INSTITUTO FEDERAL DE EDUCAÇÃO CIÊNCIA E TECNOLOGIA DE MINAS GERAIS - *CAMPUS* BETIM
BACHARELADO EM ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Arthur Carlos de Faria

REDE NEURAL APLICADA: Modelagem e Controle em Robôs Quadrúpedes

Betim
2023

ARTHUR CARLOS DE FARIA

REDE NEURAL APLICADA: Modelagem e Controle em Robôs Quadrúpedes

Trabalho de Conclusão de Curso apresentado à banca examinadora do curso de Engenharia de Controle e Automação do Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais *Campus* Betim, como parte dos requisitos para obtenção do título de Bacharel em Engenharia de Controle e Automação.

Orientador: Leandro Freitas de Abreu
Coorientador: Virgil Del Duca Almeida

Betim

2023

FICHA CATALOGRÁFICA

F224r Faria, Arthur Carlos de
Rede neural aplicada: modelagem e controle em robôs quadrúpedes / Arthur Carlos de Faria. – 2023.

82 f. : il.

Trabalho de conclusão de curso (Bacharelado em Engenharia de Controle e Automação) - Instituto Federal de Educação, Ciência e Tecnologia de Minas Gerais, Câmpus Betim, 2023.

Orientador: Prof. Dr. Leandro Freitas de Abreu
Coorientador: Prof. Me. Virgil Del Duca Almeida

1. Redes neurais. 2. Robótica. 3. Sistemas embarcados. 4. Controle. 5. Automação. I. Faria, Arthur Carlos de. II. Título.

CDU: 681.5

Ficha catalográfica elaborada pelo Bibliotecário Denísio P. Marcos CRB-6/3142

Arthur Carlos de Faria

REDE NEURAL APLICADA: modelagem e controle em robótica móvel

Trabalho de Conclusão de Curso apresentado ao curso de Engenharia de Controle e Automação do Instituto Federal de Minas Gerais Campus Betim como requisito parcial à obtenção do grau de Bacharel em Engenharia de Controle e Automação.

Betim, 13 de dezembro de 2023.

BANCA EXAMINADORA

Prof. Dr. Leandro Freitas
DAUTI – IFMG Campus Betim (orientador)

Prof. Me. Virgil Del Duca Almeida
DAUTI – IFMG Campus Betim (co-orientador)

Profª. Me. Michelle Mendes Santos
DAUTI – IFMG Campus Betim

Eng. Carlos Roberto Diniz
IHM

RESUMO

A aplicação de Redes Neurais é fundamental para identificar padrões, estabelecer correlações, classificar e prever resultados. Este estudo utiliza essas capacidades para desenvolver um Sistema Embarcado em Robótica Móvel (SERM) que consiste em duas Redes Neurais. A primeira age como um 'modelo', treinando o controlador para agir de forma preditiva, enquanto a segunda desempenha o papel do controlador. O objetivo é criar um controlador neural adaptável, capaz de aprender com o ambiente por meio dos dados sensoriais coletados. Esse controlador ajusta suas ações com base nessas informações, garantindo que as respostas atendam aos critérios de desempenho estabelecidos.

Além da implementação de modelos neurais, este estudo investigou uma abordagem integrada que conjuga técnicas de previsão e controle, explorando as potencialidades das redes neurais em um contexto de robótica. O êxito observado nas estratégias propostas sugere perspectivas para pesquisas futuras, contemplando variadas arquiteturas de redes neurais e refinamentos na dinâmica do sistema.

O sistema de controle proposto destaca-se por sua adaptabilidade, capacidade de aprendizado contínuo e eficiência operacional em ambientes caracterizados por dinâmicas desafiantes, abrindo caminhos promissores para o desenvolvimento de novos paradigmas de controladores.

Palavras-chave: Redes Neurais, Robótica Móvel, Sistema Embarcado, Controle Adaptável, Previsão, Aprendizado de Máquina, Inteligência Artificial, Ambientes Dinâmicos.

ABSTRACT

The application of Neural Networks is fundamental for pattern identification, correlation establishment, classification, and result prediction. This study leverages these capabilities to develop a Mobile Robotic Embedded System (MRES) consisting of two Neural Networks. The first acts as a 'model,' training the controller to operate predictively, while the second serves as the controller. The goal is to create an adaptable neural controller capable of learning from the environment through collected sensory data. This controller adjusts its actions based on this information, ensuring responses meet established performance criteria.

Beyond the implementation of neural models, this study investigates an integrated approach that combines prediction and control techniques, exploring the potential of neural networks in a robotics context. The success observed in the proposed strategies suggests prospects for future research, considering various neural network architectures and refinements in system dynamics.

The proposed control system stands out for its adaptability, continuous learning capability, and operational efficiency in environments characterized by challenging dynamics, paving the way for the development of new controller paradigms.

Keywords: Neural Networks, Mobile Robotics, Embedded Systems, Adaptive Control, Prediction, Machine Learning, Artificial Intelligence, Dynamic Environments.

SUMÁRIO

1 INTRODUÇÃO

A utilização de robôs com controlador embutido, ou como são conhecidos, os sistemas embarcados, está em constante crescimento. Atualmente, diversas tentativas de aprimoramento buscam capacitá-los a analisar e aprender com o ambiente circundante, almejando atingir metas específicas. Diante dessa crescente demanda por avanços na área, o presente trabalho visa contribuir para o desenvolvimento desses sistemas.

Neste contexto, foi explorada e aplicada uma técnica de aprendizado baseada em redes neurais com o objetivo de desenvolver um Sistema Embarcado em Robótica Móvel (SERM). Este sistema é projetado não apenas para predizer seus estados futuros, mas também para controlar os atuadores de maneira adaptativa, especialmente em torno do problema detalhado na Seção 1.1.

O modelo selecionado para o SERM foi um robô quadrúpede, o qual foi posicionado sobre uma superfície plana e teve a orientação de seu corpo tridimensional descrito por um sistema de coordenadas fixo, representados pelas três letras gregas: φ (phi) para o ângulo de rotação em torno do eixo z (roll), θ (theta) para o ângulo de rotação em torno do eixo y (pitch), e ψ (psi) para o ângulo de rotação em torno do eixo x (yaw), conforme a Figura 1.

Figura 1 – Sistema de Coordenadas do SERM

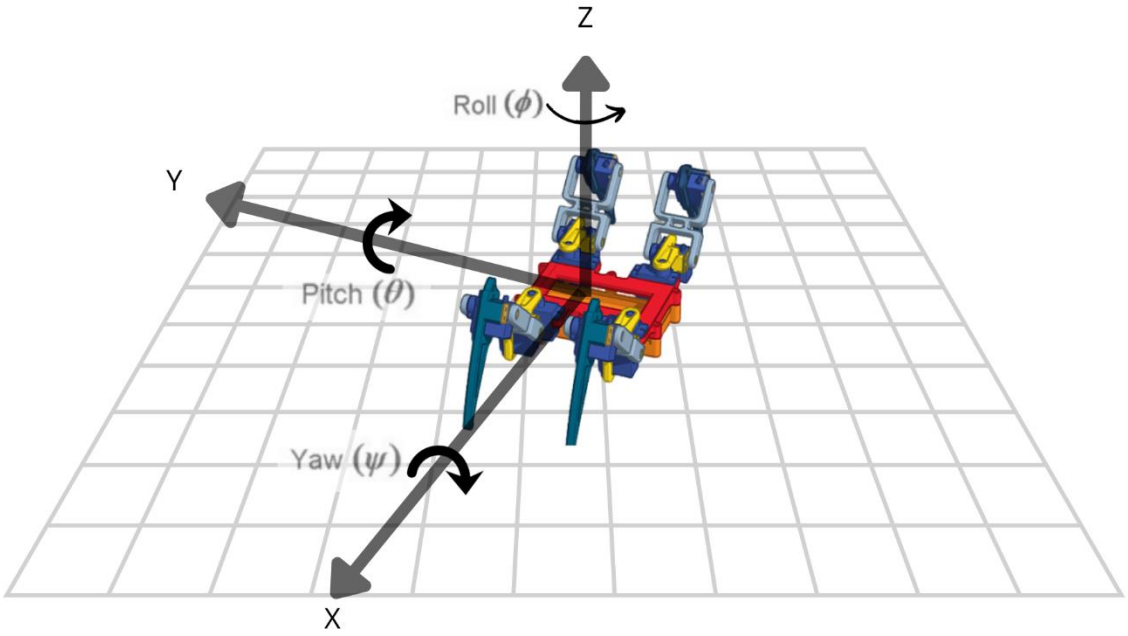


Figura 1: Sistema de Coordenadas do SERM. Fonte: Autor.

1.1. Colocação do Problema

O problema central que este trabalho se propõe a resolver reside na construção do SERM e na síntese de um controlador preditivo e adaptativo utilizando duas redes neurais, uma responsável por modelar o comportamento do robô e outra responsável por seu controle.

No escopo deste estudo, o robô quadrúpede em questão enfrenta o desafio de se manter em equilíbrio e seguir uma referência para os ângulos ψ e ψ, sendo comandado por meio de comandos enviados via Python, enquanto as respostas dos sensores e os comandos para os motores são intermediados pelo Arduino por meio da comunicação serial.

No escopo deste estudo, destaca-se que o controle SERM teve foco exclusivamente nos ângulos ψ e ψ. Essa delimitação visou direcionar a atenção para a precisão e otimização do controle desses parâmetros específicos, oferecendo uma abordagem mais especializada e eficiente na busca pela estabilidade e na execução da trajetória pré-definida do robô quadrúpede sobre a superfície plana, em torno desses eixos.

Portanto, o enfoque central deste trabalho concentrou-se no processo de implementação e na investigação da eficácia do método utilizado em alcançar os critérios de desempenho pré-determinados na Tabela 1.

1.2. Objetivos

Este trabalho se concentra na construção de dois modelos de redes neurais para controlar o SERM. O modelo I (Rede Neural Preditiva) tem como objetivo prever os próximos N estados dos sensores com base no estado inicial, enquanto o modelo II (Controlador) visa escolher os comandos que serão enviados para os seus atuadores via comunicação serial. O objetivo principal deste trabalho é aplicar, registrar e avaliar os resultados desta abordagem.

Objetivos específicos:

- Projetar e desenvolver uma rede neural capaz de prever o comportamento dos ângulos ψ (em torno de X) e θ (em torno de Y) em relação aos comandos dados;

- Desenvolver uma rede neural capaz realizar o controle de forma adaptativa;

- Realizar experimentos práticos para validar a eficácia dos algoritmos de controle desenvolvidos e das redes neurais;

- Analisar quantitativamente o desempenho dos algoritmos em termos de estabilidade, precisão e tempo de resposta, identificando áreas de melhoria e refinando as estratégias de controle conforme necessário, com base nos critérios de desempenho pré-determinados na Tabela 1.

Tabela 1 – Critérios de Desempenho

Critérios de Desempenho da Rede Preditiva		
Critério	**Faixa que atende ao critério**	**Unidade**
Precisão/Acurácia nos dados de treinamento	>=95	%
Precisão/acurácia nos dados de teste de previsão simples	>=90	%
Número de pontos previstos no teste de predição livre, com acurácia de 85%.	3	Pontos
Critérios de Desempenho do Controlador		
Critério	**Faixa que atende ao critério**	**Unidade**
Tempo de subida para entrada em degrau	<= 1	S
Tempo de acomodação para entrada em degrau	<= 1,5	S
Erro para resposta ao degrau unitário	<= 5	%
Erro para resposta à senoide	**<= 7**	**%**
Tempo de acomodação para entrada senoidal	<= 60	S

Os critérios foram definidos com base na precisão dos sensores, tempo de leitura do Arduino e tempo de resposta para a comunicação serial. Os critérios de desempenho apresentados, podem ser descritos como:

- **A precisão ou acurácia nos dados de treinamento:** Mede a proporção de predições corretas em relação ao número total de amostras de treinamento.

Uma alta precisão nos dados de treinamento sugere que o modelo está capturando bem os padrões presentes nesses dados específicos;

- **Precisão/Acurácia nos dados de teste de previsão simples:** Mede a proporção de predições corretas em relação ao número total de exemplos nos dados de teste. Uma boa precisão nos dados de teste sugere que o modelo pode generalizar bem para novos dados, não vistos durante o treinamento;
- **Número de pontos previstos no teste de predição livre:** Essa métrica é especialmente relevante para avaliar se o modelo está se comportando como um preditor trivial, isto é, se o modelo está assumindo que o ponto previsto é igual ao ponto atual ou se está realmente realizando previsões com base em inferências significativas. Essa abordagem visa identificar, visualmente, em quantas amostras a previsão torna-se incoerente. Em outras palavras, o objetivo é observar graficamente se a dinâmica prevista pelo modelo difere da dinâmica real ao longo do tempo.
- **Tempo de Subida:** O tempo de subida é uma métrica que quantifica o período necessário para que o sistema atinja a sua condição estacionária a partir do momento em que uma perturbação é aplicada. É desejável um tempo de subida curto, indicando uma rápida resposta do sistema à mudança.
- **Tempo de Acomodação:** O tempo de acomodação é o período necessário para que a resposta do sistema alcance e permaneça dentro de uma faixa específica em torno do valor final após uma perturbação. Esse tempo reflete a estabilidade e a capacidade do sistema de retornar ao seu estado de equilíbrio após uma perturbação.
- **Erro para Resposta ao Degrau Unitário:** O erro para resposta ao degrau unitário representa a diferença entre o valor final desejado e o valor final real da resposta do sistema quando submetido a uma entrada de degrau unitário, o qual pode variar de 0 a 180. É uma métrica importante para avaliar o quão bem o sistema atende às expectativas em termos de resposta ao degrau.
- **Erro para Resposta à Senoide:** Similar ao erro para resposta ao degrau unitário, o erro para resposta à senoide mede a discrepância entre a resposta desejada e a resposta real do sistema quando submetido a uma entrada senoidal.

1.3. Justificativa

A principal dificuldade enfrentada em modelar o SERM é a não linearidade introduzida pelas numerosas variáveis físicas que afetam diretamente a dinâmica do robô, como a distribuição de massa, a inércia, as características geométricas, as fricções nos motores, e outros fatores físicos que influenciam de maneira intrínseca o comportamento dinâmico do sistema. Essa complexidade é ampliada pela interação dinâmica entre essas variáveis, tornando a modelagem matemática uma tarefa desafiadora e exigindo abordagens avançadas para garantir a precisão na representação do SERM, como mostra o artigo de Souto (2007), ao tentar reproduzir

o comportamento do robô quadrúpede, exibido na Figura 2, através de um modelo matemático para estimar o estado do sistema.

Figura 2 – Robô Quadrúpede de Trabalho Correlato

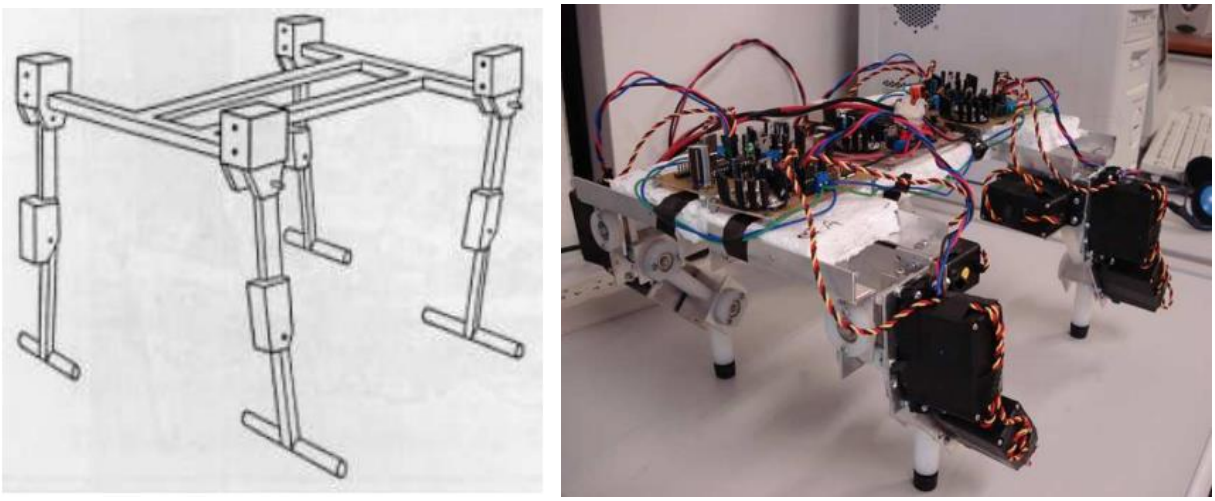


Figura 2: Robô quadrúpede. Fonte: Souto (2007).

Com isto, este trabalho busca explorar uma abordagem não convencional no projeto de controladores para sistemas robóticos, onde a modelagem e o controle são feitos utilizando técnicas de Inteligência Artificial (IA), incluindo controladores preditivos e técnicas de aprendizado por reforço com algoritmos de gradiente descendente. As Redes Neurais apresentam-se como uma escolha vantajosa à solução deste problema devido às suas capacidades de representar não linearidades e aprendizado que se adapta ao modelo e ao ambiente.

1.4 Organização do Trabalho

O trabalho está estruturado conforme a seguir.

Capítulo 2 - Fundamentação Teórica: Este capítulo aborda os métodos, teorias e trabalhos anteriores relacionados ao Controle Preditivo, Redes Neurais, Modelagem e sistemas embarcados relevantes para o contexto do robô. Além disso, serão apresentados detalhes sobre o sistema de controle e o algoritmo a ser utilizado.

Capítulo 3 - Metodologia: Neste capítulo, o trabalho é detalhado de maneira sequencial. Começa-se com a definição dos critérios a serem analisados durante os testes, seguida pela modelagem e controle do robô. O capítulo é concluído com a validação do desempenho do robô quadrúpede ao longo do treinamento e dos testes.

Capítulo 4 – Resultados e Discussões: Neste capítulo, são apresentados os resultados obtidos durante os experimentos realizados. Inicialmente, os dados

coletados durante o treinamento do SERM são avaliados, analisando-se as respostas do sistema em diferentes cenários. Em seguida, são apresentados os resultados dos testes de validação, onde o desempenho do robô é avaliado em condições diversas. As análises se concentram na interpretação dos resultados, comparando o desempenho observado com as expectativas teóricas. São abordadas as limitações do estudo e possíveis áreas para pesquisas futuras, proporcionando uma análise crítica do trabalho desenvolvido.

Capítulo 5 – Conclusão: No último capítulo, são destacados os pontos fortes e principais observações do modelo proposto. Além disso, são discutidas as implicações práticas dos resultados alcançados e como eles podem ser aplicados em contextos diversos. A conclusão também aborda as contribuições para o campo de pesquisa em robótica e encerra com reflexões sobre o impacto do trabalho realizado e sua relevância para futuros estudos na área.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo estabelece a base teórica e contextualiza a metodologia empregada neste trabalho, traçando um panorama das teorias e estudos anteriores que fundamentam a aplicação desta técnica. Compreender esses estudos anteriores não apenas fornece uma base sólida para a modelagem e controle do SERM, mas também amplia a compreensão sobre a importância de investigar e aplicar essas metodologias inovadoras na busca contínua por robôs mais inteligentes, autônomos e eficientes.

2.1 Robótica Móvel

A robótica móvel, conforme o nome sugere, é um ramo da engenharia robótica que se concentra no desenvolvimento, operação e controle de robôs capazes de se locomover em seu ambiente. Diferentemente dos robôs estacionários, que permanecem fixos em uma posição, os robôs móveis são projetados para se moverem de forma autônoma ou controlada para executar diversas tarefas em ambientes variados e dinâmicos.

Um aspecto fundamental da robótica móvel é o sistema de coordenadas Euleriano, que descreve a orientação e a posição de um objeto tridimensional em relação a um sistema de referência fixo. Conforme mostra Lage (2020), este sistema consiste em três ângulos (roll, pitch e yaw), representados pelas letras gregas φ (phi), θ (theta) e ψ (psi) que representam os movimentos de rotação em torno dos eixos X, Y e Z, respectivamente. Esses ângulos fornecem uma maneira precisa de descrever a orientação e a posição de um robô móvel em relação ao seu ambiente, permitindo um controle eficaz de seus movimentos e ações.

Na Robótica, conforme abordado por Wolf (2009), diversos temas desempenham papéis cruciais para o desenvolvimento e aplicação de sistemas eficazes. Sensores e atuadores são componentes essenciais, fornecendo dados sobre o ambiente e realizando as ações físicas necessárias para o movimento e execução de tarefas específicas. O controle de movimento envolve o desenvolvimento de algoritmos para determinar a trajetória e os movimentos precisos do robô, garantindo sua locomoção segura e eficiente em ambientes desafiadores. A aplicação de inteligência artificial e aprendizado de máquina capacita os robôs a tomar decisões autônomas com base em dados sensoriais em tempo real e a aprender com a

experiência, melhorando sua capacidade de interação com o ambiente. Esses avanços têm implicações significativas em diversos setores, incluindo logística, indústria, exploração espacial, assistência médica, agricultura, segurança e entretenimento. Em cada um desses domínios, os sistemas robóticos desempenham funções vitais, aumentando a eficiência, segurança e qualidade das operações, e possibilitando novas oportunidades e aplicações inovadoras. Assim, a robótica móvel continua a desempenhar um papel crucial no avanço tecnológico e na melhoria da qualidade de vida em todo o mundo.

Este trabalho concentrou-se especificamente no desenvolvimento e estudo de um modelo de robô quadrúpede.

2.1.1 Robôs Quadrúpedes

Manter-se em pé e responder dinamicamente às mudanças no ambiente para manter o equilíbrio são tarefas que os seres humanos executam naturalmente, muitas vezes sem perceber que se trata de uma habilidade complexa que lida com diversas variáveis. O estudo do comportamento dos robôs com pernas tem sido uma área de pesquisa crucial no campo da robótica, oferecendo conhecimentos valiosos sobre a locomoção eficiente em ambientes complexos.

O trabalho intitulado de Castro (2012), apresentado no contexto do Mestrado em Engenharia Eletrotécnica e de Computadores, explora de maneira detalhada a modelagem e simulação de um robô quadrúpede, o qual demandou uma expertise técnica significativa para realizar os cálculos detalhados e determinar parâmetros como massa, inércia e geometria. Além disso, qualquer alteração no design do robô exigiria uma revisão extensa e meticulosa desses cálculos, tornando o processo bastante inflexível.

O presente estudo adotou uma abordagem mais prática e adaptativa. As redes neurais foram treinadas para aprender padrões a partir dos dados sensoriais do robô, eliminando assim a necessidade de cálculos manuais complexos. Isso não apenas simplifica o processo de modelagem, mas também permite que o robô se adapte dinamicamente a diferentes condições, tornando-o mais versátil e capaz de lidar com cenários variados de forma autônoma. Essa abordagem economiza tempo e esforço, além de melhorar a adaptabilidade à diferentes modelos de robôs. Estes são alguns benefícios de integrar técnicas de Redes Neurais Artificiais na modelagem de sistemas robóticos complexos.

2.4 Redes Neurais Artificiais

De acordo com Heinen e Osório (2007), o desenvolvimento de robôs com pernas é uma tarefa extremamente desafiadora. Configurar manualmente os parâmetros de movimento desses robôs exige um investimento significativo de tempo por parte de especialistas humanos. Além disso, os resultados obtidos por meio dessa abordagem são frequentemente sub-ótimos e altamente sensíveis à arquitetura específica do robô.

Por este motivo, é crescente a utilização de Redes Neurais Artificiais (RNAs) para resolução de problemas semelhantes, pois são ferramentas poderosas inspiradas pelo funcionamento do cérebro humano que podem identificar padrões complexos nos dados, permitindo que os robôs ajam de forma mais inteligente e adaptativa. As RNAs consistem em camadas de neurônios artificiais interconectados, onde cada conexão possui um peso que é ajustado durante o treinamento da rede.

As redes neurais possuem as seguintes características, conforme Kovács (2006):

- **Simplicidade na Configuração -** Redes neurais podem aprender a partir dos dados coletados, eliminando a necessidade de configuração manual extensa de parâmetros, o que reduz significativamente o esforço e o tempo necessários para o desenvolvimento do modelo.

- **Captura de Padrões Complexos -** Redes neurais são capazes de capturar padrões complexos e não lineares nos dados, o que é especialmente valioso em sistemas onde as relações entre variáveis são intrincadas e difíceis de serem modeladas manualmente.

- **Robustez às Perturbações -** Redes neurais podem lidar melhor com pequenas variações e perturbações nos dados de entrada, tornando os modelos mais robustos e capazes de fornecer previsões precisas mesmo em condições variáveis.

- **Adaptabilidade Dinâmica -** Redes neurais podem se adaptar a mudanças nas condições do ambiente e nos dados de entrada ao longo do tempo. Elas conseguem ajustar seus parâmetros automaticamente à medida que novos dados são apresentados, garantindo um desempenho consistente.

- **Redução da Dependência de Especialistas -** Ao contrário dos modelos manuais, o treinamento de redes neurais não requer conhecimento especializado profundo sobre o sistema. Isso reduz a dependência de especialistas altamente qualificados e torna o processo mais acessível.

- **Generalização para Novos Dados -** Redes neurais são capazes de generalizar padrões aprendidos para dados não vistos anteriormente, permitindo que o modelo faça previsões precisas mesmo para situações não encontradas durante o treinamento.

- **Capacidade para Lidar com Dados Não Estruturados -** Redes neurais são eficazes em lidar com dados não estruturados, como texto, áudio e imagem, expandindo a aplicabilidade do modelo para uma variedade de tipos de dados.

- **Identificação de Relações Complexas -** Em sistemas com relações altamente complexas entre variáveis, as redes neurais podem identificar e modelar essas relações de maneira precisa, oferecendo uma representação mais fiel do sistema.

- **Melhoria Contínua -** Redes neurais podem ser aprimoradas continuamente à medida que mais dados se tornam disponíveis, resultando em modelos cada vez mais precisos e eficientes.

As classificações de redes neurais em termos estruturais são conceitos amplamente reconhecidos e utilizados na comunidade de aprendizado de máquina e redes neurais, como pode ser visto no trabalho de Pereira (2020). As redes neurais podem ser classificadas de acordo com a sua estrutura como:

- **Largas/Estreitas -** A largura de uma rede neural refere-se ao número de unidades em cada camada. Redes neurais largas têm muitas unidades em uma ou mais camadas, enquanto redes estreitas têm menos unidades em cada camada, conforme ilustra a Figura 3.

Figura 3 – Comparação de Rede Rasa e Profunda

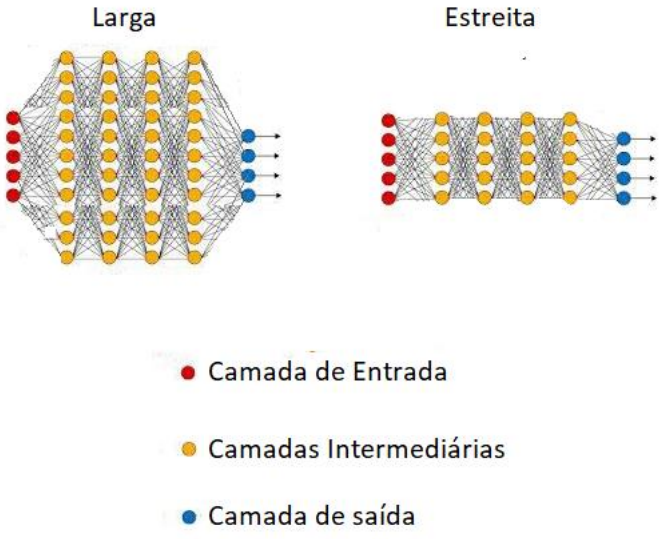


Figura 3: Imagem adaptada de Rede Neural Larga e Estreita. Fonte: "Deep Learning Book".

- **Profundas/Rasas -** A profundidade de uma rede neural se refere ao número de camadas ocultas entre a entrada e a saída. Redes neurais profundas têm várias camadas ocultas, enquanto redes rasas têm apenas uma ou duas camadas ocultas, conforme ilustra a Figura 4.

Figura 4 – Comparação de Rede Rasa e Profunda

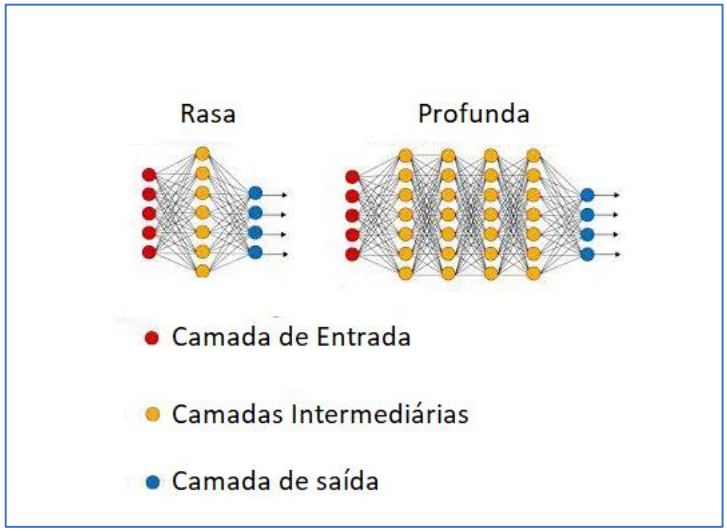


Figura 4: Imagem adaptada de Rede Neural Profunda e Rasa. Fonte: "Deep Learning Book".

Além das arquiteturas, as funções de ativação desempenham um papel crucial em redes neurais, contribuindo para a introdução de não linearidades nos modelos, permitindo-lhes capturar relações complexas nos dados. Existem diversas funções de ativação com características específicas, como pode ser visto na obra de Haykin (2001), cada uma adequada para diferentes cenários.

1. **Função Sigmoide:**
    - **Equação:** $F(x) = \frac{1}{1+e^{-z}}$
    - **Intervalo de Saída:** $(0, 1)$
    - **Aplicações:** Amplamente utilizada em camadas de saída de modelos de classificação binária, pois mapeia a entrada para uma escala entre 0 e 1, interpretável como uma probabilidade.
2. **ReLU (*Rectified Linear Unit):***
    - **Equação:** $F(x) = \max(0, x)$
    - **Intervalo de Saída:** $[0, \infty)$
    - **Aplicações:** Amplamente empregada em camadas ocultas. Eficiente e fácil de otimizar, ajuda na mitigação do problema de desvanecimento do gradiente.
3. **Função Linear:**
    - **Equação:** $F(x) = x$
    - **Intervalo de Saída:** $(-\infty, \infty)$
    - **Aplicações:** Usada em camadas de saída para problemas de regressão, onde se deseja uma saída linear em relação à entrada.

Além dessas funções, neste trabalho, foi utilizada a função *ReLU6*, similar à ReLU, mas com uma saturação para valores maiores que 6. É útil em casos em que a saída precisa ser limitada a um intervalo específico que é o caso dos atuadores do presente trabalho.

Estes conceitos são essenciais para a compreensão das estruturas empregadas neste trabalho.

2.2.1 Tipos de Redes Neurais Exploradas no Trabalho

Para a rede neural utilizada no controlador, foram testadas as seguintes estruturas:

- **Rede Estreita e Profunda (*Narrow Deep*):** Possui múltiplas camadas ocultas, mas com número limitado de neurônios em cada camada e boa capacidade para representar informações complexas e hierárquicas. Menor probabilidade de overfitting. Eficiência em termos de recursos computacionais. Possui necessidade de ajuste cuidadoso de hiper parâmetros, como mostra o trabalho de Reis (2018), sobre a otimização de parâmetros em redes neurais profundas.
- **Rede Larga e Profunda (*Wide Deep*):** Possui múltiplas camadas ocultas, cada uma com muitos neurônios e pode aprender representações ricas e complexas dos dados. Eficaz para tarefas de controle sofisticadas. Captura padrões em dados multidimensionais, porém requer recursos computacionais substanciais. Suscetível a overfitting, como mostra o trabalho de Oliveira et al. (2017). Treinamento pode ser lento devido ao grande número de parâmetros.
- **Rede Poucas Camadas e Larga (*Shallow Wide*):** Possui poucas camadas ocultas, mas com muitos neurônios em cada camada, portanto tem a vantagem de um rápido e uso eficiente de recursos. É adequada para aplicações de baixa latência e tarefas de controle simples. Em contrapartida, é limitada em tarefas complexas e cenários dinâmicos. Não ideal para variáveis de estado múltiplas.
- **Rede Estreita e Poucas Camadas (*Narrow Shallow*):** Possui poucas camadas ocultas e número limitado de neurônios em cada camada. Tem grande eficiência computacional e consumo de energia reduzido. Treinamento mais rápido para respostas rápidas do robô. Adequada para tarefas de controle simples. Entretanto é Limitada em tarefas complexas e cenários desafiadores. Não aproveita completamente o potencial das redes neurais profundas.
- **Rede Profunda, larga nas camadas iniciais e estreita nas camadas finais:** O afunilamento das camadas permite que a rede aprenda a extrair características cada vez mais abstratas e hierárquicas dos dados, exemplificado pela arquitetura apresentada pelo estudo de Ribeiro (2021).

2.2.2 Treinamento da Rede Preditiva Pelo Gradiente Descendente

No contexto das redes neurais, o gradiente descendente é usado para ajustar os pesos e os vieses da rede de forma a minimizar uma função de perda. A ideia básica é calcular o gradiente da função de perda em relação aos parâmetros da

rede e, em seguida, ajustar esses parâmetros na direção oposta ao gradiente. Isso significa que, se a função de perda está diminuindo em relação a um determinado parâmetro, esse parâmetro é aumentado para minimizar a perda, e vice-versa. Esse processo é repetido iterativamente até que a rede atinja um estado onde a função de perda é minimizada, indicando que os parâmetros foram ajustados para melhor se adequar aos dados de treinamento. Esse método de aprendizado foi utilizado pelo trabalho de Araújo (2012) para construir um sistema capaz de estimar o futuro a partir de dados do passado e presente.

No âmbito deste trabalho, busca-se emular a abordagem de Araújo ao empregar os dados armazenados para treinar uma rede neural por meio do algoritmo do gradiente descendente. A intenção é capacitar essa rede neural a prever o comportamento do SERM com base em seu estado atual e nos estados anteriores. Esse método, similar ao utilizado no trabalho referenciado, diverge no seguinte contexto: enquanto o estudo anterior se concentrava na previsão de séries temporais em um contexto financeiro, o presente trabalho visa antecipar o comportamento em um SERM.

2.2.3 Tratamento do Controlador através do Aprendizado por Reforço

O aprendizado por reforço é uma modalidade de treinamento em que um agente interage com um ambiente de maneira a alcançar uma meta ou maximizar uma recompensa ao longo do tempo, como mostra o trabalho de Monteiro e Ribeiro (2004), que, assim como o presente trabalho, utiliza o algoritmo de aprendizado por reforço em robótica móvel a fim de avaliar o seu desempenho. No trabalho de Monteiro e Ribeiro, o objetivo do agente é selecionar ações de forma a maximizar a soma ponderada das recompensas futuras. Essa soma é descontada ao longo do tempo, onde o fator de desconto temporal $\gamma\,(0 < \gamma < 1)$ controla a importância das recompensas futuras e k o intervalo de medição das recompensas de cada ação, conforme é descrito na fórmula a seguir:

$$R_t = \sum_{k=0}^{T} \gamma^k r_{t+k}\,, \qquad (1)$$

O ambiente é modelado como um Processo de Decisão Markoviano (MDP), e as interações ocorrem em passos discretos no tempo. Em um determinado estado s e ação a, o agente recebe reforço r e transita para um novo estado s' com uma certa probabilidade. O objetivo do agente é escolher ações para maximizar a soma descontada dos reforços futuros

,

$$Q^{\pi}(s,a) = E_{\pi}\{R_t | s_t = s, a_t = a\}\,, \qquad (2)$$

A função de valor de ação $Q(s,a)$ sob uma política E_{π}, representa o valor esperado da soma das recompensas quando o agente está no estado s, toma a ação a, e segue a política π a partir desse ponto. A política ótima E é aquela que maximiza $Q(s,a)$, ou seja, aquela em que a recompensa é máxima.

No processo de implementação do aprendizado por reforço na rede neural do controlador, foi adotada uma estratégia para lidar com a questão do desconto temporal. Para realizar essa tarefa, foram utilizados os valores previstos pela rede preditiva como base para aplicar o desconto temporal aos retornos recebidos pelo robô durante suas interações com o ambiente, tanto o ambiente simulado quanto no ambiente real. Essa abordagem permitiu que o robô atribuísse valores ponderados às ações tomadas, considerando não apenas as recompensas imediatas, mas também levando em conta as recompensas futuras, de forma descontada ao longo do tempo, conforme a fórmula adaptada do trabalho de Monteiro e Ribeiro (2004), a seguir:

,

$$R(t) = \gamma^{N}\, r(t-N) \ldots + \gamma^{1}\, r(t-1) + \gamma^{0}\, r(t) + \gamma^{1}\, r(t+1) + \gamma^{N}\, r(t+N)\,, \qquad (3)$$

Onde:

$R(t)$ é o valor total descontado no tempo t,
$r(t+n)$ é a recompensa obtida n passos à frente no tempo,
γ é o fator de desconto $(0 \leq \gamma \leq 1)$.

Observa-se que a importância dos valores de recompensa diminui à medida que nos afastamos do tempo atual ($t = 0$). Isso ocorre devido à aplicação do fator de desconto temporal, que atribui menos peso às recompensas futuras/passadas em relação às recompensas imediatas. Consequentemente, as recompensas distantes no tempo têm menos influência nas decisões atuais, levando a um maior foco nas ações e resultados imediatos, mas ainda levando em consideração a influência das tomadas de decisões que afetam o futuro.

A função de recompensa $R(t)$ retorna um valor entre 0 e 1, representando o quão próximo o robô está do objetivo naquele instante de tempo. Ela é calculada como 1 menos a distância euclidiana normalizada em uma escala de 0 a 1.

A fórmula utilizada para calcular a recompensa é:

$$R(t) = 1 - \sqrt{\left(\frac{\left(\psi_t - \psi_{objetivo}\right)^2 + \left(Y_t - Y_{objetivo}\right)^2}{180^2}\right)}\ , \tag{4}$$

em que ψ_t e Y_t são os ângulos medidos/previstos pelo robô no instante de tempo t, $\psi_{objetivo}$ e $Y_{objetivo}$ são os ângulos de referência desejados para o robô, 180^2 é o valor máximo possível para o erro quadrático, considerando que a diferença máxima entre os ângulos é de $180°$.

2.3 Modelagem de Sistemas Dinâmicos e Controle Preditivo

O controle preditivo emprega técnicas para antecipar o comportamento de um sistema específico, utilizando um modelo matemático. Sua principal vantagem reside na habilidade de prever o erro, permitindo a seleção da ação de controle mais apropriada, fundamentada em critérios de otimização. No trabalho de Guimarães et al. (2019), é feito o controle de forma preditiva para minimizar o erro em Conversores seguindo o seguinte algoritmo, apresentado pela Figura 3:

Figura 5 – Controle Preditivo Baseado em Modelo

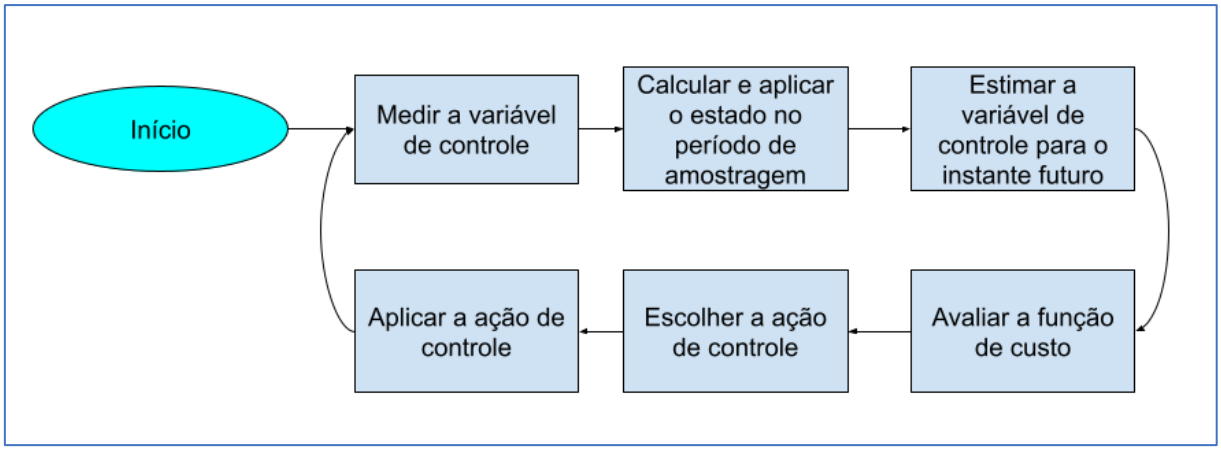
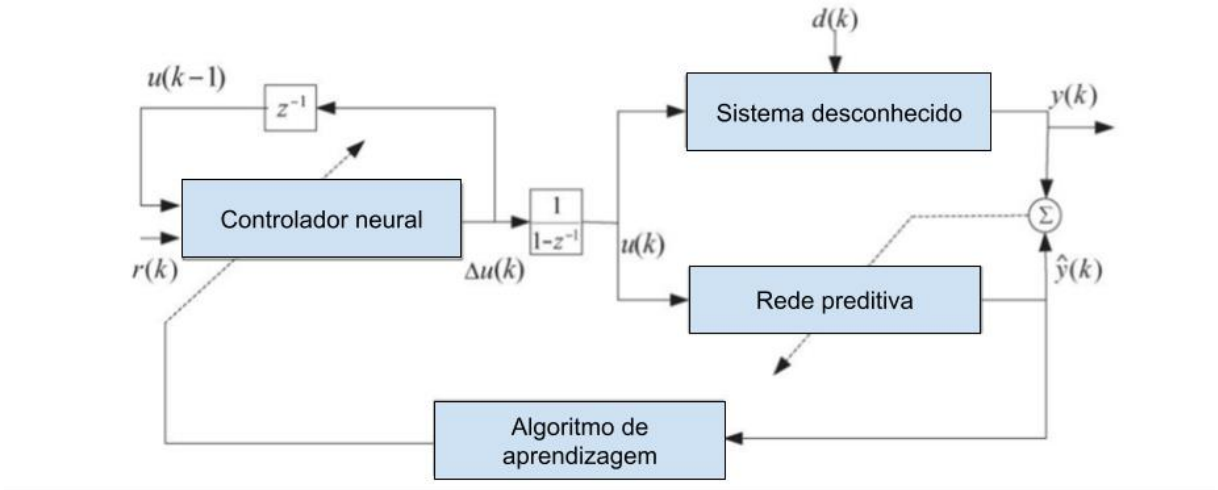


Figura 5 – Representação Gráfica do Método Utilizado no artigo “Controle Preditivo Baseado em Modelo para Conversores Formadores de Rede com Operação” (Guimarães et al., 2019).

O método ilustrado na Figura 5, combinado com elementos do modelo SDN (*Simplified Dual Network*), de Yunpeng (2012), serviu como ponto de partida para o desenvolvimento do Controlador Preditivo utilizado no SERM. Conforme ilustra a Figura 6, a SDN consiste em duas redes neurais: uma para identificação da dinâmica do sistema e outra para controle, gerando ações de controle ótimas para os atuadores. A entrada da rede de controle corresponde ao estado no instante $E = u(k-1)$, enquanto a saída representa a ação de controle que será aplicada ao sistema e, por sua vez, servirá como entrada para a rede neural preditiva. O erro, calculado como a diferença entre a saída atual $y(k)$ e a saída prevista $\hat{y}(k)$, é empregado para calibrar a rede neural preditiva. Subsequentemente, a rede neural preditiva é empregada em conjunto com um algoritmo de treinamento para gerar uma rede neural de controle. Esse processo é realizado por meio da simulação do ambiente, resultando em uma adaptação dinâmica da rede neural de controle.

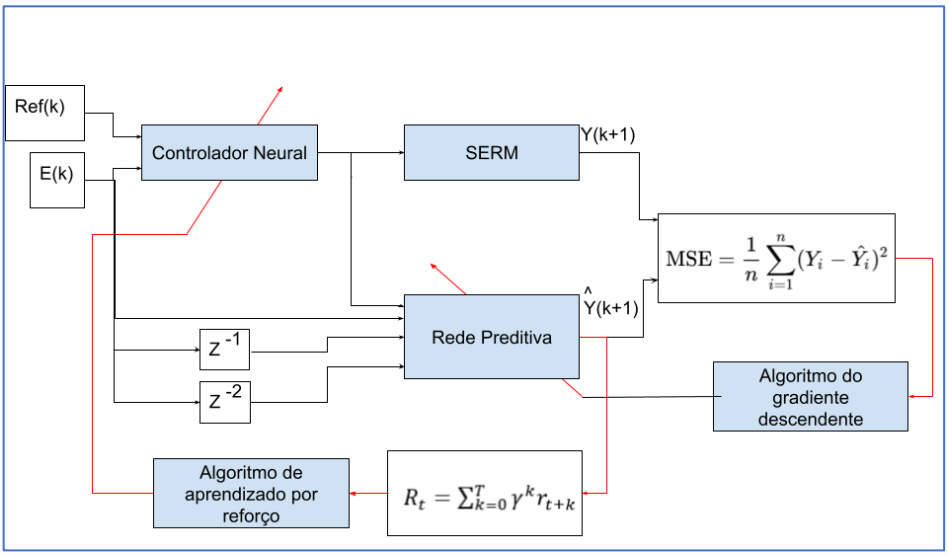
Figura 6 – Método do Modelo Interno



Fonte: Método do Modelo SDN de Yungpeng (2012)

Inicialmente, os dados provenientes do sinal aleatório, após atenderem à condição de excitação persistente, como detalhado na Seção posterior são empregados no algoritmo de treinamento da rede neural preditiva. O resultado disso é dois vetores, um que representa o comportamento do SERM $Y(k+1)$ e outro que representa o comportamento previsto pela rede preditiva. O erro entre o vetor previsto e o comportamento real do SERM é obtido através do erro médio quadrático e utilizado no algoritmo do gradiente descendente da Rede Preditiva para que esta possa aprender o comportamento do SERM. Após o treinamento, a rede neural preditiva é ajustada para refletir a relação matemática entre os dados de entrada e saída, similar à relação existente no modelo do robô físico. Em outras palavras, a rede neural é configurada simular o comportamento do SERM. Posteriormente, a rede neural preditiva resultante é utilizada como fonte de dados para a criação de um controlador neural adaptativo utilizando o algoritmo de aprendizado por reforço. A integração das duas redes, após o pré-treinamento, forma um modelo que realiza o controle do robô, conforme a Figura 7.

Figura 7 – Método do Modelo Interno Adaptado ao SERM



Fonte: Autor (2023)

2.3.1 Persistência de Excitação

Um aspecto crucial na modelagem de sistemas dinâmicos, incluindo robôs, é a condição de persistência de excitação. Esta condição garante que o sistema seja suficientemente excitado por diferentes entradas ao longo do tempo, possibilitando a identificação precisa dos parâmetros do modelo (Ioannou; Sun, 1996). A ausência de persistência de excitação pode comprometer a correta estimativa dos parâmetros do modelo, resultando em previsões imprecisas e ineficácia no controle do robô, conforme evidenciado por Torres (2021). Para atender a essa condição, o autor propôs a inclusão de um sinal senoidal de baixa amplitude e rica em frequência no sinal de controle. Essa abordagem visa assegurar que os dados utilizados no treinamento da rede neural representem de maneira precisa o comportamento do sistema, sendo essencial para o desenvolvimento deste trabalho.

3 METODOLOGIA

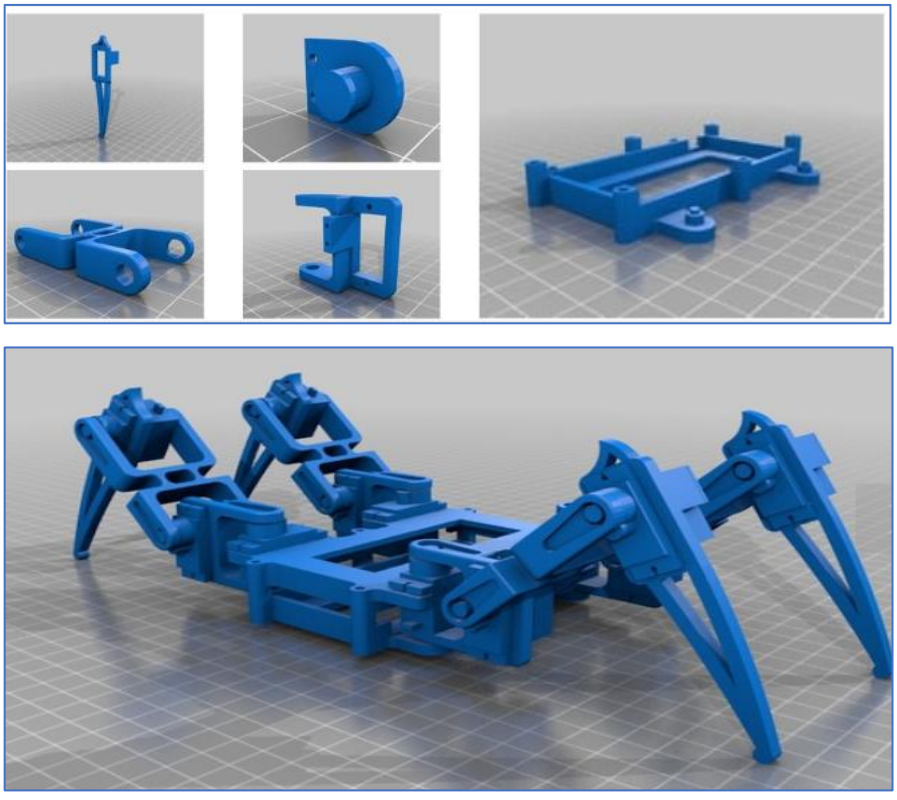
Este capítulo contempla detalhes do desenvolvimento do Controlador Preditivo para SERM, desde a montagem do sistema para coleta de dados até a implementação das redes neurais, a fim de analisar o desempenho das redes neurais em atingir os objetivos principais:

- **Seguir um sinal de referência:** O robô deve acompanhar uma referência específica para os ângulos ψ e θ, ajustando sua orientação conforme necessário para se alinhar com essa referência.
- **Prever as consequências de suas ações**: A rede neural deverá estimar a saída atual e os dados futuros, para possibilitar um controle preditivo.

3.1 Montagem do Sistema

O modelo em 3D foi originalmente criado e disponibilizado no Thingiverse (Santos, 2018). Foi utilizada uma impressora 3D para obter as peças do SERM, conforme pode ser visto na Figura 8.

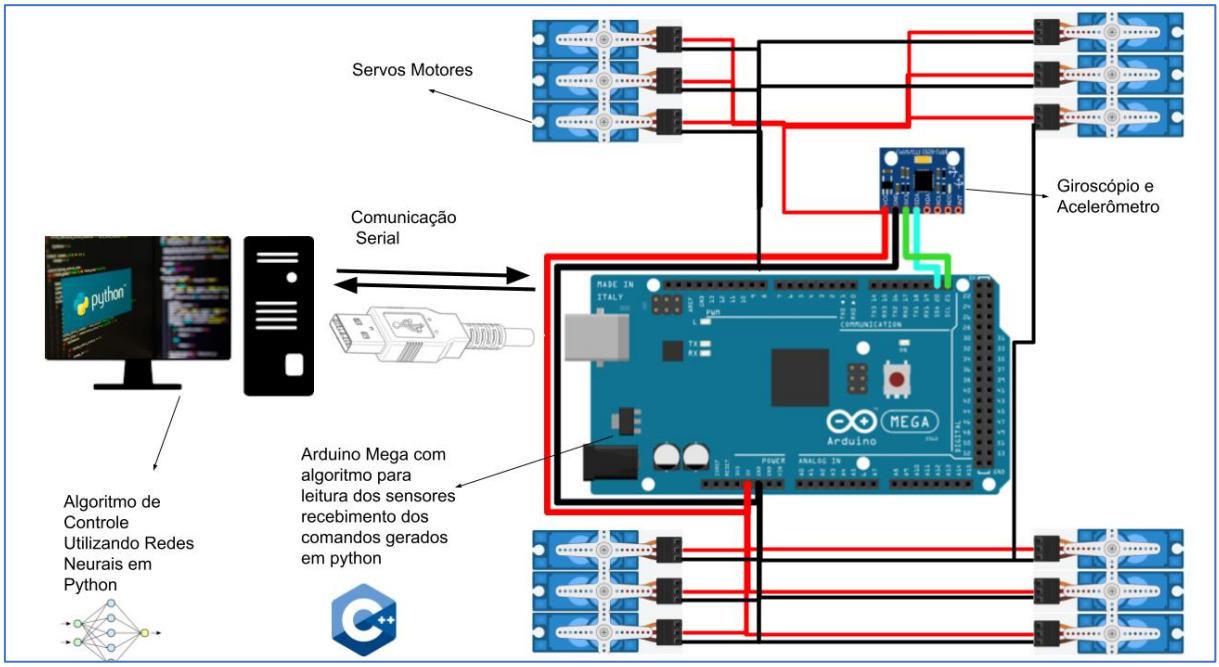
Figura 8 – Modelo de Chassi Utilizado No Desenvolvimento do SERM



Fonte: Santos. **Spider robot, quad robot, quadruped**.

A configuração do sistema é composta por um Arduino Mega, responsável por receber os comandos via comunicação serial e responder com os valores lidos pelos sensores. O Arduino é utilizado como uma ponte entre o controlador existente no computador e os atuadores/sensores do SERM.

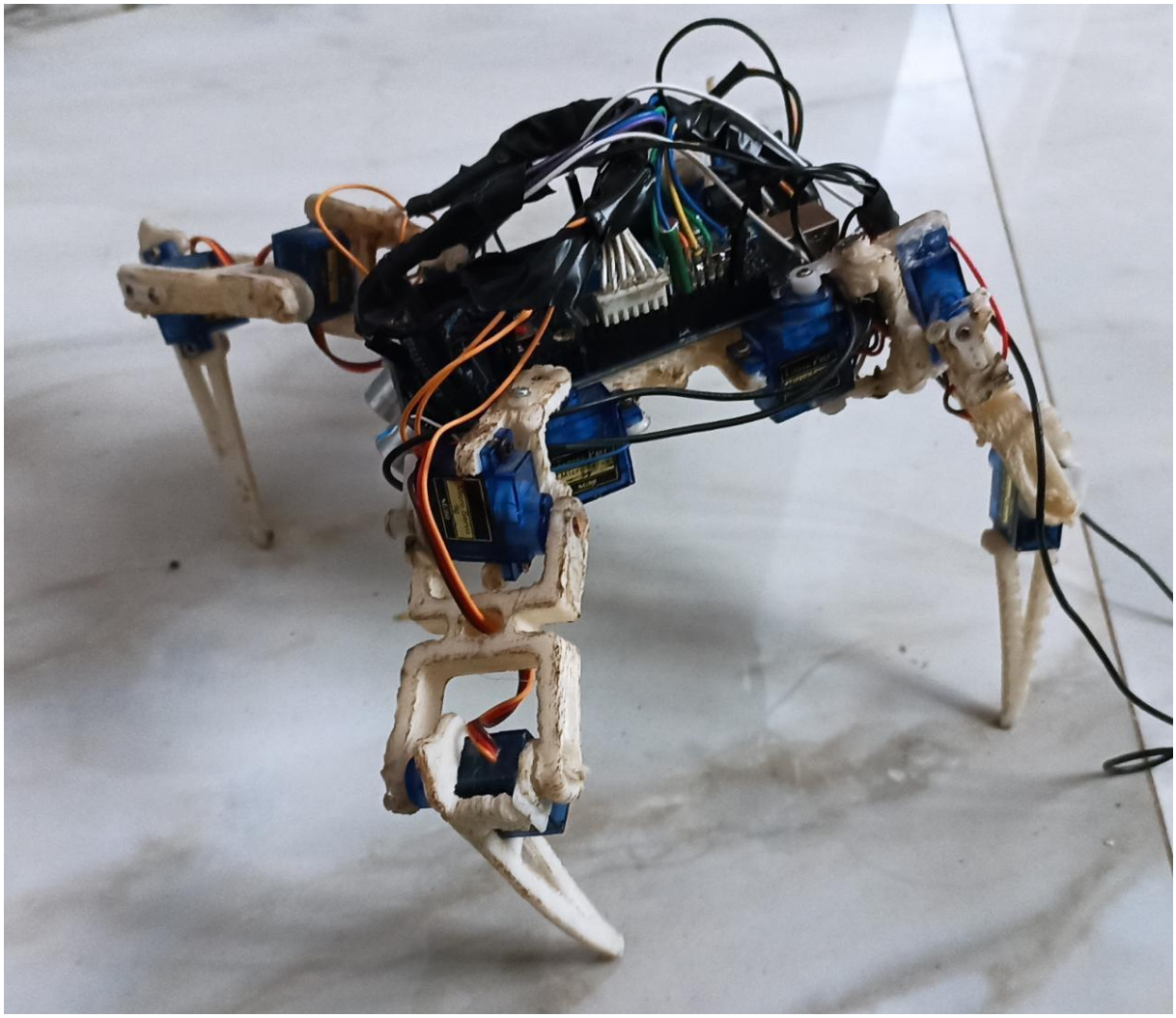
Figura 9 – Diagrama de Configuração do Sistema Embarcado em Robótica Móvel



Fonte: Autor

O esquemático apresentado na Figura 9 foi incorporado ao design do chassi representado na Figura 8. Nesse arranjo, cada grupo composto por 3 atuadores é responsável pelo movimento de uma perna do robô, totalizando assim 4 pernas e 12 atuadores no sistema. Adicionalmente, um módulo contendo acelerômetro e giroscópio foi montado na parte inferior do chassi para capturar dados sensoriais essenciais.

Figura 10 – Resultado da Montagem



Fonte: Autor.

Após a Montagem, foram feitos testes de continuidade, a alimentação do sistema, envio de comandos e a leitura dos sensores para testar os componentes e validar a montagem do sistema.

Para o desenvolvimento do código embarcado, foi utilizada a IDE do Arduino, uma plataforma simples e de fácil utilização que oferece um ambiente de programação intuitivo para o Arduino. Esta escolha foi feita devido à sua ampla aceitação na comunidade de desenvolvedores e à sua compatibilidade com a placa de controle do robô.

Para as redes neurais, optou-se por utilizar o Jupyter Notebook, uma aplicação web Open-source que permite a criação e compartilhamento de documentos que contêm código, equações, visualizações e texto narrativo. A flexibilidade do

Jupyter Notebook permitiu experimentar com diferentes arquiteturas de redes neurais, além de documentar e visualizar os resultados de forma clara e interativa.

3.2 Recursos Materiais e Ferramentas Computacionais

Os componentes utilizados para este projeto são divididos em atuadores, sensores e outros, como descritos na tabela a seguir:

Tabela 2 – Lista de Materiais

Lista de Materiais				
Atuadores				
Número	Componente	Modelo	Quantidade	Função
1	Servo Motor		12	Controle do ângulo das patas
Sensores				
Número	Componente	Modelo	Quantidade	Função
2	Módulo Sensor de Inclinação	MPU6050	1	Medir a Inclinação (ψ, θ e z do sistema
Outros				
Número	Componente	Modelo	Quantidade	Função
3	Arduino Mega	-	1	Utilizado para controle do Sistema
4	Jumpers	-	28	Utilizado para conexão dos materiais
5	Fonte 5v-3A	-	1	Alimentar o sistema
6	Filamento de impressão 3D	-	200g	Para impressão das peças
7	Espaguete termo retrátil	-	1M	Para isolar o circuito
8	Estanho para Solda	-	50g	Para soldar os componentes

Fonte: Autor.

Para fazer o controle, foi definida a utilização do Arduino Mega, pela praticidade, custo e quantidade de saídas digitais PWM A fonte a ser utilizada foi dimensionada pelo consumo de corrente do circuito. Utilizando uma fonte de bancada e um multímetro, foi aplicada uma tensão de 5V no circuito e os atuadores (motores) foram comandados para ângulos aleatórios a fim de definir qual o pico de consumo do sistema. Ao analisar o consumo do sistema, os atuadores/sensores foram submetidos a uma medição direta de corrente, revelando um valor aproximado de 2A.

Simultaneamente, o Arduino, alimentado por uma fonte independente, teve seu consumo avaliado, conforme indicado no datasheet, alcançando um máximo de 5W. A soma dos consumos individuais dos atuadores/sensores e do Arduino resultou em um consumo total do circuito de aproximadamente 3A.

Para auxiliar nos testes e modelagem, tanto do sistema a ser controlado quanto dos modelos de redes neurais, foram utilizadas as seguintes ferramentas:

- **Jupyter Notebook**: Utilizado para desenvolvimento das redes neurais a serem utilizadas no projeto;
- **IDE do Arduino**: Programação do Arduino e simulação das redes neurais;
- **Tinkercad**: Utilizado para testar o código a ser utilizado no Arduino Mega.

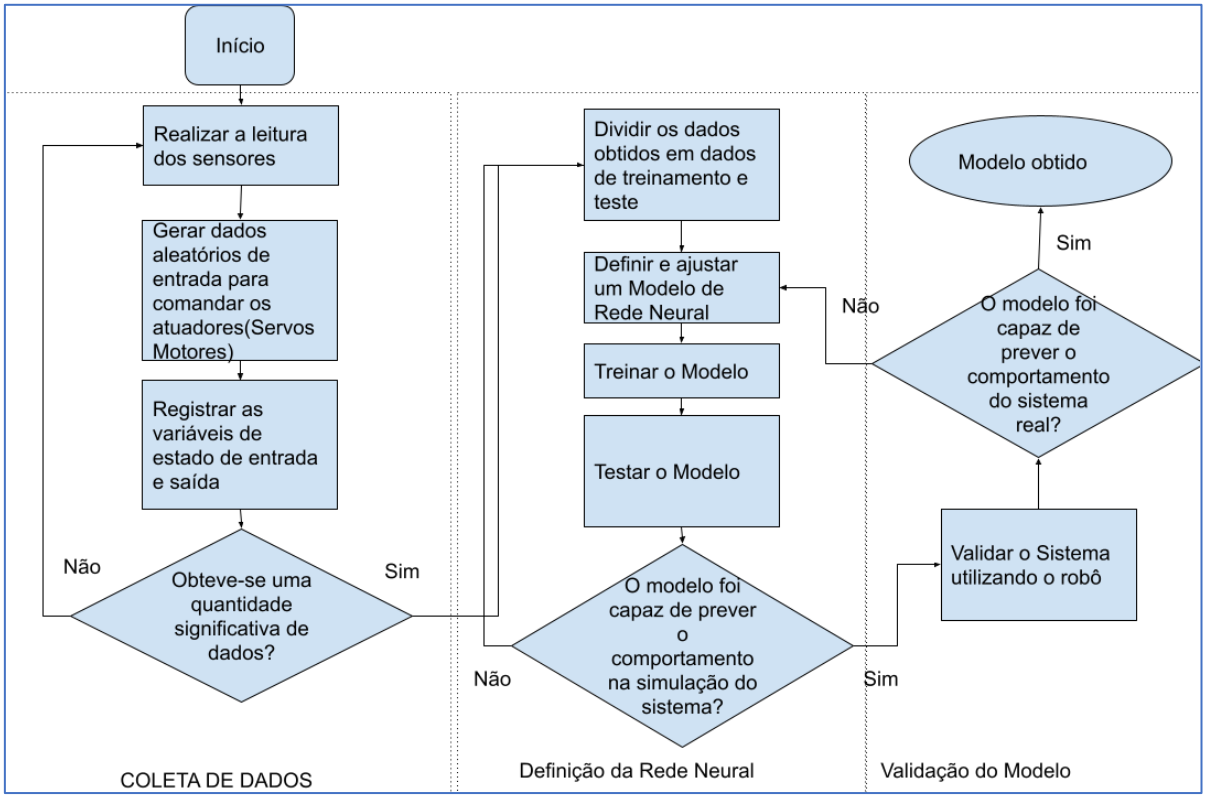
3.4 Estudo e Modelagem do Sistema Com Rede Neural Preditiva

Para modelar um sistema é preciso descrever o seu comportamento de forma a prever os estados de suas variáveis em função de perturbações e da própria dinâmica existente no sistema, isto é, cria-se uma função capaz de relacionar de forma aproximada uma entrada e uma saída, como exemplificado pelo trabalho de Castro (2012), no qual foi feita a modelagem matemática estudando a sequência de movimento dos membros de locomoção do robô, bem como as suas características geométricas, graus de liberdade, inércia, dentre outros parâmetros. O problema enfrentado por este tipo de modelagem é que “existem diversos tipos de robôs com diferentes finalidades, o que torna complicado definir um método de classificação que funcione em todos os casos” (Castro, 2012). Para enfrentar esse problema, no SERM, essa modelagem foi feita com o uso de uma rede neural, capaz de se adaptar a diferentes ambientes e robôs.

Para o robô utilizado, pode-se considerar como entrada, os estados dos sensores e comandos dados aos atuadores. Os estados são os ângulos atuais dos atuadores, a inclinação a aceleração angular e a aceleração linear nos três eixos. Os comandos emitidos correspondem aos valores angulares designados para a orientação dos servomotores.

A saída do modelo contempla os estados resultantes das variáveis de controle, especificamente a inclinação nos ângulos ψ e θ. A metodologia de modelagem adotada é delineada no fluxograma subsequente:

Figura 11 - Fluxograma do sistema projetado.



Fonte: Autor.

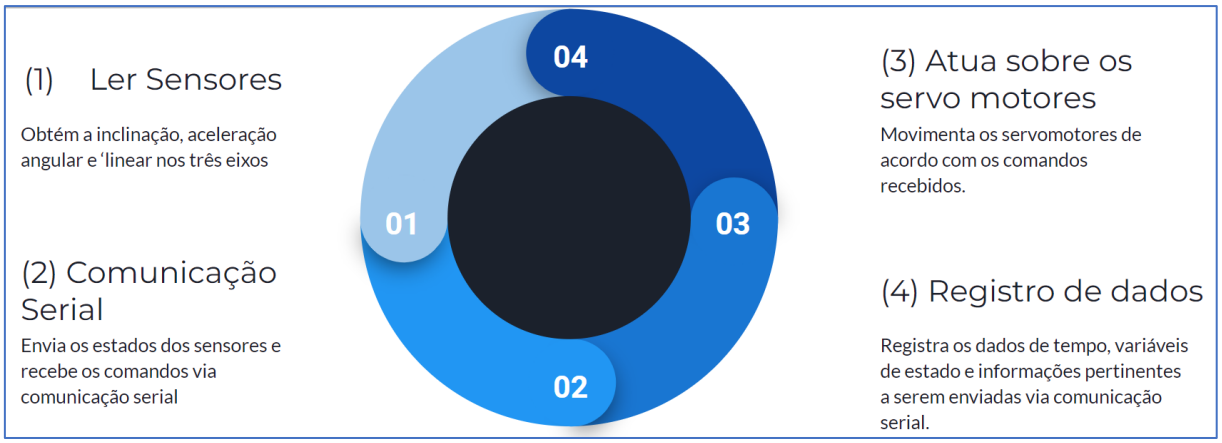
Através desta abordagem, foi possível obter um modelo que representa adequadamente o comportamento do sistema, antecipando com precisão os próximos estados com base nas variáveis de entrada, e isso foi alcançado sem a necessidade de cálculos excessivamente complexos. O modelo preditivo resultante serviu como base para a construção da rede neural responsável por controlar o sistema, utilizando o erro previsto por este modelo obtido de maneira eficaz. Conforme descrito nos itens subsequentes, partindo da coleta e processamento de dados até a obtenção da rede preditiva.

3.4.1 Coleta e Pré-processamento de Dados

A tomada de decisão é efetuada por meio das redes neurais. No entanto, para implementar essas decisões no controle do robô físico, foi empregado o Arduino Mega como uma interface facilitadora, servindo como uma ponte eficiente entre as instruções geradas pelas redes neurais e as ações executadas pelos componentes físicos do robô.

Para coletar informações do ambiente, o Arduino lê dados provenientes dos sensores, como o módulo acelerômetro e giroscópio localizados na parte inferior do chassi do robô. Esses dados são processados e, em seguida, enviados para um ambiente Python (Jupyter Notebook) por meio de comunicação serial, conforme descrito na Figura 12.

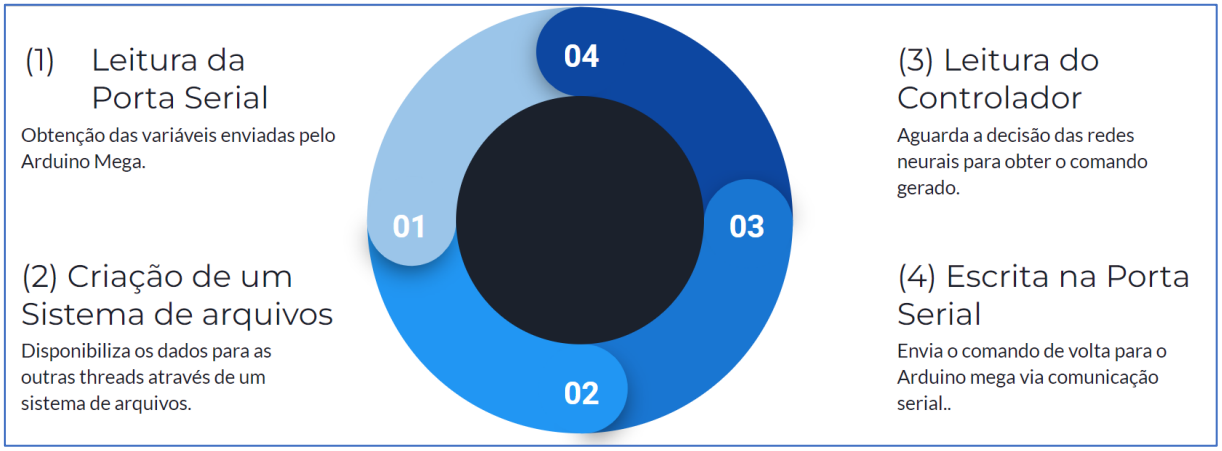
Figura 12 – Loop Arduino Mega



Fonte: Autor.

No Python, as redes neurais realizam o processamento necessário para tomar decisões com base nas informações sensoriais recebidas. Após a análise, os comandos resultantes são transmitidos de volta ao Arduino Mega por meio da comunicação serial, permitindo que o Arduino converta essas instruções em ações físicas executadas pelos atuadores do robô. Esse fluxo contínuo de dados e comandos entre o Arduino Mega e o ambiente Python trouxe a necessidade de criar uma thread dedicada à comunicação serial, com o objetivo de obter uma comunicação eficaz e em tempo real entre o sistema físico e as redes neurais, contribuindo para a operação dinâmica e adaptativa do robô quadrúpede. Conforme descrito pela Figura 13.

Figura 13 – Thread de comunicação serial em Python

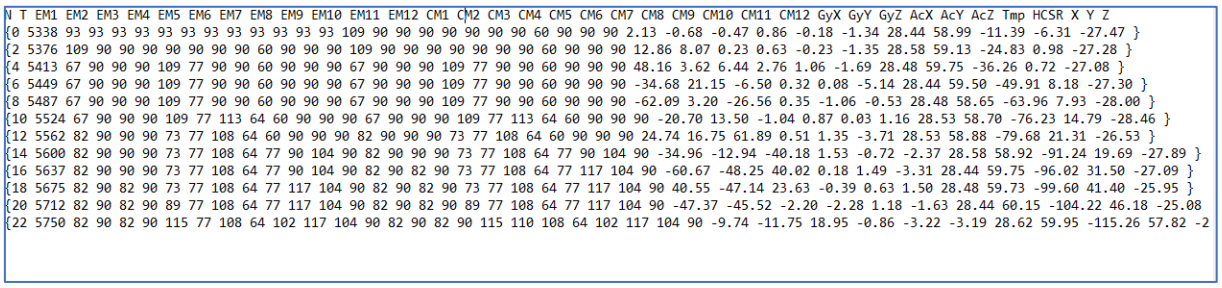


Fonte: Autor.

Estabeleceu-se uma interação entre o algoritmo implementado no Arduino Mega e o algoritmo em Python por meio de comunicação serial, viabilizando a coleta sistemática de dados. Para atender à condição de excitação persistente, foi concebido um algoritmo que gera comandos aleatórios caracterizados por baixa amplitude e espectro abrangente, seguindo uma forma de senoide ao entorno da posição na qual os atuadores aproximam da variável de controle do melhor ponto de estabilidade, ou seja, em que ψ e θ são o mais próximo de zero, em conformidade com a abordagem adotada por Torres (2021). Essa estratégia de excitação foi essencial para assegurar a persistência na variação dos estados do sistema, proporcionando uma base robusta para a aquisição de dados necessários ao desenvolvimento do modelo preditivo.

Os dados obtidos através da comunicação serial foram organizados em um arquivo de forma em que cada linha N contenha as variáveis que representam o estado em um determinado instante de tempo (t) e a linha sucessora $(N+1)$, contenha o estado lido no próximo instante de tempo $(t+1)$, conforme a Figura 11.

Figura 14 - Dados extraídos



```
N T EM1 EM2 EM3 EM4 EM5 EM6 EM7 EM8 EM9 EM10 EM11 EM12 CM1 CM2 CM3 CM4 CM5 CM6 CM7 CM8 CM9 CM10 CM11 CM12 GyX GyY GyZ AcX AcY AcZ Tmp HCSR X Y Z
{0 5338 93 93 93 93 93 93 93 93 93 93 93 93 109 90 90 90 90 90 90 60 90 90 90 2.13 -0.68 -0.47 0.86 -0.18 -1.34 28.44 58.99 -11.39 -6.31 -27.47 }
{2 5376 109 90 90 90 90 90 90 90 60 90 90 90 109 90 90 90 90 90 90 90 60 90 90 90 12.86 8.07 0.23 0.63 -0.23 -1.35 28.58 59.13 -24.83 0.98 -27.28 }
{4 5413 67 90 90 90 109 77 90 90 60 90 90 90 67 90 90 90 109 77 90 90 60 90 90 90 48.16 3.62 6.44 2.76 1.06 -1.69 28.48 59.75 -36.26 0.72 -27.08 }
{6 5449 67 90 90 90 109 77 90 90 60 90 90 90 67 90 90 90 109 77 90 90 60 90 90 90 -34.68 21.15 -6.50 0.32 0.08 -5.14 28.44 59.50 -49.91 8.18 -27.30 }
{8 5487 67 90 90 90 109 77 90 90 60 90 90 90 67 90 90 90 109 77 90 90 60 90 90 90 -62.09 3.20 -26.56 0.35 -1.06 -0.53 28.48 58.65 -63.96 7.93 -28.00 }
{10 5524 67 90 90 90 109 77 113 64 60 90 90 90 67 90 90 90 109 77 113 64 60 90 90 90 -20.70 13.50 -1.04 0.87 0.03 1.16 28.53 58.70 -76.23 14.79 -28.46 }
{12 5562 82 90 90 90 73 77 108 64 60 90 90 90 82 90 90 90 73 77 108 64 60 90 90 90 24.74 16.75 61.89 0.51 1.35 -3.71 28.53 58.88 -79.68 21.31 -26.53 }
{14 5600 82 90 90 90 73 77 108 64 77 90 104 90 82 90 90 90 73 77 108 64 77 90 104 90 -34.96 -12.94 -40.18 1.53 -0.72 -2.37 28.58 58.92 -91.24 19.69 -27.89 }
{16 5637 82 90 90 90 73 77 108 64 77 90 104 90 82 90 82 90 73 77 108 64 77 117 104 90 -60.67 -48.25 40.02 0.18 1.49 -3.31 28.44 59.75 -96.02 31.50 -27.09 }
{18 5675 82 90 82 90 73 77 108 64 77 117 104 90 82 90 82 90 73 77 108 64 77 117 104 90 40.55 -47.14 23.63 -0.39 0.63 1.50 28.48 59.73 -99.60 41.40 -25.95 }
{20 5712 82 90 82 90 89 77 108 64 77 117 104 90 82 90 82 90 89 77 108 64 77 117 104 90 -47.37 -45.52 -2.20 -2.28 1.18 -1.63 28.44 60.15 -104.22 46.18 -25.08
{22 5750 82 90 82 90 115 77 108 64 102 117 104 90 82 90 82 90 115 110 108 64 102 117 104 90 -9.74 -11.75 18.95 -0.86 -3.22 -3.19 28.62 59.95 -115.26 57.82 -2
```

Fonte: Autor.

Em seguida os dados são organizados de forma automática em um conjunto de treinamento para uma rede neural preditiva. Cada amostra do conjunto de treinamento é composta por uma sequência de valores gerados pelos atuadores em um determinado intervalo de tempo, juntamente com o valor correspondente da saída desejada (inclinação). Esses dados são usados para treinar a rede neural a fim de prever a saída desejada com base nos valores dos atuadores.

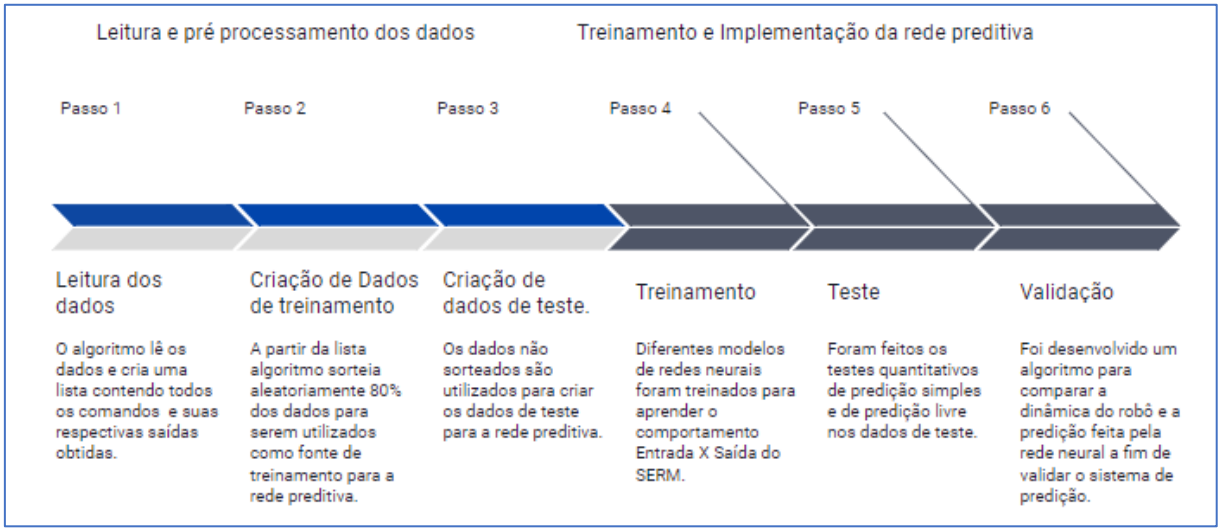
Nesse caso, os dados coletados pelo algoritmo no robô quadrúpede são utilizados para treinar uma rede neural que servirá como modelo do robô. A rede neural é capaz de simular o comportamento do robô em resposta a diferentes entradas dos atuadores.

Para isso, os dados coletados são pré-processados e organizados em um formato adequado para treinamento da rede neural. Foram utilizadas técnicas como normalização e padronização dos dados para melhorar o desempenho do modelo. Com os dados organizados, a rede neural faz uma busca no sistema de arquivos e utiliza-os para aprender o comportamento do robô no ambiente atual em resposta aos comandos aleatórios utilizados.

Após o treinamento, a rede neural pode ser utilizada para fazer predições do comportamento do robô com base em diferentes entradas dos atuadores. Essas predições foram comparadas com os dados reais do robô para avaliar a qualidade do modelo. O modelo pôde então ser refinado e aprimorado por meio de ajustes na arquitetura da rede neural e na seleção de hiper parâmetros até que seja capaz de simular com precisão o comportamento do robô quadrúpede.

O diagrama exibido na Figura 15 ilustra o processo de obtenção dos dados de treinamento e teste para utilização no treinamento.

Figura 15 – Criação dos dados de Treinamento e Teste da Rede Preditiva



Fonte: Autor.

3.4.2 Escolha da Arquitetura da Rede Preditiva

A escolha da arquitetura da rede neural preditiva envolveu a consideração cuidadosa de diversos fatores, incluindo a complexidade do sistema de controle do SERM, a necessidade de previsões precisas, e a capacidade de generalização para diferentes condições de operação. Além disso, foram analisadas as características do conjunto de dados disponível e as restrições computacionais impostas pelo ambiente embarcado.

Inspirado em trabalhos relevantes, como o de Yunpeng (2012), que abordou o controle preditivo em sistemas desconhecidos usando redes neurais, e levando em conta as particularidades do robô quadrúpede, optou-se por uma arquitetura que incorporasse elementos de redes neurais recorrentes (ESN - Echo State Network) para identificação do sistema e uma abordagem baseada em redes duplas simplificadas (SDN) para otimização.

O processo de escolha considerou a capacidade dessas arquiteturas em lidar com a complexidade dinâmica do sistema, a eficiência computacional para implementação em um ambiente embarcado, e a capacidade de adaptação a diferentes condições de operação.

A rede preditiva foi implementada mediante a criação de uma classe que herda as características da classe “module” da biblioteca torch em Python. As camadas predominantes na rede neural foram as camadas densas (fully connected), utilizando principalmente ativações ReLU (Rectified Linear Unit), amplamente adotadas em redes neurais profundas.

A função ReLU estabelece que a saída da camada é o máximo entre zero e a soma ponderada das entradas, introduzindo não-linearidade e facilitando a aprendizagem de relações complexas entre os dados. Além da ReLU, foram incorporadas camadas com ativações RELU6 e sigmoide. A função RELU6 limita o valor máximo de saída em 6, prevenindo problemas de explosão do gradiente, o que é comum em redes com muitas camadas conforme pode ser visto no trabalho de Contrani (2012). A função sigmoide, adequada para valores normalizados entre 0 e 1, foi utilizada na camada de saída da rede neural.

Tabela 3 – Funções de Ativação da Rede Preditiva

Camada	Função de Ativação
Entrada	ReLU
Camadas Internas	ReLU6
Saída	Sigmoide

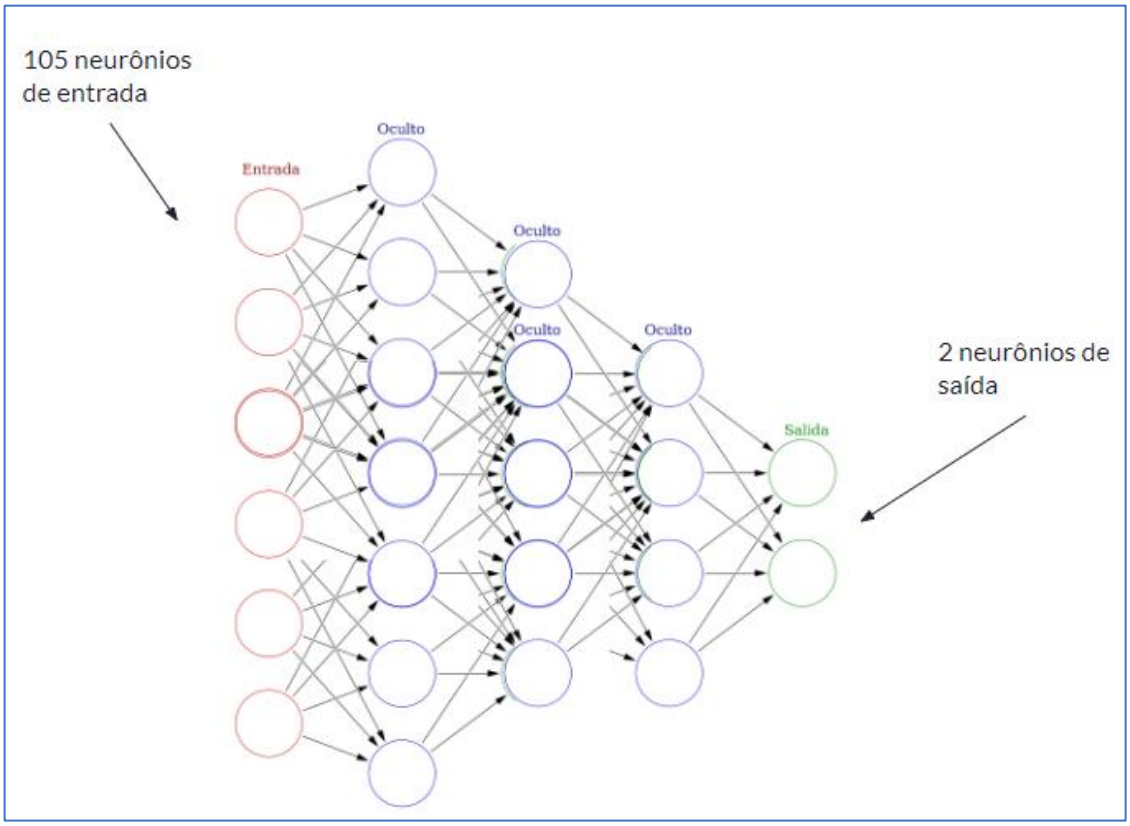
Fonte: Autor

.

Diversas configurações de redes neurais foram submetidas a testes e treinamento, conforme delineado no fluxograma apresentado na Figura 6 e no diagrama representado na Figura 10. Os resultados foram documentados e comparados com o objetivo de identificar a arquitetura mais eficaz. A arquitetura final foi selecionada após uma fase de experimentação e ajustes, chegando à uma rede profunda de largura variável que inicia com camadas compostas por 105 neurônios (correspondentes ao tamanho dos dados de entrada) e diminui gradualmente para 2 neurônios (representando o tamanho Cdos dados de saída). Nas camadas mais amplas, foi aplicada a técnica de *Dropout, comumente utilizada na comunidade acadêminca, como no trabalho de Costa (2022)*, na qual durante o treinamento, unidades (neurônios) são aleatoriamente ignoradas, ou "descartadas", com uma probabilidade determinada. Essa prática visa evitar que as unidades da rede se

tornem excessivamente dependentes de outras unidades específicas, promovendo a robustez do modelo e aprimorando sua capacidade de generalização para dados não observados.

Figura 16 – Representação da Rede Preditiva



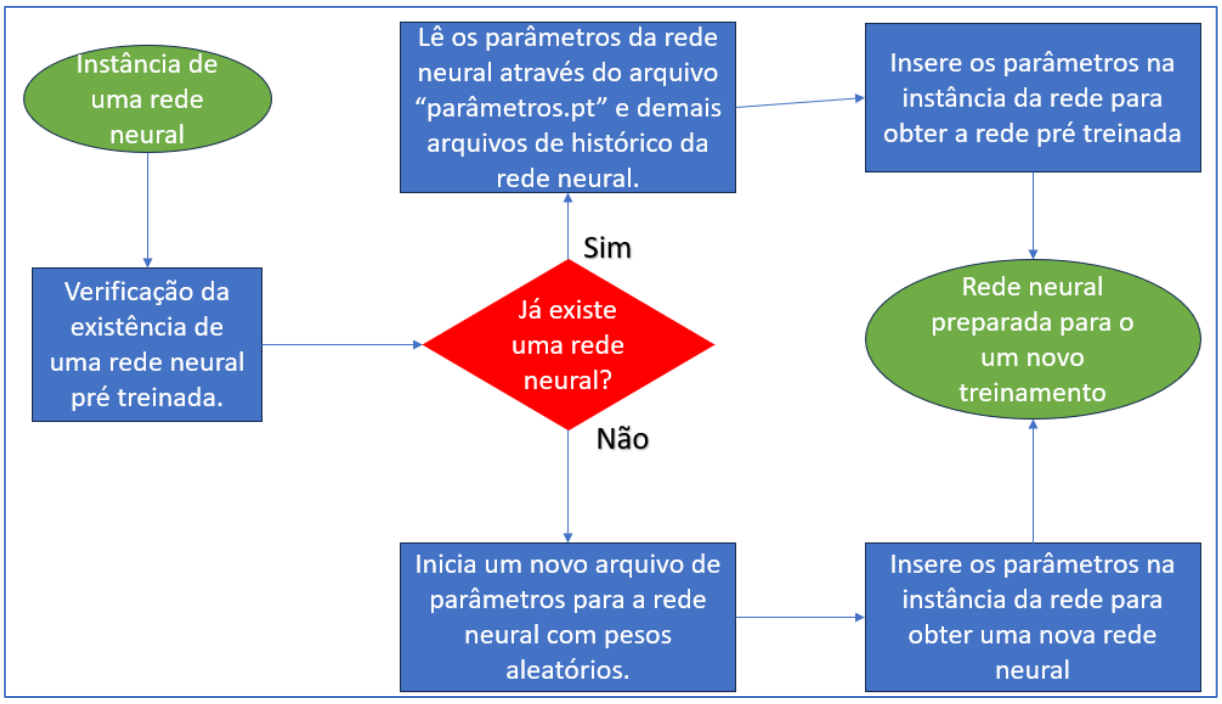
Fonte: Autor

3.4.3 Criação de Instância da Rede Preditiva

Para iniciar a Rede Neural Preditiva, uma instância da classe é criada, representando a rede neural desenvolvida para prever o comportamento do SERM.

Essa instância (objeto) funciona como o modelo da rede neural, desempenhando papéis cruciais na tomada de decisões, na previsão de estados futuros com base nos dados fornecidos e no treinamento do controlador. A etapa de criação da instância é descrita no diagrama da Figura 17.

Figura 17 – Representação da Rede Preditiva



Fonte: Autor.

Antes de iniciar o treinamento, há uma tentativa de carregar os parâmetros da rede neural do arquivo "parametrosRedePreditiva.pt", que armazena os pesos e configurações da rede. O modelo é então configurado no modo de avaliação. No modo de avaliação, a rede neural se abstém de treinamento e é exclusivamente utilizada para realizar previsões.

Posteriormente, o código extrai o melhor erro previamente salvo do arquivo "melhorErroPreditiva.txt" e o associa ao modelo. Este erro constitui uma métrica essencial para avaliação do desempenho do modelo. Caso a tentativa de carregar os parâmetros falhe, indicando a não existência de uma rede treinada, durante a primeira execução, um novo modelo é gerado.

Os parâmetros da rede neural são persistidos no arquivo "parametrosRedePreditiva.pt". Adicionalmente, é criado um arquivo

"melhorErroPreditiva.txt" com um valor inicial de erro estabelecido em um valor 1000, definido arbitrariamente. Este procedimento é crucial para a inicialização e configuração eficaz do modelo de rede neural preditiva no contexto do controle do SERM.

3.4.4 Algoritmo de treinamento da Rede Neural Preditiva

Assim como no trabalho de Araújo (2012), foi utilizado o algoritmo do gradiente descendente, utilizando como função de perda, o MSE (Mean Squared Error), que calcula a média do quadrado da diferença entre as previsões da rede neural e os dados reais do robô quadrúpede. A função MSE é comumente usada em problemas de regressão, nos quais o objetivo é prever um valor numérico contínuo, como a posição como por exemplo no trabalho de Oliveira et al. (2022), no qual utiliza-se redes neurais para a previsão de corrente elétrica em um sistema fotovoltaico.

O otimizador utilizado durante o treinamento foi o do gradiente estocástico (SGD) que adapta as taxas de aprendizagem dos pesos de cada camada da rede neural de acordo com as estatísticas de primeira e segunda ordem das derivadas parciais da função de perda. Ele atualiza os pesos da rede em cada iteração do treinamento, com base no gradiente da função de perda em relação aos pesos e em uma taxa de aprendizado fixa.

Para iniciar o treinamento, são fornecidos os seguintes parâmetros ao algoritmo:

- ***“nEpocas”:*** Recebe o número de épocas, ou seja, a quantidade de vezes em que o algoritmo deve iterarar todo o conjunto de dados de treinamento antes de salvar as conquistas, ou seja, atualizar o arquivo de parâmetros;
- ***“modelo”:*** Recebe uma instância de rede neural;
- ***“Aprendizado”:*** Recebe um valor de taxa de Aprendizado a ser utilizada no otimizador;
- ***“Arquivo”:*** Recebe o arquivo contendo os dados de treinamento;
- ***“arquivoParametros”:*** Recebe o arquivo onde deverão ser salvos os parâmetros da rede neural;
- ***“arquivoSaida1”:*** Recebe o arquivo onde deverão ser salvos os valores de erro durante o treinamento;
- ***“arquivoSaida2”:*** Recebe o arquivo onde deverão ser salvos os valores de acurácia durante o treinamento.

O algoritmo de treinamento da rede neural preditiva pode ser compreendido através do seguinte fluxograma:

Figura 18 – Representação da Rede Preditiva

Fonte: Autor.

A princípio, utiliza-se uma taxa de aprendizado alta para estimular a ampla exploração do espaço de parâmetros da rede neural, evitando que a rede neural fique presa em mínimos locais não otimizados durante o treinamento e permitindo uma busca mais abrangente no espaço de soluções. Em seguida, a rede neural realiza a previsão dos estados, procedendo ao cálculo dos gradientes da função de perda para ajustar os pesos na direção oposta. Por fim, compara-se o erro atual com o erro anterior, avaliando assim a evolução da capacidade de previsão sobre os dados de treinamento. Quando a melhora no desempenho é insignificante em relação à taxa de aprendizado atual, o treinamento é interrompido para realizar ajustes neste hiperparâmetro e permitir que a rede neural retome o aprendizado de forma mais eficiente. Esse processo de ajuste da taxa de aprendizado é repetido em um loop de forma automatizada até que ocorra a convergência do erro médio quadrático ou a rede neural não seja mais capaz de aprender.

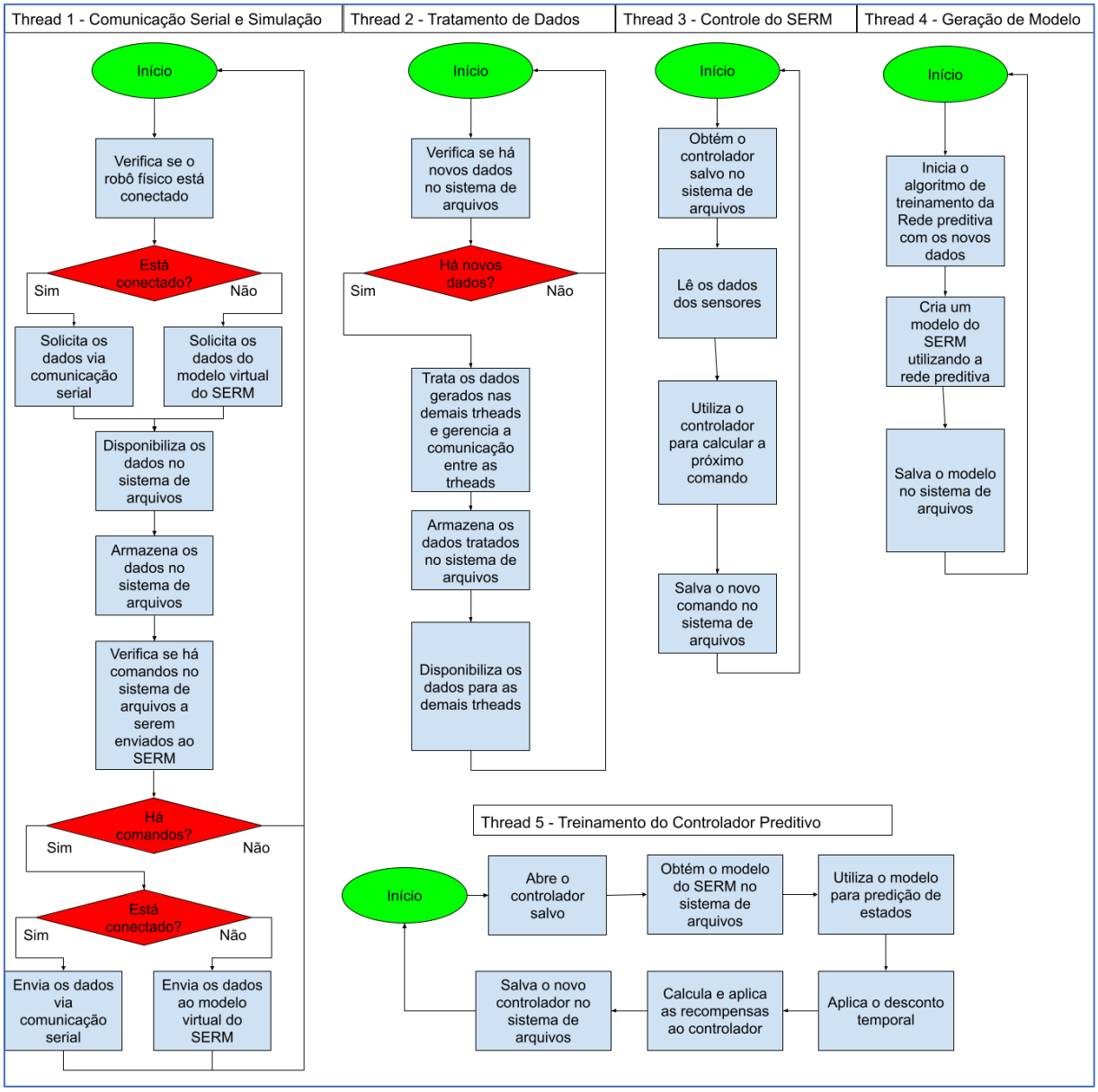
3.5 Gerenciamento de tarefas multitrheads

O sistema de controle do robô quadrúpede é caracterizado pela gestão de tarefas simultâneas, possibilitada pelo uso de threads, as quais constituem unidades independentes de execução que operam paralelamente, permitindo a realização simultânea de diversas operações essenciais ao controle do SERM. A necessidade do uso de threads se dá principalmente pela necessidade de manter a comunicação de forma contínua, mesmo durante a execução de tarefas concorrentes.

No contexto específico do SERM, a implementação de threads é essencial para coordenar atividades como a comunicação serial entre o controlador e o SERM, o tratamento de dados, gerenciamento do fluxo de informações entre as trheads, os algoritmos de treinamento online/offline, gerenciamento dos arquivos, dentre outras funções. Essa abordagem multithread oferece vantagens significativas, como a otimização do uso de recursos computacionais e a melhoria do desempenho geral do sistema.

Ao contrário da abordagem sequencial, onde uma tarefa aguarda a conclusão da outra, as threads permitem que operações ocorram de forma paralela, resultando em processos mais ágeis e responsivos. A distribuição inteligente das *thread's* permite o aproveitamento maior dos recursos disponíveis. Enquanto uma *thread* lida com operações intensivas, como a utilização das redes neurais para processar as informações e tomar decisões, outras podem continuar respondendo a eventos, como a comunicação serial com o SERM, proporcionando uma experiência mais fluída, ou seja, sem travamentos na comunicação. Além disso há a modularização de tarefas complexas, ou seja, como cada thread é responsável por uma parte independente da tarefa global, o desenvolvimento e manutenção do código, tornando-o mais estruturado e compreensível. O princípio de funcionamento das *thread's*, bem como as suas respectivas tarefas, pode ser observado no fluxograma da Figura 19.

Figura 19 – Gerenciamento de Tarefas Multitrheads



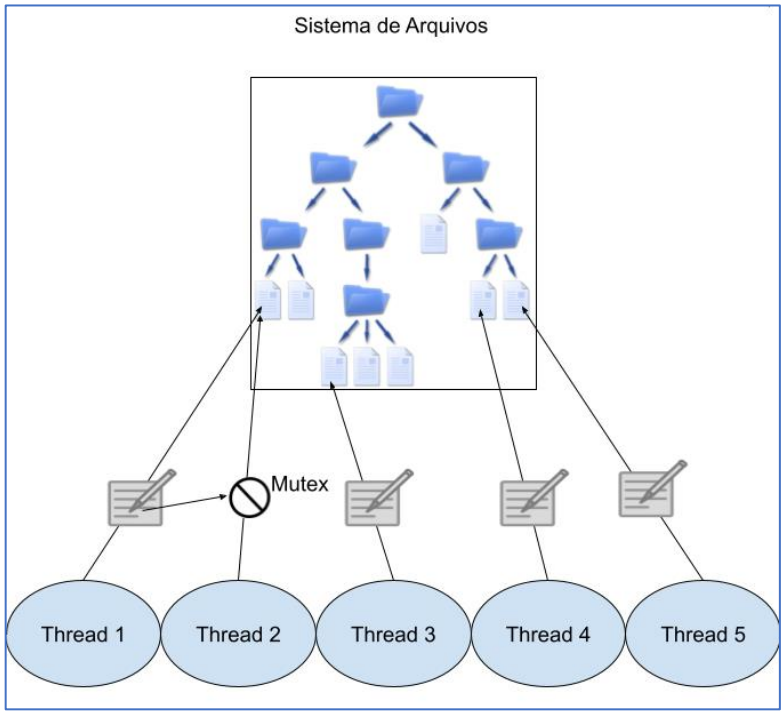
Fonte: Autor.

A utilização de threads para acessar simultaneamente o mesmo arquivo e variáveis requer a implementação de mecanismos de sincronização, a fim de evitar conflitos e inconsistências nos dados. Nesse contexto, a sincronização torna-se essencial não apenas para operações internas de threads, mas também para facilitar a comunicação entre processos. Fazer com que um processo se comunique com outro, por exemplo, envolve a utilização de mecanismos externos. Conforme observado por Strickland (2018), uma abordagem comum é gravar em um arquivo e garantir que o segundo processo leia esse arquivo, desde que haja algum tipo de

bloqueio para evitar leituras e gravações simultâneas. Assim, a aplicação consistente de mecanismos de sincronização não só mantém a integridade dos dados em operações de threads, mas também se estende de maneira fluida para a intercomunicação entre processos.

Para isso, foi utilizado o método de *mutexes* (*mutual exclusions)* ou travas, que são dispositivos de controle de acesso que garantem que apenas uma thread por vez possa modificar determinadas partes do código. Ao acessar um arquivo compartilhado, uma thread deve adquirir um mutex específico associado a esse arquivo antes de realizar operações de leitura ou escrita. Isso impede que outras threads acessem o mesmo arquivo simultaneamente, evitando potenciais condições de corrida. Para variáveis compartilhadas, um mutex também é utilizado para garantir que apenas uma thread por vez possa modificar ou acessar essas variáveis. Quando uma thread pretende modificar uma variável compartilhada, ela adquire o mutex associado a essa variável. Outras threads que tentam acessar ou modificar a mesma variável precisam esperar até que o mutex seja liberado. Esse processo está representado pela Figura 20.

Figura 20 – Representação da Rede Preditiva



Fonte: Autor.

Dessa forma, a implementação cuidadosa do uso de mutexes em torno de operações críticas (como leitura ou escrita de arquivos, modificação de variáveis compartilhadas) assegura a consistência e a integridade dos dados no ambiente multithreaded, evitando conflitos e erros.

3.5 Rede Neural de Controle para Robô Quadrúpede

A combinação de uma rede neural preditiva e uma rede neural controladora, baseada no método de Yunpeng (2012), permite a criação de um sistema inteligente capaz de tomar decisões eficientes e adaptativas em tempo real, a fim de evitar potenciais problemas e otimizar o desempenho do robô. Nesta seção, é apresentada a rede neural de controle implementada, destacando sua arquitetura, treinamento e integração com a rede neural preditiva para a obtenção de resultados promissores.

3.5.2 Princípio do algoritmo de controle preditivo e adaptativo

O sistema de controle do SERM é sustentado por uma rede neural, a qual é recompensada ou penalizada conforme comportamento da variável de controle em relação ao sinal de referência estabelecido. O algoritmo de treinamento utiliza o modelo, construído pela rede neural preditiva, para antecipar o estado futuro do SERM através de informações históricas e atuais para penalizar ou recompensar o controlador através do desconto temporal. A rede neural de controle, integrada a esse sistema, processa dados sensoriais e gera ações de controle com o objetivo de otimizar o desempenho do SERM, não apenas com base no estado presente, mas também nos próximos estados previstos. Desta forma, a rede neural de controle desempenha um papel crucial na tomada de decisões proativas, antecipando-se para gerar ações corretivas que visam melhorar a performance e evitar potenciais problemas no SERM.

No processo de controle, o controlador recebe dados da comunicação serial e emprega a rede neural para gerar um novo comando. Este comando é posteriormente armazenado em um arquivo, o qual será lido pela *thread* encarregada da comunicação serial com o SERM. Na ausência de conexão direta com o robô, a thread de comunicação encaminha os dados para o modelo virtual do SERM, a fim de simular o comportamento do sistema antes de submeter o robô físico aos testes.

A função responsável pelo controle do robô quadrúpede realiza uma série de operações essenciais para o correto funcionamento do sistema, com foco na geração de comandos para orientar o robô real. Inicialmente, a função inicia carregando o modelo de controle, denominado controlador, a partir do arquivo "parametrosControlador.pt". Este modelo é previamente treinado para gerar comandos eficazes que permitam o controle do robô em tempo real.

Em seguida, são definidos os valores de referência (*setpoint*), baseados em funções específicas, como, por exemplo, uma senoide. Estes valores representam os pontos-alvo para o robô no espaço bidimensional. A leitura dos valores atuais de inclinação do robô (ψ e θ) ocorre a partir do arquivo "inclinacao.txt", e esses valores, juntamente com os valores de referência, são registrados em arquivos separados.

A função realiza o cálculo do erro em ψ e θ, que representa a diferença entre os valores de referência e os valores atuais. Esses erros são processados para evitar problemas, visto que o sistema possui valores limitados ou cíclicos, ou seja, o valor de erro ultrapassa 180 graus e retorna ao limite inferior, criando um ciclo contínuo.

Para avaliar o desempenho do controlador, utiliza-se a métrica de distância euclidiana, que quantifica a diferença entre os valores de referência e os valores reais em ψ e θ. Essa métrica é aplicada a cada estado, abrangendo o passado, presente e futuro, e incorpora o desconto temporal. Com base nessa distância euclidiana ponderada temporalmente, o controlador é recompensado ou penalizado. Se a distância euclidiana for menor que um limiar aceitável, o controlador recebe uma recompensa positiva. Em contrapartida, se a distância euclidiana ultrapassar esse limiar, o controlador é penalizado com uma pontuação negativa. As recompensas e penalidades são então registradas no controlador, ajustando seu comportamento ao longo do tempo ao reforçar os pesos dos neurônios bem-sucedidos e atenuar os demais. Essa abordagem visa otimizar o desempenho do controlador considerando a evolução temporal dos estados.

A função também registra as ações do controlador, representadas pelo comando gerado juntamente com as coordenadas atuais e θ. O comando gerado é convertido para uma string e registrado no sistema de arquivos para ser lido e executado pelo SERM.

Além disso, ocorre a atualização e ajuste contínuo do controlador. Se o número de ações ultrapassa 1000 e a pontuação supera um valor mínimo predefinido,

o controlador colhe recompensas. Caso contrário, aplica-se penalidades para promover a aprendizagem adaptativa.

Para fins de depuração, a função gera um registro detalhado com informações sobre os valores atuais, de referência, erros, ações, pontuações e outras métricas relevantes. A exibição dessas informações no console é opcional. Essa função desempenha um papel crucial na otimização do desempenho do sistema de controle do robô quadrúpede.

3.5.3 Estrutura da Rede Neural do Controlador

Para a estrutura do controlador, foi optado por utilizar a estrutura parecida com o da rede preditiva, porém com a introdução das funções e atributos necessários para utilização no algoritmo de controle, como a política de ação, os valores de setpoint, dentre outros. Isso é possível, baseado nos seguintes motivos:

- Tipos de dados parecidos: Ambas as tarefas, previsão e controle, lidam com conjuntos de dados similares, especialmente no contexto das redes neurais. As características dos dados de entrada, como sensores e variáveis ambientais, são essencialmente as mesmas para ambas as operações. Manter a estrutura consistente permitiu uma manipulação desses dados, aproveitando as nuances específicas de cada conjunto, mas dentro de uma estrutura já adaptada às suas peculiaridades.
- Tipos de operações: Embora a previsão e o controle envolvam operações aparentemente opostas, prever futuros estados versus tomar ações com base em estados atuais, a rede neural é intrinsecamente capaz de aprender padrões complexos em qualquer direção. Ao inverter a lógica da operação, transformando a saída da rede de previsão na entrada para o controlador, podemos explorar a capacidade da rede de compreender relações causais entre dados e ações, independentemente da direção da operação.
- Facilidade na Implementação: Ter uma estrutura unificada simplificou a implementação prática do sistema. A mesma rede poderia ser treinada sequencialmente para ambas as tarefas, simplificando o fluxo de trabalho e reduzindo a complexidade do código.

3.5.5 Criação do modelo virtual do SERM

A classe “RoboVirtual()”, utilizada para criação de um modelo que representa o comportamento do SERM, é uma parte fundamental deste projeto. Ela desempenha um papel essencial na interação entre as duas redes neurais e no

treinamento do controlador do robô. Abaixo, detalhamos os principais aspectos dessa classe:

A classe realiza a inicialização, configurando diversas variáveis que representam os estados, comandos e sensores do robô virtual. Esses parâmetros incluem, entre outros, os ângulos dos servos das pernas, leituras de sensores inerciais (giroscópio e acelerômetro), distância de um sensor ultrassônico e coordenadas espaciais do robô.

Dentro dessa estrutura, encontra-se uma instância do modelo de rede neural preditiva, carregada a partir do sistema de arquivos, que contém os pesos e configurações do modelo previamente treinado.

O processo de comando e simulação é desencadeado pelo método "comandar". Esse método recebe um vetor de comandos como entrada, os quais são utilizados para atualizar os ângulos dos servos das pernas do robô virtual. Além disso, a classe simula leituras aleatórias dos sensores inerciais (giroscópio e acelerômetro).

No estágio subsequente, o método "simular" assume a responsabilidade de simular o comportamento do robô virtual com base nos comandos fornecidos. Com isso, o modelo de rede neural preditiva é empregado para prever o próximo estado do robô, levando em consideração os estados anteriores e os comandos atuais. Essa abordagem permite que o robô virtual siga uma trajetória e responda autonomamente aos comandos recebidos.

Como resultado, essa instância desempenha um papel crucial na geração de mensagens de saída, as quais encapsulam informações vitais sobre o estado do robô virtual. Essas mensagens incluem dados temporais, ângulos dos servos, leituras de sensores e coordenadas espaciais, fornecendo uma base rica para visualização do comportamento do robô virtual ou para o treinamento do controlador do robô real. Essa estrutura integrada e funcional permite uma abordagem eficaz na simulação e controle do sistema.

Essa classe desempenha um papel crucial na simulação do comportamento do robô e na criação de um ambiente de treinamento para o controlador. Ela permite que o modelo de rede neural preditiva seja usado para gerar dados de treinamento e validar o desempenho do controlador antes de implementá-lo no robô real.

3.5.6 Tratamento de dados

A função da thread de tratamento de dados no contexto do sistema consiste em processar as informações recebidas do SERM, seja ele virtual ou físico, e das demais threads, a fim de estruturá-las de maneira organizada para análise subsequente ou utilização no treinamento da rede neural. As etapas e funcionalidades centrais dessa função são detalhadas a seguir.

Esta etapa é crucial e envolve o armazenamento dos dados em arquivos separados, otimizando o acesso e uso futuro de todas as informações do sistema. Como, o ciclo atual do Arduino Mega, utilizado no SERM, estado atual dos atuadores e sensores, dentre outros.

Utilizando expressões regulares, a função identifica e isola os dados contidos entre chaves nas mensagens trocadas entre as trheads. Esses dados extraídos são posteriormente organizados em variáveis, representando informações cruciais, como número da mensagem (n), tempo (t), valores dos atuadores (EM) e (CM), dados dos sensores giroscópio (Gy) e acelerômetro (Ac) e posição inclinação (ψ, y), dentre outras

A função também gera um registro de depuração que contém informações detalhadas sobre os dados tratados. Esta função desempenha um papel fundamental na coleta, organização e armazenamento dos dados provenientes da simulação do robô quadrúpede, tornando-os prontos para análise futura.

4 RESULTADOS E DISCUSSÕES

Neste capítulo, os dados coletados, análises estatísticas e interpretações dos resultados são explorados. Através da análise quantitativa e qualitativa, busca-se compreender o impacto das estratégias adotadas, os avanços alcançados e as limitações encontradas durante a implementação do modelo de robô e das redes neurais através da apresentação das métricas como erro médio, acurácia e qualquer outra medida de desempenho relevante.

4.1 Desempenho dos Modelos preditivos

No âmbito do presente trabalho, a avaliação dos modelos foi conduzida por meio de uma abordagem sistemática de testes. Inicialmente, o conjunto de dados foi dividido em conjuntos de treinamento e teste para avaliar a capacidade de cada rede em generalizar para dados não vistos durante o treinamento. Durante a fase de treinamento, foram utilizadas métricas como o erro médio quadrático (MSE) e a acurácia para monitorar o desempenho da rede.

Com o propósito de alcançar esse objetivo, foram selecionadas diversas arquiteturas acompanhadas de distintas funções de perda e otimizadores. Essa abordagem foi adotada com o intuito de identificar quais configurações proporcionam os resultados mais destacados em termos de desempenho.

A Tabela 2 exibe a comparação dos resultados entre os modelos durante a seleção da Função de Ativação. Foram empregados quatro modelos, cada um com uma função de ativação distinta. Foram utilizadas as funções “*ReLU*”, “*Leaky*”, “*ReLU*”, Tangente Hiperbólica e Sigmoide, sendo notável que o modelo 4, o qual adotou a função "Sigmoide", destacou-se como o mais eficiente em relação aos demais.

É importante salientar que, além dos modelos explicitados nas tabelas subsequentes, uma análise abrangente de diversas combinações foi realizada. Aquelas que demonstraram variações notáveis foram categorizadas com base em características específicas, visando observar o impacto da variação de parâmetros na evolução do modelo. A Tabela 4 agrupa os modelos segundo a função de ativação, a Tabela 5 de acordo com a função de perda, a Tabela 6 em relação ao otimizador, a Tabela 7 conforme a arquitetura, e, por último, a Tabela 8 engloba os modelos que incorporaram técnicas para mitigar o overfitting.

Em cada iteração de teste/treinamento em um conjunto de dados, a acurácia foi calculada pelo algoritmo, representando a porcentagem de previsões corretas em relação ao número total de instâncias no conjunto de dados. Uma previsão foi considerada correta quando o valor do erro médio quadrático entre a previsão do modelo e o rótulo real foi menor do que 5%, conforme definido no contexto em Python. Essa métrica é obtida pela fórmula:

$$Acurácia = \frac{Número\ de\ Previsões\ Corretas}{Número\ de\ Amostras}\% \quad (5)$$

Onde:

- Número de previsões corretas: É a contagem de instâncias em que o modelo fez previsões com um erro absoluto menor que 5% em relação aos rótulos reais.
- Número de amostras: Quantidade total de instâncias no conjunto de dados em questão.

Tabela 4 - Comparação de Modelos pela Função de Ativação

Modelo	Arquitetura da Rede Neural	Função de Ativação	Função de Perda	Algoritmo de Otimização	Precisão/Acurácia No Treinamento	Tempo de Treinamento
Modelo 1	4 camadas ocultas, 5 neurônios cada	ReLU	Erro Quadrático Médio	ADAM	70%	5 min
Modelo 2	4 camadas ocultas, 5 neurônios cada	Leaky ReLU	Erro Quadrático Médio	ADAM	62%	5 min
Modelo 3	4 camadas ocultas, 5 neurônios cada	Tangente Hiperbólica (Tanh)	Erro Quadrático Médio	ADAM	75%	5 min
Modelo 4	4 camadas ocultas, 5 neurônios cada	Sigmoid	Erro Quadrático Médio	ADAM	79%	5 min

Empregando a função Sigmoide, que demonstrou resultados superiores nos testes preliminares, foram concebidos três modelos idênticos. Em cada um deles,

uma função de perda distinta foi aplicada: “Entropia Cruzada”, “Erro Médio Quadrático” e “Erro Absoluto Médio”. O modelo que apresentou o melhor desempenho foi aquele associado ao “Erro Médio Quadrático”, conforme pode ser visto na Tabela 5.

Tabela 5 - Comparação de Modelos pela Função de Perda

Nome do Modelo	Arquitetura da Rede Neural	Função de Ativação	Função de Perda	Algoritmo de Otimização	Precisão/Acurácia No Treinamento	Tempo de Treinamento
Modelo 4.1	4 camadas ocultas, 5 neurônios cada	Sigmoid	Entropia Cruzada	ADAM	75%	5 min
Modelo 4.2	4 camadas ocultas, 5 neurônios cada	Sigmoid	Erro Quadrático Médio	ADAM	79%	5 min
Modelo 4.3	4 camadas ocultas, 5 neurônios cada	Sigmoid	Erro Absoluto Médio	ADAM	56%	5 min

Em seguida, foram feitos testes utilizando diferentes otimizadores para o mesmo modelo. A comparação foi feita entre o algoritmo de otimização “ADAM” e o “SGD”. Utilizando o “SGD”, o modelo obteve um desempenho melhor, conforme pode ser visto na Tabela 6.

Tabela 6 - Comparação de Modelos pelo Otimizador

Nome do Modelo	Arquitetura da Rede Neural	Função de Ativação	Função de Perda	Algoritmo de Otimização	Precisão/Acurácia No Treinamento	Tempo de Treinamento
Modelo 4.2.1	4 camadas ocultas, 5 neurônios cada	Sigmoid	Erro Quadrático Médio	ADAM	79%	5 min
Modelo 4.2.2	4 camadas ocultas, 5 neurônios cada	Sigmoid	Erro Quadrático Médio	SGD	81%	5 min

Para a análise das arquiteturas dos modelos foi feito o seguinte teste: O teste consiste em avaliar a capacidade do modelo preditivo em antecipar com precisão o comportamento futuro com base em dados previamente observados. No contexto deste estudo, o modelo selecionado foi submetido a esse teste para verificar sua

eficácia na predição de estados subsequentes do sistema. Durante o teste, o modelo utiliza dados de entrada conhecidos e tenta prever os estados futuros do sistema, sendo posteriormente comparados com os estados reais observados. Essa abordagem permite uma avaliação direta do desempenho do modelo na tarefa de predição, fornecendo informações sobre sua capacidade de generalização para além dos dados de treinamento. Esse teste permitiu avaliar a capacidade do modelo em antecipar quantos pontos da sequência temporal podem ser previstos com precisão, considerando uma margem de erro aceitável

Foram avaliados quatro modelos distintos, abrangendo uma rede estreita e profunda, uma larga e profunda, uma estreita e rasa e uma larga e rasa. Destes, o modelo 4.2.2 (largo e profundo) destacou-se ao exibir o melhor desempenho durante o treinamento, enquanto o modelo 4.2.2.3 (estreito e profundo) apresentou o desempenho mais expressivo nos dados de teste.

Tabela 7 - Comparação de Modelos pela Arquitetura

Nome do Modelo	Arquitetura da Rede Neural	Precisão/ Acurácia No Treinamento	Tempo de Treinamento	Precisão/Acurácia nos dados de teste de Predição Simples	Teste de Predição Livre: Número de Pontos Previstos
Modelo 4.2.2.1	Estreita e rasa: 4 camadas ocultas, 13 neurônios cada	84%	8 min	72%	1
Modelo 4.2.2.2	Estreta e profunda: 12 camadas ocultas, 13 neurônios cada	93%	20 min	90%	3
Modelo 4.2.2.3	Profunda e larga: 12 camadas ocultas, 64 neurônios cada	96%	1,1 horas	89%	2
Modelo 4.2.2.4	Rasa e Larga: 4 camadas ocultas, 64 neurônios cada	85%	30min	72%	1

É importante observar na Tabela 7 que os modelos 4.2.2.2 (Estreito e Profundo) e 4.2.2.3 (Profundo e Largo) obtiveram destaque. Entretanto, apesar de o modelo Largo e Profundo alcançar o melhor desempenho nos dados de treinamento,

o seu desempenho foi inferior ao modelo mais estreito nos dados de teste. Isso indica um erro de overfitting. Com isso, pode-se concluir que o modelo Largo e Profundo se ajustou excessivamente aos dados de treinamento, resultando em um desempenho inferior ao enfrentar novos dados não vistos durante o treinamento. O modelo Estreito e Profundo, por sua vez, apresentou um desempenho mais equilibrado, destacando a importância de buscar um equilíbrio entre a complexidade do modelo e sua capacidade de generalização para dados não utilizados anteriormente.

Para este problema, foi utilizado duas técnicas: O Dropout e o afunilamento das camadas.

Posteriormente aos resultados, realizaram-se ajustes mínimos no modelo escolhido e repetiu-se os testes com o intuito de aprimorar, ainda que de forma marginal, o desempenho alcançado até então. Ao expandir as camadas iniciais e criar um modelo que gradualmente reduz sua largura, observou-se uma melhoria de 2%, alcançando 92% nos dados de teste e 95% nos dados de treinamento. O modelo escolhido para a rede preditiva, aplicando o dropout e a técnica de afunilamento das camadas, é o 4.2.2.2.2, conforme apresentado na Tabela 8. Este modelo caracteriza-se por ser uma Rede Profunda, ampla nas camadas iniciais e estreita nas camadas finais.

Tabela 8 – Ajuste final

Nome do Modelo	Arquitetura da Rede Neural	Precisão/Acurácia No Treinamento	Tempo de Treinamento	Precisão/ Acurácia nos dados de teste de Predição Simples	Teste de Predição Livre: Número de Pontos Previstos
Modelo 4.2.2.2.1	12 camadas ocultas, 13 neurônios cada	86%	40 min	90%	2
Modelo 4.2.2.2.2	Rede Profunda, larga nas camadas iniciais e estreita nas camadas finais, com Dropout.	95%	40 min	92%	5

Após o treinamento, a modelo 4.2.2.2.2 foi submetido a novos testes, nos quais foram empregados conjuntos de dados distintos daqueles utilizados durante o

processo de treinamento. Essa abordagem permitiu avaliar a capacidade da rede em realizar previsões precisas e adaptar-se a diferentes condições.

Além disso, a validação foi estendida a cenários mais desafiadores, incorporando variações nos dados de entrada e perturbações simuladas no ambiente do robô quadrúpede. Esse procedimento visou assegurar que a rede preditiva pudesse lidar eficazmente com condições adversas e imprevistas.

A avaliação da rede preditiva não se restringiu apenas a métricas quantitativas, abrangendo também uma análise qualitativa das previsões. Foi examinada a capacidade da rede em antecipar eventos críticos, como mudanças bruscas no ambiente ou no comportamento do robô.

O processo de validação contemplou ainda a comparação entre as previsões da rede preditiva e os dados reais coletados durante a operação do robô. Essa comparação permitiu identificar eventuais discrepâncias e ajustar parâmetros, aprimorando assim a precisão e confiabilidade da rede preditiva.

Em suma, a validação da rede preditiva envolveu uma abordagem abrangente que combinou métricas quantitativas, testes em condições diversas e análise qualitativa, assegurando a eficácia do modelo desenvolvido.

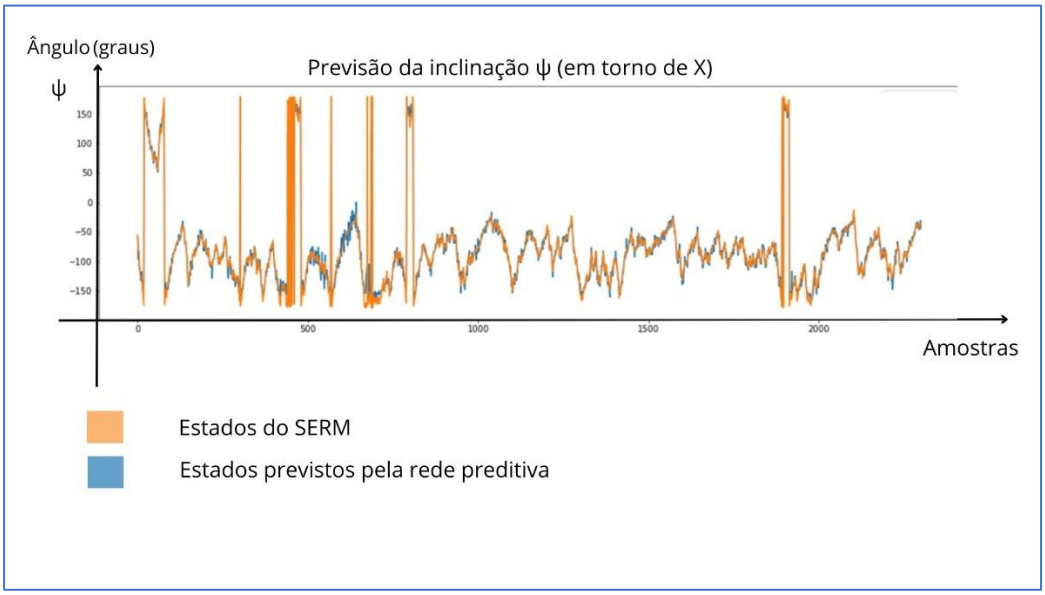
O teste inicial consistiu na aplicação de predição simples, visando antecipar o subsequente estado de cada ponto ao longo da linha temporal. Posteriormente, realizou-se uma comparação entre o estado real observado e o estado predito, sendo que os resultados para os dados de treinamento foram apresentados na Figura 18 e para os dados de teste na Figura 19. Esse procedimento permitiu avaliar a acurácia do modelo ao prever estados futuros em relação aos dados reais, fornecendo uma análise direta de sua capacidade de generalização e desempenho em condições práticas.

A Figura 21 apresenta graficamente as previsões realizadas pelo modelo 4.2.2.2, como detalhado na Tabela 6, para as amostras conhecidas, nas quais ele foi treinado, obtendo uma acurácia de 95%. O gráfico visualiza de forma elucidativa os resultados contidos na tabela, proporcionando uma representação visual das previsões e do desempenho do modelo em relação aos dados de treinamento.

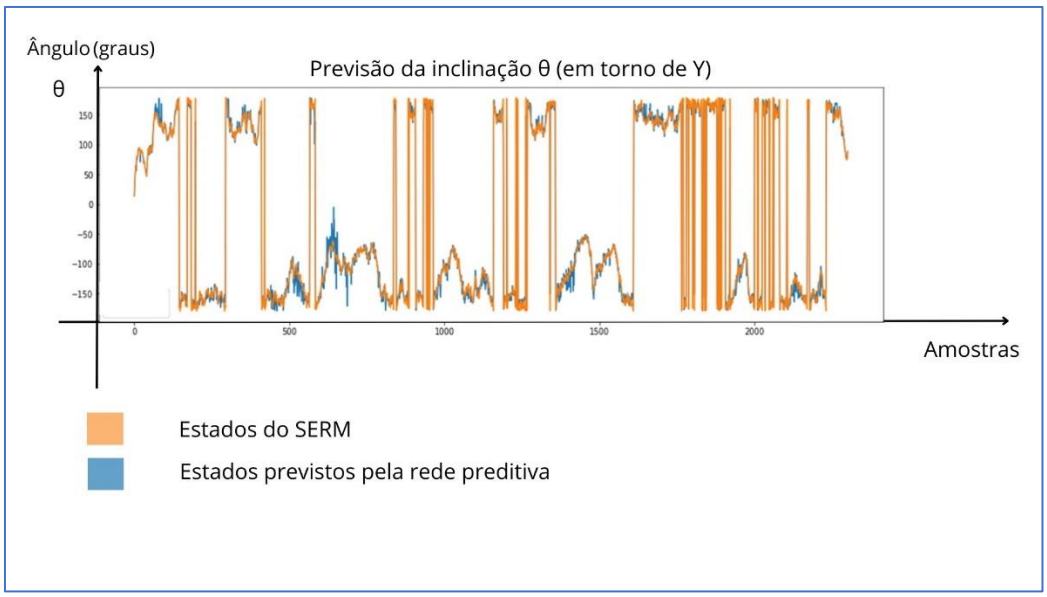
As Figuras 21 à 26 consistem em duas subfiguras, (a) e (b), representando as previsões do ângulo θ (em torno de Y) e ψ (em torno de X), respectivamente. As acurácias apresentadas foram calculadas de acordo com o erro médio quadrático entre os vetores previstos $(\psi P, \theta P)$ e os vetores reais correspondentes $(\psi R, \theta R)$. Essa

métrica proporciona uma avaliação quantitativa da precisão das previsões, considerando a diferença entre as previsões do modelo e os valores reais para os ângulos ψ e θ.

Figura 21 - Predição simples nos dados de treinamento



(a)



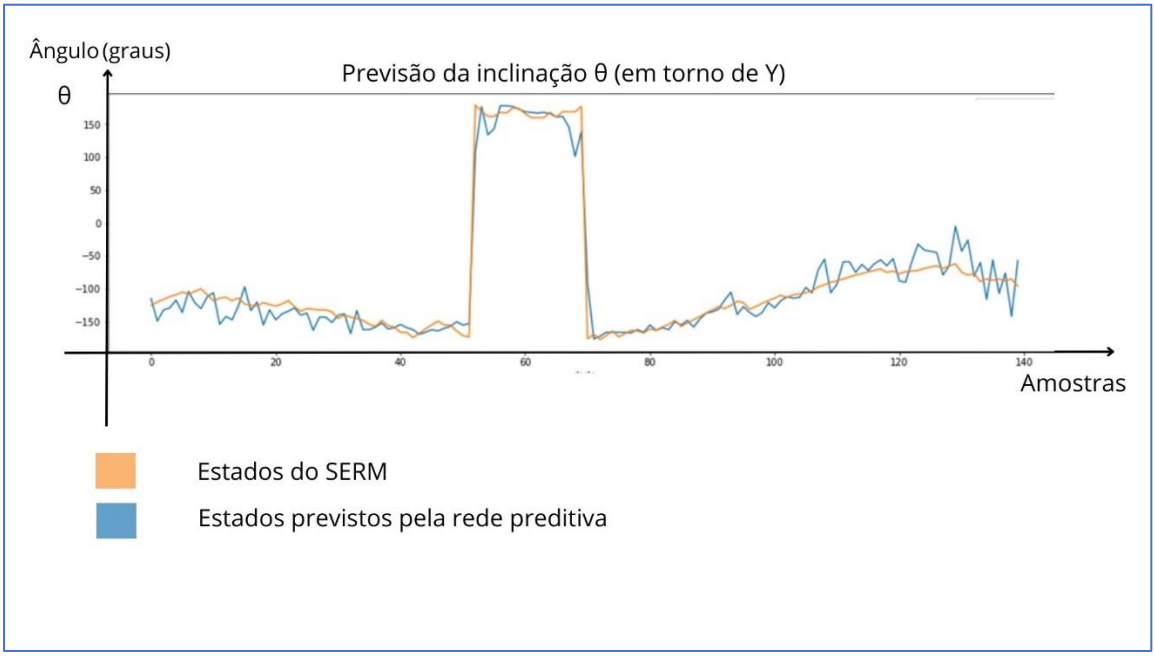
(b)

Fonte: Autor.

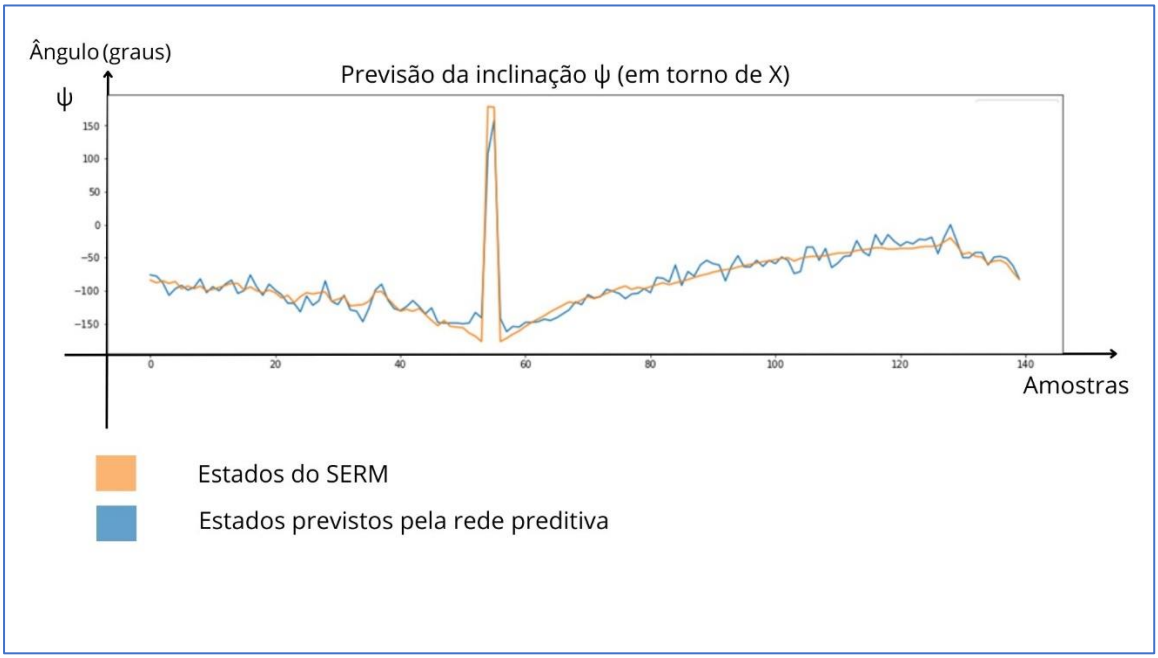
Observa-se que, nos dados de treinamento, a curva que representa as previsões mantém uma alta fidelidade em relação à curva real que descreve o comportamento do SERM. Contudo, devido ao fato de a rede neural ter conhecimento prévio desses dados durante o treinamento, é essencial adotar uma abordagem

cautelosa na avaliação do desempenho, pois não é possível discernir se o modelo efetivamente aprendeu os padrões ou se simplesmente os memorizou. Para uma avaliação mais rigorosa, é imperativo introduzir conjuntos de dados não vistos (dados de teste) e observar o desempenho da rede neural, como ilustrado na Figura 22.

Figura 22 - Predição simples nos dados de teste



(a)



(b)

Fonte: Autor.

Conforme antecipado, em face de dados inéditos, a rede neural apresentou uma diminuição de desempenho, alcançando, contudo, uma acurácia de 92%. Este resultado ainda está em conformidade com os critérios de desempenho delineados na

Tabela 1. Indicando que a rede neural demonstrou capacidade de aprendizado e generalização para previsões mais abrangentes, ou seja, para cada amostra, foi apta a prever o próximo estado de forma satisfatória.

Na avaliação dos resultados observados de acurácia nos dados de treinamento e teste, constatou-se que o modelo demonstrou uma capacidade preditiva satisfatória, refletida pela proximidade entre as duas linhas que representam os estados reais e previstos. O erro associado a essa previsão mostrou-se dentro de limites aceitáveis, com base nos critérios de desempenho avaliados até o momento. A análise dos resultados numéricos obtidos em python foi confrontada de maneira visual, mediante a observação direta dos gráficos apresentados a seguir e subsequentemente discutidos neste contexto. Essa abordagem visual permitiu uma rápida avaliação do desempenho do modelo em relação aos dados de treinamento.

Embora os resultados do teste de predição simples tenham demonstrado uma performance positiva, é imperativo adotar uma abordagem mais robusta para efetivamente validar o modelo preditivo. O teste de predição simples, ao antecipar o próximo ponto ao longo da linha temporal, pode, em determinadas circunstâncias, exibir uma eficácia aparente, uma vez que o modelo pode inadvertidamente "estimar" que o próximo ponto será igual ao anterior, comprometendo, assim, a qualidade da previsão. Quando esse cenário se manifesta, o modelo pode ser caracterizado como um preditor trivial, destacando a necessidade de métodos de avaliação mais abrangentes para garantir a robustez e a generalização efetiva do modelo preditivo.

Para contornar essa limitação, é essencial realizar um teste de predição livre. Nesse contexto, é sorteado um ponto aleatório no conjunto de dados e definido como ponto inicial, e posteriormente, o ponto sorteado é utilizado como estado inicial para prever um segundo estado que é realimentado no modelo. Esse processo é iterativamente repetido, permitindo avaliar até que ponto o modelo é capaz de realizar previsões sem depender unicamente da continuidade dos dados, proporcionando uma validação mais abrangente e confiável do seu desempenho, conforme ilustra as Figuras 20, 21 e 22. O erro aceitável para o ponto inicial é de 5%, porém dado que o ponto é realimentado, espera-se que ocorra uma propagação do erro, ou seja um novo ponto acumula o erro das previsões anteriores.

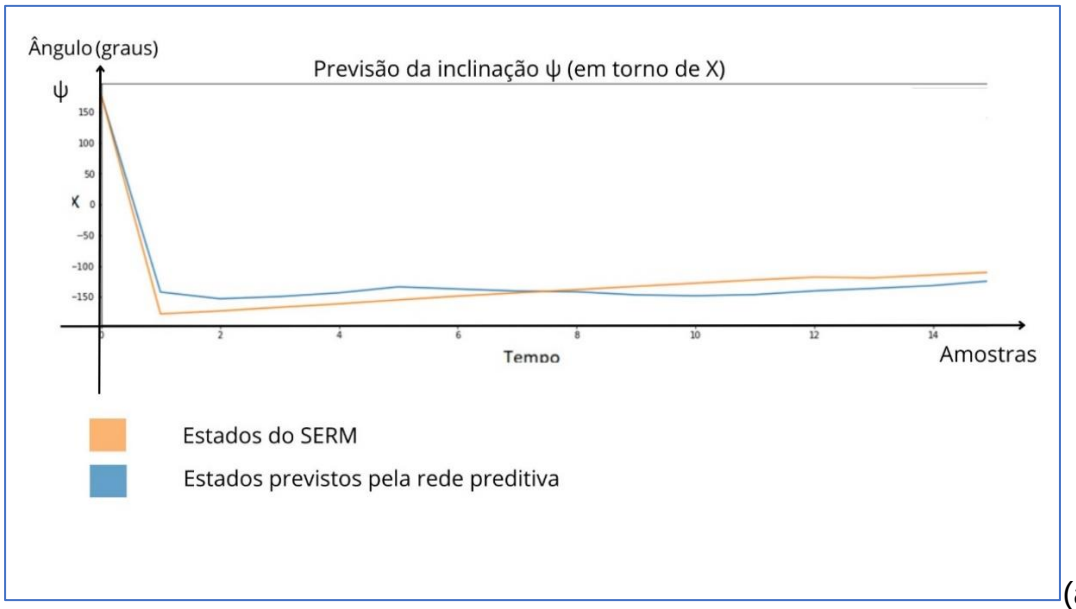
Para o teste de predição livre, é fundamental destacar que a exigência de precisão absoluta nas previsões não é o critério preponderante. O principal objetivo consiste na capacidade do modelo em antecipar adequadamente o comportamento

geral do SERM, fornecendo indicações precisas da direção do movimento e uma estimativa aproximada da magnitude da inclinação. Em outras palavras, o modelo não é estritamente avaliado pela precisão exata dos valores angulares, mas sim pela eficácia em capturar a tendência global do SERM, como a direção de sua inclinação.

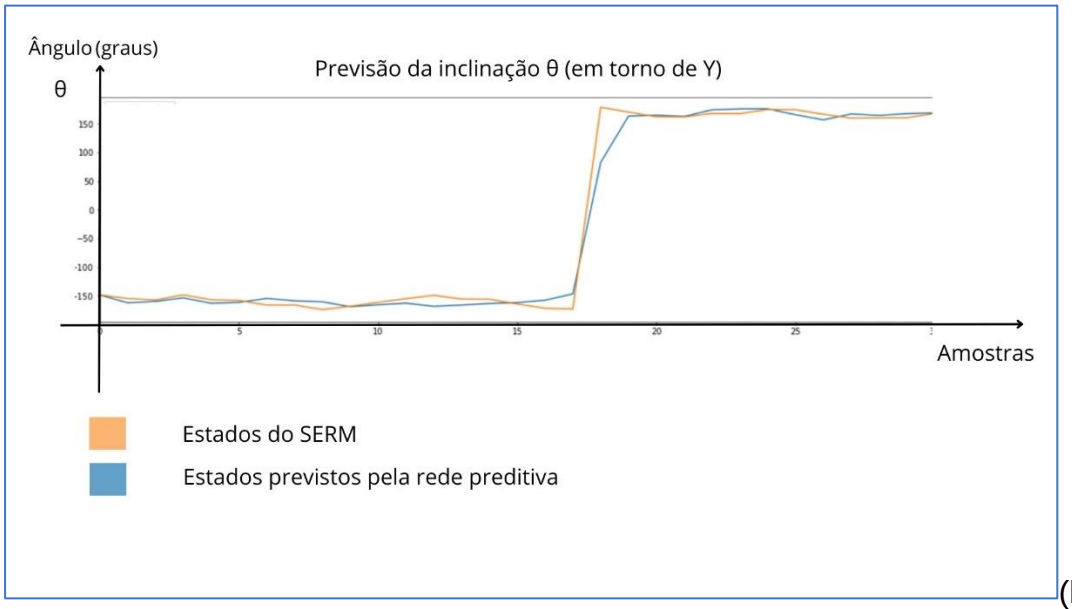
Por exemplo, durante uma inclinação do SERM, espera-se que o modelo antecipe corretamente qual é essa direção de queda e forneça uma indicação geral da magnitude, mesmo que não atinja uma previsão exata. Nesse contexto, uma acurácia reduzida de 85%, conforme a os critérios de desempenho definidos na tabela 1, seria considerada satisfatória, dada a ênfase na capacidade do modelo de discernir e antecipar com precisão os padrões de movimento do SERM, em vez de atingir previsões pontuais altamente precisas.

O algoritmo identifica o ponto em que o limite é alcançado, fornecendo como resultado a contagem de pontos previstos. Adicionalmente, é gerado um gráfico para uma avaliação visual do desempenho.

Figura 23 - Teste de predição Livre I



(a)



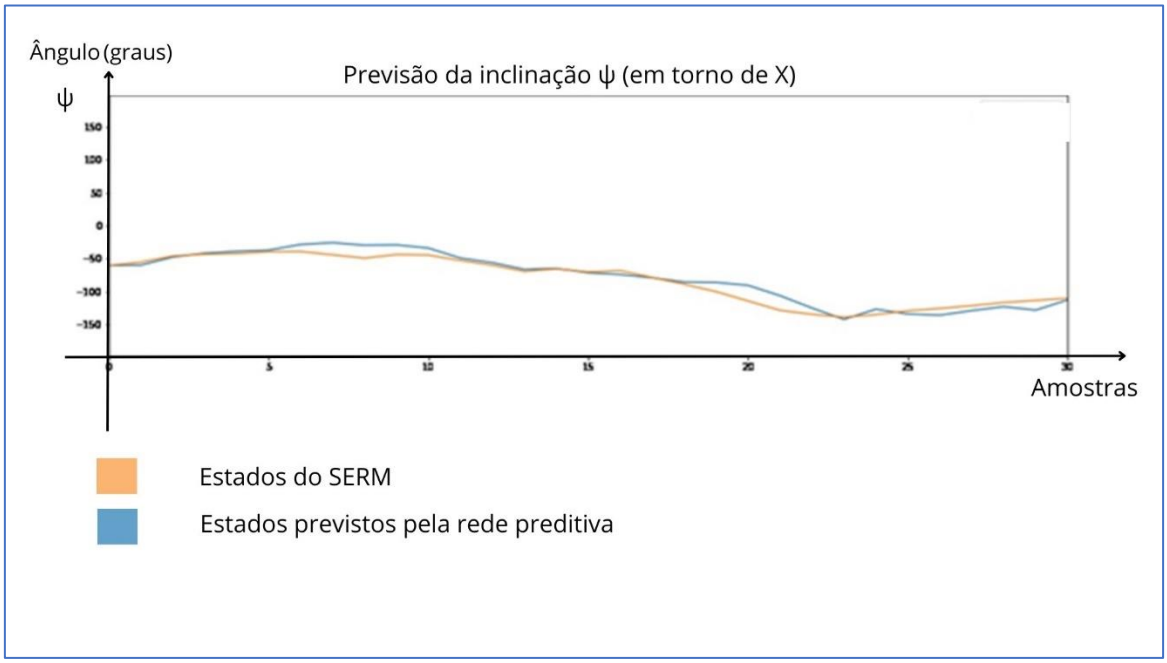
(b)

Fonte: Autor.

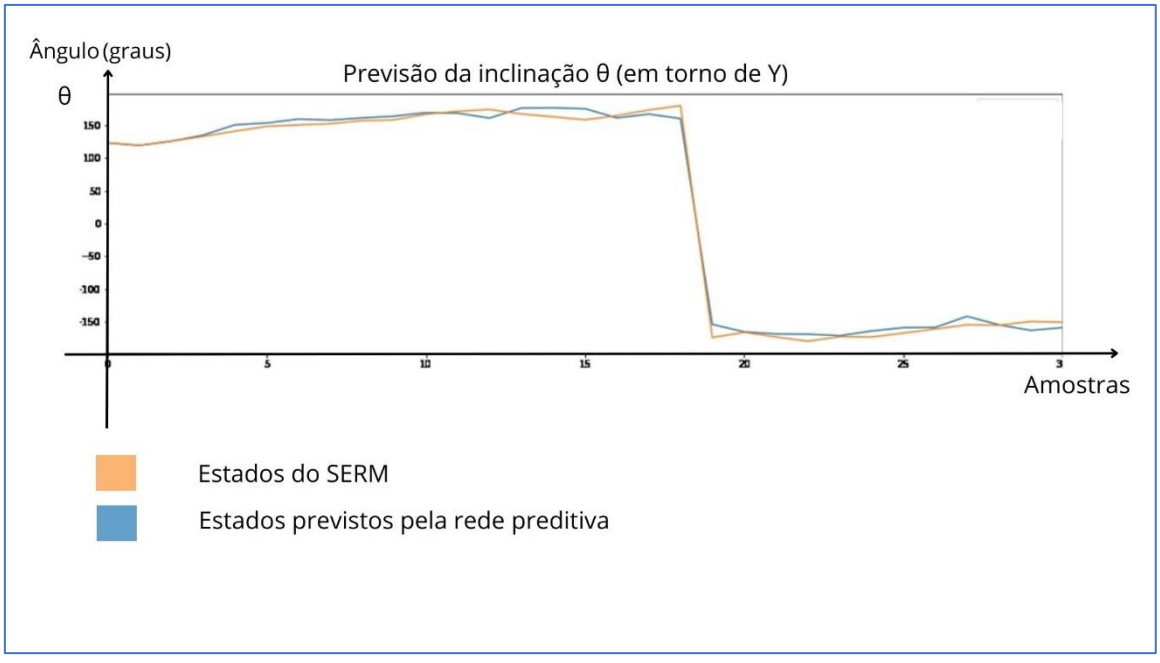
O modelo demonstrou eficácia na previsão do comportamento do SERM, conforme evidenciado na Figura 23. A análise do gráfico revelou a ocorrência de um padrão comportamental semelhante a um degrau, no qual a rede neural demonstrou eficácia ao antecipar esse comportamento. A linha que representa as previsões do modelo apresentou uma notável semelhança visual com a trajetória real, indicando uma capacidade robusta de previsão. Essa observação ressalta a habilidade do modelo em capturar intricadas nuances dinâmicas, evidenciando sua aptidão em mapear adequadamente padrões de comportamento mais complexos, como o degrau identificado no gráfico. A competência do modelo em antecipar de maneira precisa a direção da inclinação do SERM reflete sua habilidade em capturar de forma acurada

os padrões dinâmicos do sistema. No entanto, ressalta-se que um único resultado não pode ser considerado como métrica definitiva de desempenho. Para determinar a quantidade de pontos que o modelo foi capaz de prever, o algoritmo repete o teste utilizando uma variedade de dados, realizando a média da quantidade de pontos previstos. Essa abordagem mais abrangente contribui para uma avaliação mais representativa do desempenho médio do modelo em diversas situações.

Figura 24 - Teste de predição Livre II



(a)

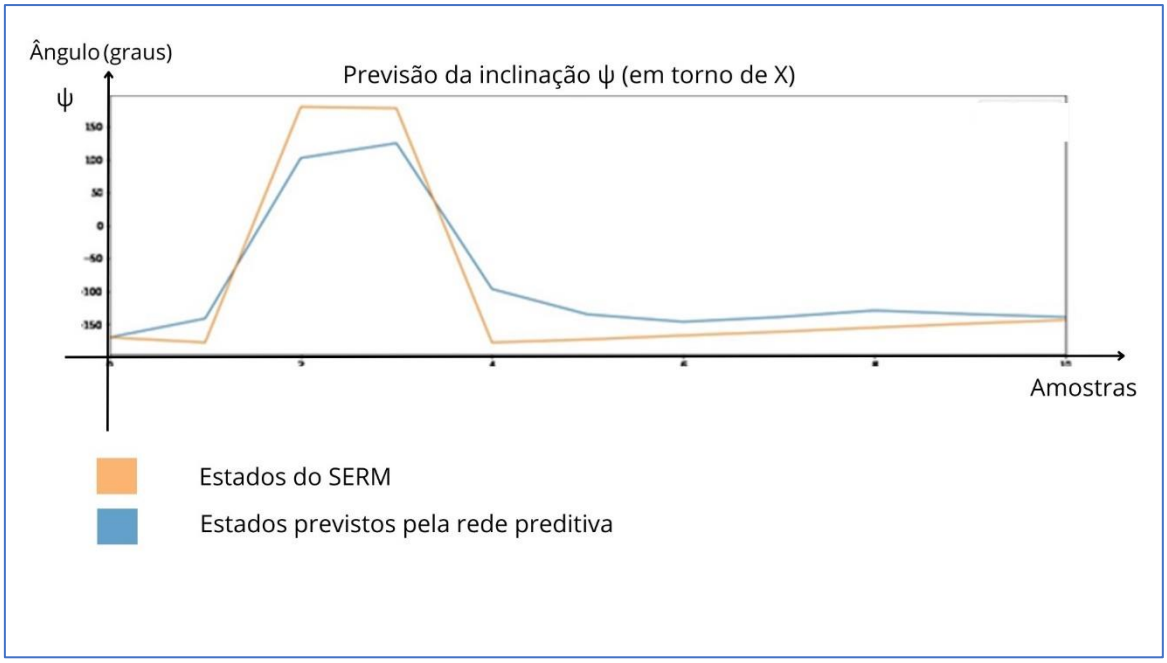


(b)

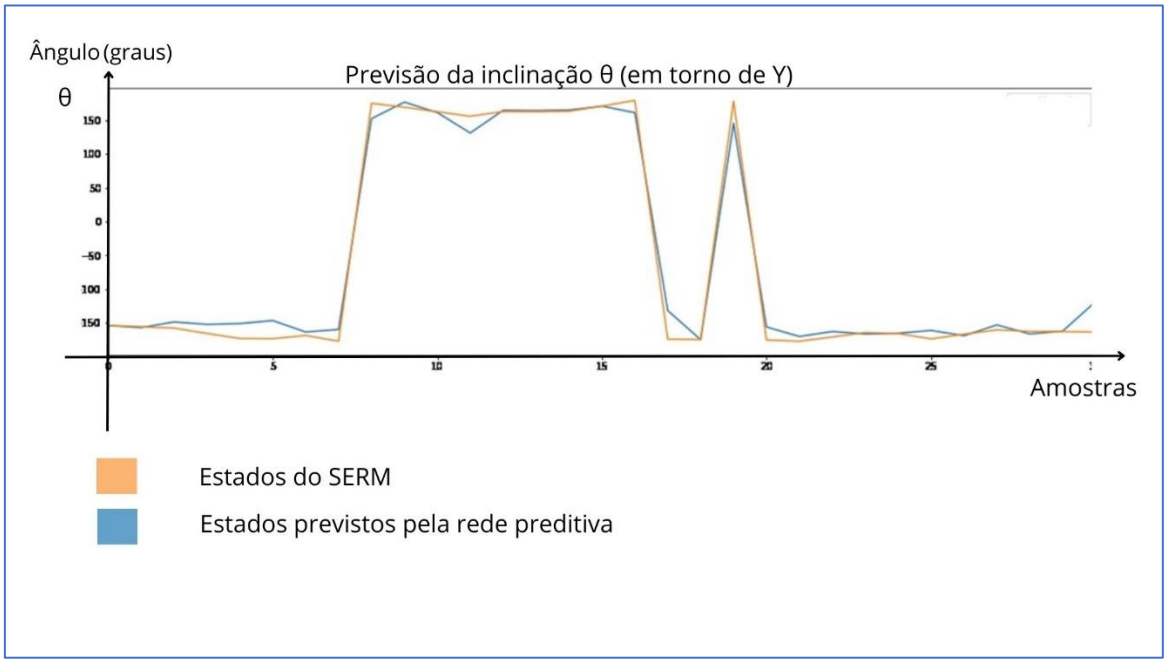
Fonte: Autor.

Na Figura 24, observa-se uma analogia comportamental, similar à situação anterior, indicando que a rede neural novamente apresentou eficácia na previsão. A linha representativa das previsões do modelo mostra uma proximidade notável com a trajetória real.

Figura 25 - Teste de predição Livre III



(a)



(b)

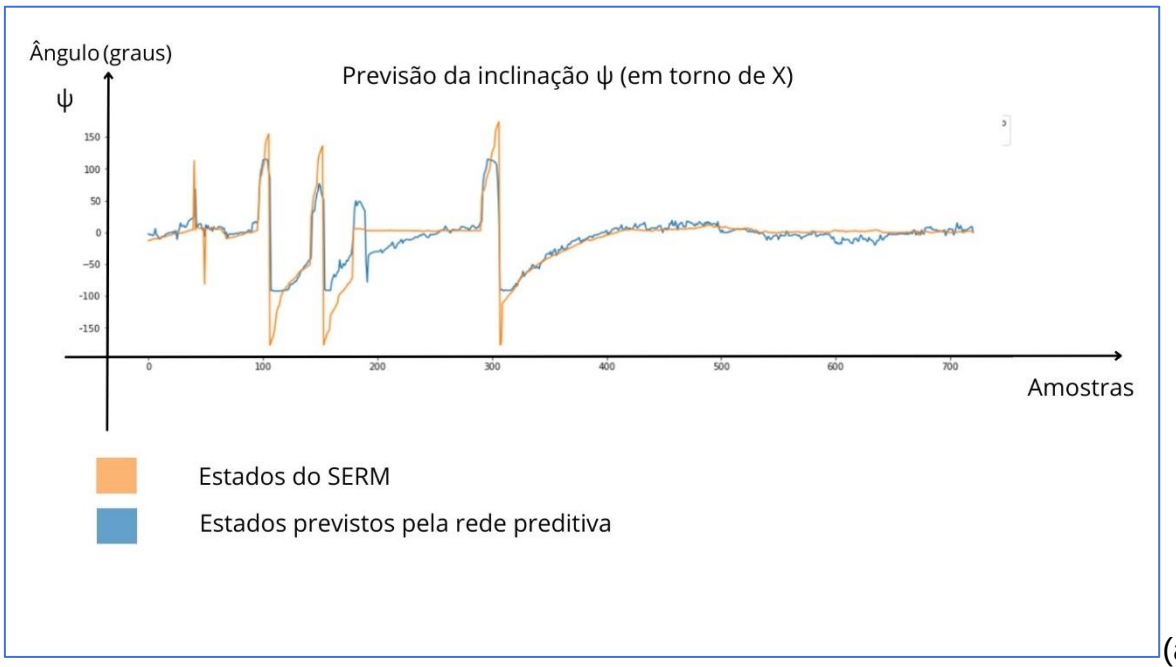
Fonte: Autor.

No gráfico representado na Figura 25, pode-se observar que, diferente dos testes anteriores, no teste 3, o SERM apresentou uma dinâmica nos dois eixos, e isso possibilitou observar a capacidade do modelo em prever estados, quando há variação nos dois parâmetros (ψ e θ). Visualmente, o modelo conseguiu representar a dinâmica conforme pode ser observado no formato das linhas geradas.

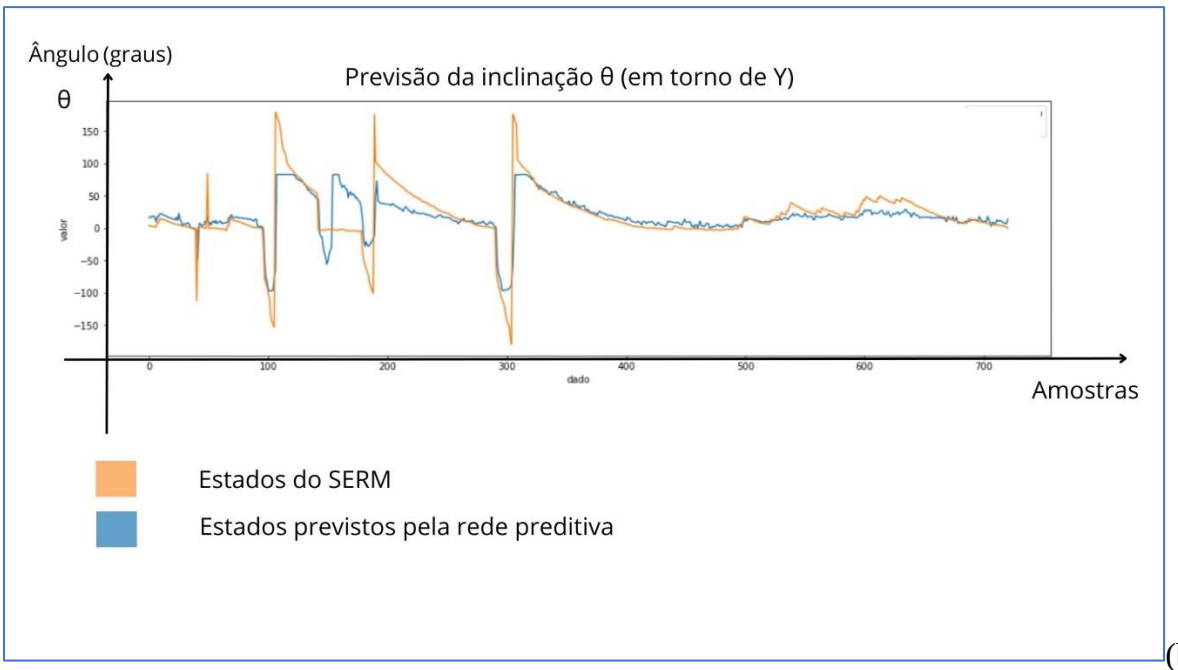
Na Figura 26, é apresentado um teste abrangente de predição livre, caracterizado por um maior número de pontos e dinâmicas mais exigentes, envolvendo variações mais bruscas. A rede preditiva, embora apresente erros significativos em pontos específicos, demonstra um comportamento que guarda semelhanças com a trajetória real. Este resultado sugere que a rede não se configura como um preditor trivial, capaz de antecipar efetivamente comportamentos complexos, apesar das limitações em determinados pontos.

O resultado global calculado utilizando o algoritmo, considerando múltiplos testes, a rede preditiva se destaca ao prever consistentemente os próximos 5 estados do SERM, mantendo uma acurácia média de 85%. Essa métrica confirma a habilidade da rede em antecipar o comportamento do sistema em cenários desafiadores, evidenciando sua robustez na previsão de dinâmicas mais exigentes.

Figura 26 - Teste de predição Livre IV



(a)



(b)

Fonte: Autor.

Os testes de predição livre representam uma etapa crucial na avaliação da eficácia do modelo preditivo. Ao submeter o modelo à tarefa desafiadora de prever pontos futuros de forma iterativa, verificou-se que os resultados foram favoráveis, e o erro associado permaneceu dentro das expectativas estabelecidas, isto é, a dinâmica presente na curva prevista demonstrou uma notável semelhança com a dinâmica real do robô, reforçando a validade do modelo preditivo. Essa correspondência entre a

previsão virtual e o comportamento real do sistema é fundamental, pois atesta a capacidade do modelo de capturar com precisão as complexidades dinâmicas do robô. Esses resultados positivos fornecem uma base sólida para a utilização do modelo virtual na projeção do controlador, indicando que o modelo é capaz de generalizar e adaptar-se às nuances do sistema real. Essa validação robusta é essencial para garantir a aplicabilidade prática do modelo preditivo no contexto do controle do robô quadrúpede.

Uma observação adicional relevante é a estabilidade do modelo sob diferentes pontos de partida. Durante os testes de predição livre, foi possível notar que o modelo manteve um desempenho consistente mesmo diante de variações nas condições iniciais. Isso sugere uma capacidade adaptativa do modelo em lidar com diferentes cenários, indicando que soube lidar com a não linearidade característica de um sistema com muitas variáveis. Além disso, a análise do comportamento do erro ao longo do tempo revelou padrões previsíveis e controláveis, o que é crucial para a implementação prática do modelo em um sistema de controle em tempo real. Essa estabilidade e previsibilidade do erro contribuem para a confiança na utilização do modelo preditivo no modelo físico, onde as condições podem ser mais dinâmicas e variáveis.

Tabela 9 – Resultado da Rede Preditiva

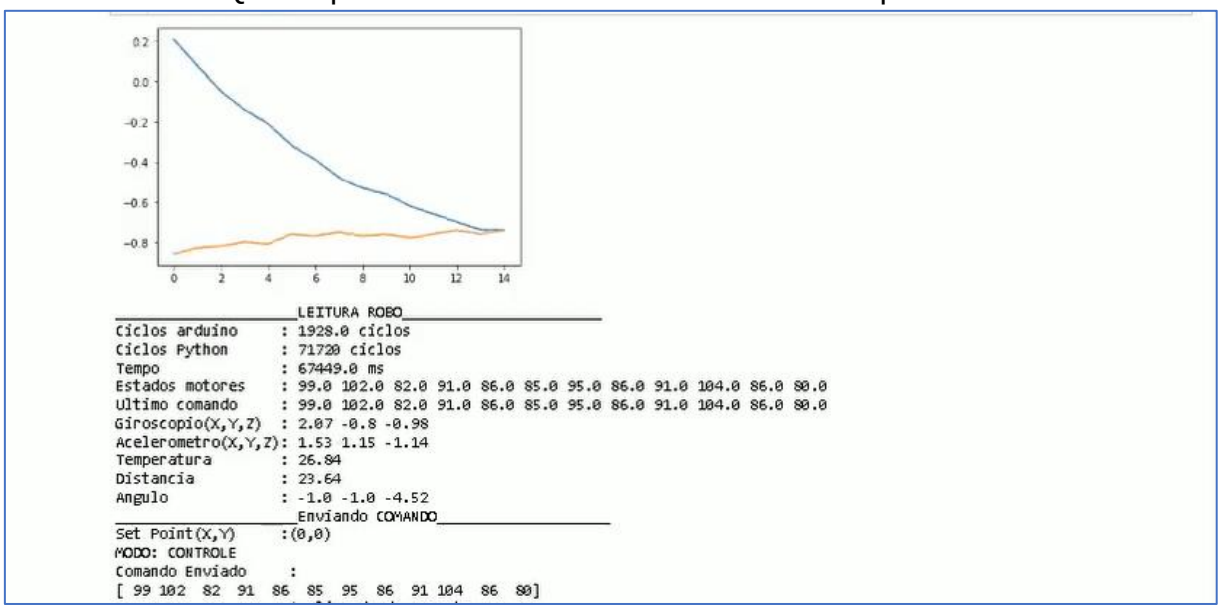
Resultado Obtido pela Rede Preditiva				
Critério	**Unidade**	**Desempenho obtido**	**Faixa que atende ao critério**	**Critério atendido?**
Precisão/Acurácia nos dados de treinamento	%	>=95	95	Sim
Precisão/acurácia nos dados de teste de previsão simples	%	>=90	92	Sim
Número de pontos previstos no teste de predição livre, com acurácia de 85%.	Pontos	3	5	Sim

4.2 Desempenho do Controlador Neural

Antes da análise dos resultados do controlador, realizou-se uma avaliação do sistema de comunicação serial e aquisição de dados do robô físico. Os dados lidos foram, o ciclo atual do programa embarcado no Arduino Mega, o ciclo atual do programa responsável pela lógica de controle em python, o tempo, o estado dos

atuadores e motores, dentre outras informações e comando reenviado, conforme pode ser visto na Figura 27.

Figura 27 – Display do Sistema exibindo a Comunicação Serial entre o Robô Quadrúpede e o Sistema de Controle em Tempo Real



Fonte: Autor.

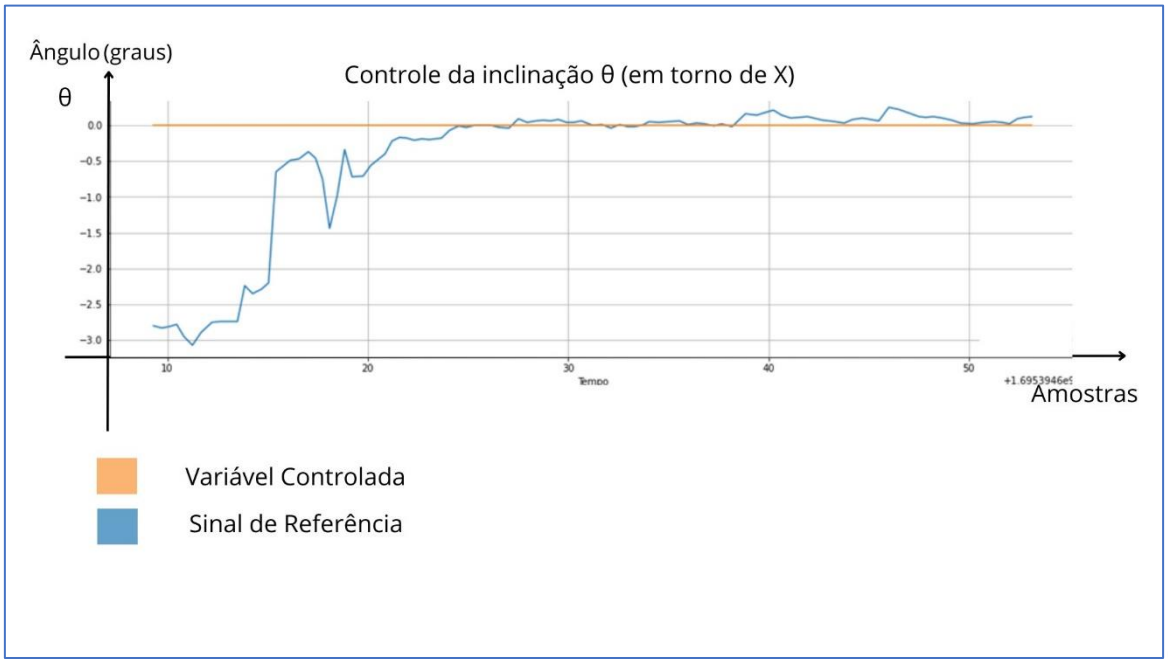
O sistema demonstrou eficiência ao possibilitar a comunicação em tempo real, processar variáveis dinâmicas e direcioná-las para o sistema de arquivo, efetuando o armazenamento em seus respectivos registros sem travamento, o que é essencial para que o robô possa agir de forma preditiva em tempo real.

Após certificar-se do funcionamento correto da comunicação serial, iniciou-se a verificação da capacidade da rede neural em guiar as ações do robô em conformidade com referências pré-determinadas. A análise do desempenho do controlador foi feita em cenários específicos, como respostas a degrau, perturbações e senoides, utilizando esses casos de teste como métricas fundamentais para avaliar a capacidade adaptativa e reativa do sistema diante de condições dinâmicas e variáveis.

O primeiro teste realizado para avaliação do controlador consistiu na análise da resposta ao degrau. O SERM foi posicionado sobre uma placa e posteriormente submetido a uma entrada em degrau, uma mudança abrupta e unitária na referência para a posição desejada, e sua resposta foi observada ao longo do tempo. A posição escolhida foi aquela em que o SERM está em equilíbrio considerando o offset existente no eixo, ou seja, $\psi = 0$ e $\theta = 5$. Este teste permitiu

avaliar a capacidade do controlador em seguir alterações repentinas na referência e analisar a estabilidade e a dinâmica do sistema em resposta a esse tipo de entrada, conforme mostra a Figura 28.

Figura 28 - Resposta ao Degrau



(a)

(b)

Fonte: Autor.

Previamente, é importante ressaltar que, ao avaliar os resultados do teste de resposta ao degrau e demais sinais exibidos nos gráficos subsequentes, a

consideração dos dois eixos do sinal foi fundamental. Diante da natureza bidimensional do sistema, a análise foi conduzida considerando o pior caso entre os eixos ψ e θ em cada situação. Em outras palavras, ao determinar parâmetros como o tempo de acomodação, tempo de subida, erro final e overshoot, foram selecionados os valores mais críticos entre os dois eixos. Como por exemplo na resposta ao degrau, o ângulo ψ alcançou a acomodação em 950 milissegundos, enquanto o θ em 50 milissegundos, logo o tempo de acomodação foi considerado como 950 milissegundos, visando abordar o cenário mais desafiador. Essa metodologia proporciona uma análise abrangente e conservadora, considerando o desempenho global do sistema em situações críticas.

A análise dos resultados do teste de resposta ao degrau revela aspectos significativos sobre o desempenho do controlador. O tempo de acomodação do sistema, mensurado em 950 milissegundos, indica o intervalo necessário para que a resposta do sistema alcance e permaneça dentro de uma faixa aceitável após a perturbação. Este parâmetro é crucial para determinar a estabilidade e a capacidade de recuperação do sistema.

De forma análoga, o erro no valor final calculado pelo algoritmo foi de 4% no ângulo ψ e 2% no ângulo θ, verifica-se a precisão do controlador em atingir a referência desejada após o estabelecimento do sistema. Um erro final reduzido é indicativo de uma resposta mais precisa e controlada.

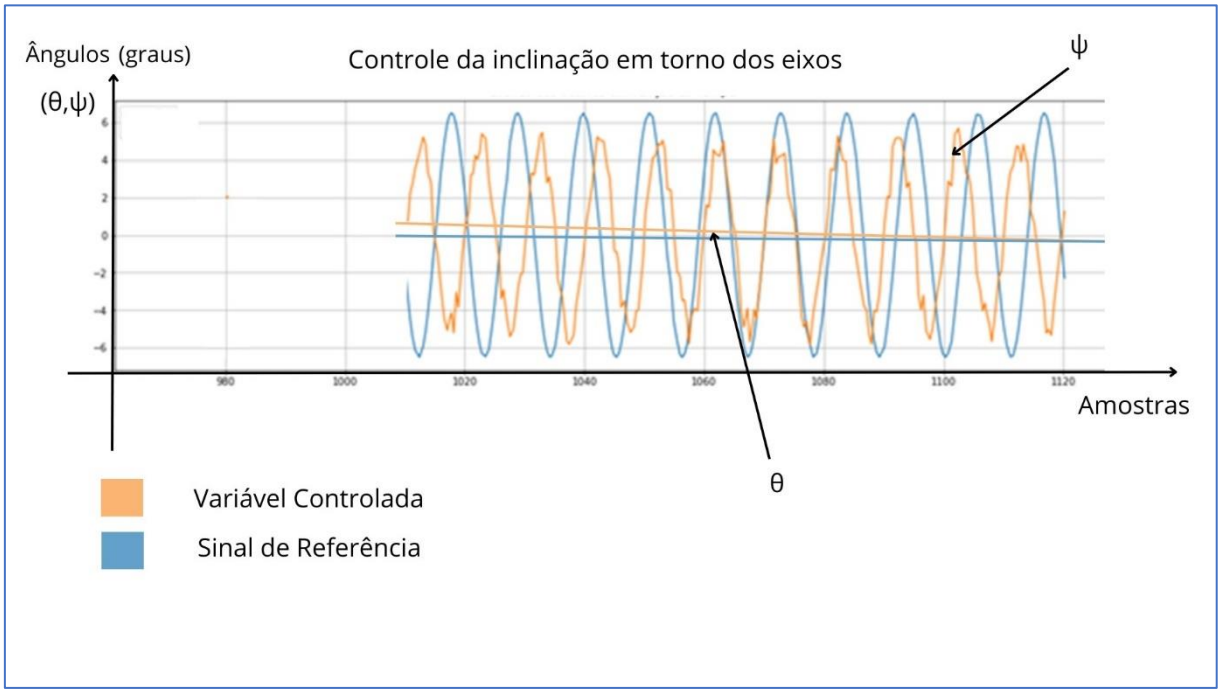
O tempo de subida, medido em 750 milissegundos, representa o intervalo necessário para que o sistema alcance pela primeira vez a referência simultaneamente nos ângulos ψ e θ após a perturbação. Este parâmetro está relacionado à agilidade e à velocidade de resposta do controlador.

Pode-se observa um overshoot no eixo que representa o ângulo ψ, de aproximadamente 12 °, que representa cerca de 400% da entrada. A magnitude desse valor antes do sistema estabilizar indica a presença de oscilações indesejadas ou instabilidades transitórias no sistema, o que não foi considerado no treinamento do controlador como um parâmetro a ser utilizado na penalidade ou recompensa e, portanto não será contabilizado para o seu desempenho, mas não deixa de ser um parâmetro a ser considerado em futuros estudos.

A análise conjunta desses parâmetros permitiu uma avaliação abrangente do desempenho do controlador na resposta ao degrau, fornecendo informações cruciais para ajustes e refinamentos no sistema de controle.

Após analisar a resposta ao degrau, foi utilizada uma senoide como referência para avaliar o comportamento do sistema diante de sinais oscilantes, a fim de avaliar o atraso, a robustez e a capacidade de rastreamento de uma trajetória do controlador cíclica. Durante a análise comparativa, são apresentadas duas situações distintas do controlador. A primeira delas foi treinada exclusivamente com dados do presente e do passado, enquanto a segunda incorporou o desconto temporal associado à rede preditiva, permitindo que o modelo fosse treinado considerando informações preditas, ou seja, dados estimados do futuro. As duas situações são apresentadas respetivamente na Figura 29 e Figura 30. Nos dois casos foi utilizado como uma referência uma senoide ψ e uma constante em θ.

Figura 29 - Resposta à Senoide Sem a Predição de Estados



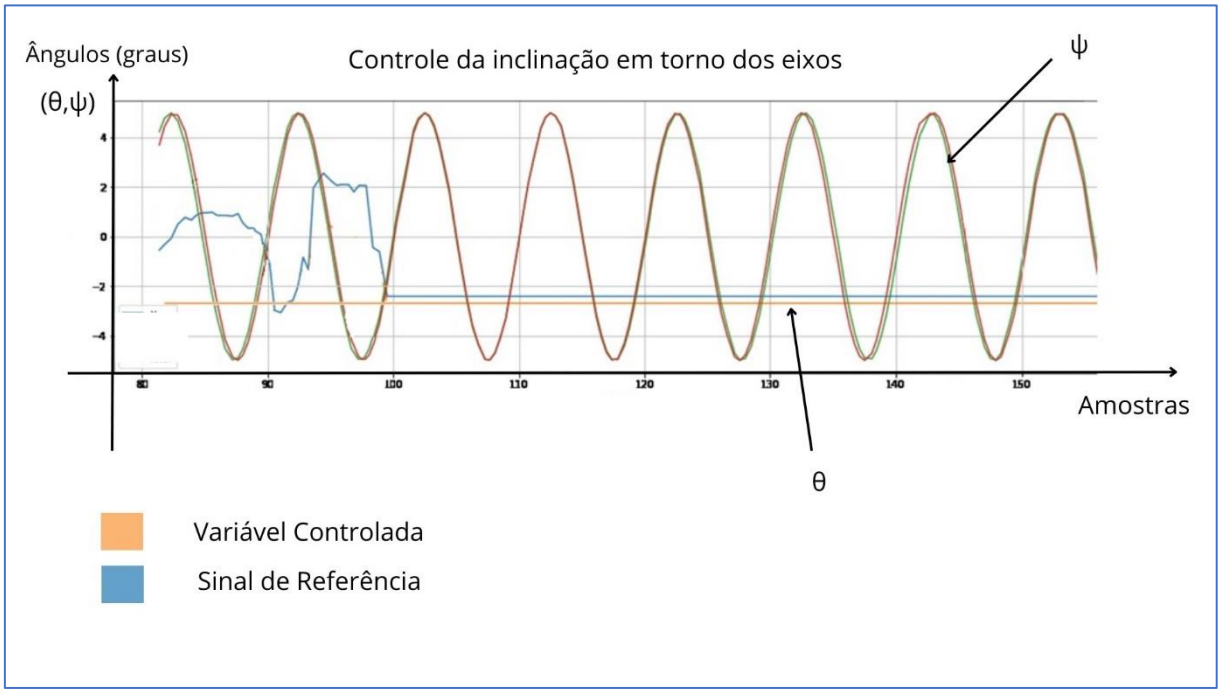
Fonte: Autor.

Ao conduzir o treinamento do controlador sem a utilização dos dados futuros no desconto temporal, ou seja, sem a assistência da rede preditiva, observa-se um desempenho inferior. O controlador apresentou um atraso considerável no processo de resposta.

Além disso, vale destacar que os dados provenientes do sensor apresentam uma magnitude considerável de ruído. Neste contexto, é interessante observar que a rede preditiva exerceu um papel crucial na mitigação desse ruído

sensorial. Consequentemente, ao realizar o controle sem a assistência da rede neural preditiva, a variável controlada revelou uma expressiva presença de ruído, como explicitado na Figura 29. Por outro lado, ao considerar a predição de estados, conforme ilustrado na Figura 30, nota-se a ausência de ruído. Embora não seja parte central do trabalho, essa evidência sugere um processo de aprendizado implícito na capacidade da rede preditiva de realizar uma filtragem eficiente nos dados sensoriais adquiridos, visto que o sensor utilizado apresentou uma alta variação nas medições realizadas em repouso.

Figura 30 - Resposta à Senoide Com a Predição de Estados



Fonte: Autor.

A utilização do desconto temporal proporcionou ao Controlador Preditivo, uma capacidade aprimorada de antecipar eventos, resultando em um desempenho superior.

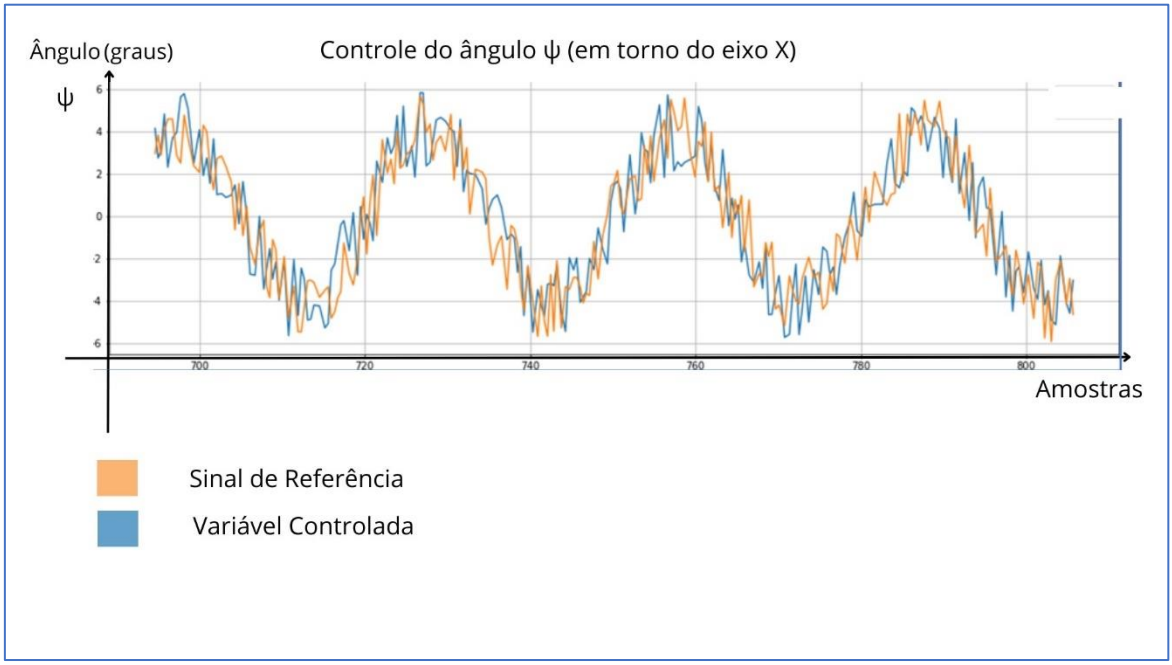
A representação visual dessas duas abordagens, evidenciada nas figuras apresentadas, destaca a diferença significativa na capacidade de seguir a senoide entre os dois modelos. O Controlador Preditivo, ao integrar informações preditas durante o treinamento, demonstrou uma resposta mais precisa e suave, alinhando-se de maneira mais próxima à referência proposta. Essa análise reforça a vantagem da

abordagem preditiva na melhoria do desempenho do controlador em cenários dinâmicos.

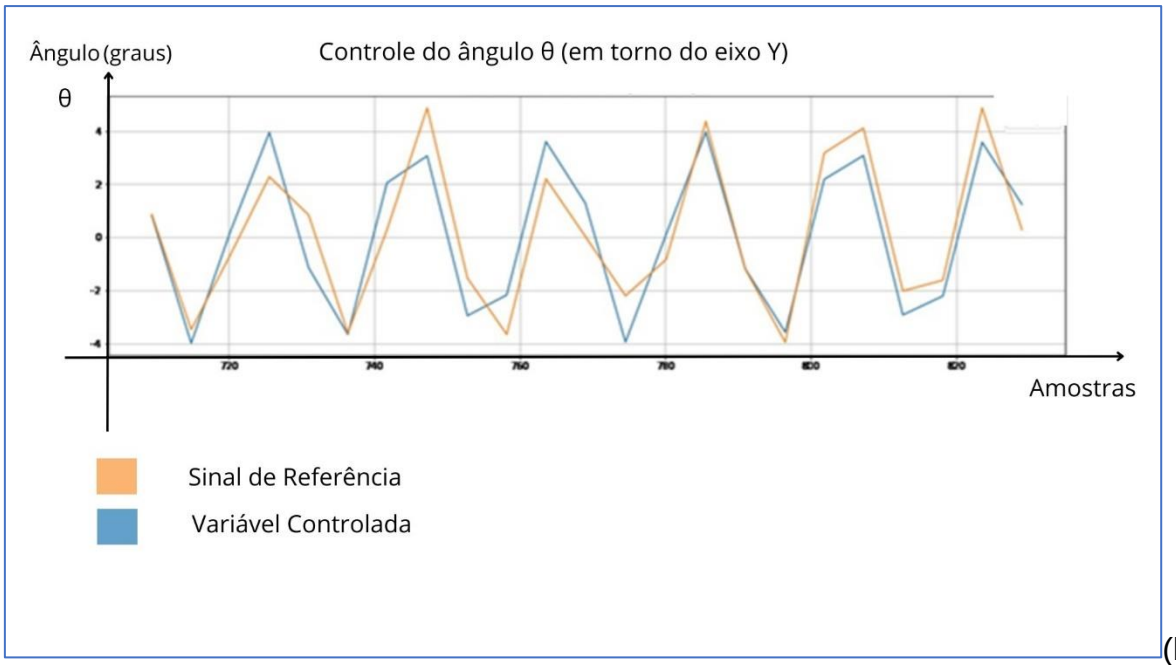
Para o sinal de referência senoidal, exibido na Figura 30, o algoritmo calculou um tempo de acomodação de 3790ms em ψ e 19840ms para θ, portanto, o tempo de acomodação do sistema foi de aproximadamente 20s, considerando o pior caso, o que atende ao critério de desempenho descrito na Tabela 1. O erro entre o sinal de referência e a senoide foi de 5% do sinal de entrada.

Por fim, foi realizado um teste adicional utilizando um sinal de referência em forma de senoide, ao qual foi acrescentado ruído. Este teste visa avaliar a robustez e a capacidade de resposta do Controlador Preditivo em condições mais desafiadoras, simulando cenários do mundo real nos quais o sistema precisa lidar com perturbações ou variações imprevistas. A adição de ruído ao sinal de referência contribui para uma análise mais abrangente da estabilidade e confiabilidade do controlador, fornecendo dados sobre seu desempenho em situações menos ideais. Este teste foi utilizado para levar o controlador e o SERM ao seu limite. O resultado foi exibido graficamente na Figura 31, onde o eixo y representa o ângulo e o eixo x o tempo em milissegundos.

Figura 31 - Resposta à Senoide com Adição de Ruído



(a)



(b)

Fonte: Autor.

Durante o teste com o sinal de referência contendo ruído, o sistema demonstrou conseguir acompanhar a variação do sinal mesmo diante das perturbações introduzidas. No entanto, foi observado um aumento na temperatura dos

atuadores. Esse fenômeno é atribuído à natureza do teste, que exigiu respostas mais dinâmicas e, consequentemente, maior esforço dos atuadores para acompanhar as variações rápidas do sinal de referência. A elevação na temperatura dos atuadores é um aspecto relevante a ser considerado, e medidas adicionais podem ser implementadas para mitigar ou monitorar esse efeito durante o funcionamento prolongado do sistema em condições similares. Essa observação ressalta a importância de avaliar não apenas o desempenho em termos de controle, mas também os impactos físicos e térmicos nas partes mecânicas do robô ao empregar estratégias mais intensivas em termos de energia.

Tabela 10 – Resultado do Controlador

Resultado Obtido pela Rede Preditiva				
Critério	**Unidade**	**Desempenho obtido**	**Faixa que atende ao critério**	**Critério atendido?**
Tempo de subida para o degrau unitário	S	0,75	<=1	Sim
Tempo de acomodação para o degrau unitário	S	0,95	<=1,5	Sim
Erro para resposta ao degrau unitário	%	4	<=5	Sim
Erro para resposta à senoide	%	5	<=7	Sim
Tempo de acomodação para entrada senoidal	s	20	60	Sim

4.3 Considerações Finais

Ao explorar diferentes arquiteturas de rede e estratégias de treinamento, foi possível otimizar o desempenho do modelo preditivo, destacando a importância de considerar informações previstas no treinamento do controlador. A utilização do desconto temporal revelou-se uma técnica valiosa, conferindo ao controlador preditivo a capacidade aprimorada de antecipar eventos e melhorar sua resposta diante de perturbações no ambiente, devido aos seguintes resultados:

- **Desempenho Satisfatório em Treinamento**

A rede conseguiu aprender padrões sutis nos dados de treinamento e se adaptar eficazmente às nuances presentes. Isso foi evidenciado por uma rápida convergência durante o treinamento, alcançando o desempenho desejado.

- **Validação nos dados de Teste**

  Durante o processo de desenvolvimento do modelo preditivo, foi observado um desafio significativo relacionado ao overfitting. O modelo inicial, uma 'Rede Estreita e Profunda', mostrou-se propenso a overfitting devido à complexidade dos dados. Esta arquitetura profunda, embora fosse capaz de capturar as nuances dos dados de treinamento, acabou se ajustando excessivamente a esses dados específicos. Como resultado, quando novos dados foram introduzidos, o modelo não conseguiu generalizar eficazmente, levando a previsões imprecisas.

  A abordagem de afunilar as camadas, associado à técnica de *dropout*, não apenas resolveu o problema do overfitting, mas também permitiu ao modelo manter um equilíbrio entre a complexidade dos dados e a capacidade de generalização para novos conjuntos de dados, resultando em previsões mais precisas e confiáveis.

- **Generalização e Precisão no Controle**

  Durante testes, o controlador preditivo continuou a superar as expectativas, demonstrando excelente capacidade de generalização para dados não observados anteriormente. Isso sugere que a rede está efetivamente capturando os padrões subjacentes e não apenas memorizando os dados de treinamento. O afunilamento das camadas, reduzindo gradualmente o número de neurônios das camadas de entrada para as camadas de saída, permitiu que a rede neural se ajustasse de maneira mais precisa às características específicas dos dados, evitando o overfitting e aumentando a generalização para novos dados.

O código-fonte dos algoritmos, tanto do SERM quanto das redes neurais, juntamente com os conjuntos de dados de teste e treinamento, imagens, vídeos,

modelos 3D e outros recursos relacionados, foram disponibilizados publicamente no repositório no GitHub do perfil “ArthurCarlosF”, através do endereço “https://github.com/ArthurCarlosF/RepositorioTcc2”. Essa iniciativa visa promover a transparência e a replicabilidade da pesquisa, permitindo que outros pesquisadores e entusiastas da área possam acessar, explorar e utilizar os recursos desenvolvidos neste trabalho. A disponibilidade desses materiais no GitHub não apenas compartilha os resultados obtidos, mas também fomenta a colaboração e a disseminação do conhecimento no âmbito da robótica e inteligência artificial.

5 CONCLUSÃO

O presente trabalho buscou avaliar a implementação e análise de uma abordagem preditiva e adaptativa baseada em redes neurais para um Sistema Embarcado em Robótica Móvel (SERM). A partir do desenvolvimento e treinamento de uma rede neural preditiva, foi possível antecipar eventos futuros, proporcionando ao controlador uma capacidade aprimorada de resposta. A utilização do desconto temporal revelou-se uma estratégia eficaz para otimizar o desempenho do Controlador Preditivo, resultando em melhorias significativas em relação ao modelo convencional.

Os testes realizados, incluindo respostas a degrau, senoides e perturbações, ofereceram dados valiosos sobre o comportamento do sistema proposto. A capacidade do modelo em seguir referências, mesmo em presença de ruídos, destaca a sua aplicabilidade em cenários dinâmicos e imprevisíveis. A observação do aquecimento dos atuadores durante operações prolongadas fornece informações cruciais para futuras considerações de segurança e eficiência operacional.

A contribuição deste trabalho vai além da simples implementação de modelos; ele oferece uma abordagem integrada que une previsão e controle, explorando o potencial das redes neurais em ambientes robóticos. O sucesso das estratégias propostas abre caminho para futuras pesquisas na otimização contínua dos algoritmos, considerando diferentes arquiteturas de rede, técnicas de regularização e aprimoramentos na dinâmica do sistema.

Em síntese, a abordagem preditiva e controladora baseada em redes neurais apresentada neste trabalho demonstra promissoras perspectivas para o avanço em novos controladores, destacando-se pela capacidade de adaptação, aprendizado contínuo e eficiência operacional em ambientes dinâmicos e desafiadores.

Em adição, uma direção promissora para futuras pesquisas é a implementação do sistema proposto em um ambiente completamente embarcado. Ao transferir o modelo preditivo e o controlador para um *“raspberry”* ou hardware similar, seria possível explorar ainda mais as capacidades do SERM em cenários mais práticos. A integração de sensores e atuadores diretamente no sistema do controlador, sem a necessidade

de um computador, permitiria uma operação com maior mobilidade e velocidade, aumentando a robustez e a eficácia do controle em situações de alta dinâmica. Ademais, considerações sobre eficiência energética e requisitos de processamento devem ser levadas em conta para garantir a viabilidade do sistema embarcado em aplicações reais. Portanto, sugere-se que trabalhos futuros sejam voltados para a adaptação e otimização do modelo para implementação embarcada, visando a sua aplicação em uma ampla gama de aplicações robóticas.

6 REFERÊNCIAS BIBLIOGRÁFICAS

Araújo, Ricardo. **Proposta de uma Classe de Perceptrons Híbridos com Aprendizagem baseada em Gradiente Descendente**. Programa de Pós-Graduação em Ciência da Computação, Universidade Federal de Pernambuco, 2012. Disponível em: https://repositorio.ufpe.br/bitstream/123456789/2842/1/arquivo9424_1.pdf. Acesso em: 15/11/2022.

Aurélio, Marco. **Algoritmos Genéticos: Princípios e Aplicações**. ICA: Laboratório de Inteligência Computacional Aplicada. ICA: Laboratório de Inteligência Computacional Aplicada, 1999. Disponível em: http://www.inf.ufsc.br/~mauro.roisenberg/ine5377/Cursos-ICA/MQ-intro_apost.pdf. Acesso em: 15/11/2022.

Castro, Tomás F. **Simulação de Robôs Quadrúpedes Utilizando o SimMechanics**. Repositório Científico do Instituto Politécnico do Porto, 2012. Disponível em: https://recipp.ipp.pt/handle/10400.22/7168. Acesso em: 15/11/2022.

Contarini, Gabriel. **Redes neurais LSTM e modelo GARCH: uma abordagem conjunta para previsão de retornos**. Instituto de Economia, Universidade Federal do Rio de Janeiro, 2020. Disponível em: https://pantheon.ufrj.br/handle/11422/14800. Acesso em: 05/10/2023.

Costa, Artur T. V. C. **Comparação de Métodos de Regularização no Treinamento de Tedes Neurais Artificiais Aplicado a Uma Torre de Resfriamento de Água**. Lume, 2023. Disponível em: https://www.lume.ufrgs.br/handle/10183/25581. Acesso em: 06/10/2023.

Faria, Arthur C. **Rede Neural Aplicada: Modelagem e Controle em Robótica Móvel**. Git Hub, 2023. Disponível em: https://github.com/ArthurCarlosF/RepositorioTcc2. Acesso em: 28/11/2023.

Guimarães, Rayane A. et al. **Controle Preditivo Baseado em Modelo para Conversores Formadores de Rede com Operação Ilhada**. Scholar, 2019.

Disponível em: https://proceedings.science/proceedings/100113. Acesso em: 14/09/2023.

Haykin, Simon. **Redes Neurais, Princípios e Práticas**. 2° Edição. Books: bookman, 2001.

Lage, Eduardo. **Ângulos de Euler**. Revista Elementar, 2020. Disponível em: https://rce.casadasciencias.org/rceapp/static/docs/artigos/2020-043.pdf. Acesso em: 05/02/2024.

Ioannou, P. A.; Sun, J. **Robust Adaptive Control**. IEEE, 1996. Disponível em: https://ieeexplore.ieee.org/document/4171550 . Acesso em: 14/09/2023.

Kovács, Zsolt. **Redes Neurais Artificiais**. 4° Edição. São Paulo: Livraria da Física, 2006.

Monteiro, Sildomar; Ribeiro, Carlos. **Desempenho de algoritmos de aprendizagem por reforço sob condições de ambiguidade sensorial em robótica móvel**. Scielo Brasil, 2004. Disponível em: https://www.scielo.br/j/ca/a/3YPYK8P6ff97tgJwJn8xhXs/. Acesso em: 25/06/2022.

Oliveira, Josenalde. **Controle Adaptativo Indireto por Modelo de Referência e Estrutura Variável**. Universidade Federal do Rio Grande do Norte, 2003. Disponível em: https://repositorio.ufrn.br/bitstream/123456789/27034/1/josenaldeBO2003.pdf. Acesso em: 25/06/2022.

Oliveira, Leandro et al. **As Redes Neurais Aplicadas à Previsão de Corrente Elétrica de um Sistema Fotovoltaico**. Revista FSA, 2022. Disponível em: https://web.s.ebscohost.com. Acesso em: 06/10/2023.

Oliveira, Prabhát K. et al. **Classificação de Eletrocardiograma utilizando Redes Neurais**. Doity, 2017. Disponível em: https://doity.com.br/media/doity/submissoes. Acesso em: 14/09/2023.

Osório, Fernando; Heinen, Milton. **Controle Inteligente do Caminhar de Robôs Móveis Utilizando Algoritmos Genéticos e Redes Neurais Artificiais**. ENIA, 2007. Disponível em: https://www.cos.ufrj.br/~ines/enia07_html/pdf/27434.pdf. Acesso em: 15/11/2022.

Pan, Yunpeng; Wang, Jun. ***Predictive Control of Unknown Nonlinear Dynamical Systems Based on Recurrent Neural Networks***. IEEE, 2012. Disponível em: https://ieeexplore.ieee.org/abstract/document/6029334. Acesso em: 15/11/2022.

Pereira, Robertson da Silva. **Redes Heterogêneas para Classificação de Produtos em Notas Fiscais Eletrônicas de Compras Públicas**. SBC OPEN LIB, 2020. Disponível em: https://repositorio.cgu.gov.br/bitstream/1/64722/3/TCC_Roberston.pdf. Acesso em: 15/11/2023.

Rauber, Thomas. **Redes Neurais Artificiais**. Research Gate , 2005. Disponível em: https://www.researchgate.net/profile/Thomas-Rauber-2. Acesso em: 15/11/2022.

Reis, Carlos H. **Otimização de Hiperparâmetros em Redes Neurais Profundas**. Git Hub, 2018. Disponível em: https://carlos-henreis.github.io/files/Monografia_TFG.pdf. Acesso em: 14/09/2023.

Ribeiro, Eduardo da Silva. **LDMAT: Função de Custo Baseada em Matrizes de Distâncias para o Treinamento de Redes Neurais Convolucionais**. Repositório Institucional da UFMG, 2021. Disponível em: https://repositorio.ufmg.br/bitstream/1843/39773/4/teseEduardoDaSilvaRibeiroPDFA.pdf. Acesso em: 15/11/2022.

Santos, Ana P. et al. **Aplicação de Representações Em Blocos Interconectados em Identificação Caixa-Cinza de Sistemas Dinâmicos Não Lineares**. Research Gate, 2010. Disponível em: https://www.researchgate.net/profile/Luciana-Margoti. Acesso em: 15/11/2022.

Santos, Daniel. **Spider robot, quad robot, quadruped**. Thingiverse, 2018. Disponível em: https://www.thingiverse.com/thing:2796820/files. Acesso em: 25/06/2022.

Souto, Rafael**. Modelagem Cinemática de um Robô Quadrúpede e Geração de Seus Movimentos Usando Filtragem Estocástica**. ENE UnB, 2007. Disponível em: http://www2.ene.unb.br/gaborges/arquivos/pf.rafael.fontes.2007.1.pdf. Acesso em: 25/06/2022.

Strickland, James R**. Um processo, vários threads**. Springer, 2018. Disponível em: https://link.springer.com/chapter/10.1007/978-1-4842-3414-3_8. Acesso em: 20/10/2023.

Torres, Luiz. **Controle adaptativo de sistemas dinâmicos: uma proposta para o relaxamento da condição de excitação persistente**. UNIPÊ, 2021. Disponível em: https://www.editoracientifica.com.br/artigos/controle-adaptativo-de-sistemas-dinamicos-uma-proposta-para-o-relaxamento-da-condicao-de-excitacao-persistente. Acesso em: 15/11/2022.

Wolf, Denis F.; Simões, Eduardo do Valle; Osório, Fernando S.; Junior, Onofre T. **Robótica Móvel Inteligente: Da Simulação às Aplicações no Mundo Real**. XXIV Jornada de Atualização em Informática (JAI), 2009. Disponível em: https://osorio.wait4.org/publications/2009/CL_JAI2009_Completo.pdf. Acesso em: 05/02/2024.