CEEL - ISSN 2596-2221
Universidade Federal de Uberlândia
13 a 17 de dezembro de 2021

ANÁLISE SIMULADA DE CIRCUITOS ELÉTRICOS COM LINGUAGEM PYTHON

Resumo **- O objetivo do presente artigo é instruir e apresentar, progressivamente, estudos e técnicas para manipulação de sistemas de controle de circuitos elétricos, através de suas funções de transferência e da manobra de grandezas contidas nestes sistemas, como tensão, corrente, indutância e carga, e seus componentes, sejam eles passivos, como resistores, indutores e capacitores, ou ativos, como amplificadores operacionais. Destarte, por meio das manipulações feitas através de sistemas de controle, conseguir-se-á circuitos com características espefícicas desejadas, como por exemplo, a estabilidade dos valores de tensão ou potência, mediante a integração de módulos elétricos, que possam ser conectados, atuando de forma análoga a um diagrama de blocos, atingindo assim uma voltagem ou potência de saída do sistema dita equilibrada.**

Palavras-chave **- Circuitos elétricos, corrente, estabilidade, manipulação, sistemas de controle, tensão.**

SIMULATED ANALYSIS OF ELECTRICAL CIRCUITS WITH PYTHON LANGUAGE

Abstract **- The purpose of this article is to progressively instruct and present studies and techniques for the manipulation of control systems of electrical circuits, through their transfer functions and the maneuvering of quantities contained in these systems, such as voltage, current, inductance and load, and its components, whether passive, such as resistors, inductors and capacitors, or active, such as operational amplifiers. Thus, through the manipulations made through control systems, circuits with specific desired characteristics will be achieved, such as the stability of voltage or power values, through the integration of electrical modules, which can be connected, acting in a way analogous to a block diagram, reaching a so-called balanced output voltage or power of the system.**

Keywords **- Control systems, current, electrical circuits, manipulation, stability, voltage.**

I. INTRODUÇÃO

Todos e quaisquer dispositivos elétricos são alimentados por uma corrente elétrica e uma tensão. Através da manipulação destas duas grandezas, terá-se um comportamento elétrico desejado.

Uma solda de arco elétrico, por exemplo, consiste de uma elevada tensão elevadae corrente com um pequeno valor nominal. Dito isso, mesmo sendo requisitado pela solda um alto valor de tensão, se esta demorar a estabilizar e oscilar para valores ainda maiores, a qualidade da solda será prejudicada e dessa forma criará-se um ponto de tensão na estrutura metálica que, futuramente, poderá romper-se e causar um provável acidente, Wainer [1].

Destarte, as manipulações a serem feitas para obeter-se comportamentos desejados de tensão e corrente ocorrerão através dos sistemas de controle e suas funções de transferências. Portanto, procurou-se, através deste projeto e artigo, criar módulos elétricos que possam ser conectados analogamente a um diagrama de blocos, para que tais sistemas alcançem comportamentos ditos estáveis.

Portanto, ao desenvolver dos estudos, encontrou-se uma dificuldade em uma modelagem de um circuito elétrico no domínio da frequência tanto na parte teórica quanto no desenvolvimento dos cálculos, bem como a visualização desse sistema na aplicação da engenharia. Em circuitos mais simples, RLC, por exemplo, tem-se por objetivo um panorama geral, contudo quando parte-se para um circuito mais próximo da realidade, a complexidade aumenta e a dificuldade de se calcular e os erros se propagam exponencialmente.

Tendo em vista a facilidade e praticidade dos estudos, buscou-se no uso de linguagem de programação o início e a base da busca por essas soluções. Por se tratar de dados complexos e exigir gráficos, buscou-se na linguagem *python* as soluções para os problemas abordados anteriormente. Por se tratar de uma linguagem *open source*, flexível, robusta e é amplamente usada na análise científica, trazendo uma vantagem sobre as demais linguagens. Com o código desenvolvido, basta o usuário digitar as funções de transferência do circuito em questão, para que o algoritmo traga em seu *output* as soluções já calculadas.

II. FUNDAMENTAÇÃO TEÓRICA

A. *Modelagem no Dominío do Tempo*

As variavéis de estado de um sistema são encontradas através dos elementos armazenadores de energia. Nos circuitos elétricos, são o capacitor e o indutor. Como desenvolvido por Nise [2], obtem-se as seguintes equações de estado:

$$\dot{x} = Ax + Bu \tag{1}$$

$$y = Dx + Eu \tag{2}$$

$$A = \begin{bmatrix} 0 & 1 \\ -1/LC & -R/L \end{bmatrix} \tag{3}$$

$$B = \begin{bmatrix} 0 \\ 1/L \end{bmatrix} D = \begin{bmatrix} -1/C - R \end{bmatrix} \tag{4}$$

Onde:
x = vetor de estados
$\dot{x}$ = derivada do vetor de estado em relação ao tempo
y = vetor de saída
u = vetor de entrada ou de controle
A = matriz do sistema
B = matriz de entrada
D = matriz de saída
E = matriz de realimentação
C = capacitância
L = indutância

Em que A é a matriz de estado, B a de entrada, D de saída e E a matriz de alimentação direta, que está sendo considerada como zero por o sistema estar em estado estacionário no tempo igual a zero segundos.

B. Estabilidade

Um sistema é dito estável se a resposta natural aproximar-se de zero enquanto que o tempo tende ao infinito. Caso contrário o sistema é instável. O sistema é marginalmente estável quando estabiliza com um certo valor ou continua oscilando sem alcançar uma estabiliade.

C. Lugar das Raízes

O método do lugar das raízes permite que seja feita de a análise gráfica de um sistema. É especialmente útil para sistemas de malha fechada, em que ganhos podem alterar diretamente no lugar dos polos (além dos zeros) do sistema.

D. Linguagem Python

Python é uma linguagem de programação de alto nível, dinâmica, interpretada, modular, multiplataforma e também orientada a objetos, isto é, uma forma específica de organizar softwares onde a grosso modo, os procedimentos estão submetidos às classes, o que possibilita maior controle e estabilidade de códigos para projetos de grandes proporções, sendo esta a vertente utilizada para o desenvolvimento do script norteador do projeto.

Para a criação da calculadora, precisou-se pesquisar e percorrer uma sólida teoria sobre o assunto em questão. Dentre os assuntos abordados buscou-se aprofundar na modelagem de um sistema no domínio do tempo e da frequência, respostas no domínio do tempo, critérios de estabilidades, erros de regime permanente e lugar das raízes. Utilizou-se de recursos bibliográficos à documentações de pacotes *python* para maior aprofundamento nos conceitos e formas de se calcular o sistema.

III. METODOLOGIA

A abordagem introdutória ao tema decorreu-se através de pesquisas descritivas explanando-se bibliografias, artigos científicos e bibliotecas de controle em linguagem python que elaboram os sistemas de controle dos níveis de tensão, corrente e potência em um circuito ou rede elétrica. De forma quantitativa e comparativa, procurou-se compreender os métodos para se atingir, com êxito, a manipulação desejada e a constatação da viabilidade de se equilibrar tais níveis desejados no circuito.

A posteriori, desfrutando de tema, problemática e referências escolhidas, tendo estas como mola propulsora do estudo desenvolvido, buscou-se organizar e dissecar as formas e estudos necessários para controlar o sistema teórico e simulado dos circuitos elétricos de acordo com as informações e ensinamentos coletados nas bibliografias citadas [2 e 3], ambas discorrendo sobre os circuitos elétricos de primeira e segunda ordem e as aplicações necessárias para se estabilizar uma dada variável elétrica.

Posto isso, com usufruto das bibliografias referenciadas e citadas acima, além da linguagem de programação python, através da biblioteca *control*, buscou-se demonstrar e constatar, gráfico e algebricamente, a estabilização de uma dada variável no circuito ao decorrer do tempo e a interação com a realimentação deste circuito.

Por fim, para que fosse bem constatada e finalmente concretizada as elucidações e estudos a despeito das perturbações e viabilidade do controle de variáveis elétricas num circuito, abordou-se progressivamente a modelagem do circuito elétrico no domínio da frequência, diagramação de blocos e fluxo de sinais, comportamento no tempo, sistemas de primeira e seguda ordem, critérios de estabilidade, erros de estado estacionário e métodos para localização dos lugares das raízes.

IV. RESULTADOS

A. Bibliotecas

Buscou-se através do presente código realizar a análise de sistemas de controle quaisquer de forma prática, sendo estes de segunda ordem, utilizando-se a linguagem de programação python e tendo como principal referência a biblioteca *control*, além de outras bibliotecas fundamentais para análise matemática e gráfica, como *numpy* e *matplotlib*, respectivamente.

Abaixo, tem-se a importação de pacotes e bibliotecas necessários para o pleno funcionamento do script, como citado acima.

Bibliotecas.

```
import matplotlib.pyplot as plt
import control as ct
import numpy as np

import time
import warnings
```

```
%matplotlib inline

warnings.filterwarnings('ignore')
```

B. *Classe e função construtora*

Destarte, para função construtora da classe voltada à resolução do problema, tem-se as variáveis de estado de entrada necessárias para a determinação da função transferência do sistema, o ganho *K* e o sistema em si, obtido através das matrizes de estado.

Classe e função construtora.

```
class MyControl():

    def __init__(self):
        self.variaveisEstado = 0
        self.K = 0 # Ganho K
        self.sys = 0 # Sistema principal.
        self.controlSys = 0 # Sistema colocado para
            fazer o controle do principal.
        self.serie = 0 # Colocados em serie.
        self.feed = 0 # Processo final, ja
            adicionado o feedback.
        # Escolher criar o sistema pela funo de
            transferencia ou pelas variaveis de
            estado:
        self.A = 0
        self.B = 0
        self.C = 0
```

C. *Funções de transferência e o sistema*

Pode-se iniciar o estudo do sistema, por meio de interações com o usuário, através do fornecimento da função de transferência já pronta, como vemos no trecho de código abaixo:

Funções de transferência e o sistema.

```
    # Criar sistema a partir da funcao de
        transferencia.
    def criarSistemaTransf(self):
        num = []
        den = []
        nNum = int(input("Quantos coeficiente tem o
            numerador? "))
        nDen = int(input("Quantos coeficiente tem o
            denominador? "))
        print("Para ordens nos quais os coeficientes
            sao zero, informe zero!")
        for x in range(0, nNum):
            x = float(input("Informe, na ordem
                certa, os coeficientes do numerador:
                "))
            num.append(x)
        num = np.array(num)
        for x in range(0, nDen):
            x = float(input("Informe, na ordem
                certa, os coeficientes do
                denominador: "))
            den.append(x)
        den = np.array(den)
        self.sys = ct.tf([[num]], [[den]])
        print(self.sys)
```

D. *Variáveis de estado e o sistema*

Pode-se iniciar o estudo do sistema, também, por meio de interações com o usuário, através do fornecimento das variáveis de estado que compõem o sistema, como vemos no trecho de código abaixo:

Funções de transferência e o sistema.

```
    # Criar sistema a partir das equacoes de estado.
    def criarSistemaEstado(self):
        # Cria as matrizes de estado. Respeitar a
            ordem dos inputs de acordo com a matriz
            de referencia.
        self.variaveisEstado = int(input("Sao
            quantas variveis de estado? "))

        # Matriz de estado A
        listA = []
        for x in range(0,int(self.variaveisEstado*
            self.variaveisEstado)):
            x = int(input("Informe, na ordem certa,
                os valores da matriz de estado: "))
            listA.append(x)
        self.A = np.array(listA, dtype = float)
        self.A.resize(self.variaveisEstado,
            self.variaveisEstado)
        self.A = self.A.tolist()
        print("Matriz de estado: \n",self.A)

        # Matriz de entrada B
        listB = []
        for x in range(0, int(self.variaveisEstado)):
            x = int(input("Informe, na ordem certa,
                os valores da matriz de entrada: ",))
            listB.append(x)
        self.B = np.array(listB, dtype = float)
        self.B.resize(self.variaveisEstado, 1)
        self.B = self.B.tolist()
        print("Matriz de entrada: \n", self.B)

        # Matriz de sada C
        listC = []
        for x in range(0, int(self.variaveisEstado)):
            x = int(input("Informe, na ordem certa,
                os valores da matriz de saida: ",))
            listC.append(x)
        self.C = np.array(listC, dtype = float)
        self.C.resize(1, self.variaveisEstado)
        self.C = self.C.tolist()
        print("Matriz de saida C: \n",self.C)
        self.sys = StateSpace(self.A, self.B,
            self.C, 0)
        print(self.sys)
```

```
        self.sys = ct.tf(self.sys)
        print(self.sys)
```

E. *Análise isolada do sistema obtido*

Posteriormente, analisa-se o sistema capturado pela interação com o usuário, obtendo sua resposta natural e posições de polos e zeros, caso existam, no plano complexo:

Análise isolada do sistema.

```
    def analiseIsolada(self):
        print("Plot da funcao de transferencia: \n")
        time.sleep(0.5)
        plt.figure(1)
        fig, ax = plt.subplots()
        yout, T = impulse(self.sys)
        plt.plot(T.T, yout.T)
        ax.set_title("Resposta Natural")
        ax.set_ylabel('volts')
        ax.set_xlabel('tempo(s)')
        plt.show(block=False)
        time.sleep(1)

        # Polos e zeros.
        #print("Polos e zeros: \n")
        time.sleep(0.5)
        #damp(self.sys)
        #pzmap(self.sys)
        time.sleep(1)

        # Root-Locus (Lugar das raizes)
        print("Lugar das raizes: \n")
        time.sleep(1)
        rlocus(self.sys)
```

Tem-se, portanto, para a função de transferência descrita abaixo, de acordo com o trecho de código acima, a resposta natural do sistema assim como o mapeamento de seus polos e zeros no plano complexo:

$$\frac{-24s+24}{s^2+23s-24}$$

Figura 1: Resposta natural para função transferência.

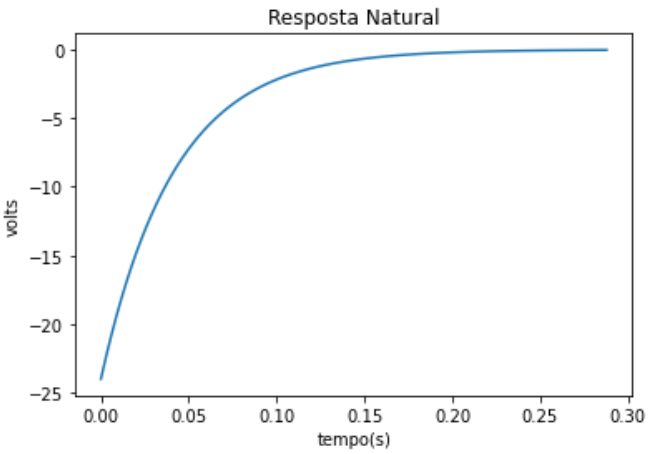
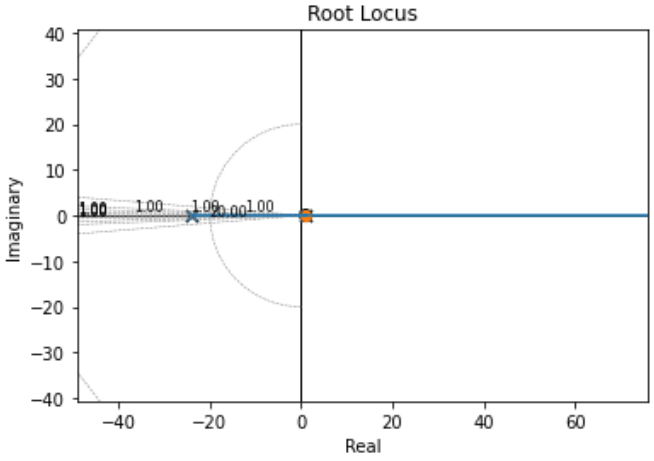


Figura 2: Polos e zeros no plano complexo.



F. *Realimentação do sistema*

Desta forma, pode-se aplicar uma realimentação, caso necessário, à resposta natural com o método *feedBackSimples()*. Pode-se, por exemplo, adicionar uma entrada degrau $\frac{1}{s}$ ou qualquer outro sistema que seja usado para fazer o controle. Dito isso adiciona-se a realimentação para que seja feita a análise de lugar das raízes na qual avaliamos a estabilidade do sistema dado um ganho K.

Realimentação do sistema.

```
    def feedBackSimples(self):
        self.feed = ct.feedback(self.sys, sys2=1,
            sign= 1)
        print("Funcao de transferencia com
            realimentacao: \n")
        print(self.feed)
        time.sleep(2)
        print("Plot com realimentacao: \n")
        time.sleep(1)
        plt.figure(2)
        fig, ax = plt.subplots()
        youf, T = impulse(self.feed)
        plt.plot(T, youf.T)
        ax.set_title("Realimentacao")
        ax.set_ylabel('volts')
        ax.set_xlabel('tempo(s)')
        plt.show()
        time.sleep(2)
        #print("Polos e zeros: \n")
        time.sleep(1)
        #damp(self.feed)
        #pzmap(self.feed)
        time.sleep(1)

        print("Lugar das raizes com realimentacao:
            \n")
        ct.rlocus(self.feed)
```

G. *Adição de sistemas*

Assim, dado um outro sistema, obtido através da interação com o usuário, neste caso $\frac{1}{s}$ ou degrau unitário, adicionando-o à resposta natural com os métodos *sistemaControle()* e *emSerie()*, respectivamente, obtem-se uma resposta de estabilização ainda mais rápida.

Adição de um novo sistema qualquer.

```
def sistemaControle(self):
    num = []
    den = []
    nNum = int(input("Quantos coeficiente tem o
        numerador? "))
    nDen = int(input("Quantos coeficiente tem o
        denominador? "))
    print("Para ordens nos quais os coeficientes
        sao zero, informe zero!")
    for x in range(0,nNum):
        x = float(input("Informe, na ordem
            certa, os coeficientes do numerador:
            "))
        num.append(x)
    num = np.array(num)
    for x in range(0,nDen):
        x = float(input("Informe, na ordem
            certa, os coeficientes do
            denominador: "))
        den.append(x)
    den = np.array(den)
    self.controlSys = ct.tf([[num]], [[den]])
    print(self.controlSys)

# Anlise desse sistema junto a resposta natural
def emSerie(self):
    self.serie = ct.series(self.controlSys,
        self.sys)
    print(self.serie)
    print("Plot em serie: \n")
    time.sleep(1)
    plt.figure(3)
    fig, ax = plt.subplots()
    yout, T = impulse(self.serie)
    plt.plot(T.T, yout.T)
    ax.set_xlabel('tempo(s)')
    ax.set_ylabel('volts')
    ax.set_title('Resposta Degrau')
    plt.show(block=False)
    time.sleep(1)
    print("Polos e zeros: ")
    time.sleep(1)
    #damp(self.serie)
    #pzmap(self.serie)
    time.sleep(1)
    print("O esperado  que em 'Open-loop' os
        ganhos nao alterem no lugar das
        raizes.")
    time.sleep(1)
    ct.rlocus(self.serie)
```

Portanto, a resposta ao degrau e o mapeamento de polos e zeros no plano complexo obtidos com uma redução de 10 vezes no tempo de estabilização, através da entrada degrau unitária, são:

Figura 3: Resposta ao degrau unitário.

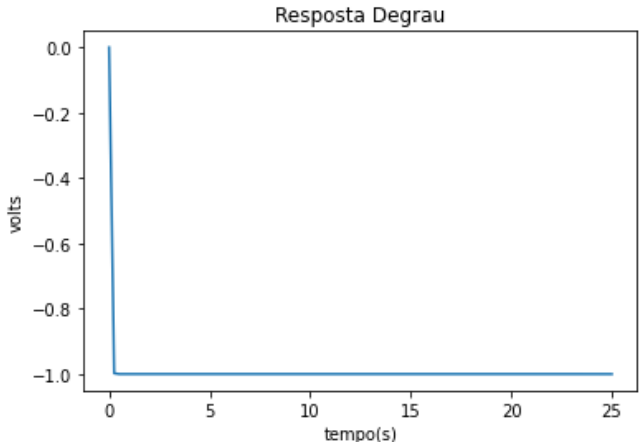
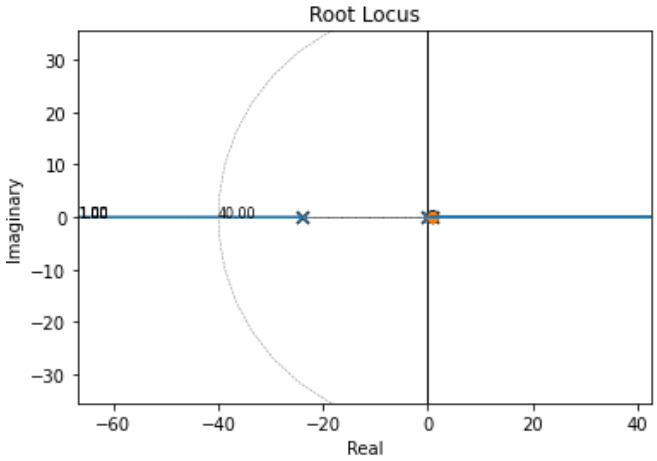


Figura 4: Polos e zeros no plano complexo.



H. *Realimentação do novo sistema*

Por fim, é possível também adicionar uma realimentação ao novo sistema, obtido através da adição do degrau unitário ao sistema original.

Realimentação do sistema final.

```
self.serie = ct.series(self.controlSys,
    self.sys)
print(self.serie)
print("Plot em serie: \n")
time.sleep(1)
plt.figure(3)
fig, ax = plt.subplots()
yout, T = impulse(self.serie)
```

```
        plt.plot(T.T, yout.T)
        ax.set_xlabel('tempo(s)')
        ax.set_ylabel('volts')
        ax.set_title('Resposta Degrau')
        plt.show(block=False)
        time.sleep(1)
        print("Polos e zeros: ")
        time.sleep(1)
        #damp(self.serie)
        #pzmap(self.serie)
        time.sleep(1)
        print("O esperado  que em 'Open-loop' os
            ganhos nao alterem no lugar das
            raizes.")
        time.sleep(1)
        ct.rlocus(self.serie)

    def emSerieComFeedBack(self):
        self.feed = ct.feedback(self.serie, sys2=1,
            sign= 1)
        print(self.feed)
        print("Plot em srie e com realimentacao: \n")
        time.sleep(1)
        plt.figure(1)
        fig, ax = plt.subplots()
        yout, T = impulse(self.feed)
        plt.plot(T.T, yout.T)
        ax.set_xlabel('tempo(s)')
        ax.set_ylabel('volts')
        ax.set_title('Resposta com Realimentacao')
        plt.show(block=False)
        time.sleep(1)
        #print("Polos e zeros: \n")
        #damp(self.feed)
        #pzmap(self.feed)
        time.sleep(1)
        print("O esperado eh que em 'Closed-loop' os
            ganhos alterem no lugar das raizes.")
        ct.rlocus(self.feed)
```

V. CONCLUSÃO

Conclui-se que, as manipulações e obtenção da função de transferência para atingir-se comportamentos desejados, ditos estáveis, para as grandezas elétricas em circuitos é passível de ser realizada de forma simulada algébrica e graficamente, através da linguagem de programação python debruçada, principalmente, sob o auxílio da biblioteca *control*, como elucidam os resultados apresentados acima.

Constata-se também que a manipulação e acréscimo de novos sistemas, ganhos ou realimentaões ao sistema original e seus sistemas subsequentes trazem comportamentos controlados em uma taxa de tempo médio por estabilização cada vez menores, ou seja, melhores. Sendo o script como um todo uma ferramenta de grande utilidade para controle algébrico e estudo de sistemas a ele aplicados.

Em uma solda, por exemplo, é possível obter-se resultados de tensão estáveis através dos ganhos de tensão de entrada. Em casos mais especifícos, é possível alcançar a estabilidade do arco elétrico mais rápido caso seja aplicado uma realimentação. São todas possibilidades que podem ser testadas e vislumbradas de forma prática com a aplicação dos métodos desenvolvidos neste trabalho.

Ademais, estuda-se a possibilidade do interfaceamento do script, facilitando a usabilidade homem-máquina do mesmo. Tal interface será desenvolvida através do framework Django, utilizando linguagem python e prototipado via aplicação Figma, para estudos da interface e experiência de usuários (UI e UX).

Buscará-se, para a ampliação dessa calculadora, o uso de inteligência artificial com *machine learning* e *deep learning*. A posteriori, a criação de redes neurais para o processamento de uma imagem, para que não precise digitar a equação, ampliando assim apenas a foto do circuito. O software buscaria na base de dados soluções existentes semelhantes ao modelo que se queira resolver e resolveria o problema. Destaca-se também a criação de um aplicativo *mobile* e *desktop* com interfaces gráficas e melhor interação com o usuário.

Para isso, as ferramentas a serem usadas são: *python*, para o desenvolvimento de *machine learning*, Kotlin e Java para o desenvolvimento *mobile*, todos eles integrados a um banco de dados SQL. Também destaca-se o uso de Docker com uso de *containers* para subir as aplicações Java e Kotlin, Maven e Spring Boot. A parte de *frontend* também pode ser feita usando recursos web com o uso de linguagens HTML, CSS, e *frameworks* Flask ou Django.

REFERÊNCIAS

[1] WAINER, Emílio. *Soldagem: processos e metalurgia. 4ª edição. Editora Edgar Blucher LTDA*. São Paulo, SP, 2004.

[2] NISE, Norman S. *Engenharia de sistemas de controle. 6ª edição. GEN - grupo editorial nacional.* Rio de Janeiro, 2012.

[3] DORF, Richard C. *Sistemas de controle modernos. 8ª edição. LTC Editora.* Rio de Janeiro, RJ, 2001.

[4] SADIKU Matthew N. O. *Fundamentals of electric circuits. 5ª edição. Connect Learn Succeed.* New York, NY, 2013.

[5] Python Control Systems Toolbox. Disponível em: `https://python-control.readthedocs.io/en/0.8.3/intro.html`

[6] Numpy documentation. Disponível em: `https://numpy.org/doc/`

[7] Matplotlib: Visualization with Python. Disponível em: `https://matplotlib.org/`

[8] Script completo desenvolvido no projeto. Disponível em: `https://github.com/adrohms/ArtigoSC`