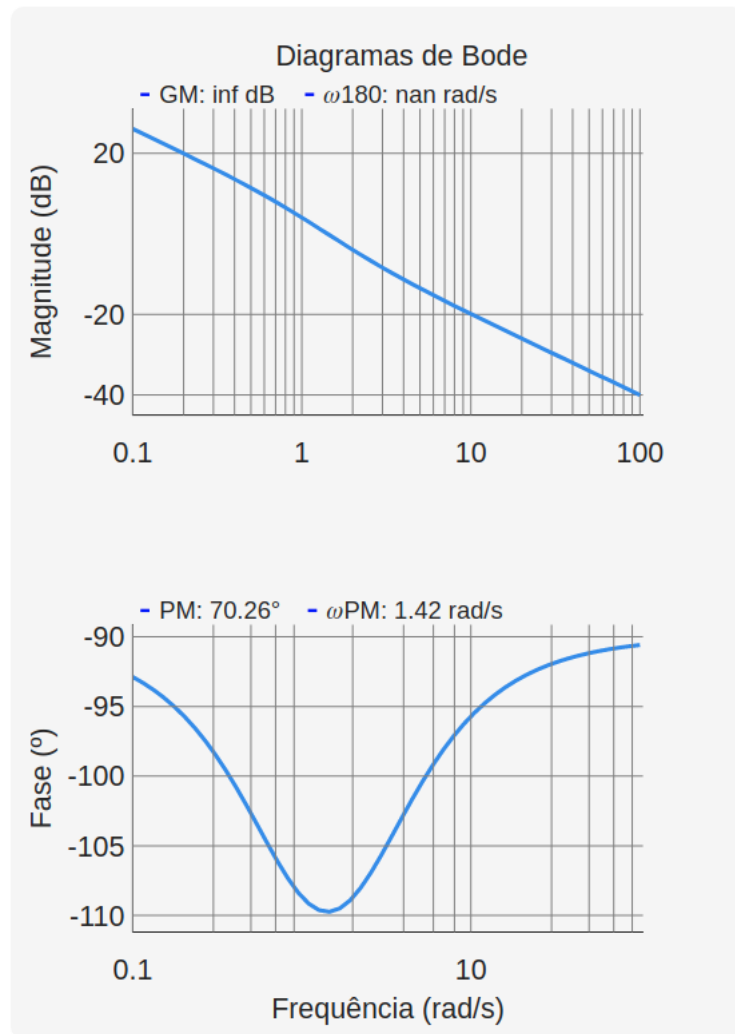
Diagramas de Bode

Nas Figs. 23 e 24 são representados os Diagramas de Bode obtidos por Ximenes (2021) e pelo *Achilles Control* respectivamente. Nota-se que a principal contribuição foi o acréscimo das informações: margem de ganho (GM) e a frequência crítica (ω_{180}); e, margem de fase (PM) e a frequência de cruzamento de ganho (ω_{PM}). Também, foram acrescentados os nomes dos eixos. Além disso, o eixo das frequências foi reposicionado para a parte inferior do gráfico.

Figura 23 – Diagrama de Bode gerado por Ximenes (2021).



Fonte: Ximenes (2021)

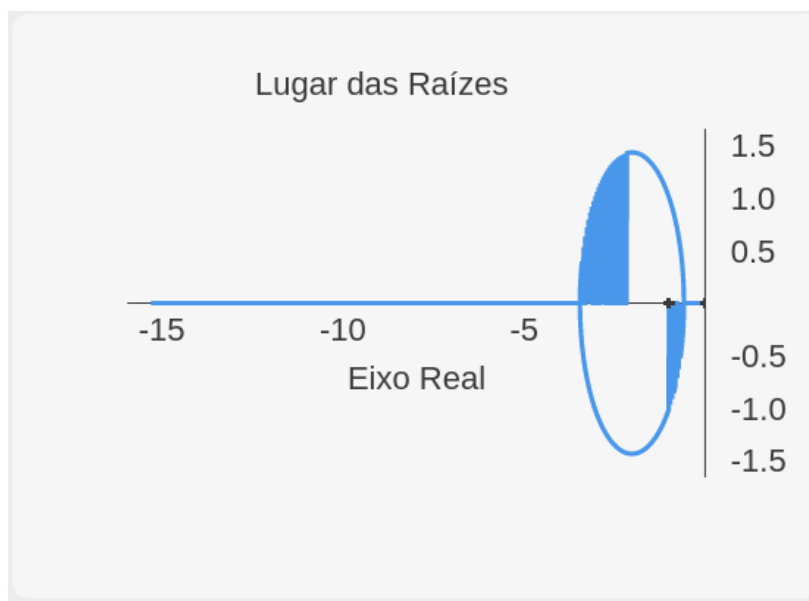
Figura 24 – Diagrama de Bode gerado pelo *Achilles Control*.

Fonte: autoria própria, 2021

Lugar das Raízes

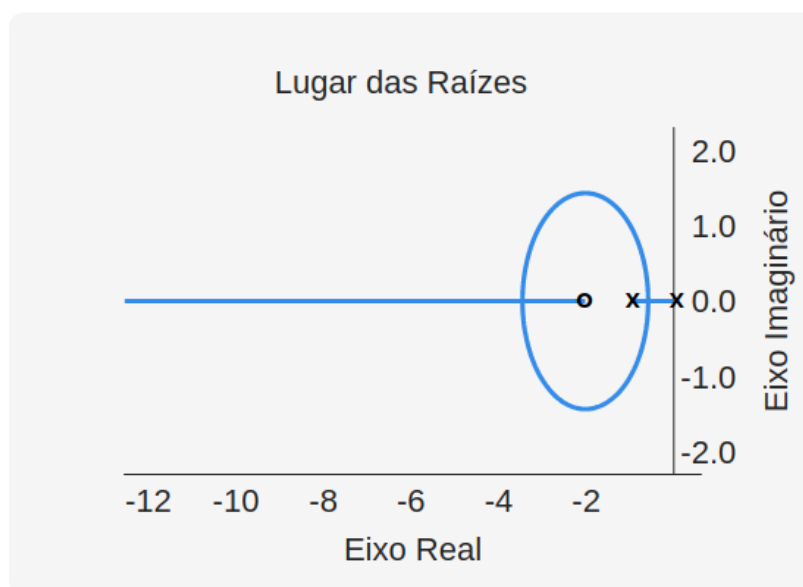
O lugar das raízes, mostrado nas Figs. 25 e 26, é o gráfico onde verifica-se a maior quantidade de correções. No desenvolvimento deste trabalho, foram consertadas as falhas de construção do gráfico gerado por Ximenes (2021), como comentadas na Seção 3.2.2. Além disso, foram acrescentadas informações relacionadas ao coeficiente de amortecimento (ζ) e a frequência natural (ω_n), como pode ser visto na Fig. 27.

Figura 25 – Lugar das Raízes gerado por Ximenes (2021).



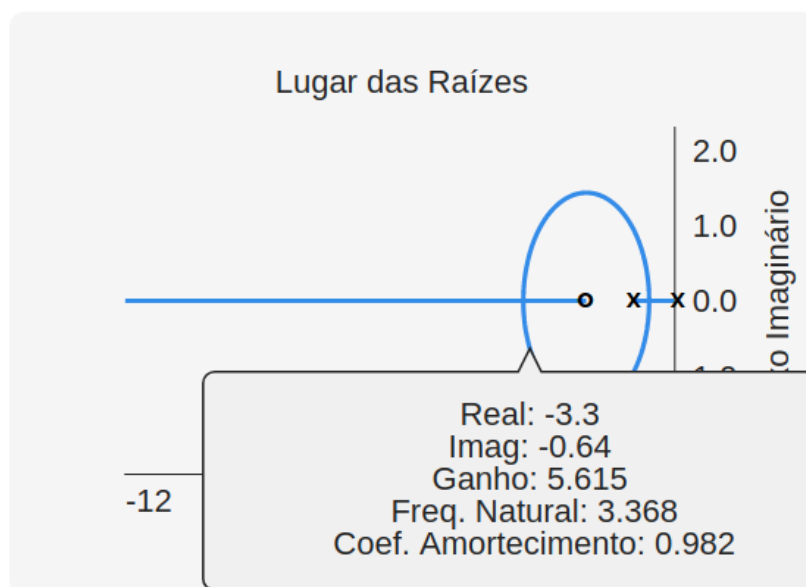
Fonte: Ximenes (2021)

Figura 26 – Lugar das Raízes gerado pelo *Achilles Control*.



Fonte: autoria própria, 2021

Figura 27 – Informações adicionais no Lugar das Raízes gerado pelo *Achilles Control*.



Fonte: autoria própria, 2021

4.2 Resultados e Melhorias Futuras

É possível afirmar, com base nas comparações, que a plataforma alcançou seu objetivo inicial. Pode-se perceber que os dados mostrados estão condizentes com os da plataforma de Ximenes (2021) com contribuições interessantes. Contudo, há necessidade de algumas melhorias para o *software* abordar mais conteúdos e, por isso, são listadas a seguir algumas dessas possíveis melhorias:

- Melhorar tratamento de erros:
 - Tratar erros de comunicação com o servidor e outros tipos de erro que não são tratados atualmente.
- Adição de mais testes unitários.
- Inclusão de outras informações como o Diagrama de Nyquist.
- Inclusão de análise e projeto por variáveis de estados.
- Adição da possibilidade de incluir atraso.
- Verificar possibilidade de cálculo em tempo real, permitindo alterações mais rápidas nos gráficos.
- Adição de análise de sistemas com resposta invertida (zeros no semi-plano direito).
- Converter o site para uma versão *mobile* para que o usuário tenha uma experiência agradável em outros aparelhos.
- Verificar a infraestrutura da plataforma para que seja possível disponibilizar o site para o público.

No próximo capítulo serão apresentadas as considerações finais em relação ao trabalho exposto neste relatório.

5 Considerações Finais

Este trabalho de conclusão de curso teve como objetivo adicionar contribuições à plataforma *web* que auxilia projetos de sistemas de controle lineares desenvolvida por Ximenes (2021). Para isso, foram estudados tanto os princípios da Teoria de Controle Analógico quanto sobre tecnologias em alta no mercado atual para implementar novas funcionalidades na plataforma.

Foram utilizadas tecnologias gratuitas para disponibilizar uma plataforma *web* de fácil utilização que auxilie alunos no aprendizado de projeto e análise de sistemas de controle. Todas as mudanças realizadas no site foram pensadas para melhorar a experiência de aprendizado do aluno. Desse modo, foi alterada a interface gráfica do site para que possuía similaridade com um jogo. A página inicial possui um botão de "Começar" que redireciona o usuário para a tela de entradas, onde a depender do que for inserido pelo aluno, gera um resultado diferente na próxima página. Por fim, existe a funcionalidade "Check Bounds" que verifica se o projeto do controlador está de acordo com as especificações do projetista, ou seja, verifica se o usuário ganhou ou perdeu o "jogo". Então, percebe-se que foi utilizada a técnica de gamificação para motivar o usuário a utilizar a plataforma.

Alguns desafios da execução deste trabalho foram a compreensão das tecnologias utilizadas, principalmente relacionadas ao *front end* da aplicação, que foi completamente remodelado. Também, foi desafiador realizar os testes de integração que foram feitos para garantir que o servidor *back end* e o *front end* se comuniquem corretamente.

Por fim, este trabalho de conclusão de curso demandou conhecimentos das disciplinas de graduação, principalmente de Controle Analógico, Introdução a Programação e Técnicas de Programação. Além disso, permitiu o desenvolvimento de habilidades complementares à formação curricular, como a aquisição de conhecimento em relação a desenvolvimento de *software*. Também, espera-se que o projeto seja continuado para abordar mais conceitos referentes à Teoria de Controle Analógico. Desse modo, foram propostas possíveis futuras melhorias a serem incrementadas na plataforma *Achilles Control*.

Referências

XIMENES, P. H. O. T. (2021). Desenvolvimento de uma plataforma web para projetos de sistemas de controle lineares. Dissertação (Monografia) - Curso de Engenharia Elétrica, Departamento de Engenharia Elétrica, Universidade Federal de Campina Grande, Campina Grande, 2021. Disponível em: <<https://sites.google.com/a/dee.ufcg.edu.br/cgee/projeto-de-engenharia/relatorios>>. Acesso em: 27 jun. 2021.

Getting Started with the Control System Designer. MathWorks, 2021. Disponível em: <<https://www.mathworks.com/help/control/ug/getting-started-with-the-control-system-designer.html>>. Acesso em: 02 ago. 2021.

BISHOP, R. H., DORF, R. C. (2009). Sistemas de controle modernos. Brasil: LTC.

OGATA, K. (2011). Engenharia de controle moderno. Brasil: PRENTICE HALL BRASIL.

FRANKLIN, G. F., et al. (2013). Sistemas de Controle para Engenharia - 6ed. Brasil: Bookman Editora. p. 189-191.