UNIVERSIDADE FEDERAL DO CEARÁ

CENTRO DE TECNOLOGIA

DEPARTAMENTO DE ENGENHARIA ELÉTRICA

CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

VINÍCIUS REBOUÇAS CORTÊZ

USO DE APRENDIZADO DE MÁQUINA NA ESTIMAÇÃO DE PARÂMETROS DO CONTROLADOR PID-2DOF APLICADO NO CONTROLE DE VELOCIDADE DE UM MOTOR DE INDUÇÃO TRIFÁSICO

FORTALEZA

2023

VINÍCIUS REBOUÇAS CORTÊZ

USO DE APRENDIZADO DE MÁQUINA NA ESTIMAÇÃO DE PARÂMETROS DO CONTROLADOR PID-2DOF APLICADO NO CONTROLE DE VELOCIDADE DE UM MOTOR DE INDUÇÃO TRIFÁSICO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Orientadora: Prof. Dra. Laurinda Lúcia Nogueira dos Reis

Coorientador: Dr. Darielson Araújo de Souza

FORTALEZA

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

C858u Cortêz, Vinícius Rebouças.
Uso de Aprendizado de Máquina na Estimação de Parâmetros do Controlador PID-2DOF Aplicado no Controle de Velocidade de um Motor de Indução Trifásico / Vinícius Rebouças Cortêz. – 2023.
62 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Centro de Tecnologia, Curso de Engenharia Elétrica, Fortaleza, 2023.
Orientação: Profa. Dra. Laurinda Lúcia Nogueira dos Reis.
Coorientação: Prof. Dr. Darielson Araújo de Souza.

1. Machine Learning. 2. PID-2DOF. 3. Controle. 4. Braço Mecânico. I. Título.

CDD 621.3

VINÍCIUS REBOUÇAS CORTÊZ

USO DE APRENDIZADO DE MÁQUINA NA ESTIMAÇÃO DE PARÂMETROS DO CONTROLADOR PID-2DOF APLICADO NO CONTROLE DE VELOCIDADE DE UM MOTOR DE INDUÇÃO TRIFÁSICO

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia Elétrica do Centro de Tecnologia da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia Elétrica.

Aprovada em:

BANCA EXAMINADORA

Prof. Dra. Laurinda Lúcia Nogueira dos Reis (Orientadora)
Universidade Federal do Ceará (UFC)

Dr. Darielson Araújo de Souza (Coorientador)
Instituto Federal de Educação, Ciência e Tecnologia do Ceará (IFCE)

Prof. Dr. Josias Guimarães Batista
Instituto Federal do Ceará (IFCE)

À minha família, por sua capacidade de acreditar em mim e investir em mim. Mãe, seu cuidado e dedicação foi que deram, em alguns momentos, a esperança para seguir. Pai, sua presença significou segurança e certeza de que não estou sozinho nessa caminhada. Irmã sua companhia e presença sempre foram cruciais para tudo.

AGRADECIMENTOS

Primeiramente à minha família por me apoiarem ao longo dessa jornada, sem eles não chegaria nem na metade de onde estou.

À Prof. Dr. Laurinda Lúcia Nogueira dos Reis por me orientar neste trabalho e nos conhecimentos cruciais para conclusão dela.

Ao Dr. Darielson Araújo de Souza pela ajuda e prontidão sempre presentes.

À todos os amigos que fiz ao longo do curso e todo o conhecimento e momentos trocados.

Ao Grupo de Pesquisa em Automação, Controle e Robótica (GPAR) por permitir o contato pratico com o mundo do controle.

Ao Alanio F. Lima por ser parceiro na caminhada durante minha estadia no laboratório.

Agradeço à todos os professores por me proporcionar o conhecimento não apenas racional, mas a manifestação do caráter e afetividade da educação no processo de formação profissional, por tanto que se dedicaram a mim, não somente por terem me ensinado, mas por terem me feito aprender.

E por ultimo mas não menos importante, à todos os demais que de alguma maneira me guiaram, ajudaram ou apoiaram para chegar aqui.

“O insucesso é apenas uma oportunidade para recomeçar com mais inteligência.”

(Henry Ford)

RESUMO

Este estudo investiga o uso de *Machine Learning (ML)* na estimação dos parâmetros de um controlador Proporcional-Integral-Derivativo de *2 Degree of Freedom* (2 Graus de Liberdade) aplicado a um motor de indução trifásico acoplado a um sistema de controle de um braço robótico cilíndrico. Ao usar métodos de sintonia clássico de um controlador PID-2DOF como ponto de partida, em que verificou-se os resultados em termos das seguintes especificações como, tempo de subida, tempo de estabelecimento e sobressinal. Os Os resultados através dos testes realizados mostram que o modelo de *Machine Learning* fornece resultados com uma precisão adequada, sendo capaz de identificar parâmetros que aumentam a velocidade e robustez do controle do sistema. Este trabalho ressalta a contribuição no uso algoritmo do *Machine Learning* para a utilização em sistemas de controle, destacando-se sua aplicação em determinados cenários de teste e fornecendo uma base para futuras pesquisas e melhorias, incluindo a exploração de diferentes conjuntos de dados e modelos de *Machine Learning*.

Palavras-chave: *Machine Learning*. PID-2DOF. Controle. Braço mecânico

ABSTRACT

This study investigates the use of *Machine Learning* (ML)in estimating the parameters of a Proportional-Integral-Derivative 2 Degree of Freedom controller applied to a three-phase induction motor coupled to a cylindrical robotic arm control system. Using classical tuning methods for a PID-2DOF controller as a starting point, the results were checked in terms of the following specifications: rise time, settling time and overshoot. The results of the tests carried out show that the *Machine Learning* model provides results with adequate precision, being able to identify parameters that increase the speed and robustness of the system's control. This work highlights the contribution of the *Machine Learning* algorithm for use in control systems, highlighting its application in certain test scenarios and providing a basis for future research and improvements, including the exploration of different data sets and *Machine Learning* models.

Keywords: *Machine Learning*. PID-2DOF. Control. Mechanical arm

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

GPAR	Grupo de Pesquisa em Automação, Controle e Robótica
ML	*Machine Learning*
PID-2DOF	Proporcional-Integral-Derivativo de 2 Graus de Liberdade
PID	Proporcional-Integral-Derivativo
MLP	*Multi-Layer Perceptron*
ExtraTrees	*Extremely Randomized Trees*
TCC	Trabalho de Conclusão de Curso
GDL	Graus de Liberdade
MSE	*Mean Squared Error*
MAE	*Mean Absolute Error*
RMSE	*Root Mean Squared Error*
UFC	Universidade Federal do Ceará
DSP	*Digital Signal Processing*
ReLU	*Rectified Linear Unit*

LISTA DE SÍMBOLOS

A	Amperes
V	Volts
cv	Cavalo-Vapor
s	Frequência na Transformada de Laplace
RPM	Rotação Por Minuto

SUMÁRIO

1 INTRODUÇÃO

A área de engenharia elétrica tem sido marcada por rápidos avanços tecnológicos, com o intuito de otimizar e aprimorar os sistemas de controle de motores elétricos. Nesse contexto, o presente Trabalho de Conclusão de Curso (TCC) tem como objetivo explorar a aplicação de técnicas de *Machine Learning (ML)*, ou aprendizado de máquina, para a estimação de parâmetros de um controlador Proporcional-Integral-Derivativo de *2 Degree of Freedom* (2 Graus de Liberdade) em sistemas de controle de motores elétricos.

Os controladores do tipo Proporcional-Integral-Derivativo (PID) são amplamente utilizados na indústria para garantir um controle preciso e eficiente dos motores elétricos (FACCIN, 2004, p. 2-3). No entanto, a tarefa de ajustar os parâmetros dos controladores PID de forma otimizada representa um desafio, principalmente devido à complexidade dos sistemas de controle de motores elétricos.

Diante dessa problemática, a utilização de técnicas de ML tem se mostrado promissora, pois permite automatizar o processo de estimação dos parâmetros PID-2DOF. Com o emprego de algoritmos de aprendizado de máquina, é possível analisar grandes volumes de dados e identificar padrões, possibilitando um ajuste mais eficiente dos controladores.

1.1 Motivação

Este trabalho visa contribuir para o avanço das técnicas de controle de motores elétricos, proporcionando um melhor desempenho e otimização dos sistemas de controle existentes.

1.2 Problemática

A dificuldade na otimização dos parâmetros dos controladores PID-2DOF em sistemas de controle de motores elétricos motiva a busca por abordagens inovadoras, como o uso de ML.

1.3 Objetivo Geral

Explorar a aplicação de técnicas de ML na estimação de parâmetros de controladores PID-2DOF em sistemas de controle de motores elétricos.

1.4 Objetivos Específicos

- Realizar uma análise detalhada da aplicação de ML na estimação de parâmetros PID-2DOF.
- Comparar os resultados obtidos por técnicas tradicionais e técnicas baseadas em ML.
- Verificar melhorias na precisão e eficiência da estimação de parâmetros PID-2DOF por meio do uso de técnicas de aprendizado de máquina.

2 FUNDAMENTAÇÃO TEÓRICA

O controle de sistemas de motores elétricos sempre foi algo primordial no meio cientifico e empresarial, sendo amplamente demandados e utilizados, fazendo-se então a sua otimização algo de particular importância (RITA, 2016, p. 1-3). Tendo em vista isso será abordado a seguir os principais conceitos necessários para realização deste TCC.

2.1 Principais Tipologias de Controladores

Serão abordadas a seguir as principais topologias de controladores encontradas na literatura.

2.1.1 Controlador Proporcional-Integral-Derivativo

O controlador PID é um dos mais utilizados na indústria devido à sua simplicidade e eficiência no controle de sistemas lineares e não lineares. Ele é caracterizado pela combinação de três ações de controle: proporção, integral e derivativo (ÅSTRÖM; HÄGGLUND, 2009).

- A ação proporcional (P) é responsável por ajustar a saída do controlador de acordo com o erro presente entre o valor de referência e o valor medido. Isso permite que o controlador minimize o erro presente no sistema de controle.
- A ação integral (I) integra o erro ao longo do tempo, corrigindo desvios sustentados do valor de referência. Essa ação é importante para eliminar erros de offset, que podem ocorrer devido a variações nos parâmetros do sistema.
- A ação derivativa (D) analisa a taxa de variação do erro e ajusta a saída do controlador para evitar oscilações rápidas ou instabilidades. Ela é vital especialmente em sistemas nos quais a resposta rápida é essencial.

O controlador PID é amplamente utilizado na indústria por ser capaz de fornecer uma resposta rápida e precisa em sistemas de controle de motores elétricos. Sua base teórica é bem estabelecida e amplamente estudada, o que permite sua implementação eficiente em diferentes aplicações.

2.1.2 Controlador Proporcional-Integral-Derivativo de 2 Graus de Liberdade

O controlador Proporcional-Integral-Derivativo de *2 Degree of Freedom* (2 Graus de Liberdade) é uma extensão do controlador PID padrão que foi desenvolvida para solucionar algumas limitações identificadas no PID convencional. Em um controlador PID-2DOF, são adicionados dois Graus de Liberdade (GDL): um para a ação proporcional e integral (PI) e outro para a ação derivativa (D) (DASH *et al.*, 2014, p. 4).

Essa abordagem permite que diferentes pesos sejam atribuídos às ações PI e D, o que melhora o desempenho e a estabilidade do sistema de controle. Com o controlador PID-2DOF, é possível ajustar individualmente as ações PI e D para atingir um melhor desempenho e resposta do sistema.

Uma aplicação comum desses controladores é na regulação da velocidade de motores elétricos, tanto em aplicações industriais como em veículos elétricos. O ajuste adequado dos parâmetros do controlador PID ou PID-2DOF permite alcançar uma resposta rápida às mudanças de carga, melhorando assim a eficiência energética e reduzindo o desgaste mecânico do motor (BINGI *et al.*, 2018).

2.1.3 Domínio de tempo do sistema

Dentro do mundo de controle existem dois domínios de tempo a serem trabalhados, o continuo e o discreto, e a escolha sobre em qual domínio trabalhar é crucial para o sucesso do projeto. O controle de tempo contínuo é geralmente mais fácil de analisar matematicamente e pode ser mais intuitivo em certos casos, porém em muitos casos, não descreve bem o modelo desejado. Enquanto o controle de tempo discreto é frequentemente utilizado em sistemas digitais e computacionais e costuma representar bem o sistema.

Uma técnica muito comum é de se trabalhar em domínio de tempo continuo mas discretizar o sinal ao final, especificamente para sistemas de controle PID e PID-2DOF o método de Tustin para discretização demonstra um bom meio termo entre esforço e desempenho (SOARES, 1996, p. 112-113).

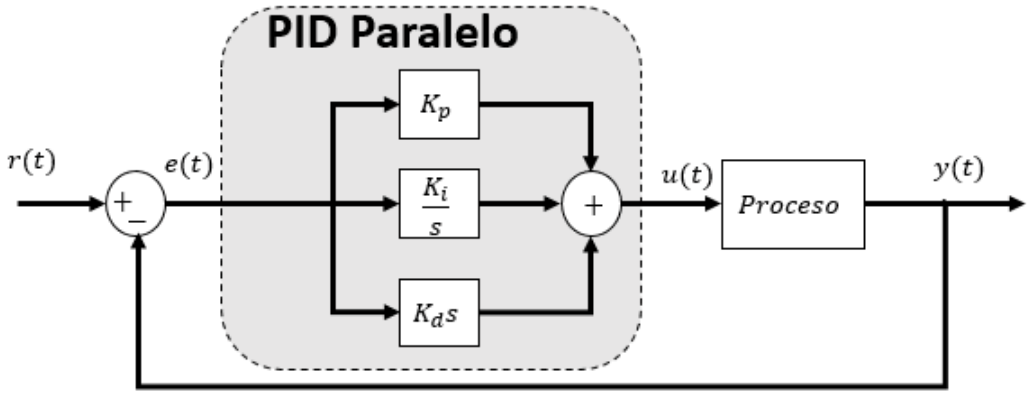
O método de Tustin, também conhecido como discretização Bilateral, consiste em substituir a frequência de Laplace s pela equação abaixo, onde Ts é o tempo de amostragem.

$$s = \frac{2}{T_s}\frac{z-1}{z+1} \tag{2.1}$$

2.1.4 Equacionamento do Sistema

O controle a ser estudado neste projeto será um controlador PID paralelo clássico com um incremento de um controlador auxiliar para dar o segundo GDL para que ao combinar os dois será obtido o controlador PID-2DOF desejado. O controlador PID será o referente ao diagrama da Figura 1 que pode ser descrito pela equação 2.4 (ÅSTRöM; HäGGLUND, 1995).

Figura 1 – Diagrama do Controlador PID Clássico.



Fonte: Própria autoria.

Onde podemos definir que:

$$K_i = \frac{K_p}{T_i} \tag{2.2}$$

$$K_d = K_p \times T_d \tag{2.3}$$

Logo a equação do controlador PID clássico é:

$$C(s) = K_p + \frac{K_i}{s} + K_d s \tag{2.4}$$

Agora será adicionado um controlador auxiliar X(s) para formar o PID-2DOF juntamente com os pontos de ajuste para superar as limitações do PID clássico, de maneira que as funções de transferência da planta, do controlador PID e da controlador auxiliar do PID-2DOF serão as equações 3.1, 2.7 e 2.8 respectivamente. De maneira tal que o sistema será formado pelo diagrama da Figura 2 (HELENO *et al.*, 2022). Além do controlador auxiliar será adicionado na porção Derivativa do controlador um filtro derivativo de primeira ordem para reduzir ruídos e melhorar a resposta do sistema.

$$C(s) = bK_p + \frac{K_i}{s} + \frac{cK_d s}{T_f s + 1} \tag{2.5}$$

$$X(s) = (1-b)K_p + \frac{(1-c)K_d s}{T_f s + 1} \tag{2.6}$$

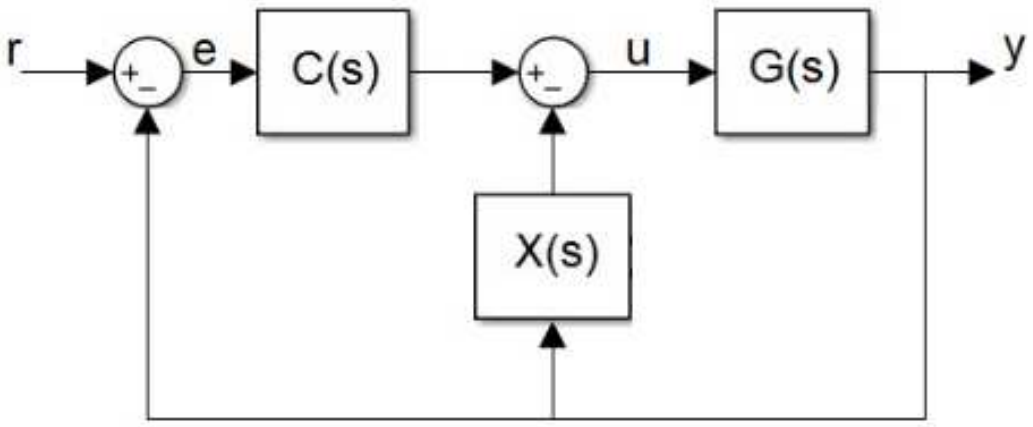
Agora, aplicando a discretização Bilinear, ou método de Tustin, em ambas equações anteriores, ou seja, fazendo a substituição da equação 2.1 e simplificando, é encontrado que:

$$C(z) = bK_p + \frac{K_i T_s z + K_i T_s}{2z - 2} + \frac{2cK_d z - 2cK_d}{(2T_f + T_s)z + (T_s - 2T_f)} \tag{2.7}$$

$$X(z) = (1-b)K_p + \frac{2(1-c)K_d z - 2(1-c)K_d}{(2T_f + T_s)z + (T_s - 2T_f)} \tag{2.8}$$

O Ts adotado para todo o projeto será de 0,2 segundos, escolhido devido à ser o valor padrão utilizado em outros projetos dentro da mesma bancada utilizada.

Figura 2 – Diagrama do Controlador PID-2DOF.



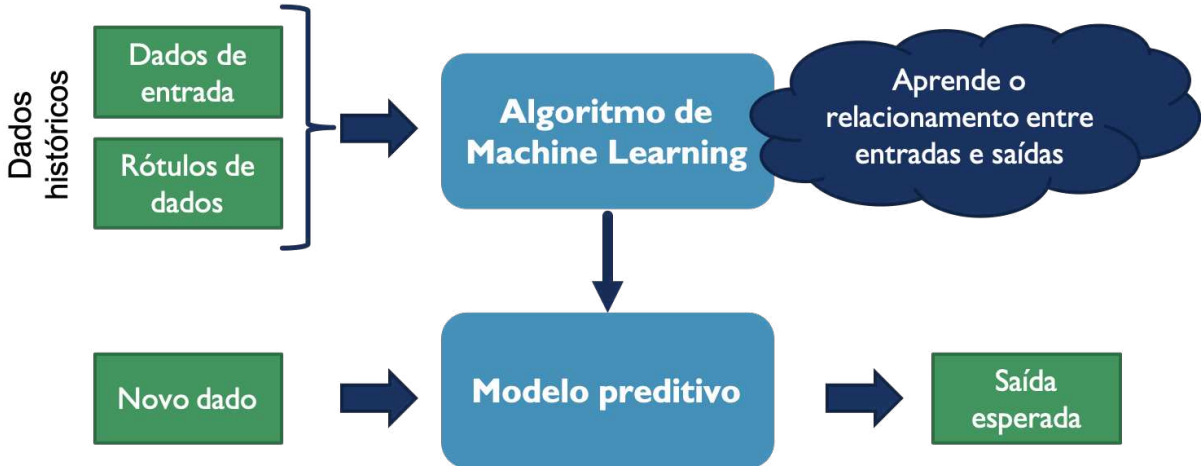
Fonte: (HELENO *et al.*, 2022).

2.2 Algoritmo de *Machine Learning*

Machine Learning, ou inteligencia computacional aplicada, consiste no desenvolvimento de algoritmos que permitem aos sistemas aprenderem e melhorarem seu desempenho em tarefas específicas a partir da análise de dados, sem programação explícita (ZHOU, 2021). Em vez de seguir regras fixas, os sistemas de ML são treinados usando exemplos e dados,

permitindo-lhes fazer previsões, tomar decisões ou identificar padrões com base em informações históricas, como mostrado na figura 3.

Figura 3 – Fluxograma de treinamento de ML.



Fonte: (ESCOVEDO, 2020).

Os modelos de ML podem ser divididos entre modelos de regressão ou classificação. Onde regressão consiste em uma analise numérica para descrever uma relação linear ou não das entradas com as saídas, permitindo fazer analises preditivas. Já a classificação consiste em separar conjuntos de dados com base nas características de suas entradas, permitindo fazer diferenciações e agrupamentos de dados, além das suas classificações (BREIMAN, 2017).

Além disso os modelos de ML podem ser divididos entre modelos de treinamento supervisionado e não-supervisionado. Onde para treinamentos supervisionados o sistema é treinado com amostras de dados de saídas conhecidas, enquanto não-supervisionadas o sistema é treinado sem essas amostras de saídas desejadas conhecidas (BATISTA *et al.*, 2003).

2.2.1 Principais Tipologias de Modelos de ML

Uma das técnicas de ML utilizadas para modelar e prever comportamentos é o modelo de Árvore de Decisão. Esse modelo consiste em uma estrutura de árvore em que cada nó interno representa uma decisão a ser tomada, com base em uma certa condição, e cada nó folha representa o resultado ou classificação final (MONARD; BARANAUSKAS, 2003). O modelo de Árvore de Decisão pode ser utilizado tanto para problemas de regressão, onde busca-se prever um valor numérico, quanto para problemas de classificação, onde busca-se agrupar os dados em categorias.

Outra técnica importante é o *Ensemble Learning*, que consiste em combinar vários

modelos de ML para obter um modelo mais robusto e geralmente mais preciso. Existem diferentes métodos de *Ensemble Learning*, como o *Boosting* e *Random Forest* (OSHIRO, 2013). O *Boosting*, por sua vez, treina modelos sequencialmente, onde cada modelo tenta corrigir os erros do modelo anterior. Já o *Random Forest* é um método que utiliza múltiplas árvores de decisão para realizar a classificação ou regressão e combina seus resultados através de votação ou média (OSHIRO, 2013).

Os principais modelos de ML e os analisados no TCC são os abordados abaixo:

O *Multi-Layer Perceptron* (MLP)é um tipo de rede neural artificial formada por múltiplas camadas de neurônios interconectados. Cada neurônio recebe informações dos neurônios da camada anterior e passa adiante a sua própria saída para os neurônios da camada seguinte. Essa estrutura permite ao MLP aprender e extrair características de padrões complexos nos dados. É um algoritmo bastante utilizado para classificação e regressão em problemas de aprendizado supervisionado (TAUD; MAS, 2018). As redes MLP utilizam a técnica de *backpropagation*, ou retro-propagação onde o valor do erro do treinamento é utilizado de maneira retroativa para ajustar os pesos durante o treinamento.

O algoritmo de *Bagging* com uso de árvores de decisão é uma técnica de *Ensemble Learning*, onde várias instâncias de um mesmo algoritmo são treinadas em paralelo e suas respostas são combinadas para chegar a um resultado final (BREIMAN, 1996, p. 5-8). No caso específico de *Bagging*, múltiplas árvores de decisão são treinadas em diferentes conjuntos de dados amostrais, que são obtidos através de *bootstrap*. A combinação dos resultados das árvores ajuda a reduzir o *overfitting*, aumentando a precisão de classificação.

O algoritmo de *Extremely Randomized Trees (ExtraTrees)* é uma extensão do algoritmo *Random Forest*, considerado um modelo de *Ensemble Learning*. Ele utiliza múltiplas árvores de decisão, como no *Random Forest*, mas de forma mais aleatória. Ao construir cada árvore, uma amostra aleatória de características é selecionada, além de serem considerados apenas um subconjunto dos dados de treinamento. Esse método cria modelos mais diversos e promove o aumento da eficiência computacional (GEURTS *et al.*, 2006, p. 5-7).

O *Voting* é uma técnica essencial no *Ensemble Learning*, que combina previsões de diversos modelos de ML para aprimorar a precisão e a robustez das decisões. Semelhante ao algoritmo ExtraTrees, que cria árvores de decisão de forma aleatória para diversificar os modelos, o *Voting* reúne previsões de diferentes modelos, cada um com abordagens distintas (PHYO *et al.*, 2022). A diversidade resultante reduz o viés e a variância nas previsões, tornando o resultado

mais confiável, sendo essencial escolher modelos diversificados para melhorar o desempenho do *ensemble*.

2.2.2 Otimização de modelo de ML

Os hiperparâmetros em um modelo de acrfullML referem-se aos parâmetros ou configurações que não podem ser aprendidos automaticamente pelo algoritmo durante o treinamento. Eles são definidos pelo usuário antes que o treinamento comece e podem afetar significativamente o desempenho do modelo (LOPES *et al.*, 1996).

Existem diversos tipos de hiperparâmetros que podem ser encontrados em diferentes algoritmos de ML, incluindo o número de camadas e neurônios em uma rede neural, a taxa de aprendizado utilizada no processo de treinamento, os critérios de parada e os métodos de regularização. Esses hiperparâmetros precisam ser ajustados de forma adequada para otimizar os resultados do modelo (REIS, 2021).

Um algoritmo muito utilizado para otimização de hiperparâmetros é a otimização bayesiana. A otimização bayesiana é uma abordagem que busca encontrar os valores ideais dos hiperparâmetros através de um processo iterativo de tentativa e erro. Nesse processo, uma função de perda é definida para medir o desempenho do modelo com determinados hiperparâmetros (CROCOMO; DELBEM, 2011). Com base nessa função, a otimização bayesiana busca encontrar a melhor combinação de hiperparâmetros para minimizar ou maximizar essa função.

A otimização bayesiana utiliza uma técnica chamada modelo de regressão bayesiana para estimar a função de perda em relação aos hiperparâmetros e gerar sugestões de próximas configurações a serem testadas. Essas sugestões são obtidas a partir de uma distribuição probabilística que considera o conhecimento prévio sobre os resultados de configurações já testadas (FRAZIER, 2018). Dessa forma, a otimização bayesiana é capaz de explorar diferentes conjuntos de hiperparâmetros de forma mais eficiente do que métodos tradicionais de busca exaustiva.

Em resumo, os hiperparâmetros em um modelo de ML são configurações que afetam o desempenho do modelo, mas que não são aprendidos automaticamente durante o treinamento (SNOEK *et al.*, 2012). A otimização bayesiana é um algoritmo utilizado para encontrar os melhores hiperparâmetros através de um processo iterativo de tentativa e erro, utilizando um modelo de regressão bayesiana para estimar a função de perda e fornecer sugestões de melhores configurações a serem testadas. Essa abordagem é mais eficiente em relação a métodos tradicionais de busca exaustiva.

2.2.3 Avaliação de modelo de ML

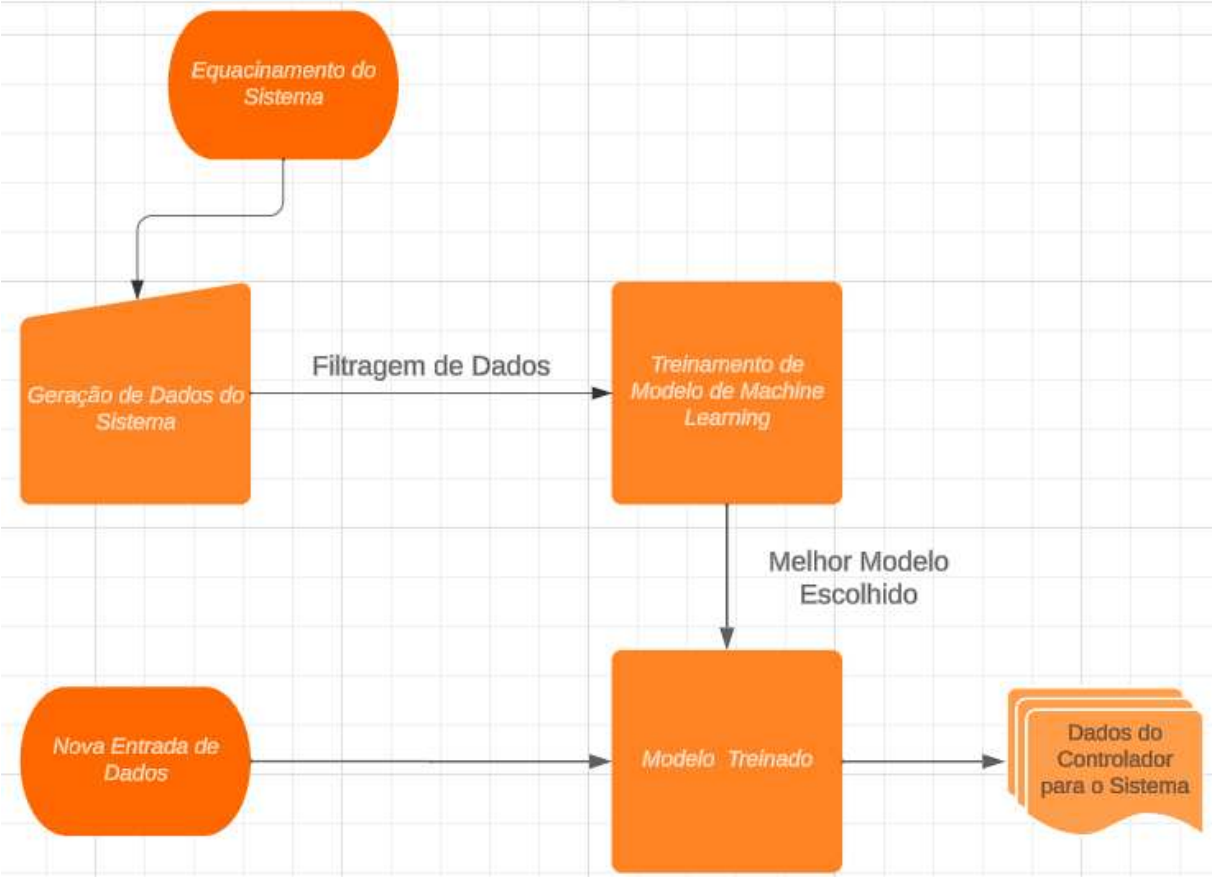
As métricas de treinamento e teste de *Machine Learning* são medidas que permitem avaliar a performance de um modelo preditivo em relação aos dados de treinamento e teste. A seguir, será abordados as principais métricas e que serão utilizadas mais adiante: *Mean Squared Error* (MSE), *Mean Absolute Error* (MAE) e *Root Mean Squared Error* (RMSE).

- *Mean Squared Error* (MSE): é uma métrica que calcula a média do quadrado das diferenças entre os valores preditos pelo modelo e os valores reais do conjunto de dados. Quanto menor o valor do MSE, melhor é a capacidade do modelo em fazer previsões precisas. O MSE é muito útil para lidar com outliers, uma vez que ele amplifica os erros (CHAO *et al.*, 2022).
- *Mean Absolute Error* (MAE): é uma métrica que calcula a média das diferenças absolutas entre os valores preditos pelo modelo e os valores reais do conjunto de dados. Diferentemente do MSE, o MAE não considera o quadrado das diferenças e, por isso, tende a ser menos sensível a outliers. Similar ao MSE, quanto menor o valor do MAE, melhor é a capacidade do modelo em fazer previsões precisas (CHAO *et al.*, 2022).
- *Root Mean Squared Error* (RMSE): é uma métrica que calcula a raiz quadrada do MSE. Essa métrica é útil para termos uma noção mais intuitiva do erro, uma vez que é expressa na mesma unidade da variável de saída. Similarmente ao MSE, quanto menor o valor do RMSE, melhor é a capacidade do modelo em fazer previsões precisas (CHAO *et al.*, 2022).

3 METODOLOGIA

Neste capitulo, será descrita uma análise que realizada em uma bancada teórica do Grupo de Pesquisa em Automação, Controle e Robótica (GPAR) da Universidade Federal do Ceará (UFC). A planta em estudo consiste em um braço mecânico cilíndrico com três Graus de Liberdade (GDL), acionado por três motores elétricos DC, com sensores de corrente, transformadores de potência e uma placa de *Digital Signal Processing* (DSP). O treinamento de modelos de *Machine Learning* (ML) para o controle de velocidade de um dos motores foi realizado, utilizando dados gerados por simulações da planta. O processo de coleta, geração e filtragem dos dados, além do treinamento de modelos MLP, Árvore de Decisão, *Bagging* e ExtraTrees, foi detalhadamente explicado, incluindo a otimização dos hiperparâmetros. A análise das métricas de desempenho, como *Mean Squared Error* (MSE), *Mean Absolute Error* (MAE) e *Root Mean Squared Error* (RMSE), permitiu avaliar a eficácia dos modelos na previsão dos parâmetros do controlador PID-2DOF. As técnicas de *Ensemble Learning*, como *Bagging* e ExtraTrees, mostraram desempenho satisfatório, contribuindo para a compreensão e controle da planta estudada, conforme o fluxograma presente na Figura 4.

Figura 4 – Fluxograma.



Fonte: Autoria própria.

3.1 Planta Estudada

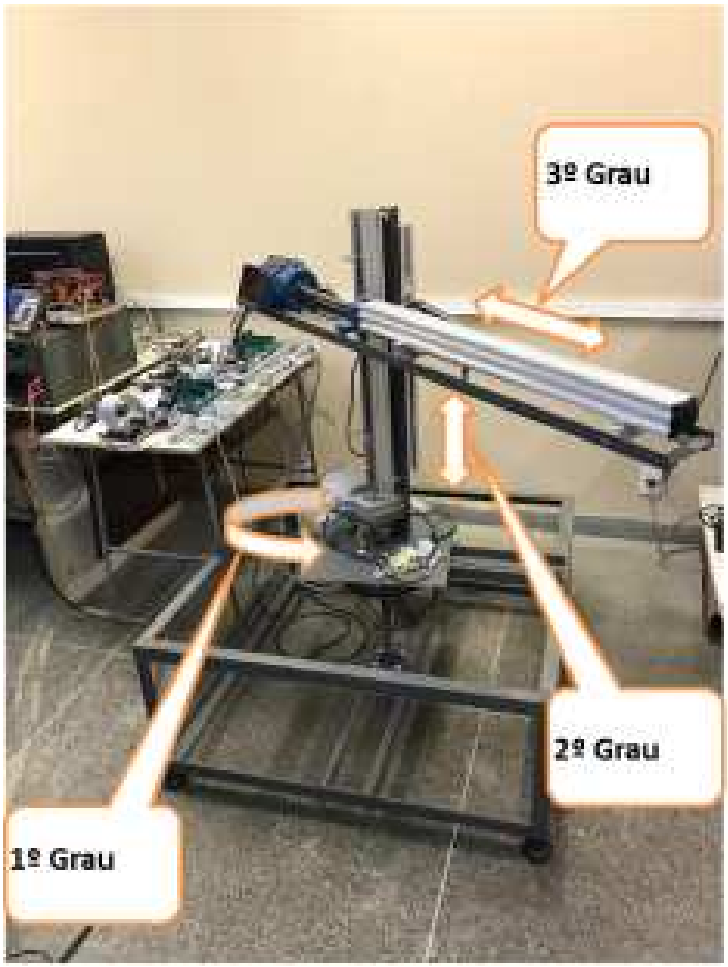
A planta estudada em questão é uma das bancadas teóricas do Grupo de Pesquisa em Automação, Controle e Robótica (GPAR) da Universidade Federal do Ceará (UFC), um grupo de estudo do qual o autor fez parte durante sua graduação. A bancada em questão consiste um braço mecânico cilíndrico com três Graus de Liberdade (GDL), além do acionamento da garra, controlado por três motores elétricos DC, além de contar com sensores de corrente, transformadores de potência e uma placa *Digital Signal Processing* (DSP), ou placa Processadora de Sinais Digitais. Porém por questão de generalização, será escolhido o treinamento do modelo de ML para o controle de velocidade de um desses motores, o mais externo possível. Tal bancada será melhor descrita a seguir:

Primeiramente na Figura 5 vemos o braço mecânico cilíndrico utilizado na bancada. Enquanto na Figura 6 vemos que a junta 1 do manipulador é acionada por um motor com potência nominal de 0,5 cv, tensão nominal de 220/380 V, 4 polos e corrente nominal de 1,18 A, com conexão em delta devido à tensão de até 220 V utilizada, e tendo em vista seu número de polos

podemos concluir que sua velocidade máxima sem carga na frequência da rede brasileira é de 1800 RPM. O dimensionamento do motor foi realizado para garantir a movimentação eficaz da estrutura do manipulador (BATISTA *et al.*, 2019).

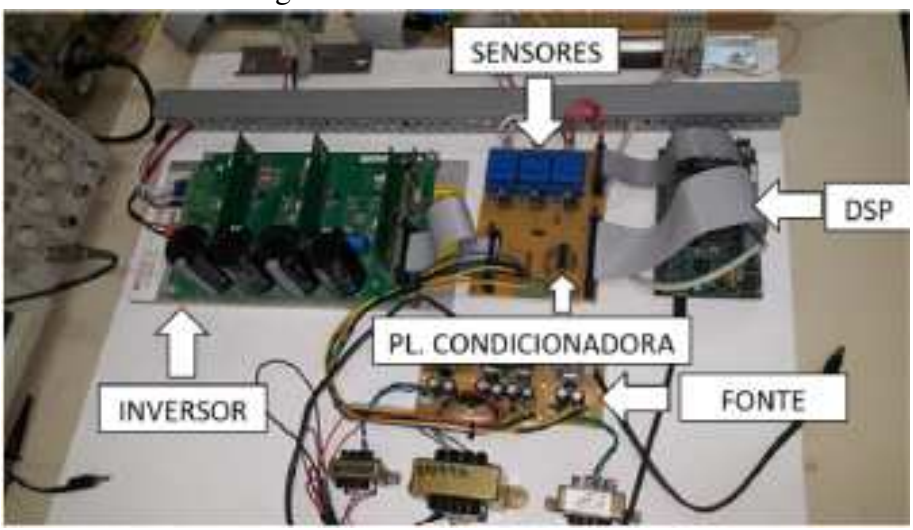
Nesse contexto, empregou-se uma *Digital Signal Processing* (DSP) da *Texas Instruments*®, modelo TMS320F2812. A principal vantagem dessa escolha é o alto desempenho, com a capacidade de executar 150 MIPS (milhões de instruções por segundo), além de oferecer suporte intrínseco à modulação em vetores espaciais (SVPWM - *Space Vector Modulation based on Pulse Width Modulation*).

Figura 5 – Bancada.



Fonte: (HELENO *et al.*, 2022).

Figura 6 – Sensores da Bancada.



Fonte: (HELENO *et al.*, 2022).

A identificação do motor da primeira junta pode ser descrita como a função de transferência contínua, na equação 3.1 abaixo:

$$G(z) = \frac{2,283z - 0,07834}{z^2 - 0,377z - 0,5939} \tag{3.1}$$

Essa função na equação 3.1 representa o modelo da junta identificado através do Método dos Mínimos Quadrados não Recursivos na bancada estudada presente no laboratório do GPAR (HELENO *et al.*, 2022).

3.2 Coleta de Dados

Nesta seção do capítulo será mostrada como a geração, simulação, aquisição e preprocessamento dos dados utilizados no treinamento dos modelos de ML.

3.2.1 Geração de Dados

Para geração dos dados, foi feito um *script* usando a ferramenta *Matlab*, no qual com o conhecimento das equações do sistema, presentes nas equações 2.8 e 2.7, foi montado uma simulação do sistema, ou seja, esse *script* no Apêndice A escolhe valores aleatórios, de maneira uniformemente distribuído, para as variáveis de controle simula o bancada através de suas funções de transferência e simula a resposta degrau da banca, onde salva as características métricas da resposta degrau num documento *excel*.

As variáveis de entrada e de saída estão descritas abaixo:

Variáveis de entrada do modelo de ML:

- `Amplitude_degrau`: Representa a amplitude do sinal de degrau que é aplicado à planta durante a simulação. Indica o tamanho da perturbação que está sendo introduzida no sistema.
- `RiseTime`: Representa o tempo de subida da resposta do sistema ao degrau. Indica quanto tempo leva para o sistema atingir uma determinada porcentagem da resposta final após uma mudança no *setpoint*.
- `SettlingTime`: Representa o tempo de assentamento, que é o tempo que o sistema leva para entrar e permanecer dentro de uma faixa aceitável após uma perturbação ou mudança no *setpoint*.
- `Overshoot`: Representa a porcentagem de sobressinal na resposta do sistema ao degrau. Indica o quanto a resposta ultrapassa o valor final antes de estabilizar.

Variáveis de saída do modelo de ML:

- `Kp`: Representa o ganho proporcional do controlador PID. Este é um parâmetro que ajusta a ação proporcional do controlador.
- `Ti`: Representa o tempo integral do controlador PID. É o inverso de K_i, o ganho integral. Controla a ação integrativa do controlador.
- `Td`: Representa o tempo derivativo do controlador PID. É o inverso de K_d, o ganho derivativo. Controla a ação derivativa do controlador.
- `b`: Representa o peso do ponto de ajuste no termo proporcional do controlador PID-2DOF. Isso ajusta a influência do ponto de ajuste no termo proporcional.
- `c`: Representa o peso do ponto de ajuste no termo derivativo do controlador PID-2DOF. Isso ajusta a influência do ponto de ajuste no termo derivativo.
- `Tf`: Representa o tempo de filtro derivativo do controlador PID-2DOF. Isso controla o filtro derivativo aplicado ao termo derivativo.

A escolha dessas variáveis de entrada e saída ocorreu pois o objetivo é treinar um modelo que dado um sistema descrito, ou seja, uma saída com as características desejadas escolhidas, o modelo nos retorno o parâmetros do controlador PID-2DOF que gerem uma saída com essas especificações.

Portanto, para se obter um quantidade de dados com diferentes valores de entradas

e saídas, foi feito o *script Matlab* que num loop finito, a cada iteração escolhe os valores das variáveis de entrada aleatoriamente mas com distribuição igual entre os limites superior e inferior, simula o sistema com essas entradas definidas, valendo-se das funções de transferência anteriormente descritas e salva numa variável do tipo *table* as entradas escolhidas e as saídas geradas para cada entrada, onde posteriormente essa variável foi exportada num arquivo de planilha. Esse processo foi repetido diversas vezes a ponto de se obter uma base numerosa de dados brutos. O limite escolhido para a Amplitude do degrau foi de 20 RPM(uma velocidade miníma usual) e 1800 RPM(velocidade máxima sem carga), enquanto os valores dos parâmetros do controlador foram escolhidos como valores próximos à valores usuais encontrados por métodos clássicos de ajuste de controlador na planta em questão (HELENO *et al.*, 2022). Tudo isso está explicitado na Tabela 1, tais valores limites foram escolhidos através de controles anteriores feitos com a bancada, buscando evitar vieses durante o treinamento.

Tabela 1 – Limites das Variáveis Selecionadas

Variável	Limite Mínimo	Limite Máximo
Amplitude_degrau	20	1800
Kp	0	1
Ti	0	1
Td	0	1
b	0	1
c	0	1
Tf	0	1

Fonte: Autoria própria.

O código Matlab que gerou esses dados está posto no Apêndice A.

3.2.2 Filtragem e Teste de Dados

Após adquirir os dados via software, utilizando a linguagem de programação *Python*, utilizando as bibliotecas abaixo:

- *SciPy* é uma biblioteca para computação científica que fornece funções estatísticas avançadas, como testes de hipóteses, análise de variância e distribuições estatísticas.
- *Pandas* é uma biblioteca poderosa para manipulação e análise de dados em formato tabular. Ela oferece estruturas de dados flexíveis, como o DataFrame, e funcionalidades para limpeza, transformação e exploração de dados.
- *NumPy* é essencial para computação numérica eficiente em Python. Ele fornece estruturas de dados como arrays e matrizes, juntamente com funções para operações matemáticas,

facilitando cálculos científicos.

- *Matplotlib* é uma biblioteca de visualização em Python, e *pyplot* é um módulo dela. Ele é amplamente utilizado para criar gráficos e visualizações, permitindo representações visuais de dados.
- *Seaborn* é uma biblioteca baseada no *Matplotlib*, projetada para criar visualizações estatísticas atraentes. Ela simplifica a criação de gráficos complexos e oferece estilos visuais aprimorados.

Essas bibliotecas são fundamentais para a análise de dados, estatísticas e visualização no contexto do código compilado.

Para a geração dos dados foi utilizado os valores máximos e mínimos de cada uma das variáveis, na Tabela 2.

Tabela 2 – Limites das Variáveis Simuladas

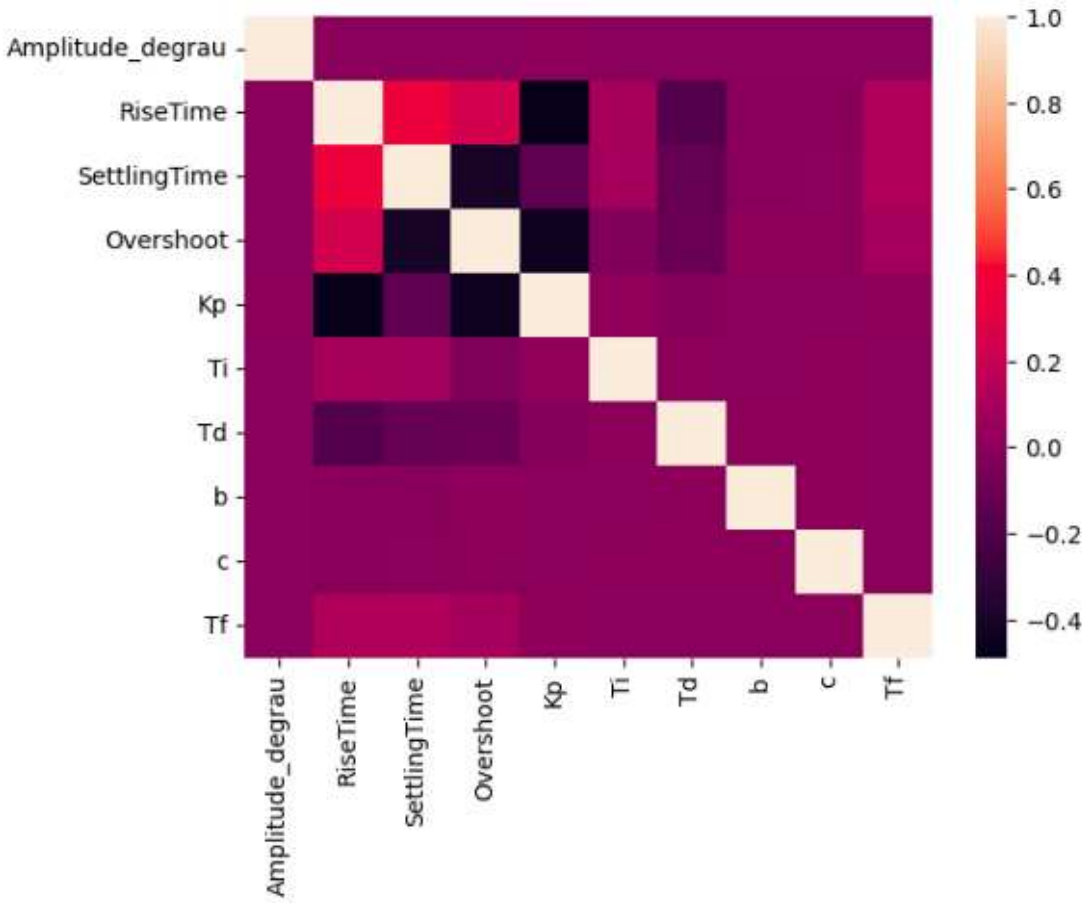
Variável	Limite Mínimo	Limite Máximo
Amplitude_degrau	20,0196	1799,9819
Kp	0,0006	1,0000
Ti	0,0000	1,0000
Td	0,0000	1,0000
b	0,0000	1,0000
c	0,0000	1,0000
Tf	0,0000	1,0000
RiseTime	0,0059	3,8311
SettlingTime	0,8306	10,0000
Overshoot	0,0000	3686,7580

Fonte: Autoria própria.

Após a geração foi excluídas linhas duplicadas ou com alguma das colunas vazias, além de excluir *outliers*, após tudo isso restou uma base de 10 colunas e 92483 linhas, para essa base será mostrado um mapa de calor com as correlações entre as colunas, mostrado na Figura 7.

Vale ressaltar que um mapa de calor de correlações de variáveis é uma representação gráfica em cores que visualiza a força e direção das relações lineares entre diferentes variáveis. Cores mais intensas indicam correlações mais fortes, sendo útil para identificar padrões e *insights* em conjuntos de dados.

Figura 7 – Mapa de Calor das Correlações



Fonte: Própria autoria.

Nesse mapa de calor, evidencia que os dados não estão muito correlacionados entre si, foi feito outras gerações de dados, com diferentes limites de geração, com mais correlações, mas os menores valores de MSE foram obtidos com essa base, portando o mapa de calor do conjunto de dados utilizado seria o da Figura 7.

3.3 Treinamento e Validação dos Modelos de *Machine Learning*

Após o pré-processamento dos dados foi feito o treinamento das principais tipologias de ML, os algoritmos de treinamento foram feitos em *Python* utilizando a plataforma *Colaboratory* do *Google*.

As bibliotecas em *Python* utilizada para os treinamentos dos diferentes modelos foram:

– *sklearn.model_selection.train_test_split*: Uma função do *scikit-learn* para dividir os dados em conjuntos de treinamento e teste, essencial para avaliar o desempenho do modelo.

- *sklearn.preprocessing.StandardScaler*: Classe que realiza a padronização dos recursos, garantindo que as características tenham média zero e desvio padrão um.
- *keras.models.Sequential*: Classe que permite a construção sequencial de modelos de redes neurais no *Keras*.
- *keras.layers.Dense*: Camada densamente conectada (totalmente conectada) em uma rede neural.
- *keras.optimizers.Adam*: Otimizador Adam, amplamente utilizado para ajuste de modelos de aprendizado profundo.
- *keras.layers* e *keras.callbacks.EarlyStopping*: Módulos que contêm várias camadas e a capacidade de interromper o treinamento prematuramente, respectivamente.
- *sklearn.ensemble.ExtraTreesRegressor*: Regressor de ExtraTrees, um algoritmo de aprendizado ensemble para tarefas de regressão.
- *sklearn.ensemble.VotingRegressor*: Regressor de Votação, que combina vários regressores para melhorar o desempenho preditivo.
- *sklearn.metrics.mean_squared_error, mean_absolute_error, r2_score*: Métricas comuns para avaliar o desempenho de modelos de regressão.
- *math.sqrt*: Função para calcular a raiz quadrada, utilizada para calcular o MSE.
- *sklearn.tree.DecisionTreeRegressor*: Regressor de Árvore de Decisão, utilizado para construir modelos baseados nesse algoritmo.
- *sklearn.ensemble.BaggingRegressor*: Regressor de *Bagging*, uma técnica de ensemble que combina modelos treinados em subconjuntos aleatórios dos dados de treinamento.

As tipologias escolhidas foram duas clássicas, MLP e Árvore de Decisão, e duas técnicas de *Ensemble Learning*, as tipologia de *Baggings* e ExtraTrees, onde essas duas últimas foram escolhidas por conta do seu bom desempenho como não linearidades.

Todos os modelos, exceto o de *Baggings*, tiveram seus hiperparâmetros otimizados utilizando um algoritmo de otimização bayesiana. Onde o modelo de Baggings não foi possível a otimização bayesiana devido à limitações de memoria RAM disponibilizada no Colaboratory, portanto seus hiperparâmetros foram otimizados de maneira empírica.

Adiante no projeto será mostrado os treinamentos e validações destes quatro modelos de ML.

Para todos os modelos, primeiramente foi divididos os dados entre as colunas de variáveis de entrada(x) e de saída(y) da maneira descrita na subseção 3.2.1. Após isso foi feito

foi escolhido 25% dos dados de todas as variáveis para usar como validação enquanto o restante para treinamento, onde os valores das variáveis de entrada foram normalizados para otimizar os treinamentos (JO, 2019). O script para essa separação e normalização se encontra no Apêndice B.

Além disso, ao final de todas os treinamentos de modelos, foi feito uma validação com a parte dos dados não utilizados no treinamento onde busca-se alguma principais métricas para análise, como está parte é igual para todos os modelos, esse trecho de código para validação está no Apêndice C

3.3.1 Treinamento e métricas da Multi-Layer Perceptron

Para esse modelo primeiramente é necessário definir o numero de camadas, onde a função de ativação das camadas ocultas é a função *LeakyReLU*, enquanto a camada de saída possuí a função de ativação Linear, por se tratar de uma regressão (PEDAMONTI, 2018). Isso é evidenciado na tabela 3.

A função de ativação *LeakyReLU* é uma variação da função *Rectified Linear Unit* (ReLU), amplamente usada em redes neurais e aprendizado profundo. A *LeakyReLU* resolve um problema comum do ReLU, que é a "morte"de neurônios, onde a saída é sempre zero para entradas negativas (DUBEY; JAIN, 2019). Na *LeakyReLU*, para entradas negativas, a função retorna uma pequena inclinação negativa em vez de zero, controlada pelo parâmetro alpha.

A função Linear consiste em transmitir os valores da camada anterior adiante

Sobre os hiperparâmetros das camadas individualmente, foi explicitado na Tabela 3.

Tabela 3 – Rede MLP

Camada	Número de Neurônios	Função de Ativação	Alpha
Camada de Entrada	256	LeakyReLU	0,0100
Camada Oculta 1	256	LeakyReLU	0,0100
Camada Oculta 2	256	LeakyReLU	1,0000
Camada de Saída	6	Linear	-

Fonte: Própria autoria.

Além desses hiperparâmetros foi escolhido o otimizador Adam, que é um otimizador adaptativo que ajusta efetivamente as taxas de aprendizado com base nas características dos gradientes dos parâmetros (ZHANG, 2018), esse otimizador foi escolhido por ser usado com frequência para treinamentos de MLP e se demonstrou satisfatório. Isso ajuda a acelerar o treinamento de redes neurais e torná-lo mais robusto, tornando-o uma escolha popular em muitas

aplicações de aprendizado de máquina e aprendizado profundo, para o caso estudados, onde seus hiperparâmetros estão presentes na Tabela 4.

Tabela 4 – Parâmetros do Adam

Hiperparâmetro	Valor
Taxa de Aprendizado	0,0010
Beta 1	0,4500
Beta 2	0,5000

Fonte: Própria autoria.

Além disso essa rede treinará por 300 épocas, revisando 300 vezes o conjunto de dados e treinando com blocos de 128 dados cada vez, ou *batch size* de 128.

Por último foi separado 25% dos dados de treinamento para validações durante o treinamento, além de um *script* para evitar o *overffiting*, fazendo com que caso o MSE diminuir ao longo de 10 épocas o treinamento é encerrado.

Todo a definição dessa rede está no Apêndice D, e após a rede treinada, o *script* de teste do Apêndice C foi executado e retornou as seguintes métricas para a rede MLP, na tabela 5.

Tabela 5 – Métricas da MLP

Métrica	Valor
MSE	0,0681
MAE	0,2610
RMSE	0,2176

Fonte: Própria autoria.

3.3.2 Treinamento e Métricas da Árvore de Decisão

Para esse modelo de ML foi feito a otimização para encontrar os hiperparâmetros, a otimização bayesiana explicada no capítulo 2, seriam os abaixo:

- **max_depth** (Profundidade Máxima): A profundidade máxima de uma árvore de decisão refere-se ao número máximo de níveis de divisão que a árvore pode atingir. É um hiperparâmetro crítico que controla a complexidade da árvore. Uma profundidade mais alta permite que a árvore capture relações mais complexas nos dados de treinamento, mas também aumenta o risco de overfitting, onde a árvore se ajusta excessivamente aos dados de treinamento, prejudicando sua capacidade de generalização.
- **min_samples_split** (Mínimo de Amostras para Divisão): Este hiperparâmetro determina o número mínimo de amostras necessárias em um nó para que uma divisão (split) seja

realizada. Uma divisão ocorre quando a árvore tenta criar um novo nó filho. Estabelecer um valor maior para min_samples_split pode levar a árvores mais rasas, evitando divisões em nós com um número muito baixo de amostras, o que pode reduzir o overfitting.

– **min_samples_leaf** (Mínimo de Amostras em Folhas): Este hiperparâmetro define o número mínimo de amostras necessárias em uma folha da árvore. As folhas representam os nós terminais da árvore onde as previsões são feitas. Aumentar min_samples_leaf resulta em folhas com mais amostras e, consequentemente, em árvores mais rasas. Isso ajuda a evitar overfitting, pois limita o número de folhas com previsões com base em um pequeno número de amostras, o que pode levar a estimativas instáveis.

Tendo em vista isso, o valores do hiperparâmetros escolhidos estão na Tabela 6, onde as métricas de desempenho desse modelo está na Tabela 7 e o algoritmo desse treinamento no Apêndice E.

Tabela 6 – Parâmetros do DecisionTree

Hiperparâmetro	Valor
max_depth	64
min_samples_split	20
min_samples_leaf	1

Fonte: Própria autoria.

Tabela 7 – Métricas da DecisionTree

Métrica	Valor
MSE	0,1106
MAE	0,2639
RMSE	0,3325

Fonte: Própria autoria.

3.3.3 Treinamento e Métricas da Bagging

Para o modelo de *Bagging* será treinada varias *DecisionTrees* em sequencia e será escolhida a melhor delas, tal como descrito na seção 2.2. Tendo em visto isso, os mesmos hiperparâmetros do *DecisionTree* existem no *Bagging* em adição com os hiperparâmetros abaixo:

– **n_estimators** (Número de Estimadores): O hiperparâmetro n_estimators se refere ao número de estimadores (modelos base) que compõem o ensemble *Bagging*. Em um modelo *Bagging*, várias cópias do mesmo modelo base são treinadas com conjuntos de dados ligeiramente diferentes (amostras de bootstrap) e, em seguida, suas previsões são

combinadas. O valor de n_estimators determina quantos modelos base estão incluídos no ensemble. Aumentar esse valor pode melhorar a capacidade do ensemble de generalizar os dados, mas também aumenta o custo computacional.

- **max_samples** (Máximo de Amostras): O hiperparâmetro max_samples controla a fração de amostras a serem usadas para treinar cada estimador individual no modelo Bagging. Ele define a proporção das amostras de treinamento que serão selecionadas aleatoriamente para cada modelo base. O valor padrão de 1.0 indica que todas as amostras estão disponíveis para treinamento. No entanto, definir um valor menor que 1.0 permite a criação de amostras de bootstrap, o que pode introduzir variação no treinamento e reduzir o risco de overfitting.
- **bootstrap** (Amostragem com Reposição): O hiperparâmetro bootstrap é um valor booleano que determina se a amostragem é realizada com reposição. Se bootstrap for definido como True, as amostras são selecionadas aleatoriamente com reposição, o que significa que uma mesma amostra pode ser escolhida mais de uma vez para treinamento de um estimador individual. Isso introduz variação nos dados de treinamento de cada estimador. Se bootstrap for False, a amostragem é realizada sem reposição, ou seja, cada amostra é usada apenas uma vez em cada estimador.
- **random_state** (Semente Aleatória): O hiperparâmetro random_state é uma semente aleatória que controla a aleatoriedade na seleção de amostras durante o treinamento dos estimadores. Ao definir random_state com um valor específico (como 42), você garante que os resultados sejam reproduzíveis, ou seja, o mesmo conjunto de amostras é selecionado em diferentes execuções do modelo Bagging, o que é importante para a consistência dos resultados em experimentos repetíveis.

Tendo em vista isso, o valores do hiperparâmetros escolhidos estão na Tabela 8, onde as métricas de desempenho desse modelo está na Tabela 9 e o algoritmo desse treinamento no Apêndice F.

Tabela 8 – Parâmetros do Bagging

Hiperparâmetro	Valor
max_depth	64
min_samples_split	20
min_samples_leaf	1
n_estimators	1000
max_samples	0,8
bootstrap	True
random_state	42

Fonte: Própria autoria.

Tabela 9 – Métricas da Bagging

Métrica	Valor
MSE	0,0849
MAE	0,2915
RMSE	0,2395

Fonte: Própria autoria.

3.3.4 Treinamento e Métricas da Extremely Randomized Trees

Para o modelo de ExtraTrees, por ser um modelo de *Ensemble Learning* com base no modelo de *DecisionTree* assim como o *Bagging*, implica que ambos modelos possuem os mesmos hiperparâmtros, onde a diferença entre eles seria na combinação das *DecisionTrees*.

A principal diferença entre o *Bagging* e o método ExtraTrees reside na abordagem de criação das árvores de decisão. O *Bagging* utiliza árvores de decisão base, treinadas em subconjuntos aleatórios dos dados de treinamento, introduzindo variância por meio da amostragem de dados. Em contraste, oExtraTrees adiciona uma camada adicional de aleatoriedade, criando árvores extremamente aleatórias que fazem divisões de características de forma altamente aleatória (TRAN *et al.*, 2023). Essa abordagem mais agressiva aumenta a independência e diversidade entre as árvores, tornando o ExtraTrees especialmente eficaz em problemas com alta dimensionalidade ou dados ruidosos, embora possa introduzir um leve viés devido à aleatoriedade extrema.

Tendo em vista isso, o valores do hiperparâmetros escolhidos estão na Tabela 10, onde as métricas de desempenho desse modelo está na Tabela 11 e o algoritmo desse treinamento no Apêndice G.

Tabela 10 – Parâmetros do ExtraTrees

Hiperparâmetro	Valor
max_depth	128
min_samples_split	40
min_samples_leaf	1
n_estimators	2000
max_samples	0,8461
bootstrap	True
random_state	42

Fonte: Própria autoria.

Tabela 11 – Métricas da ExtraTrees

Métrica	Valor
MSE	0,0711
MAE	0,2667
RMSE	0,2216

Fonte: Própria autoria.

3.3.5 Treinamento e Métricas do Voting

Considerando os desempenhos positivos dos métodos anteriores, com exceção da *DecisionTree* pura, optou-se por combiná-los em um único modelo utilizando a técnica de *Voting*. Essa abordagem envolve o treinamento separado e paralelo dos métodos, seguido pela obtenção de resultados e previsões como a média ponderada dos modelos. Os valores das métricas de desempenho são analisados considerando esse modelo combinado.

Tendo em vista isso, o valores das métricas de desempenho de todos os modelos está na Tabela 12 e o algoritmo desse treinamento no Apêndice H.

Tabela 12 – Métricas dos Modelos

Modelo	MSE	MAE	RMSE
MLP	0,0681	0,2610	0,2176
DecisionTree	0,1106	0,2639	0,3325
Bagging	0,0849	0,2915	0,2395
ExtraTrees	0.0705	0,2667	0,2216
Voting	0,0706	0,2657	0,2211

Fonte: Própria autoria.

Durante o processo de treinamento e validação, vemos que o modelo de MLP se demonstra bem mais performáticos que os demais modelos, mesmos os combinacionais com ele.

Por conta disso, o modelo de ML escolhido para estimar os parâmetros do controlador PID-2DOF será o MLP, no próximo capitulo será demonstrado a estimação propriamente dita desses parâmetros.

Porém vale ressaltar que o modelo de ExtraTrees se demonstrou muito satisfatório também, devendo então ser considerado para analises futuras de problemas parecidos.

4 RESULTADOS

Para esse capítulo foi feito a previsão de alguns valores escolhidos empiricamente para buscar uma onda com boas características, tomando como base alguns testes feitos na mesma bancada (HELENO *et al.*, 2022).

Portando, foi escolhido como teste, 4 resultados, os dois melhores, ou seja, mais rápidos controladores de malha aberta, apesar da impossibilidade de um comparação direta e dois com características escolhidas por fins demonstrativos, na Tabela 13, e considerando um degrau de 200 RPM.

Tabela 13 – Testes Escolhidos

Teste	Amplitude	RiseTime	SettlingTime	Overshoot
Teste 1	200	0,1342	0,9947	15,2600
Teste 2	200	0,3903	0,4875	9,6800
Teste 3	200	2,0000	4,0000	10,0000
Teste 4	200	2,0000	5,0000	15,0000

Fonte: Própria autoria.

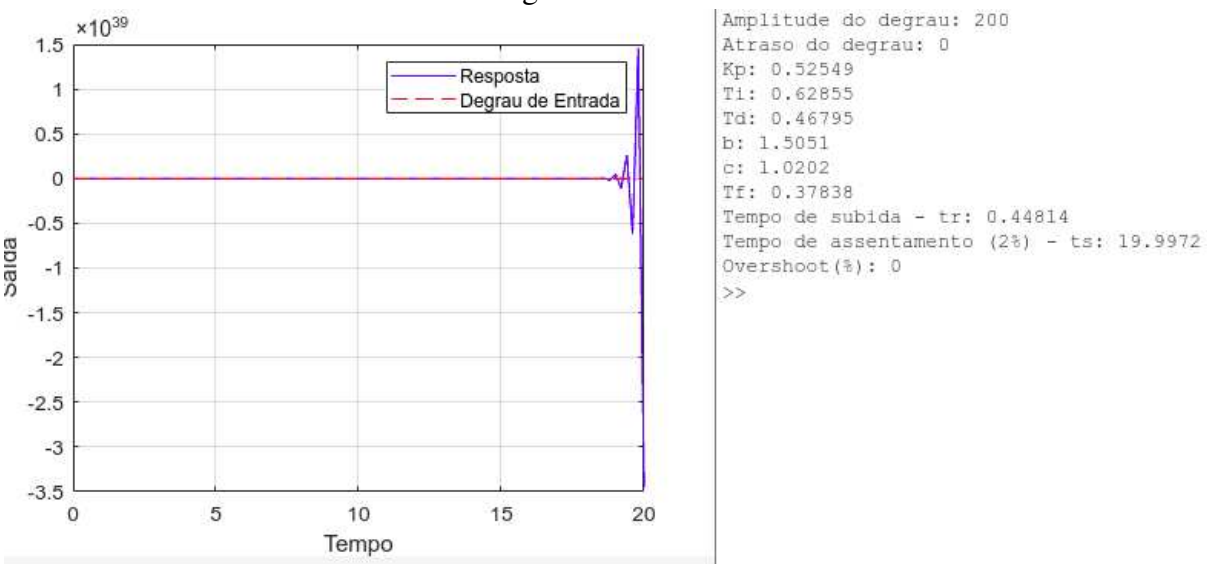
Para testar o desempenho da saída do modelo de ML foi adaptado o código no apêndice A para testar uma única situação e imprimir seu resultado na tela.

4.1 Teste 1

Para esse teste gerar o novo input corresponde te para o modelo tal qual mostra a tabela 13, e seguindo o modelo de código do apêndice I.

Após gerado o teste para essa situação, foi obtida a resposta e formato de onda nas Figura 8.

Figura 8 – Teste 1



Fonte: Própria autoria.

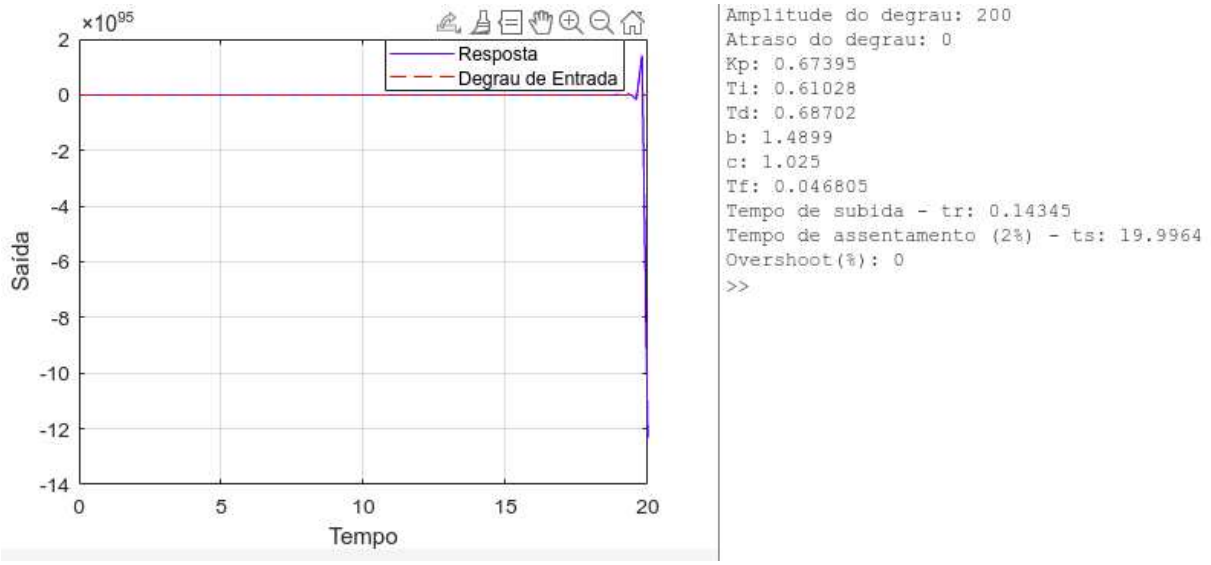
Como podemos perceber na Figura 8, foi obtido um resultado instável que não converge para a referência pedida, isso se da devido ao modelo ter atribuídos alguns valores negativos aos parâmetros do controlador, na tentativa de encontrar um formato de onda tão ideal quanto o solicitado.

4.2 Teste 2

Para esse teste gerar o novo input corresponde te para o modelo tal qual mostra a tabela 13, e seguindo o modelo de código do apêndice I.

Após gerado o teste para essa situação, foi obtida a resposta e formato de onda nas Figura 9.

Figura 9 – Teste 2



Fonte: Própria autoria.

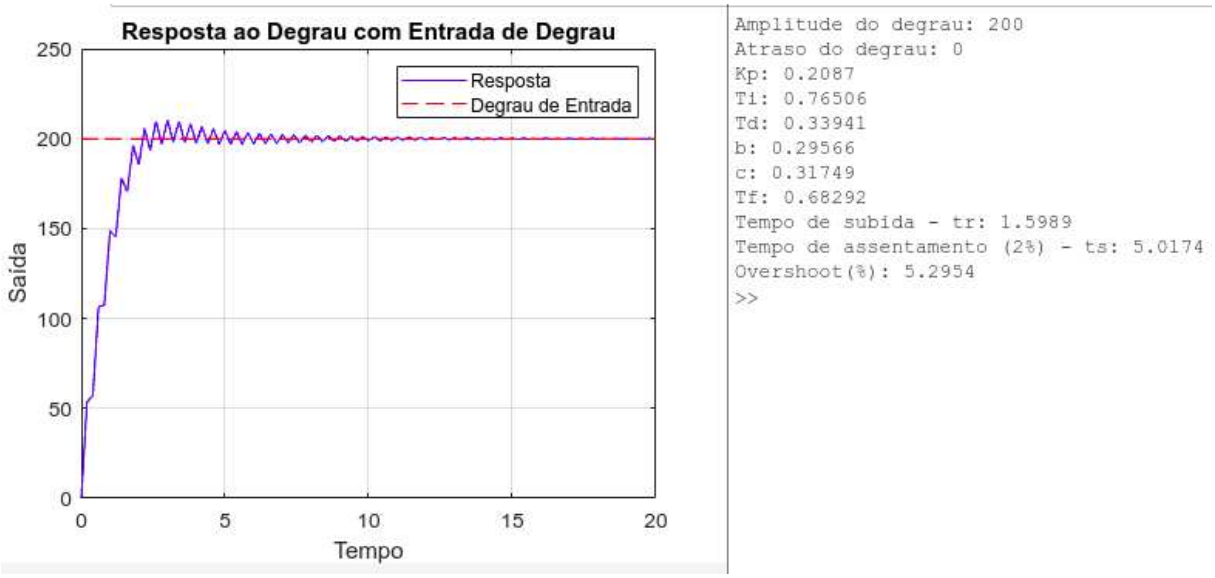
Como podemos perceber na Figura 9, foi obtido um resultado bem semelhante ao anterior, mas apresentando agora uma tentativa de controle antes da instabilidade, porém ainda o sistema não foi capaz de gerar uma controle como o solicitado.

4.3 Teste 3

Para esse teste gerar o novo input corresponde te para o modelo tal qual mostra a tabela 13, e seguindo o modelo de código do apêndice I.

Após gerado o teste para essa situação, foi obtida a resposta e formato de onda nas Figura 10.

Figura 10 – Teste 3



Fonte: Própria autoria.

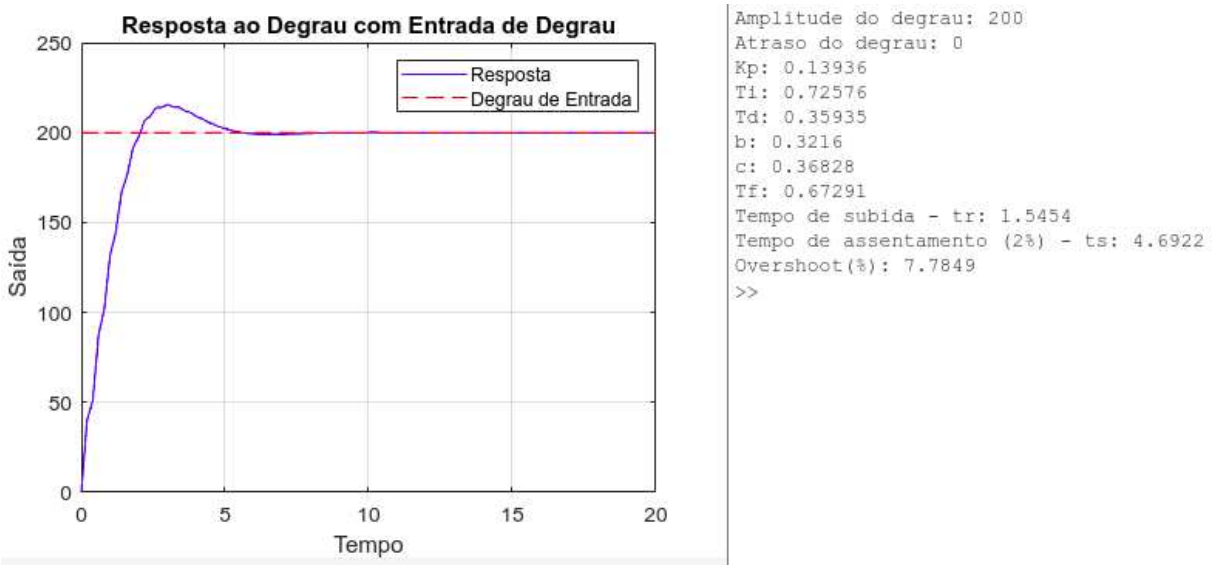
Como podemos perceber na Figura 10, foi obtido um resultado estável e controlado, inclusive sendo bem próximo do solicitado, mas com um overshoot consideravelmente mais alto que o pedido. Porém podemos concluir que para para SettlingTimes superiores a 1 segundo nosso sistema pode ser controlado e o modelo descreve isso.

4.4 Teste 4

Para esse teste gerar o novo input corresponde te para o modelo tal qual mostra a tabela 13, e seguindo o modelo de código do apêndice I.

Após gerado o teste para essa situação, foi obtida a resposta e formato de onda nas Figura 11.

Figura 11 – Teste 4



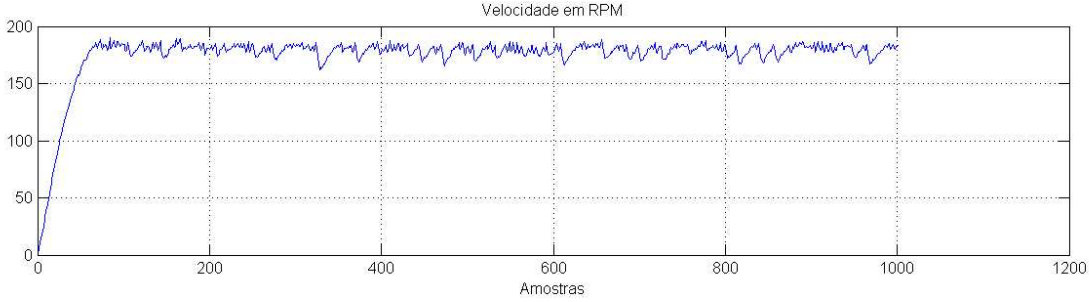
Fonte: Própria autoria.

Como podemos perceber na Figura 11, foi obtido um resultado bem próximo do pedido com métricas dentro do solicitado, mostrando que para as respostas mais lentas o sistema pode sim ser controlado com eficacia, desde que respeitemos os limites particulares e inerentes ao próprio sistema.

4.5 Testes Experimentais

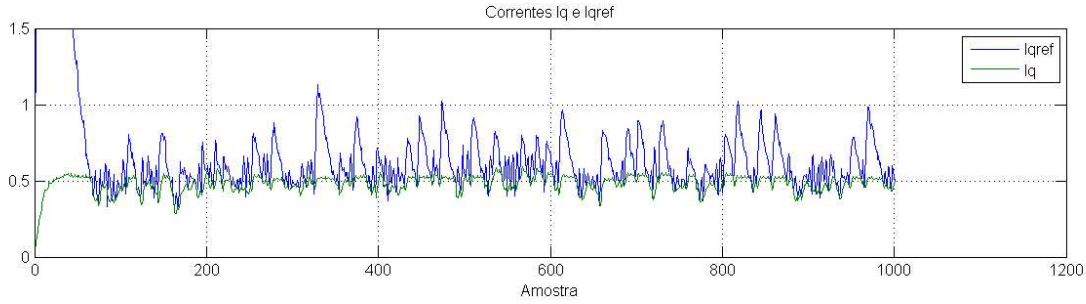
Tendo em vista o desempenho em software do modelo no Teste 4, foi feito alguns testes na bancada experimental, para ver como o controlador encontrado no Teste 4 se comportaria ao ser feito de maneira pratica. Após a aplicação do controlador encontrado pelo modelo de ML onde capturamos a resposta do sistema tanto a um degrau de velocidade, na Figura 12, as suas correntes de controle, na Figura 13, a resposta a uma onda quadrada de 100 RPM com delay, na Figura 14 e as correntes para essa ultima resposta na Figura 15.

Figura 12 – Degrau de velocidade.



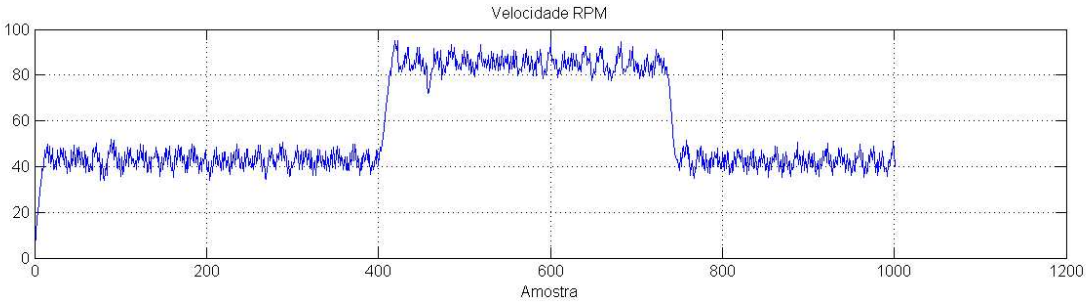
Fonte: Própria autoria.

Figura 13 – Correntes de controle do degrau.



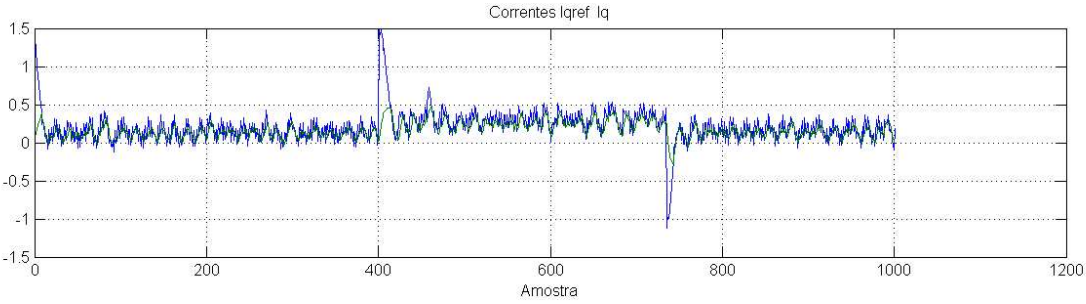
Fonte: Própria autoria.

Figura 14 – Onda quadrada de velocidade.



Fonte: Própria autoria.

Figura 15 – Correntes de controle da onda quadrada.



Fonte: Própria autoria.

Vemos que as velocidades em RPM tendem a se manter na referencia dada, e as suas correntes de controle estão acompanhando as velocidades, mantendo uma referencia para velocidades constantes, mas para quando a mudança na velocidade vemos o pico nas correntes para descrever isso, conforme podemos ver nas Figuras 14 e 15 respectivamente.

Pode se concluir então que o quarto teste foi o mais assertivo, como o modelo utilizado foi o mesmo, é intuitivo que essas diferenças ocorreram devido a particularidades do sistema, foi encontrado então, através dos parâmetros retornados pelo modelo de ML utilizado,

um controlador com desempenho mais que satisfatório para o sistema proposto, sendo capaz de controlar a bancada experimental.

A rede já treinada está presente, assim como seu código de geração na integra está no repositório do *Github* presente no link GitHub - Machine Learning Braco Robotico.

5 CONCLUSÕES E TRABALHOS FUTUROS

O presente trabalho explora a aplicação do *Machine Learning* (ML) na estimativa de parâmetros PID-2DOF em sistemas de controle de motores. Inicialmente, utilizamos um controlador PID-2DOF sintonizado por métodos clássicos como ponto de referência, destacando os resultados obtidos em termos de *RiseTime*, *SettlingTime* e *Overshoot*. Esses controladores clássicos foram selecionados com base em critérios específicos(HELENO *et al.*, 2022).

Foi selecionado então quatro cenários de teste, sendo dois com melhor desempenho do controlador clássico e referente a um controle de uma versão idealizada do sistema, e dois com características mais desafiadoras. Usamos esses cenários para avaliar o desempenho da saída do modelo ML, tal qual mostrado no capitulo 4.

Ao analisar tais resultados obtidos é demonstrado que a aplicação do modelo de ML na estimativa de parâmetros PID-2DOF em sistemas de controle de motores é promissora. O modelo mostra capacidades de previsão precisas na maioria dos casos, sem contar de ter sido capaz de encontrar parâmetros que forneceram um sistema rápido e robusto, dentro das limitações inerentes ao próprio sistema. Portanto, conclui-se que o treinamento de um modelo de ML é uma abordagem alternativa e eficiente em comparação com o método clássico para o ajuste do controlador PID-2DOF.

Tendo em vista isso, foram realizados testes práticos na bancada experimental para avaliar o desempenho do controlador obtido no Teste 4. A aplicação do controlador encontrado pelo modelo de *Machine Learning* (ML) foi analisada em diferentes cenários. A resposta do sistema foi capturada em um degrau de velocidade (Figura 12), onde observamos a estabilidade e tempo de resposta do sistema. Além disso, as correntes de controle associadas a esse degrau foram registradas (Figura 13).

Outro teste consistiu na aplicação de uma onda quadrada de 100 RPM com delay, apresentada na Figura 14, que proporcionou *insights* sobre o comportamento do controlador em situações de variação mais complexa de velocidade. As correntes de controle correspondentes a essa onda quadrada foram registradas e estão representadas na Figura 15.

Os resultados indicam que as velocidades em RPM tendem a manter-se próximas à referência estabelecida, e as correntes de controle acompanham as variações de velocidade, mantendo uma resposta adequada para velocidades constantes. Contudo, ao observar mudanças abruptas na velocidade, é evidenciado um pico nas correntes de controle, indicando uma resposta transiente mais acentuada, conforme demonstrado nas Figuras 14 e 15.

Estes resultados práticos corroboram a eficácia do modelo de ML na sintonia do controlador PID-2DOF para sistemas de controle de motores, destacando a capacidade do modelo em lidar com situações dinâmicas e complexas.

No entanto, é fundamental considerar as limitações e desafios específicos associados a determinadas situações de teste. Desta forma, este trabalho contribui para a compreensão do potencial e das nuances da integração do ML em sistemas de controle, abrindo caminho para futuras pesquisas e melhorias neste campo dinâmico e inovador, dentre elas é possível destacar as seguintes:

Testes com diferente conjunto de dados, pois o melhor conjunto de dados encontrado apresentou correlações baixas entre as variáveis de saídas e entradas, portanto alterações e refinamentos na base de dados poderia ser benéfico para o desempenho do modelo.

Além disso, buscar descrever de maneira mais precisa o sistema físico poderiam apresentar resultados mais precisos, como trabalhar diretamente no tempo discreto ao invés de discretizar o sistema como feito aqui, ou até mesmos testar outros métodos de discretização.

Ademais, os testes com diferente modelos de ML, como Redes Neurais Recorrentes ou Algoritmos Genéticos, poderia ser muito promissor, pois há diversos modelos e algorítimos que podem ser uteis para variados problemas, sem contar com o teste de diferentes técnicas de Ensemble para combinar diferentes modelos. Este trabalho testou o Ensembles apenas entre modelos baseados em MLP e baseados em Árvores de Decisão.

Para finalizar, a controle na bancada física, sem depender de software demonstraria muito o real desempenho do modelo aqui demonstrado.

REFERÊNCIAS

ÅSTRÖM, K. J.; HÄGGLUND, T. **Control PID avanzado**. Madrid: Pearson, 2009.

BATISTA, G. E. d. A. P. *et al.* **Pré-processamento de dados em aprendizado de máquina supervisionado**. Tese (Doutorado) – Universidade de São Paulo, 2003.

BATISTA, J.; COSTA, J.; SOUZA, D.; FILGUEIRAS, L.; JúNIOR, J.; JúNIOR, A.; REIS, L. Modelagem dinâmica e simulação de um controlador pid e lqr para um manipulador cilíndrico. **Anais do SBAI**, 2019.

BINGI, K.; IBRAHIM, R.; KARSITI, M. N.; HASSAN, S. M.; HARINDRAN, V. R. A comparative study of 2dof pid and 2dof fractional order pid controllers on a class of unstable systems. **Archives of Control Sciences**, Committee of Automatic Control and Robotics PAS, p. 635–682, 2018.

BREIMAN, L. Bagging predictors. **Machine learning**, Springer, v. 24, p. 123–140, 1996.

BREIMAN, L. **Classification and regression trees**. New York: Routledge, 2017.

CHAO, H.; ZHANG, J.; YAN, P. Regression metric loss: Learning a semantic representation space for medical images. In: WANG, L.; DOU, Q.; FLETCHER, P. T.; SPEIDEL, S.; LI, S. (Ed.). **Medical Image Computing and Computer Assisted Intervention – MICCAI 2022**. Cham: Springer, 2022. (Lecture Notes in Computer Science, v. 13438), p. 427–436. Acessado em: 22 de novembro de 2023. Disponível em: https://doi.org/10.1007/978-3-031-16452-1_41.

CROCOMO, M. K.; DELBEM, A. C. B. Otimização bayesiana com detecção de comunidades. 2011.

DASH, P.; SAIKIA, L. C.; SINHA, N. Comparison of performances of several cuckoo search algorithm based 2dof controllers in agc of multi-area thermal system. **International Journal of Electrical Power & Energy Systems**, Elsevier, v. 55, p. 429–436, 2014.

DUBEY, A. K.; JAIN, V. Comparative study of convolution neural network's relu and leaky-relu activation functions. In: MISHRA, S.; SOOD, Y.; TOMAR, A. (Ed.). **Applications of Computing, Automation and Wireless Systems in Electrical Engineering**. Singapore: Springer, 2019. (Lecture Notes in Electrical Engineering, v. 553), p. 76. Acessado em: 18 de novembro de 2023. Disponível em: https://doi.org/10.1007/978-981-13-6772-4_76.

ESCOVEDO, T. **Machine Learning: Conceitos e Modelos — Parte I: Aprendizado Supervisionado***. Medium, 2020. Acessado em: 02 de dezembro de 2023. Disponível em: https://tatianaesc.medium.com/machine-learning-conceitos-e-modelos-f0373bf4f445.

FACCIN, F. Abordagem inovadora no projeto de controladores pid. 2004.

FRAZIER, P. I. A tutorial on bayesian optimization. **arXiv preprint arXiv:1807.02811**, 2018.

GEURTS, P.; ERNST, D.; WEHENKEL, L. Extremely randomized trees. **Machine learning**, Springer, v. 63, p. 3–42, 2006.

HELENO, F.; BATISTA, J. G.; SOUZA, D.; LIMA, A. D.; LAURINDA, L.; JúNIOR, A.; DIAS, E. Controle e identificação de parâmetros de uma junta de um manipulador com base em pid, pid 2-dof e mínimos quadrados. In: . Fortaleza: [*S. n.*], 2022.

JO, J.-M. Effectiveness of normalization pre-processing of big data to the machine learning performance. **The Journal of the Korea institute of electronic communication sciences**, Korea Institute of Electronic Communication Science, v. 14, n. 3, p. 547–552, 2019.

LOPES, H. F.; SCHMIDT, A. M.; MOREIRA, A. R. B. Estimação de hiperparâmetros em modelos de previsão. Instituto de Pesquisa Econômica Aplicada (Ipea), 1996.

MONARD, M. C.; BARANAUSKAS, J. A. Indução de regras e árvores de decisão. **Sistemas Inteligentes-fundamentos e aplicações**, sn, v. 1, p. 115–139, 2003.

OSHIRO, T. M. **Uma abordagem para a construção de uma única árvore a partir de uma Random Forest para classificação de bases de expressão gênica**. Tese (Doutorado) – Universidade de São Paulo, 2013.

PEDAMONTI, D. Comparison of non-linear activation functions for deep neural networks on mnist classification task. **arXiv preprint arXiv:1804.02763**, 2018.

PHYO, P.-P.; BYUN, Y.-C.; PARK, N. Short-term energy forecasting using machine-learning-based ensemble voting regression. **Symmetry**, MDPI, v. 14, n. 1, p. 160, 2022.

REIS, C. H. Otimização de hiperparâmetros em redes neurais profundas. **Minas Gerais**, 2021.

RITA, D. J. Controle de processos usando redes neurais artificiais : uma aplicação experimental. **Ufsc.br**, 2016. Acessado em: 12 de novembro de 2023. Disponível em: https://repositorio.ufsc.br/handle/123456789/157974?show=full.

SNOEK, J.; LAROCHELLE, H.; ADAMS, R. P. Practical bayesian optimization of machine learning algorithms. **Advances in neural information processing systems**, v. 25, 2012.

SOARES, P. M. O. d. R. Discretização de controladores contínuos. 1996.

TAUD, H.; MAS, J. Multilayer perceptron (mlp). **Geomatic approaches for modeling land change scenarios**, Springer, p. 451–455, 2018.

TRAN, Q.-H.; NGUYEN, H.; BUI, X.-N. Novel soft computing model for predicting blast-induced ground vibration in open-pit mines based on the bagging and sibling of extra trees models. **CMES-Computer Modeling in Engineering & Sciences**, v. 134, n. 3, 2023.

ZHANG, Z. Improved adam optimizer for deep neural networks. In: IEEE. **2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)**. Banff, 2018. p. 1–2.

ZHOU, Z.-H. **Machine learning**. Singapore: Springer Nature, 2021.

ÅSTRöM, K.; HäGGLUND, T. **PID Controllers: Theory, Design, and Tuning**. Research Triangle Park: Instrument Society of America, 1995. v. 2. 59–120 p.

APÊNDICE A – CÓDIGO DA GERAÇÃO DE DADOS

Código-fonte 1 – Geração de Dados

```
close all;
clear;
clc;

tic;
%Variaveis Fixas
Ts = 0.2; % Tempo de amostragem
Tempo_degrau = 0; % Tempo de atraso do degrau

numerador = [2.283, -0.07834];
denominador = [1, -0.377, -0.5939];
for i = 1:1:25000
    disp(i);
    % Defina os valores do degrau
    rng('shuffle');
    Amplitude_degrau = unifrnd(20, 1800); % Altura do
       degrau

    % Parametros do controlador PID 2-DOF
    b = unifrnd(0, 1); % Peso do ponto de ajuste no termo
       proporcional
    c = unifrnd(0, 1); % Peso do ponto de ajuste no termo
       derivado
    Tf = unifrnd(0, 1); % Tempo de filtro derivativo

    % Parametros do controlador PID
    Kp = unifrnd(0, 1); % Ganho proporcional
    Ti = unifrnd(0, 1); % Tempo integral
    Td = unifrnd(0, 1); % Tempo derivativo
```

```
27
28
29      % FUNCOES DE TRANSFERENCIA
30      % Controlador PID
31      Ki = Kp/Ti;
32      Kd = Kp*Td;
33
34      P = b*Kp;
35
36      I = tf([Ki*Ts, Ki*Ts], [2, -2], Ts);
37
38      D = tf([2*c*Kd, -2*c*Kd], [(2*Tf +Ts), -(2*Tf +Ts)], Ts
            );
39
40      C = P + I + D;
41
42      % Segundo Grau de Liberdade
43      P = (1-b)*Kp;
44
45      I = 0;
46
47      D = tf([2*(1-c)*Kd, -2*(1-c)*Kd], [(2*Tf +Ts), -(2*Tf +
            Ts)], Ts);
48
49      X = P + I + D;
50
51      % Simulacao da planta e resposta ao degrau
52      G = tf(numerador, denominador, Ts);
53      sistema_controlado = G*C*feedback(1,G*(C+X));
54
55      % Simule a resposta ao degrau
56      if (Tempo_degrau <= 0)
```

```
57          t_sim = 0:Ts:10; % Tempo de simulacao
58      else
59           t_sim = 0:Ts:5*Tempo_degrau; % Tempo de simulacao
60      end
61      degrau = Amplitude_degrau * (t_sim >= Tempo_degrau);
62
63      [y, t] = lsim(sistema_controlado, degrau, t_sim);
64
65      infos = stepinfo(y, t);
66      RiseTime = infos.RiseTime;
67      SettlingTime = infos.SettlingTime;
68      Overshoot = infos.Overshoot;
69      Undershoot = infos.Undershoot;
70      if (i == 1)
71          % Criando uma tabela com as variaveis e seus
              rotulos de coluna
72          tabela = table(Amplitude_degrau, Tempo_degrau, Kp,
              Ti, Td, b, c, Tf, RiseTime, SettlingTime,
              Overshoot, Undershoot, 'VariableNames', {'
              Amplitude_degrau', 'Tempo_degrau', 'Kp', 'Ti', '
              Td', 'b', 'c', 'Tf', 'RiseTime', 'SettlingTime',
               'Overshoot', 'Undershoot'});
73      else
74          nova_linha = [Amplitude_degrau, Tempo_degrau, Kp,
              Ti, Td, b, c, Tf, RiseTime, SettlingTime,
              Overshoot, Undershoot]; % Novos valores para
              cada coluna
75      % Criar uma nova tabela com a nova linha
76      nova_tabela = array2table(nova_linha, 'VariableNames',
           {'Amplitude_degrau', 'Tempo_degrau', 'Kp', 'Ti', '
         Td', 'b', 'c', 'Tf', 'RiseTime', 'SettlingTime', '
         Overshoot', 'Undershoot'});
```

```
77
78      % Concatenar a nova tabela com a tabela existente
79      tabela = [tabela; nova_tabela];
80      end
81
82  end
83
84  % % Exibindo a tabela
85  % disp(tabela);
86
87  % Especifique o nome do arquivo Excel de saida
88  dados_excel = 'dados.xlsx';
89
90  % Use a funcao writetable para salvar a tabela como um
        arquivo Excel
91  writetable(tabela, dados_excel, 'Sheet', 'Planilha1');
92  % Calculo do tempo de treinamento
93  tempo_decorrido = toc;
94  disp('Tempo de processamento do treinamento: ');
95  disp([num2str(tempo_decorrido), ' segundos']);
96  disp([num2str(tempo_decorrido/60), ' minutos']);
```

APÊNDICE B – SCRIPT DE SEPARAÇÃO DE DADOS

Código-fonte 2 – Separação de dados

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

url = r'/content/drive/MyDrive/TCC/novo/Dados finais.xlsx'
dados = pd.read_excel(url)
# Selecionar as colunas de entrada
colunas_entrada = ['Amplitude_degrau', 'Tempo_degrau', '
   RiseTime', 'SettlingTime', 'Overshoot']
x = dados[colunas_entrada]

# Selecionar as colunas de saida
colunas_saida = ['Kp', 'Ti', 'Td', 'b', 'c', 'Tf']
y = dados[colunas_saida]

x_train, x_test, y_train, y_test = train_test_split(x, y,
   test_size=0.25, random_state=58)

# Criar objetos de MinMaxScaler para cada conjunto
scaler = StandardScaler()
x_train_normalized = scaler.fit_transform(x_train)
x_test_normalized = scaler.fit_transform(x_test)

print(f"Dados Treinamento - {x_train.shape}\nDados Teste -
   {x_test.shape}\n ")
```

APÊNDICE C – SCRIPT DE VALIDAÇÃO

Código-fonte 3 – Validação de modelos

```
from sklearn.metrics import mean_squared_error,
    mean_absolute_error, r2_score
from math import sqrt

# Faca previsoes no conjunto de teste
y_pred = modelRegressor.predict(x_test_normalized)#MUDAR
    AQUI O MODELO A SER AVALIADO

# Avalie o desempenho do modelo
test_loss = mean_squared_error(y_test, y_pred)
test_mae = mean_absolute_error(y_test, y_pred)
rmse = np.sqrt(test_loss)
r2 = r2_score(y_test, y_pred)

# Exiba as metricas
print("Metricas do BaggingRegressor")
print(f"Test MSE: {test_loss}")
print(f"Test MAE: {test_mae}")
print(f"Test RMSE: {rmse}")
print(f"Test R2: {r2}")
```

APÊNDICE D – SCRIPT DE TREINAMENDO DA MLP

Código-fonte 4 – Treinamento de MLP

```
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam
from keras.layers import *
from keras.callbacks import EarlyStopping
mlp = Sequential()
mlp.add(Dense(units=256, input_shape=[4], activation=
    LeakyReLU(alpha=0.01)))
mlp.add(Dense(units=256, activation=LeakyReLU(alpha=0.01)))
mlp.add(Dense(units=256, activation=LeakyReLU(alpha=1.0)))
mlp.add(Dense(units=6, activation='linear'))

# Compile o modelo MLP com otimizador, funcao de perda e
    metricas apropriados
mlp.compile(optimizer=Adam(learning_rate=0.001, beta_1
    =0.45, beta_2=0.5), loss='mean_squared_error', metrics=[
    'mean_squared_error'])

# Adicione um callback de parada antecipada para monitorar
    a perda de validacao
early_stopping = EarlyStopping(monitor='val_loss', patience
    =10, restore_best_weights=True)

# Treine o modelo MLP
history = mlp.fit(x_train_normalized, y_train, epochs=300,
    batch_size=128, callbacks=[early_stopping],
    validation_split=0.25, verbose=False)
```

APÊNDICE E – SCRIPT DE TREINAMENDO DA DECISIONTREE

Código-fonte 5 – Treinamento de DecisionTree

```
from sklearn.tree import DecisionTreeRegressor

# Hiperparametros da Arvore de Decisao
max_depth = 64
min_samples_split = 20
min_samples_leaf = 1

# Crie o modelo de Arvore de Decisao
decisionTreeRegressor = DecisionTreeRegressor(max_depth=
    max_depth, min_samples_split=min_samples_split,
    min_samples_leaf=min_samples_leaf)

# Treine o modelo de Arvore de Decisao
decisionTreeRegressor.fit(x_train_normalized, y_train)
```

APÊNDICE F – SCRIPT DE TREINAMENDO DA BAGGING

Código-fonte 6 – Treinamento de Bagging

```
from sklearn.tree import DecisionTreeRegressor  # Importe o
    regressor de Arvore de Decisao
from sklearn.ensemble import BaggingRegressor

# Hiperparametros da Arvore de Decisao
max_depth = 64
min_samples_split = 20
min_samples_leaf = 1

# Crie o modelo de Arvore de Decisao
base_model = DecisionTreeRegressor(max_depth=max_depth,
   min_samples_split=min_samples_split, min_samples_leaf=
   min_samples_leaf)

# Hiperparametros do Bagging Regressor (mantidos os mesmos)
n_estimators = 1000
max_samples = 0.8
bootstrap = True
random_state = 42

# Resto do codigo e igual
baggingRegressor = BaggingRegressor(base_model,
   n_estimators=n_estimators, max_samples=max_samples,
   bootstrap=bootstrap, random_state=random_state)

# Treine o modelo
baggingRegressor.fit(x_train_normalized, y_train)
```

APÊNDICE G – SCRIPT DE TREINAMENDO DA EXTRATREES

Código-fonte 7 – Treinamento de ExtraTrees

```
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.metrics import mean_squared_error,
   mean_absolute_error, r2_score
from math import sqrt

# Crie e treine o modelo ExtraTreesRegressor
extraTreesRegressor = ExtraTreesRegressor(n_estimators
   =2000, max_depth=128, min_samples_split=40,
   min_samples_leaf=1, max_features=0.8461, random_state
   =42)
extraTreesRegressor.fit(x_train_normalized, y_train)
```

APÊNDICE H – SCRIPT DE TREINAMENDO DO VOTING

Código-fonte 8 – Treinamento de Voting

```
voting_regressor = (mlp  + extra  + bag )/3
# Faca previsoes usando o modelo de ensemble
y_pred = voting_regressor

# Avalie o desempenho do modelo de ensemble
mse = mean_squared_error(y_test, y_pred)
rmse = sqrt(mse)
mae = mean_absolute_error(y_test, y_pred)
r2 = r2_score(y_test, y_pred)

# Fa a a previsao
predicted_output_1 = mlp.predict(new_input_normalized)
predicted_output_2 = extraTreesRegressor.predict(
   new_input_normalized)
predicted_output_3 = baggingRegressor.predict(
   new_input_normalized)

predicted_output = (
    predicted_output_1 + predicted_output_2  +
    predicted_output_3
    )/3
```

APÊNDICE I – SCRIPT DE NOVO TESTE

Código-fonte 9 – Novo Teste

```
# Pre-processamento da linha de entrada (aplique a mesma normalizacao)
new_input_normalized = scaler.transform(new_input)  # Use o mesmo scaler que foi usado para normalizar os dados de treinamento

# Faca a previsao
predicted_output_1 = extraTreesRegressor.predict(new_input_normalized)
predicted_output_2 = mlp.predict(new_input_normalized)
predicted_output_3 = baggingRegressor.predict(new_input_normalized)
predicted_output = (
    predicted_output_1 + predicted_output_2  +
    predicted_output_3
    )/3
# Realiza a previsao com a RNA treinada

# Crie um DataFrame com rotulos de coluna
predicted_df = pd.DataFrame(predicted_output, columns=colunas_saida)  # "colunas_saida"    uma lista de nomes de colunas de saida

# Exiba o DataFrame com os valores previstos
print("Valores Preditos:")
print(predicted_df)
```