Universidade de Brasília

Faculdade de Ciências e Tecnologias em Engenharia

Máquinas de vetores de suporte para controle de próteses utilizando interfaces miocinéticas e sua implementação em SoC FPGA

Felipe Sidrone da Silva Dionisio

TRABALHO DE CONCLUSÃO DE CURSO

ENGENHARIA ELETRÔNICA

Brasília

2025

Universidade de Brasília
Faculdade de Ciências e Tecnologias em Engenharia

Máquinas de vetores de suporte para controle de próteses utilizando interfaces miocinéticas e sua implementação em SoC FPGA

Felipe Sidrone da Silva Dionisio

Trabalho de Conclusão de Curso submetido ao curso de Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Orientador: Prof. Dr. Daniel Mauricio Muñoz Arboleda
Coorientador: Prof. Dr. Carlos Eduardo da Silva Santos

Brasília
2025

FICHA CATALOGRÁFICA

Sidrone da Silva Dionisio, Felipe.

Máquinas de vetores de suporte para controle de próteses utilizando interfaces miocinéticas e sua implementação em SoC FPGA / Felipe Sidrone da Silva Dionisio; orientador Daniel Mauricio Muñoz Arboleda; coorientador Carlos Eduardo da Silva Santos. -- Brasília, 2025.

87 p.

Trabalho de Conclusão de Curso (Engenharia Eletrônica) -- Universidade de Brasília, 2025.

1. SVM. 2. Aprendizado de máquina. 3. Otimização multiobjetivo. 4. Seleção de variáveis. 5. Dataset.I. Mauricio Muñoz Arboleda, Daniel, orient. II. Eduardo da Silva Santos, Carlos, coorient. III. Título.

Universidade de Brasília
Faculdade de Ciências e Tecnologias em Engenharia

Máquinas de vetores de suporte para controle de próteses utilizando interfaces miocinéticas e sua implementação em SoC FPGA

Felipe Sidrone da Silva Dionisio

Trabalho de Conclusão de Curso submetido ao curso de Engenharia Eletrônica da Universidade de Brasília, como requisito parcial para obtenção do grau de Bacharel em Engenharia Eletrônica.

Trabalho aprovado. Brasília, 12 de dezembro de 2025:

Prof. Dr. Daniel Mauricio Muñoz Arboleda,
UnB/FCTE
Orientador

Prof. Dr. Gilmar Silva Beserra,
UnB/FCTE
Examinador interno

Prof. Dr. Guillermo Alvarez Bestard,
UnB/FCTE
Examinador interno

Agradecimentos

Acima de tudo, agradeço a Deus e à Virgem Maria, por toda saúde, inteligência, resiliência e tantas oportunidades. Agradeço aos meus pais, Cláudia e Alcino, que abriram mão de seus sonhos para que eu pudesse alcançar os meus. Obrigado por todo cuidado, carinho e pelo suporte que me possibilitou conquistas inimagináveis.

Demonstro minha eterna gratidão a quem eu carinhosamente chamo de "Dona Lúcia", minha segunda mãe e uma das pessoas responsáveis pelo meu desenvolvimento durante a infância e adolescência, e que hoje permanece cuidando de mim junto a Deus.

Agradeço aos meus avós: Ana, Joaquim, Maria e Quintino. Agradeço à minha família pelo café quente de todo fim de tarde, seguido de palavras de incentivo que sempre acompanharam a minha trajetória profissional. Um agradecimento especial ao meu pequeno afilhado Heitor, que renova minhas energias com um abraço apertado ao fim do dia.

Sou grato por todos os amigos que fiz durante a trajetória acadêmica. Além de compartilharmos sinceras risadas, a excelência de cada um de vocês sempre me serviu de inspiração. Muito obrigado, Matheus Luiz, Pedro Tristão, Lucas Marques, Erick Rollemberg, Gabriel Capparelli e Enrique Unternbaumen.

A presença dos meus grandes amigos sempre trouxe leveza à minha jornada, celebrando comigo cada pequena conquista. Kayo Felipe, João Pedro, Enzo Brenner, Luana Araújo e Mariana Rezende, obrigado por tornarem cada fase da vida mais leve.

Em especial, agradeço ao amor da minha vida, Thamires Rodrigues, uma pessoa única que Deus me presenteou com a companhia diária. Obrigado por nunca me deixar pensar em desistir, pelo cuidado e por todo o incentivo incondicional.

Por fim, agradeço a todos os profissionais que se dedicam ao desenvolvimento e à viabilização das interfaces miocinéticas, em especial aos pesquisadores do Laboratório de Microeletrônica da Universidade de Brasília. Manifesto minha gratidão ao meu orientador, Prof. Dr. Daniel Mauricio Muñoz Arboleda; ao meu coorientador, Prof. Dr. Carlos Eduardo da Silva Santos; e ao meu colega de trabalho, Prof. Paulo de Carvalho.

Resumo

O desenvolvimento de algoritmos e sistemas embarcados para aplicações de Interface Humano-Máquina voltadas ao controle de próteses apresenta desafios significativos, devido a restrições de tempo de resposta, capacidade de processamento, consumo de energia, disponibilidade de recursos, tamanho e peso. Buscando atender a esses requisitos com elevada eficiência computacional, este trabalho propõe a implementação de Regressores de Vetores de Suporte (SVR), utilizando *kernel* linear e *kernel* RBF, em um SoC FPGA para estimar o deslocamento de ímãs em aplicações de interface miocinética.

Para o treinamento dos modelos de aprendizado de máquina, foi construída uma base de dados experimental contendo informações sobre o campo magnético associado a cada posição dos ímãs, de modo a gerar modelos capazes de localizá-los em tempo de execução. A complexidade da solução é reduzida por meio de uma análise estatística que identifica as variáveis mais relevantes para cada regressor, permitindo minimizar simultaneamente o erro empírico e o custo computacional. Além disso, a comunicação entre a FPGA, onde os regressores estão implementados, e o microcontrolador responsável pela leitura dos sensores é realizada via protocolo SPI, garantindo alta velocidade e paralelismo na aquisição dos dados.

Os resultados obtidos demonstram como as características dos diferentes conjuntos de dados impactam o treinamento dos modelos, além de evidenciar o potencial dos SVRs implementados em hardware para estimar, em tempo real, o deslocamento de ímãs, configurando-se como uma alternativa precisa e eficiente para o controle de próteses por meio de interfaces miocinéticas.

Palavras-chave: SVM. Aprendizado de máquina. Otimização multiobjetivo. Seleção de variáveis. Dataset.

Abstract

The development of algorithms and embedded systems for Human–Machine Interface applications aimed at prosthesis control presents significant challenges due to constraints on response time, processing capability, energy consumption, resource availability, size, and weight. To meet these requirements with high computational efficiency, this work proposes the implementation of Support Vector Regressors (SVR), using linear kernel and RBF kernel, on an SoC FPGA to estimate magnet displacement in myokinetic interface applications.

For training the machine learning models, an experimental dataset was constructed containing information on the magnetic field associated with each magnet position, enabling the creation of models capable of localizing them during runtime. The complexity of the solution is reduced through a statistical analysis that identifies the most relevant variables for each regressor, allowing the simultaneous minimization of empirical error and computational cost. In addition, communication between the FPGA, where the regressors are implemented, and the microcontroller responsible for reading the sensors is performed via the SPI protocol, ensuring high speed and parallelism in data acquisition.

The results demonstrate how the characteristics of different datasets influence model training and highlight the potential of hardware-implemented SVRs for estimating magnet displacement in real time, emerging as a precise and efficient alternative for prosthesis control through myokinetic interfaces.

Keywords: SVM. Machine learning. Multi-objective optimization. Feature selection. Dataset.

Lista de figuras

Figura 2.1 Ilustração dos componentes de uma interface miocinética. *Fonte: Retirado de (Tarantino et al., 2017)* 23
Figura 2.2 Conjunto de dados linearmente separável. Retirado de (Santos, 2019) . . 25
Figura 2.3 Transformação realizada pela função de *kernel*. Retirado de (Santos, 2019) 26
Figura 2.4 Na esquerda, um modelo de SVR com margens de tolerância ε definida pelas linhas tracejadas, a função do regressor $\hat{f}$, os dados x_i. Na direita, a função de perda. Retirado de (Santos, 2019) 28
Figura 2.5 Arquitetura genérica em uma FPGA. Fonte: Retirado de (Pedroni, 2004) 31
Figura 2.6 Arquitetura de um SoC FPGA. Retirado de (AMD, 2025c) 32
Figura 3.1 Fluxograma do processo de emulação de uma interface miocinética. Representação da entrada de dados pelo usuário, processamento e conversão desses parâmetros em sinal PWM para controle dos motores. 37
Figura 3.2 Fluxograma do processamento dos dados para estimar a posição dos ímãs e controlar a mão robótica. Apresenta a comunicação das unidades de aquisição com FPGA e a comunicação da FPGA com a mão robótica. . . 38
Figura 3.3 Interface miocinética com estrutura em formato de antebraço e placas de aquisição de dados na parte externa. 39
Figura 3.4 Interior da interface miocinética, apresentando os ímãs (marcadores magnéticos) e os servomotores responsáveis por emular o movimento muscular. 40
Figura 3.5 Placa de controle do protótipo da interface miocinética, responsáveis por receber os parâmetros do movimento e converter em sinais de modulação por largura de pulso. 41
Figura 3.6 À esquerda, as duas metades da plataforma de testes separadas; à direita, as duas metades parafusadas, configuração utilizada na realização dos testes. 41
Figura 3.7 Figura editada para ilustração da plataforma e mão robótica aplicadas em conjunto. 42
Figura 3.8 Movimento do ímã com incrementos constantes. 44
Figura 3.9 Movimento do ímã com incrementos variáveis. 45
Figura 3.10 Arquitetura da implementação de SVR em hardware 46
Figura 3.11 Arquitetura SVR *kernel* Linear 47
Figura 3.12 Arquitetura SVR *kernel* RBF 48
Figura 3.13 Ambiente Gráfico NIOTS. Retirado de (Santos, 2019) 49
Figura 4.1 Tela inicial do aplicativo 53

Figura 4.2 Tela de movimentos personalizados 54
Figura 4.3 Tela de movimentos padronizados 55
Figura 4.4 Diagrama de blocos para implementação dos módulos SPI em VHDL. . 57
Figura 4.5 Conexão física entre a FPGA e as placas de aquisição como parte da implementação do protocolo SPI. 58
Figura 4.6 Configuração do ILA e demonstração de uma coleta de dados 59
Figura 4.7 Análise de timing do módulo SPI. 59
Figura 4.8 Consumo de potência módulo SPI implementado em conjunto com o ILA. 60
Figura 4.9 Consumo de potência de cada módulo SPI individualmente e do ILA. . 60
Figura 4.10 Exemplo de conjunto de treinamento para o ímã 1. 63
Figura 4.11 Exemplo de conjunto de validação para o ímã 1. 64
Figura 4.12 Configuração do NIOTS para treinamento dos modelos. 65
Figura 4.13 Exemplo de resultados do treinamento apresentado pelo NIOTS. 66
Figura 4.14 Resultados do treinamento com 144 entradas para os regressores 1 ao 4. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF. . 67
Figura 4.15 Resultados do treinamento com 144 entradas para os regressores 5 ao 8. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF. . 67
Figura 4.16 Resultados do treinamento com 10 *features* para os regressores 1 ao 4. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF. 68
Figura 4.17 Resultados do treinamento com 10 *features* para os regressores 1 ao 8. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF. 68
Figura 4.18 Layout da implementação SVR linear. Vermelho: FIFO e o recurso de seleção de features; Amarelo: SVR para o ímã 1; Verde: SVR para o ímã 2; Rosa: SVR para o ímã 3; Azul: SVR para o ímã 4; Ciano: SVR para o ímã 5; Cinza: ocupação do ILA-core. 72
Figura 4.19 Simulação de 5 regressores de vetores de suporte com *kernel* linear . . . 73
Figura 4.20 Relatório de consumo energia total para o *Kernel* linear. 74
Figura 4.21 Relatório de consumo de energia discriminado por módulos 74
Figura 4.22 Análise de timing - Clock em 100 MHz 75
Figura 4.23 Comparação entre o valor real e a saída de cada regressor. Em vermelho a saída real, em preto o valor calculado pelo MATLAB (64 bits) e em azul o valor calculado pelo FPGA (27 bits) 76

Figura 4.24 Layout da implementação SVR RBF. Vermelho: FIFO e o recurso de seleção de features; Amarelo: SVR para o ímã 1; Verde: SVR para o ímã 2; Rosa: SVR para o ímã 3; Azul: SVR para o ímã 4; Ciano: SVR para o ímã 5; Cinza: ocupação do ILA-core. 77
Figura 4.25 Simulação de 5 regressores de vetores de suporte com *kernel* RBF com marcadores de tempo nas bordas de subida dos sinais de *ready*, que indicam uma saída válida. 78
Figura 4.26 Relatório de consumo energia total para o *kernel* RBF 79
Figura 4.27 Relatório de consumo de energia discriminado por módulos para a implementação com *kernel* RBF 79
Figura 4.28 Análise de timing para regressor com *kernel* RBF - Clock em 100 MHz . 80
Figura 4.29 Comparação entre o valor real e a saída de cada regressor RBF. Em vermelho a saída real, em preto o valor calculado pelo FPGA (27 bits) . . . 80

Lista de tabelas

Tabela 2.1 Comparação entre diferentes tecnologias de interface Humano-Máquina para controle de próteses. Adaptado de (Mendez *et al.*, 2022) 22
Tabela 2.2 Comparação entre bases de dados utilizadas para treinar e validar modelos de aprendizado de máquina para estimar deslocamento de ímãs. 34
Tabela 2.3 Comparação entre diferentes implementações de modelos de aprendizado de máquina para estimar deslocamento de ímãs. 35
Tabela 3.1 Exemplo de posição do ímã para um movimento de 2 Hz, amplitude de 5 mm e $\Delta t = 0.05$ s. 44
Tabela 3.2 Posição do ímã para A = 5 mm, $f = 2$ Hz e $\Delta t = 0{,}05$ s. 45
Tabela 4.1 Resultado de MSE e quantidade de vetores de suporte para 144 entradas e 10 entradas com kernel linear. 69
Tabela 4.2 Resultado de MSE e quantidade de vetores de suporte para 144 entradas e 10 entradas com kernel RBF. 69
Tabela 4.3 Resultado de MSE e quantidade de vetores de suporte para 384 entradas e 10 entradas com kernel linear. 70
Tabela 4.4 Resultado de MSE e quantidade de vetores de suporte para 384 entradas e 10 entradas com kernel RBF. 70
Tabela 4.5 Ocupação de recursos de cada módulo dos regressores de vetores de suporte com *Kernel* linear implementados em um SoC FPGA Zynq 7020 da AMD-Xilinx. 72
Tabela 4.6 Diferença entre o valor calculado por cada regressor implementado no Vivado e em MATLAB. Esse diferença foi calculada para uma amostra do conjunto de dados 75
Tabela 4.7 MSE entre o valor real e o valor calculado pelo regressor. 75
Tabela 4.8 Ocupação de recursos de cada módulo dos regressores de vetores de suporte com *Kernel* RBF implementados em um SoC FPGA Zynq 7020 da AMD-Xilinx. 77
Tabela 4.9 MSE entre o valor real e o valor calculado pelo regressor para o *kernel* RBF. 80
Tabela 4.10 Comparação entre diferentes implementações de modelos de aprendizado de máquina para estimar deslocamento de ímãs, incluindo os resultados obtidos neste trabalho. 81

Lista de abreviaturas e siglas

AXI	*Advanced eXtensible Interface*
BLE	*Bluetooth Low Energy*
CLB	*Configurable Logic Block*
CORDIC	*COordinate Rotation Digital Computer*
DE	*Differential Evolution*
DSP	*Digital Signal Processor*
EMG	*Eletromiografia*
FPGA	*Field Programmable Gate Array*
HDL	*Hardware Description Language*
IMES	*Implantable Myoelectric Sensors*
LMA	*Levenberg–Marquardt algorithm*
LUT	*Look-Up Table*
ML	*Machine Learning*
MLP	*Multilayer Perceptron*
MLPS	*Magnet localizations per second*
MMs	*Magnetic Markes*
MODE	*Multi-Objective Differential Evolution*
MSE	*Mean Squared Error*
NIOTS	*Nature-Inspired Optimization Tools for SVM*
PCA	*Principal Component Analysis*
PLL	*Phase-Locked Loop*
RBF	*Radial-Basis Function*
RBFNN	*Radial-Basis Function Neural Network*
RTL	*Register-Transfer Level*
SMO	*Sequential Minimal Optimization*
SoC	*System-on-Chip*
SQUIDs	*Superconducting Quantum Interference Devices*
SVD	*Singular Value Decomposition*
SVM	*Suport Vector Machine*
SVR	*Support Vector Regression*

Sumário

1 Introdução 14
1.1 Descrição do problema 14
1.2 Perguntas de pesquisa 17
1.3 Justificativa 18
1.4 Objetivos 19
1.4.1 Objetivo geral 19
1.4.2 Objetivos específicos 19
1.5 Organização do documento 19

2 Fundamentação teórica 21
2.1 Interface miocinética para controle de próteses 21
2.1.1 Comparação entre as diferentes técnicas estudadas para implementação de interface humano-máquina 21
2.1.2 Interface Miocinética 22
2.2 Máquina de Vetores de Suporte 23
2.2.1 Diferenças de Máquinas de Vetores de Suporte como classificadores e regressores 25
2.2.2 Funções de *kernel* 25
2.2.3 Treinamento das SVMs 27
2.2.4 Função preditora 28
2.3 Análise de Componentes Principais 28
2.4 Seleção univariada de variáveis 29
2.5 Hardware reconfigurável 30
2.5.1 FPGA 30
2.5.2 SoC FPGA 32
2.5.3 Analisador lógico integrado 32
2.5.4 Fluxo de projeto 33
2.6 Estado da arte 34
2.7 Conclusão do capítulo 36

3 Metodologia 37
3.1 Proposta geral 37
3.2 Desenvolvimento da Interface Miocinética 39
3.3 Modelagem dos movimentos dos servomotores 43
3.3.1 Movimento de rampa 43
3.3.2 Movimento senoidal 44
3.4 Arquitetura da implementação de SVR em hardware 46

3.4.1 Arquitetura de SVR com *kernel* linear . . . 47
3.4.2 Arquitetura de SVR com *kernel* RBF . . . 48
3.5 Ferramentas auxiliares . . . 48
3.5.1 NIOTS . . . 49
3.5.2 Otimização multiobjetivo . . . 50
3.5.3 Bibliotecas para operações ariméticas em ponto flutuante e cálculo de funções transcendentais . . . 51
3.6 Conclusão do capítulo . . . 51

4 Resultados . . . 52
4.1 Construção da base de dados para treinamento e validação dos modelos . . . 52
4.1.1 Interface gráfica para controle da interface miocinética . . . 53
4.1.2 Firmware da unidade de controle . . . 55
4.1.3 Implementação do protocolo SPI entre FPGA e unidades de aquisição 56
4.1.4 Dataset gerado a partir dos dados de variação de campo magnético . 61
4.2 Treinamento dos modelos SVR . . . 65
4.2.1 Treinamento dos modelos sem seleção de *features* . . . 66
4.2.2 Treinamento dos modelos aplicando seleção de *features* . . . 67
4.2.3 Treinamento com o dataset produzido na *Scuola Superiore Sant'Anna* 69
4.3 Implementação dos modelos SVR em FPGA . . . 70
4.3.1 Implementação de SVR com *kernel* linear . . . 71
4.3.2 Implementação de SVR com *kernel* RBF . . . 76
4.4 Comparação dos resultados obtidos com trabalhos correlatos . . . 81

5 Conclusões . . . 82
5.0.1 Trabalhos futuros . . . 83

Referências . . . 85

1 Introdução

1.1 Descrição do problema

A amputação de membros superiores compromete significativamente a capacidade de um indivíduo de realizar atividades essenciais do dia a dia. Restaurar as habilidades motoras de controle da mão humana, seja para realizar gestos, manipular objetos ou executar tarefas mais complexas, representa um dos maiores desafios para a engenharia de reabilitação e neurociência aplicada. O êxito neste desafio está condicionado a dois elementos essenciais:

a) Uma mão artificial que reproduza movimentos semelhantes aos movimentos naturais da mão humana;

b) Uma interface Humano-Máquina que capture sinais biológicos do usuário, processe e os converta em sinais que o sistema de controle da mão artificial seja capaz de interpretar.

A interface Humano-Máquina realiza a comunicação entre o usuário e a mão artificial, portanto, deve permitir o controle da prótese de forma intuitiva e com o mínimo de esforço por parte da pessoa amputada. Atualmente, as interfaces utilizadas para controle de próteses comerciais empregam a técnica de eletromiografia (EMG), devido a sua acessibilidade e à estratégia de controle intuitiva. No entanto, essa tecnologia apresenta limitações quanto à aquisição dos sinais e à viabilidade de controle simultâneo de múltiplos graus de liberdade (Esposito *et al.*, 2021).

Estudos recentes (Tarantino *et al.*, 2017) apresentam uma alternativa ao controle baseado em sinais elétricos, propondo a implantação de ímãs permanentes (MMs, sigla em inglês) para medir o deslocamento muscular, tecnologia denominada Interface Miocinética. O uso de ímãs reduz o consumo de energia, aumenta a imunidade a ruídos e permite o controle independente de vários graus de liberdade, requisitos-chave em aplicações biomédicas.

O controle de próteses por meio da interface miocinética pode ser realizado de duas formas: controle direto ou localização precisa de cada ímã. A primeira abordagem utiliza reconhecimento de padrões de movimento com base na variação do campo magnético, como apresentado em (Moradi *et al.*, 2022) e (Cipriani *et al.*, 2023). Nessa abordagem, os movimentos são divididos em classes conforme os padrões de leitura do campo magnético. A segunda abordagem considera a posição exata de cada ímã, possibilitando estratégias de controle capazes de implementar movimentos contínuos. A localização precisa de cada ímã, por sua vez, depende da resposta à seguinte pergunta: como determinar o deslocamento de um ímã a partir da variação do campo magnético por ele gerado?

Existem duas abordagens principais para estimar esse deslocamento:

a) Resolver o problema inverso da magnetostática, como abordado no estudo apresentado em (Tarantino *et al.*, 2017);

b) Utilizar métodos baseados em dados e modelos de aprendizado de máquina para mapear medições magnéticas experimentais aos deslocamentos dos ímãs (Mendez *et al.*, 2022) e (Mendez *et al.*, 2024).

Nesse contexto, torna-se essencial o desenvolvimento de algoritmos eficientes capazes de estimar, em tempo real, as posições de múltiplos ímãs. A elaboração dessas ferramentas permite uma implementação com equilíbrio adequado entre precisão, exatidão, tempo de processamento, graus de liberdade, consumo de energia e de recursos.

No estudo de (Tarantino *et al.*, 2017), os ímãs foram modelados como dipolos magnéticos pontuais, e sua localização foi realizada por meio do algoritmo de otimização *Levenberg–Marquardt* (LMA) (Kaltenbach, 2022). Embora essa abordagem forneça uma solução viável para o problema inverso da magnetostática, apresenta uma limitação relevante: o algoritmo não possui tempo de execução fixo, o que compromete sua aplicabilidade em sistemas com restrições de tempo real. Adicionalmente, o algoritmo de rastreamento empregado demonstrou-se incapaz de acompanhar a taxa de amostragem dos sensores. Enquanto os sensores eram capazes de registrar aproximadamente 75 amostras por segundo (uma amostra a cada 13 ms), o tempo necessário para localizar quatro ímãs era de 45 ms, tornando o método menos atrativo para aplicações que demandam respostas rápidas.

Em (Clemente *et al.*, 2019), foi desenvolvido um sistema embarcado capaz de rastrear até cinco ímãs em aproximadamente 4 ms, empregando LMA. Foram utilizados 32 sensores de campo magnético, com um consumo de energia total de 980 mW (550 mW para a unidade de aquisição e 430 mW para a unidade computacional). Nesse caso, o tempo necessário para amostrar as leituras dos sensores e enviá-las para a unidade computacional foi cerca 24 ms, limitando a taxa de saída para aproximadamente 29 ms.

Uma arquitetura paralela e modular foi proposta em (Ianniciello; Gherardini; Cipriani, 2024) para reduzir a latência desse sistema. Essa nova abordagem, também utilizando LMA, apresentou um consumo de energia de 600 mW para uma unidade de aquisição até 1.2 W para oito unidades de aquisição, cujo tempo de processamento e rastreamento variam de 153 μs para um ímã até 21.3 ms para 8 ímãs. Esse período é somado ao tempo de envio dos dados, variando de 10,4 ms para uma unidade de aquisição até 25,8 ms para 8 unidades de aquisição.

No estudo apresentado por (Mendez *et al.*, 2022) foram utilizadas FPGAs (*Field Programmable Gate Arrays*) para reduzir o tempo de rastreamentos dos ímãs, unindo o paralelismo proporcionado pelas FPGAs ao paralelismo das Redes Neurais Artificiais. Nesse trabalho foram abordados os modelos de regressão Linear e RBFNN (*Radial-Basis Function Neural Network*). O modelo RBFNN obteve um erro de localização inferior a 720 μm em 95% do tempo, com latência de 12,07 μs, enquanto o modelo linear atingiu um erro máximo de

520 μm com latência de 4,8 μs. O consumo de potência foi de 209 mW para o modelo linear e 545 mW para o RBFNN. Embora o modelo linear tenha apresentado melhor acurácia, o RBFNN demonstrou maior precisão, sendo a repetibilidade dos resultados um fator relevante na aplicação à interface miocinética. No entanto, o modelo RBFNN apresenta desvantagens em relação à maior latência, maior uso de recursos de hardware e maior consumo de energia.

Em (Mendez *et al.*, 2024) foram utilizadas técnicas de Reconfiguração Dinâmica Parcial em Tempo de Execução em um Sistema em Chip (SoC, sigla em inglês) baseado em processador ARM e FPGA, com o objetivo de melhorar a eficiência dos sistemas abordados em (Mendez *et al.*, 2022). Em termos de desempenho e consumo de energia, a arquitetura proposta implementou o Modelo Linear, o Modelo RBFNN e um Modelo *Perceptron Multicamadas* (MLP, sigla em inglês) para estimar os deslocamentos dos ímãs. Em relação ao processamento de dados, os dados dos sensores magnéticos foram processados em um conjunto reduzido de características por meio da Análise de Componentes Principais (PCA, sigla em inglês). Essa implementação foi capaz de rastrear cinco ímãs em aproximadamente 9,5 ms de tempo total de execução. O consumo total de energia foi estimado em 1719 mW, dos quais 1514 mW foram dedicados ao núcleo ARM Cortex-A9, e 162 mW foram dissipados pela lógica programável.

No estudo apresentado em (Mendes, 2024), técnicas de Reconfiguração Dinâmica Parcial com *prefetching* permitiram a execução de um regressor em uma partição simultaneamente à reconfiguração de outra. Essa metodologia foi implementada em FPGA e em um Sistema em Chip (SoC, sigla em inglês) baseado em processador ARM e FPGA, permitindo comparar a eficiência computacional entre os dois cenários. O modelo implementado em SoC FPGA realizou a localização de 5 ímãs em 6.67 ms consumindo 1.635 W de potência, enquanto o modelo implementado em FPGA executou a mesma tarefa em 5.97 ms, dissipando 1.203 W.

O estudo apresentado em (Feitosa, 2023) obteve como resultado o desenvolvimento de placas de aquisição de dados compostas por sensores de efeito *Hall*, responsáveis por captar a variação do campo magnético gerado pelo deslocamento dos ímãs e enviar os dados a um microcontrolador ESP-WROOM-32 por meio do protocolo SPI. Essas placas, denominadas Unidades de Aquisição, serão utilizadas no desenvolvimento desta pesquisa.

O primeiro ensaio clínico utilizando a interface miocinética é apresentado em (Moradi *et al.*, 2022). Nessa abordagem foi empregada uma máquina de estados finitos na qual, à medida que o ímã se desloca, o campo magnético ultrapassa determinados *thresholds*, alternando entre estados. Em cada estado é executado um gesto específico, tal como abrir a mão, fechá-la ou mantê-la em repouso. Essa abordagem foi um importante avanço para o desenvolvimento da tecnologia, contudo apresentou limitações de movimento por não ser capaz de rastrear a posição dos ímãs.

Um novo ensaio clínico apresentado em (Cipriani *et al.*, 2023) exibiu uma abordagem utilizando um classificador de reconhecimento de padrões com três classes (abertura da mão, fechamento da mão e repouso), utilizando Máquina de Vetores de Suporte (SVM, sigla em inglês) Lineares. Esse estudo apresentou taxa de classificação de 100% com um tempo de execução entre 1.25s e 3s, incluindo o tempo para alcançar a posição desejada e um tempo de espera de 1s. O mesmo estudo implementou cinco classes distintas, apresentando taxa de classificação de 77% para um tempo de 4,05s e 98% para um tempo de 6,35s. O hardware utilizado é descrito em (Ianniciello; Gherardini; Cipriani, 2024). A unidade de processamento utiliza um processador ARM Cortex-M7 600 MHz, enquanto a transmissão dos dados necessita de aproximadamente 23,6 ms utilizando o protocolo I2C.

Reunindo os principais avanços apresentados nos trabalhos anteriores, o presente estudo propõe a implementação de Máquinas de Vetores de Suporte (SVM, sigla em inglês) como regressores em FPGA. O objetivo desta aplicação é estimar o deslocamento de ímãs para aplicações em interfaces miocinéticas em tempo real.

1.2 Perguntas de pesquisa

A motivação deste trabalho baseia-se em três questões fundamentais.

a) Como emular interfaces miocinéticas para simular ímãs implantados em músculos, de forma que o deslocamento desses ímãs represente contrações voluntárias dos músculos de pessoas amputadas, e a variação do campo magnético possa fornecer uma base de dados para o treinamento e a validação de modelos de aprendizagem de máquina?

b) É viável implementar Regressores de Vetores de Suporte em FPGAs para estimar, em tempo real, o deslocamento de ímãs e realizar o controle de próteses por meio de uma interface miocinética?

c) A implementação de SVMs apresenta vantagens em relação a outras abordagens de aprendizado de máquina na estimativa do deslocamento de ímãs? É possível otimizar o desempenho computacional — em termos de tempo de execução e uso de recursos — mantendo a precisão na localização dos ímãs, quando comparado aos trabalhos apresentados em (Mendez *et al.*, 2024) e (Mendez *et al.*, 2022)?

1.3 Justificativa

Para atender à necessidade de uma resposta rápida aos estímulos do usuário, este trabalho propõe a implementação de Regressores de Vetores de Suporte (SVR, sigla em inglês) em hardware. Movimentos convencionais da mão humana ocorrem tipicamente em frequências não superiores a 2 Hz, enquanto indivíduos que executam atividades de alta velocidade, como pianistas, podem atingir frequências máximas de até 10,5 Hz (Mendez, 2021). Dessa forma, a taxa de aquisição dos sensores, que estabelece os requisitos de tempo real do sistema, sendo igual ou superior a 41 Hz, é suficiente para representar fielmente o movimento humano, de acordo com o critério estabelecido pelo Teorema da Amostragem de Nyquist–Shannon.

A demanda por cálculos rápidos e paralelismo justifica o uso de FPGAs que, além de realizarem operações em velocidades superiores às de processadores convencionais, permitem a entrada paralela de dados enviados por múltiplas placas de aquisição e a execução simultânea de diversos regressores. No trabalho apresentado em (Ianniciello; Gherardini; Cipriani, 2024), o tempo de transferência de dados da unidade de aquisição para a unidade de processamento, bem como o tempo de execução do algoritmo, varia conforme o número de ímãs monitorados e placas de aquisição utilizadas. Essa limitação é mitigada pela execução paralela proporcionada pelo FPGA.

Aplicações em Interfaces Humano-Máquina requerem modelos de aprendizagem de máquina com alta capacidade de generalização e baixa complexidade computacional, devido às restrições de processamento, tempo e consumo de energia. As Máquinas de Vetores de Suporte se destacam por sua ampla capacidade de generalização e flexibilidade, proporcionada pelo uso de diferentes funções de *kernel* (Cortes; Vapnik, 1995). O *kernel* linear apresenta menor complexidade de implementação, enquanto o *kernel* RBF pode oferecer maior acurácia em algumas aplicações, a depender das características do conjunto de dados. Dessa forma, é possível comparar o desempenho de ambos em diferentes cenários e com marcadores distintos. No trabalho de (Mendez *et al.*, 2024), é demonstrado que alguns ímãs apresentaram menor erro de localização para modelos específicos, ressaltando a importância de empregar diferentes modelos para localizar vários ímãs simultaneamente.

A seleção univariada de variáveis (técnica utilizada para reduzir a quantidade de dados de entrada e simplificar o modelo) será empregada nesta implementação como alternativa à Análise de Componentes Principais (PCA). Os trabalhos de (Mendes, 2024) e (Mendez *et al.*, 2024) utilizaram o *PCA* para redução de dimensionalidade. Apesar de eficaz na redução do número de variáveis, esse método exige que a cada novo vetor de entrada, uma multiplicação com a matriz de componentes principais seja realizada. No caso dos trabalhos citados, a multiplicação resulta em uma matriz de dimensão 384×25, o que representa um custo computacional elevado. Por outro lado, a técnica de seleção de variáveis proposta realiza uma análise estatística *offline*, permitindo a implementação do modelo considerando somente

as variáveis mais relevantes. Dessa forma, o esforço computacional durante a execução do modelo pode ser significativamente reduzido em comparação ao *PCA*.

1.4 Objetivos

1.4.1 Objetivo geral

Implementar Regressores de Vetores de Suporte em um SoC FPGA para estimar o deslocamento de ímãs, de modo a possibilitar o desenvolvimento de técnicas para controle de próteses de mão robótica por meio de interfaces miocinéticas. A comparação dos resultados obtidos com os avanços recentes na área permite a avaliação da viabilidade dessa abordagem em cenários reais.

1.4.2 Objetivos específicos

a) Desenvolver um firmware para controlar servomotores com movimento parametrizável em forma de rampa e senoide, nos quais amplitude e frequência são entradas fornecidas pelo usuário. Os motores deslocam ímãs de forma a reproduzir o comportamento dos ímãs implantados nos músculos do antebraço. Essa etapa tem por objetivo gerar um *dataset* experimental para o treinamento de regressores por vetores de suporte (SVR) e para validar os modelos implementados em hardware;

b) Desenvolver uma interface gráfica para permitir que o usuário forneça os parâmetros para o firmware de controle dos servomotores, permitindo a inserção de estímulos com diferentes frequências, amplitudes e formas de onda.

c) Desenvolver um código em VHDL que implemente o protocolo SPI em um FPGA e implementar um barramento SPI para cada uma das placas, possibilitando a recepção simultânea dos dados provenientes das placas de aquisição;

d) Utilizar os dados coletados para realizar o treinamento dos regressores de vetores de suporte utilizando algoritmos bioinspirados, obtendo os parâmetros para a implementação dos regressores em hardware;

e) Implementar regressores de vetores de suporte com *kernel* Linear e *kernel* RBF em um SoC FPGA para estimar, em tempo real, o deslocamento dos ímãs;

1.5 Organização do documento

Este documento está organizado em cinco capítulos. O Capítulo 1 apresenta uma visão geral do problema e do estado da arte no desenvolvimento de interfaces miocinéticas, destacando os principais objetivos e as questões centrais a serem investigadas. O Capítulo

2 trata da fundamentação teórica, aprofundando-se nas vantagens do uso de interfaces miocinéticas e detalhando a plataforma empregada, os conceitos de hardware reconfigurável, as Máquinas de Vetores de Suporte (SVMs), as técnicas de redução de dimensionalidade e o estado da arte. O Capítulo 3 propõe uma metodologia para a implementação de SVMs em FPGA, descrevendo o cenário de testes e coleta de dados, além de apresentar a abstração da implementação em hardware por meio de diagramas de blocos e uma explicação sobre a plataforma NIOTS, utilizada para obter os hiperparâmetros da SVM. O Capítulo 4 aborda os resultados e o Capítulo 5 apresenta as conclusões trabalho desenvolvido e sugestões para trabalhos futuros.

2 Fundamentação teórica

O capítulo 2 apresenta, em detalhes, os principais conceitos utilizados no desenvolvimento da proposta de solução. Inicialmente, é descrito o funcionamento de uma Interface Miocinética, seguido de uma comparação com outras abordagens aplicadas ao desenvolvimento de Interfaces Humano-Máquina. Na sequência, abordam-se as Máquinas de Vetores de Suporte, destacando-se as diferenças entre seu uso como classificadores e como regressores, bem como entre o *kernel* linear e o *kernel* RBF. Discutem-se, ainda, o método de redução de dimensionalidade *PCA* e a técnica de seleção de variáveis univariadas. Por fim, apresenta-se a estrutura do FPGA e o fluxo de projeto no desenvolvimento de aplicações em de hardware reconfigurável.

2.1 Interface miocinética para controle de próteses

Uma Interface Homem-Máquina tem como objetivo converter os sinais gerados pela intenção de movimento do usuário em sinais de controle. No contexto desta pesquisa, trata-se de sinais destinados ao controle de uma prótese, de forma intuitiva e com o mínimo de esforço por parte do usuário. Diversas técnicas vêm sendo exploradas para o desenvolvimento dessas interfaces, muitas das quais baseiam-se na análise dos sinais elétricos produzidos pela atividade muscular.

2.1.1 Comparação entre as diferentes técnicas estudadas para implementação de interface humano-máquina

A eletromiografia superficial consiste na aplicação de eletrodos sobre a pele para captar os sinais elétricos gerados pela contração muscular (Luca, 1997). Essa tecnologia é amplamente utilizada pela praticidade de implementação e estratégias de controle simplificadas, no entanto, essa técnica apresenta limitações na aquisição de dados provenientes de músculos profundos e é suscetível ruídos devido à barreira imposta pela pele. Uma alternativa que supera essas limitações é a eletromiografia implantada, em que sensores mioelétricos implantáveis (IMES, sigla em inglês) são inseridos diretamente nos músculos (Weir *et al.*, 2009). Contudo, por se tratarem de dispositivos eletrônicos ativos, esses sensores requerem uma fonte de alimentação para transmitir os sinais por indução a um receptor externo.

Outra abordagem promissora é o uso de eletrodos neurais para captar informações diretamente do sistema nervoso periférico (Micera; Carpaneto; Raspopovic, 2010). Além disso, estudos recentes têm investigado a aplicação de dispositivos supercondutores de interferência quântica (SQUIDs, sigla em inglês) (Zuo *et al.*, 2020), capazes de detectar

campos magnéticos extremamente fracos gerados pelos músculos esqueléticos humanos. A utilização de SQUIDs permite medições de altíssima sensibilidade, embora sua utilização dependa de ambientes altamente controlados.

A Tabela 2.1 apresenta uma comparação entre diferentes tecnologias estudadas para o controle de próteses, considerando: precisão, custo e a capacidade da técnica de obter informações sobre a mobilidade dos músculos resíduais do membro amputado.

Tabela 2.1 – Comparação entre diferentes tecnologias de interface Humano-Máquina para controle de próteses. Adaptado de (Mendez *et al.*, 2022)

Tecnologia	**Variável observada**	**Precisão**	**Informações musculares**	**Custo**
EMG superficial	Campo elétrico da atividade muscular	Baixa	Baixa	Baixo
EMG implantável	Campo elétrico da atividade muscular	Média	Média	Médio
Eletrodo neural	Campo elétrico da atividade neural	Média	Média	Alto
Sensor SQUID	Campo magnético da atividade muscular	Alta	Baixa	Alto
Interface miocinética	Deslocamento muscular	Alta	Média	Médio

2.1.2 Interface Miocinética

Uma abordagem promissora para o controle de próteses é o uso de marcadores magnéticos implantados (MMs, sigla em inglês). Tais dispositivos substituem a transdução de sinais elétricos pela detecção do campo magnético gerado por ímãs permanentes inseridos nos músculos, tecnologia conhecida como interface miocinética (Tarantino *et al.*, 2017). Como esses ímãs se movem de acordo com contração muscular, sua posição pode ser localizada por sensores externos, permitindo estimar de forma precisa o deslocamento muscular em tempo real. Isso possibilita um controle contínuo e proporcional de múltiplos graus de liberdade, preservando a naturalidade dos movimentos. Por serem passivos, os MMs não requerem alimentação elétrica, tornando o sistema mais estável, com maior imunidade a ruídos e de baixa manutenção.

A aplicação de interfaces miocinéticas baseia-se na premissa de que indivíduos ampu-

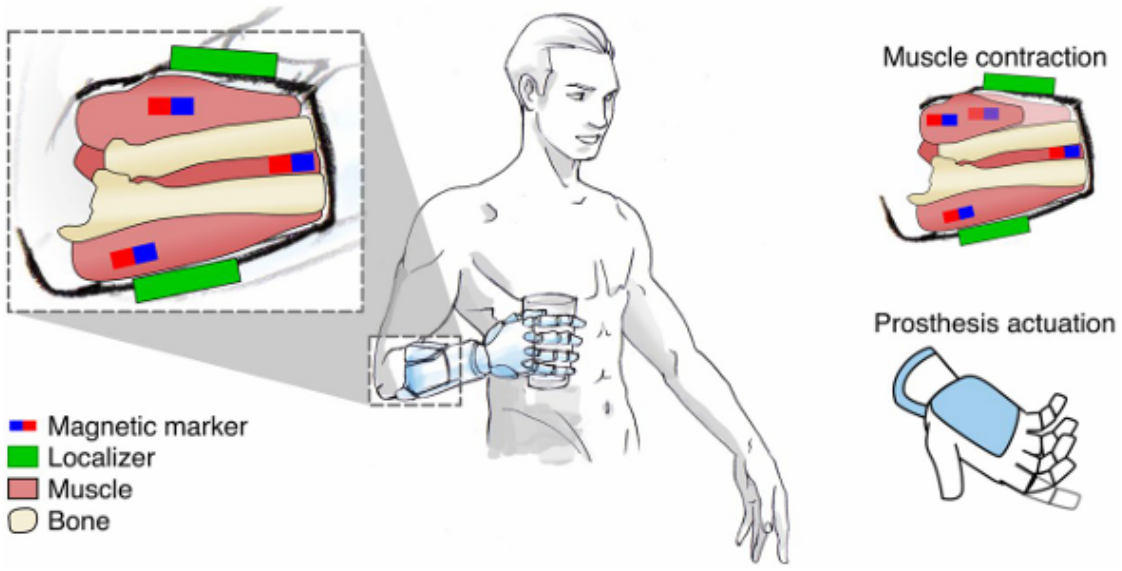


Figura 2.1 – Ilustração dos componentes de uma interface miocinética. *Fonte: Retirado de (Tarantino et al., 2017)*

tados são capazes de ativar voluntariamente contrações distintas em seus músculos residuais, permitindo que algoritmos mapeiem os padrões de sinal gerados para classes de movimento. A premissa também permite que os algoritmos estimem com precisão o deslocamento de ímãs, a fim de gerar comandos de controle para os motores da prótese.

A Figura 2.1 apresenta uma ilustração do funcionamento da interface miocinética. É possível observar os ímãs implantados, sendo que o deslocamento desses ímãs fornece uma medida direta da contração muscular. Os localizadores representados na cor verde são sensores de efeito *hall* para aquisição de dados e unidades de processamento para estimar a posição dos ímãs. Esses dispositivos são fixados externamente ao membro, sendo capazes de monitorar esses deslocamentos e convertê-los em sinais de controle para a mão artificial. Diversas técnicas podem ser implementadas para estimar a posição dos ímãs para monitorar os deslocamentos, entre elas, a Máquina de vetores de suporte, apresentada na seção 2.2.

2.2 Máquina de Vetores de Suporte

Máquina de Vetores de Suporte (SVM, sigla em inglês) é uma técnica de aprendizado de máquina supervisionado baseada na teoria das margens largas. O resultado do treinamento de uma SVM é uma função que recebe um conjunto de entradas e tem por objetivo calcular uma saída com erro mínimo em relação aos dados utilizados durante o treinamento e aproximar dentro de uma determinada tolerância dados que não foram apresentados anteriormente.

As SVMs são obtidas a partir de um conjunto de dados conhecidos que apresentam

rótulos ou estão associados a um valor específico, esses dados são denominados conjunto de treinamento. Assim, as SVMs buscam classificar dados que não pertencem ao conjunto de treinamento com base nos dados previamente apresentados. Quanto maior o êxito em classificar esses dados, maior é a capacidade de generalização do modelo. A complexidade do modelo está associada à quantidade de elementos necessários para compor a função classificadora. Um modelo que demanda mais pontos do conjunto de treinamento que outro é considerado mais complexo, assim como uma operação exponencial é mais complexa que uma multiplicação. Isso ocorre porque casos que exigem mais operações matemáticas também demandam maior consumo de recursos de hardware, energia e tempo de processamento.

No treinamento das SVMs, busca-se minimizar a complexidade e maximizar a capacidade de generalização, o que caracteriza um problema multiobjetivo. Como esses objetivos são contraditórios, deve-se encontrar um compromisso entre ambos que seja adequado para a aplicação em questão. Para obter uma SVM com alta capacidade de generalização e baixa complexidade, é necessário selecionar o parâmetro de regularização (C), os parâmetros do *kernel* (γ) e a largura do tubo e-insesitive (ε). Estes parâmetros em conjunto são denominados Hiperparâmetros (Burges, 1998) e a definição de seus valores ótimos é denominada Problema de Otimização de Hiperparâmetros.

O parâmetro de regularização (C) penaliza os pontos cujo erro excede (ε). Quando (C) tende ao infinito, o modelo passa a valorizar excessivamente os dados de treinamento, reduzindo o erro, porém diminuindo a capacidade de generalização e podendo resultar em sobreajuste. Por outro lado, quando (C) tende a zero, a SVM tolera tantos erros que perde a capacidade de classificar adequadamente os dados (Santos, 2019).

O objetivo das SVMs é definir um hiperplano (que, em duas dimensões, é uma reta) que separe os dados do conjunto de treinamento, denominado hiperplano classificador. São definidos dois hiperplanos paralelos a ele, chamados de margens; quanto maior a distância entre eles, maior a capacidade de generalização. As margens são determinadas por pontos do conjunto de treinamento, denominados vetores de suporte (Santos, 2019).

A Figura 2.2 apresenta um conjunto de dados linearmente separável, ou seja, é possível traçar uma reta (ou um plano, em dimensões superiores) que separe os dados em classes distintas. Nessa figura, observam-se as classes “estrela” e “círculo”; a reta contínua que separa as classes é o hiperplano classificador, e as retas pontilhadas representam as margens. Dessa forma, durante o treinamento das SVMs, devem ser identificados os vetores de suporte que possibilitam maximizar a distância entre essas margens.

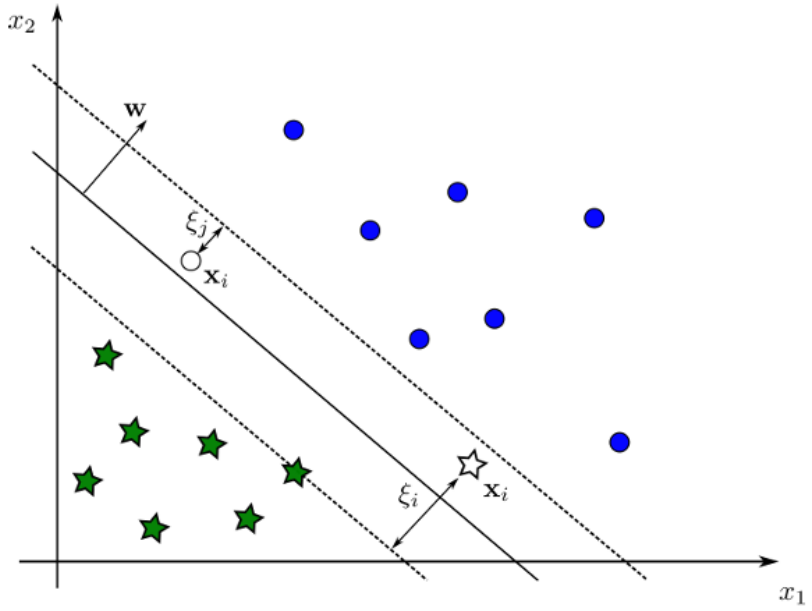


Figura 2.2 – Conjunto de dados linearmente separável. Retirado de (Santos, 2019)

2.2.1 Diferenças de Máquinas de Vetores de Suporte como classificadores e regressores

As Máquinas de Vetores de Suporte podem ser utilizadas para classificar conjuntos de dados, de forma que os dados de treinamento sejam separados em classes e a SVM determina a qual classe um novo dado de entrada pertence.

Um outro cenário é a utilização de Máquina de Vetores de Suporte como regressor, chamado de Regressores de Vetores de Suporte (SVR). Nesse caso, o objetivo não é mais separar os dados por classes, mas encontrar uma função interpole o conjunto de dados com uma margem de tolerância ε. Enquanto os classificadores apresentam saídas discretas, os regressores apresentam saídas contínuas, que é o caso abordado neste trabalho. É implementado um regressor para cada ímã, e a saída de cada regressor representa o deslocamento do respectivo ímã, um valor contínuo expresso em milímetros.

Tanto os regressores quanto os classificadores podem ser implementados com diferentes funções de *kernel*. O presente trabalho avalia o desempenho e eficiência de SVRs implementados com *kernel* linear e *kernel* RBF.

2.2.2 Funções de *kernel*

As funções de *Kernel* são ferramentas matemáticas responsáveis por representar os dados de entrada em um espaço de dimensão maior, ou seja, essas funções realizam uma transformação no espaço dos dados de entrada, mapeando-os para um espaço de maior dimensão. No caso do uso de SVMs como classificadores, o interesse em realizar essa operação é a representação dos dados em um espaço em que sejam linearmente separáveis. No caso

dos SVRs, o *kernel* mapeia os dados em um espaço de maior dimensão, de modo que, no espaço transformado, uma função linear possa modelar relações que são não lineares no espaço original. A Figura 2.3 apresenta um exemplo de um conjunto de dados transformado para um espaço de dimensão maior, em que são linearmente separáveis.

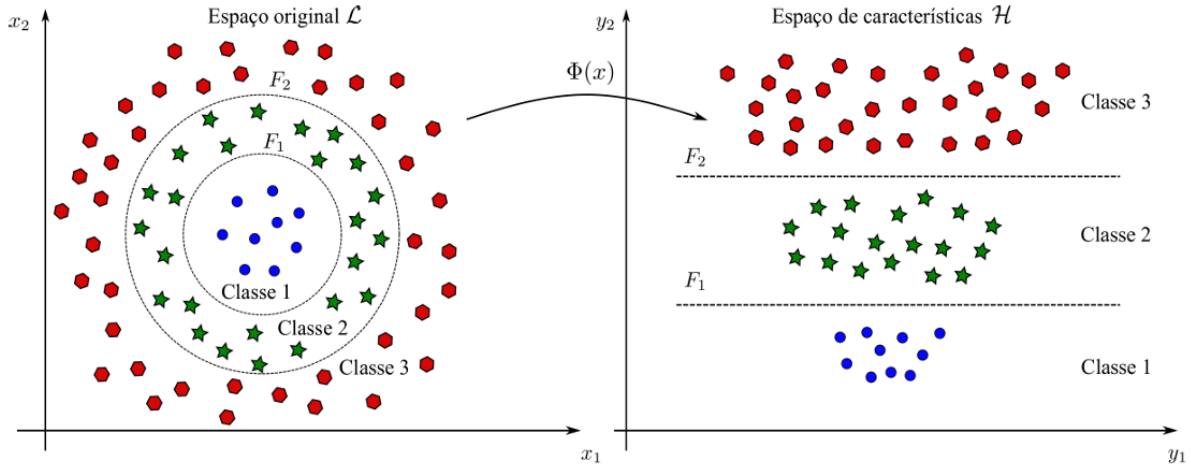


Figura 2.3 – Transformação realizada pela função de *kernel*. Retirado de (Santos, 2019)

Técnicas de aprendizado de máquina aplicadas a interfaces humano-máquina exigem modelos com alta capacidade de generalização e com baixa complexidade, devido às restrições de processamento e consumo de energia desses sistemas. As Máquinas de Vetores de Suporte atendem a esses requisitos devido a sua capacidade de generalização e flexibilidade das suas funções de *kernel*, que permite o projetista avaliar a resposta de modelos com diferentes funções e comparar com o custo computacional, baseando sua decisão nos requisitos da aplicação e nos recursos disponíveis. Este trabalho aborda o uso do *kernel* RBF, pela sua elevada capacidade de generalização, e do *kernel* linear, pela sua baixa complexidade de implementação em hardware.

O cálculo do *kernel* linear é definido pela operação de produto interno, que envolve as operações de soma e multiplicação apresentadas na Equação (2.1) sendo sua implementação em hardware muito eficiente em termos de velocidade e consumo de energia.

$$K_L(x,y) = x \cdot y \tag{2.1}$$

Em que x é um vetor de suporte e y o vetor de entrada de dados.

As SVR com *kernel* RBF são populares por possuir apenas um parâmetro a ser ajustado e captar relações não-lineares complexas de qualquer conjunto de treinamento (Burges, 1998). O *kernel* RBF é dado pela Equação (2.2).

$$K_{RBF}(x,y) = exp\left(-\frac{\|x-y\|^2}{\gamma}\right) \tag{2.2}$$

onde x é um vetor de suporte, y o vetor de entrada de dados e γ é uma constante maior que zero a ser definida.

Em casos extremos quando o valor de γ tende a zero, o modelo tende a considerar todos os pontos de treinamento como vetores de suporte, gerando modelos complexos e com baixa capacidade de generalização (Burges, 1998). Estas características ressaltam a importância da escolha adequada do hiperparâmetros da SVR.

2.2.3 Treinamento das SVMs

As SVMs possuem um modelo matemático para obtenção dos pesos, apresentado na Equação (2.3). Esse modelo consiste em um Problema de Otimização Quadrático que busca maximizar as margens.

$$\begin{aligned} \max \quad & L_r(\alpha_i) = -\tfrac{1}{2}\sum_{i,j=1}^{N} K(\mathbf{x}_i, \mathbf{x}_j)\overline{\alpha}_i\overline{\alpha}_j - \varepsilon\sum_{i=1}^{N}\overline{\alpha}_i + \sum_{i=1}^{N} y_i\overline{\alpha}_i \\ \text{s.a.} \quad & \sum_{j=1}^{n}\overline{\alpha}_i = 0, \quad 0 \leq \alpha_i, \alpha_i^{\star} \leq C, \end{aligned} \tag{2.3}$$

onde, $\overline{\alpha}_i = \alpha_i - \alpha_i^{\star}$, sendo α_i e $\alpha_i^{\star}$ as variáveis de decisão. O termo ε é chamado parâmetro ε-insensitive, esse parâmetro define uma zona de tolerância em torno da função interpolada pelo regressor. Se o valor calculado pelo regressor em um ponto x resulta em um valor dentro da zona de tolerância, definida pela soma do valor real da função com o parâmetro ε, esse dado não é penalizado no treinamento. Esse parâmetro, em conjunto com o parâmetro C, está relacionado com a acurácia e complexidade do modelo.

C é um parâmetro de regularização que penaliza as predições que apresentem erros fora da zona de tolerância. Esse parâmetro define quanto é tolerado que os dados se distanciem de ε. Considerando o parâmetro C e um *kernel* específico, quando ε tender a infinito o número de vetores de suporte no limite tende a zero, mas degenera a acurácia da função preditora $\hat{f}$. Por outro lado, quando o parâmetro ε tende a zero, o número de vetores de suporte tende a N (cardinalidade do conjunto de treinamento), o que aumenta chances da ocorrência de *overfitting* (Steinwart; Christmann, 2008).

A largura da região definida por ε está diretamente relacionada à precisão do valor calculado pela função preditora. Quanto maior o valor de ε, maior a tolerância a erros, o que produz aproximações mais grosseiras, porém com menos vetores de suporte. Por outro lado, valores muito pequenos de ε tendem a gerar modelos com muitos vetores de suporte, resultando em uma função excessivamente ajustada aos dados de treinamento (Santos, 2019).

A Figura 2.4 ilustra uma função preditora $\hat{f}$, as margens de tolerância definida por ε e o vetor de dados x_i, além disso é apresentada também a função de perda, que estabelece a penalidade para os dados fora da tolerância e é definida pela Equação 2.4.

$$P(t) = \begin{cases} |t| - \varepsilon, & |t| > \varepsilon \\ 0, & \text{caso contrário} \end{cases} \tag{2.4}$$

em que $t = f - \hat{f}$ é a diferença entre o valor calculado pelo modelo e o valor real.

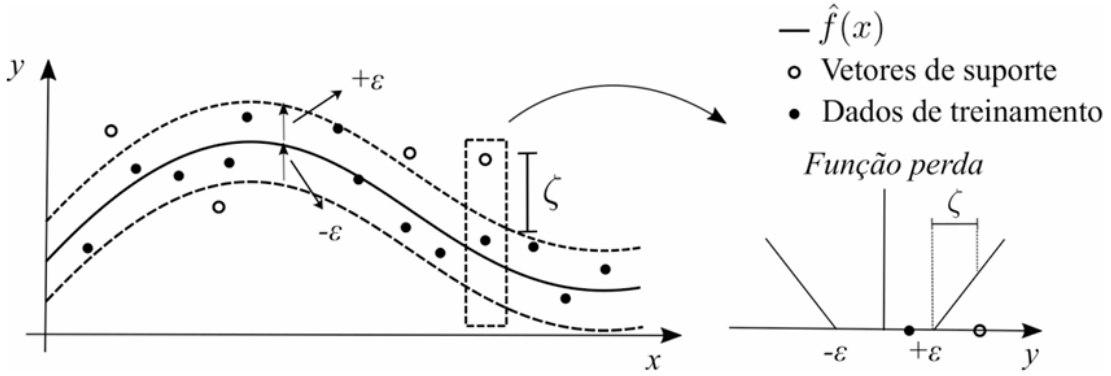


Figura 2.4 – Na esquerda, um modelo de SVR com margens de tolerância ε definida pelas linhas tracejadas, a função do regressor $\hat{f}$, os dados x_i. Na direita, a função de perda. Retirado de (Santos, 2019)

2.2.4 Função preditora

A função preditora $\hat{f}$, apresentada na Equação 2.5, é o modelo a ser implementado em hardware, em que a saída esperada deve corresponder ao deslocamento de um ímã. Essa função é definida por um conjunto de vetores de suporte (SV, sigla em inglês), compostos pelos parâmetros α_i, α_i^* e ρ que são definidos pela otimização dos hiperparâmetros. K é o *kernel*, que neste estudo serão abordados os tipos linear e RBF, definidos pelas Equações 2.1 e 2.2, respectivamente.

$$\hat{f}(x) = \sum_{i \in SV} (\alpha_i - \alpha_i^*) K(x_i, x) + \rho \tag{2.5}$$

A complexidade da implementação dessa função em hardware depende da cardinalidade do conjunto de vetores de suporte e das operações necessárias para implementar a função de *kernel* abordada, assim como a dimensionalidade de cada vetor de suporte e dos vetores de entrada. Diversas técnicas podem ser aplicadas para reduzir a dimensionalidade, ou seja, a quantidade de elementos de cada vetor de entrada. A seção 2.3 aborda a Análise de Componentes Principais e a seção 2.4 aborda uma análise estatística para determinar quais entradas são mais relevantes no cálculo da saída.

2.3 Análise de Componentes Principais

Análise de Componentes Principais (PCA, sigla em inglês) é uma técnica de redução de dimensionalidade cujo objetivo é reduzir o número de dados processados, mantendo a informação mais relevante para a análise. Esse método realiza operações matemáticas (Equa-

ções 2.6 e 2.7), de modo que a informação contida nos dados de entrada seja representada por um conjunto reduzido de dados.

A Equação 2.6 apresenta a multiplicação da matriz X, que contém os dados de entrada, pela matriz V_K, resultando em uma aproximação de dimensionalidade reduzida (T_K) do conjunto de dados X. Em cada coluna K da matriz V_K estão os componentes principais, eles formam uma base ortogonal de um espaço de dimensão K.

Na Equação 2.7 é aplicada a técnica de *Singular Value Decomposition* (SVD), utilizada na etapa de treinamento para determinar os componentes principais. SVD decompõe a matrix $\mathbf{X} \in \mathbb{C}^{n \times m}$. Nessa equação os termos $\mathbf{U} \in \mathbb{C}^{n \times n}$ e $\mathbf{V} \in \mathbb{C}^{m \times m}$ são matrizes unitárias, e $\mathbf{V}$ contém uma base ortonormal dos autovetores correspondentes, que é utilizada na Equação 2.7 (Brunton; Kutz, 2019). O termo H denota a transposta conjugada complexa.

$$\mathbf{T}_K = \mathbf{X}\mathbf{V}_K \tag{2.6}$$

$$\mathbf{X} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^H \tag{2.7}$$

Nos trabalhos apresentados em (Mendez *et al.*, 2024) e (Mendes, 2024) foi utilizado PCA para redução de dimensionalidade, isso permitiu a redução de um conjunto de dados de 384 variáveis (dados dos eixos x,y e z de 128 sensores) para 25 entradas. Por outro lado, essa abordagem requer uma multiplicação de uma matriz de 384 colunas (matriz X com os as variáveis de entrada) por uma matriz de 25 colunas (componentes principais) para cada novo vetor de entrada. Esse processo de multiplicação matricial impacta na latência e no consumo de recursos do FPGA, portanto, foi proposto a técnica de seleção univariada de variáveis, apresentada na seção 2.4.

2.4 Seleção univariada de variáveis

O problema da seleção de variáveis consiste na escolha de um subconjunto de variáveis independentes que maximize a capacidade de generalização do modelo, ao mesmo tempo em que minimize sua complexidade. Muitas vezes as variáveis independentes são redundantes, acrescentando pouca ou nenhuma informação ou, ainda, adicionando majoritariamente ruídos e aumentando a complexidade dos modelos.

Na Equação (2.8), é avaliada a correlação linear entre cada variável de entrada e a saída. As variáveis que apresentam maior covariância em relação à média dos dados, ou seja, que provocam maior variação na saída, são consideradas mais relevantes. Quando uma variável é estatisticamente independente da saída, sua correlação é nula, resultando em $\rho_i = 0$, o que leva ao seu descarte no processo de seleção de variáveis. O valor de ρ_i é uma forma padronizada da covariância, obtida por meio da divisão da covariância pelo produto da variância dos dados de entrada e saída

$$\rho_i = \frac{E[(x_i - \bar{x}_i)(y - \bar{y})]}{\sigma_{x_i}\sigma_y} \tag{2.8}$$

onde $\bar{x}_i$, σ_{x_i} são a média e a variância da *feature* x_i, respectivamente, e $\bar{y}$ e σ_y são a média e a variância da saída, respectivamente. Em seguida, é realizado um teste de hipóteses para análise de ρ_i.

A seleção univariada de variáveis realiza uma análise estatística offline, antes do treinamento, de forma que sejam consideradas somente as variáveis de entrada mais relevantes, reduzindo a dimensionalidade dos dados e simplificando o modelo. Dessa forma, implementa-se apenas o modelo simplificado com as variáveis selecionadas, o que representa uma maior eficiência computacional.

O resultado da aplicação dessa técnica é a redução no número de elementos do vetor de entrada. As unidades de aquisição, compostas por placas com diversos sensores de efeito *Hall*, geram 144 dados para serem processados pelos regressores. A técnica de seleção de variáveis reduz esse número de modo que, se o projetista optar por selecionar 10 variáveis, o vetor de entrada passará a ter somente 10 elementos, assim como os vetores de suporte, em vez de 144, representando 134 operações matemáticas a menos.

Esse método pode apresentar limitações por considerar somente as relações lineares e não considera as relações entre duas ou mais variáveis, fatores que podem fornercer informações importantes para a estimação da variável y.

Após a seleção das variáveis mais relevantes, que são denominadas *features*, os modelos são treinados e implementados em hardware reconfigurável. A seção 2.5 apresenta uma explicação sobre FPGAs.

2.5 Hardware reconfigurável

Diversos tipos de hardware reconfigurável estão disponíveis no mercado. Neste trabalho, o foco será nas FPGAs, devido à sua elevada flexibilidade, alto desempenho e ampla aplicabilidade em prototipagem.

2.5.1 FPGA

Field Programmable Gate Arrays (FPGAs) são um tipo de circuitos integrados projetado para permitir a reconfiguração do hardware. Diferente dos circuitos integrados convencionais, que possuem uma estrutura fixa após a fabricação, as FPGAs permitem alterar a lógica implementada e a conexão entre seus componentes internos, possibilitando a implementação física de circuitos digitais conforme descrito pelo projetista.

Essa flexibilidade é proporcionada pelos *Configurable Logic Blocks* (CLBs), blocos configuráveis capazes de implementar operações lógicas. Esses CLBs são interligados por

matrizes de interconexões também reconfiguráveis (AMD, 2025a), como ilustrado na Figura 2.5. Além da possibilidade de reconfiguração total, algumas famílias de FPGAs permitem ainda a reconfiguração parcial dinâmica, ou seja, é possível modificar apenas uma região do circuito enquanto o restante continua em funcionamento.

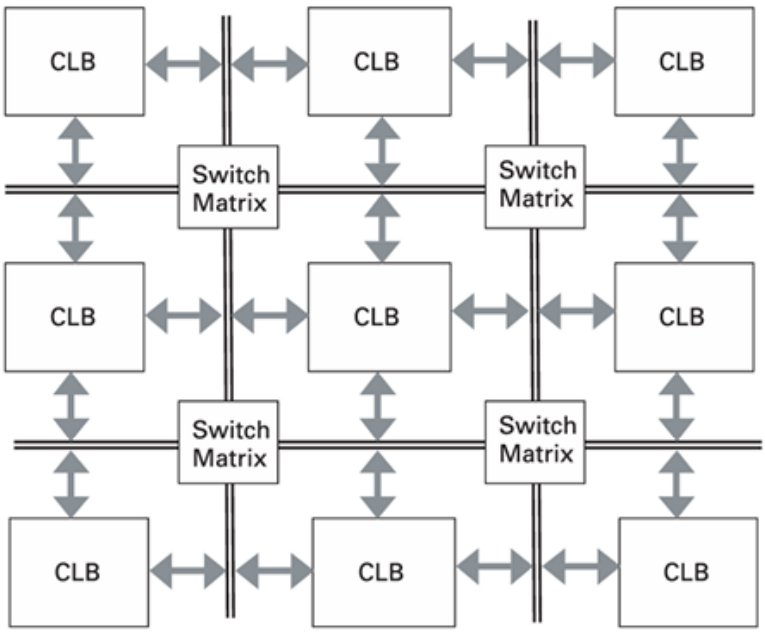


Figura 2.5 – Arquitetura genérica em uma FPGA.
Fonte: Retirado de (Pedroni, 2004)

Diferentes fabricantes adotam arquiteturas distintas para os CLBs. Nas FPGAs da AMD/Xilinx, cada CLB é composto internamente por *Look-Up Tables* (LUTs), *flip-flops* e multiplexadores. As LUTs são responsáveis por implementar lógica combinacional; os *flip-flops* funcionam como registradores, implementando lógica sequencial e os multiplexadores são usados para direcionar as saídas das LUTs para os CLBs vizinhos.

Além desses elementos, as FPGAs também incluem diversos componentes programáveis que desempenham funções específicas, como blocos de memória RAM, *Digital Signal Processors* (DSPs), Phase-Locked Loops (PLLs) e interfaces de comunicação de alta velocidade. No caso da AMD/Xilinx, destaca-se a interface *AXI* (*Advanced eXtensible Interface*). Outro elemento fundamental são os *I/O Blocks*, que correspondem aos pinos de entrada e saída responsáveis pela comunicação da FPGA com o ambiente externo, permitindo a recepção e/ou transmissão de dados.

FPGAs são amplamente empregadas em aplicações de tempo real que demandam respostas rápidas e alta eficiência, devido à sua capacidade de executar diversas operações em paralelo. Esse paralelismo permite processar grandes volumes de dados em intervalos curtos, atendendo aos requisitos de baixa latência presentes em sistemas embarcados de dinâmica rápida.

2.5.2 SoC FPGA

System-on-Chip (SoC) FPGA é a integração de uma FPGA com um processador, geralmente ARM, no mesmo encapsulamento. Essa tecnologia possibilita o desenvolvimento de projetos Hardware/Software, nos quais a FPGA executa tarefas que demandam alta velocidade e paralelismo, enquanto o processador de propósito geral realiza tarefas programadas por linguagens de programação convencionais.

A AMD disponibiliza a plataforma Vitis para a programação do processador, permitindo o uso de linguagens convencionais como C/C++, em conjunto com o Vivado, responsável pela reconfiguração da FPGA por meio de linguagens de descrição de hardware (HDLs). A família de FPGAs Zynq-7000, da AMD-Xilinx, é um exemplo de SoC FPGA que integra um processador ARM Cortex-A9, com arquitetura apresentada na figura 2.6.

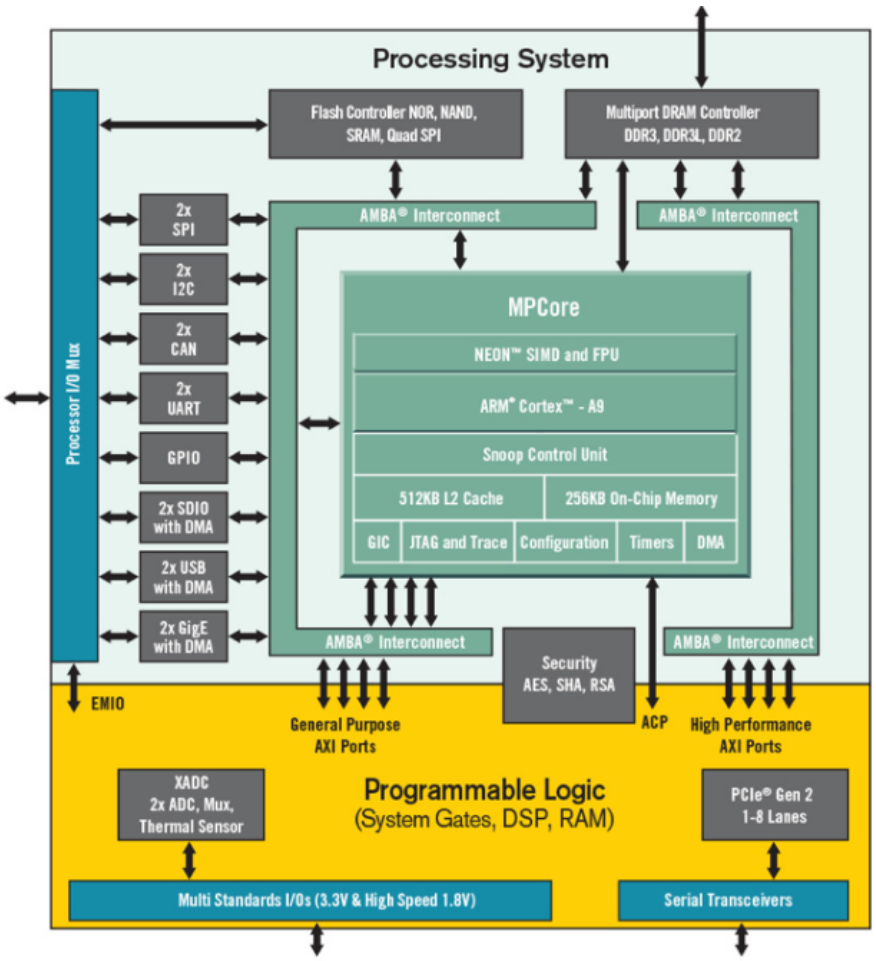


Figura 2.6 – Arquitetura de um SoC FPGA. Retirado de (AMD, 2025c)

2.5.3 Analisador lógico integrado

O analisador lógico integrado (ILA, sigla em inglês) é um núcleo personalizável que pode ser utilizado para monitorar sinais internos de um projeto (AMD, 2025b). Trata-se de

uma ferramenta incluída no software Vivado, disponibilizada na forma de *IP core*.

Ao instanciar o ILA, definem-se os sinais de entrada que serão monitorados, sendo que pelo menos um deles deve atuar como *trigger*. O ILA é implementado fisicamente na FPGA, ocupando, portanto, uma quantidade significativa dos recursos disponíveis do chip. Esse recurso é utilizado para testes, permitindo verificar se as saídas da implementação física estão coerentes com aquelas obtidas em simulação, além de possibilitar o armazenamento automático dos dados em arquivos.

Os consumos de energia e de recursos associados ao ILA devem ser desconsiderados no cálculo do consumo total do sistema, uma vez que se trata de um recurso exclusivamente voltado à etapa de testes e que não estará presente na implementação final.

2.5.4 Fluxo de projeto

O fluxo de projeto de um circuito digital implementado em FPGA envolve três etapas principais: codificação, síntese e implementação.

Antes da etapa de codificação, é uma boa prática preparar o projeto por meio do desenvolvimento de diagramas de blocos que descrevam claramente o fluxo de sinais, nomes, tipos e tamanhos. Essa etapa preliminar facilita a codificação e proporciona uma análise mais clara do funcionamento do sistema a ser projetado.

As três etapas mencionadas podem ser realizadas no software Vivado, da AMD/Xilinx. Na fase de codificação, utilizam-se linguagens de descrição de hardware (HDLs, sigla em inglês), sendo as duas principais o VHDL e o Verilog, amplamente utilizadas nos ambientes acadêmico e industrial. Para validar a lógica desenvolvida antes da implementação física é utilizado um ambiente de simulação, o qual permite rastrear individualmente cada sinal do circuito.

Na etapa de síntese, a ferramenta infere as equações booleanas correspondentes, produzindo um diagrama de blocos com os componentes necessários e suas interconexões. Essa fase é conhecida como Síntese em Nível de Transferência de Registradores (RTL, sigla em inglês).

Já na etapa de implementação, o Vivado realiza a otimização do circuito, posiciona os componentes no *chip* e estabelece as conexões entre eles. Esse processo é conhecido como *Place and Route* (PAR).

É fundamental selecionar corretamente o modelo do *chip* que será utilizado, assim como fornecer o arquivo de *constraints*, que define as portas de entrada e saída e a frequência de *clock* do sistema. Com base nessas informações, a ferramenta realiza análises de utilização de recursos, temporização e consumo de potência.

2.6 Estado da arte

O trabalho apresentado em (Clemente *et al.*, 2019) descreve o hardware utilizado para a construção de um *dataset* destinado ao treinamento de modelos de aprendizado de máquina para localização de ímãs, desenvolvido pelo Instituto de Bio-robótica da *Scuola Superiore Sant'Anna*. A geração desse *dataset* consiste em movimentar os ímãs segundo trajetórias conhecidas; por exemplo, ao movimentá-lo com incrementos constantes, obtém-se um movimento em rampa. Conhecendo-se a amplitude, a frequência e o tipo de movimento aplicado, é possível determinar a posição do ímã em relação à posição inicial em cada instante de tempo. Com essas informações, associam-se as leituras dos sensores de efeito *Hall* à posição correspondente do ímã, de modo que tais dados possam ser utilizados no treinamento e validação de modelos de aprendizado de máquina.

O trabalho de (Feitosa, 2023), por sua vez, apresenta uma arquitetura distinta para as unidades de aquisição, utilizando o protocolo SPI, enquanto o *dataset* de (Clemente *et al.*, 2019) emprega o protocolo I2C e as estruturas onde essas placas são acopladas possuem formato físico diferente. Além disso, surge a necessidade de coletar amostras com movimentos simultâneos dos ímãs, uma vez que o *dataset* citado contempla apenas amostras em que um ímã se movimenta por vez. Diante dessas diferenças, este trabalho propõe a construção de um novo *dataset* com características próprias. As principais diferenças entre os dois conjuntos são apresentadas na Tabela 2.2.

Tabela 2.2 – Comparação entre bases de dados utilizadas para treinar e validar modelos de aprendizado de máquina para estimar deslocamento de ímãs.

Base de Dados	(Clemente *et al.*, 2019)	Este trabalho
Quantidade de Unidades de Aquisição	4	4
Quantidade de sensores por unidade de aquisição	32	12
Total de valores adquiridos a cada amostra	384	144
Área coberta por cada unidade de aquisição	17.01 cm^2	4.86 cm^2
Protocolo de comunicação	I2C	SPI
Tempo de transmissão por unidade de aquisição	23.9 ms	576 μs
Quantidade de ímãs	5	8
Os ímãs se movimentam ao mesmo tempo?	Não	Sim
Formas de onda	Rampa	Rampa e senoide
Amplitude e frequência do movimento variam?	Não	Sim

No Capítulo 1 foram apresentados os estudos desenvolvidos com foco nas interfaces miocinéticas. A Tabela 2.3 apresenta um resumo desses trabalhos, filtrando apenas aqueles que utilizam a abordagem baseada na estimativa da posição do ímã, uma vez que essa será a técnica adotada nesta pesquisa.

Na Tabela 2.3, observam-se, nas colunas, a referência do trabalho analisado, uma breve descrição do hardware utilizado e do algoritmo implementado, além do consumo de energia. Esse consumo considera a parte lógica descrita em cada um dos trabalhos somada ao consumo de potência de 550 mW por cada unidade de aquisição, totalizando 2.2 W (Clemente *et al.*, 2019). Em seguida, apresenta-se o tempo de processamento para a localização dos ímãs, indicando entre parênteses a quantidade de ímãs localizados nesse intervalo, e o tempo de transmissão entre as unidades de aquisição e a unidade de processamento.

Observa-se que, quando aplicadas FPGAs, o tempo de processamento permanece fixo independentemente da quantidade de unidades de aquisição, pois o SPI é implementado em paralelo. Já no trabalho descrito em (Clemente *et al.*, 2019), o tempo de transmissão é quatro vezes maior que nos demais, pois não há paralelismo: para cada unidade de aquisição adicionada são somados 23,9 ms. Em seguida, é apresentada a eficiência computacional, expressa em localizações de ímã por segundo por Watt. Na última coluna, apresenta-se o *dataset* utilizado para treinar e validar os modelos.

Tabela 2.3 – Comparação entre diferentes implementações de modelos de aprendizado de máquina para estimar deslocamento de ímãs.

	Descrição	**Consumo de energia (W)**	**Tempo de processamento para localização de ímã**	**Tempo de transmissão (ms)**	**Eficiência computacional (MLPS/W)**	**Base de dados**
(Clemente *et al.*, 2019)	ARM Cortex M7 @600 MHz/ LMA	2.63	0.8 ms (1 mm)	95.6	3.94	(Clemente *et al.*, 2019)
(Mendez *et al.*, 2022)	SoC FPGA / regressor linear	2.409	4.8 μs (1 mm)	23.9	17.37	(Clemente *et al.*, 2019)
(Mendez *et al.*, 2022)	SoC FPGA / regressor RBFNN	2.745	12.07 μs (1 mm)	23.9	15.26	(Clemente *et al.*, 2019)
(Mendez *et al.*, 2024)	SoC FPGA / Reconfiguração Dinâmica	3.919	9.52 ms (5 mms)	23.9	38.18	(Clemente *et al.*, 2019)

O trabalho apresentado em (Mendez *et al.*, 2022) utiliza FPGAs, resultando em um aumento significativo da eficiência computacional em decorrência do paralelismo proporcionado por esse recurso. No entanto, esse estudo não explora técnicas de redução de dimensionalidade, as quais podem tanto aumentar a eficiência computacional quanto reduzir o erro empírico. Por sua vez, no estudo apresentado em (Mendez *et al.*, 2024), a técnica

de redução de dimensionalidade *PCA* é aplicada em conjunto com a reconfiguração dinâmica parcial, possibilitando um ganho considerável de eficiência computacional. Contudo, essa abordagem apresenta a desvantagem de exigir, a cada novo conjunto de entrada, uma multiplicação matricial de dimensão 384×25, intrínseca ao PCA, operação que implica um elevado custo computacional.

2.7 Conclusão do capítulo

Esse capítulo apresentou os principais conceitos necessários para o entendimento desta pesquisa, destacando as vantagens das interfaces miocinéticas em comparação com outras tecnologias. O método de seleção univariada de variáveis foi abordado de forma a evidenciar seu potencial como alternativa promissora ao uso do PCA em implementações em hardware. As máquinas de vetores de suporte foram detalhadas, assim como as diferenças entre o *kernel* linear e o *kernel* RBF, com ênfase em suas principais vantagens e desafios. Além disso, foram discutidas as FPGAs, que constituem o objeto de estudo desta pesquisa, e o fluxo de projeto adotado para a implementação dos regressores em hardware. Por fim, foi apresentada uma análise do estado da arte, evidenciando as características dos trabalhos semelhantes.

3 Metodologia

A metodologia proposta tem como objetivo coletar dados referentes à variação do campo magnético gerado pelo movimento dos ímãs, rotulando-os com a posição de cada ímã no momento da coleta de cada amostra e utilizar esse conjunto de dados para treinar modelos de Regressores de Vetores de Suporte (SVR) e validar esses modelos em cenários de teste. Para a coleta dos dados e validação dos modelos treinados, propõe-se a utilização uma maquete que emula um antebraço humano com ímãs implantados. Além disso, são apresentadas arquiteturas projetadas para implementação em VHDL dos regressores, com o intuito de estimar, em tempo real, o deslocamento dos ímãs.

3.1 Proposta geral

O desenvolvimento desta pesquisa é dividido em dois módulos principais:

- Emular uma interface miocinética, de modo que os ímãs se movimentem ao receber comandos do usuário (Figura 3.1). Esses ímãs, idealmente, são os ímãs implantados nos músculos de pessoas amputadas. A função desse protótipo é emular o movimento humano. O hardware e firmware responsáveis por movimentar esses motores são, em conjunto, chamados unidade de controle.
- Processar os dados da variação dos campos magnéticos para calcular o deslocamento dos ímãs e controlar uma mão robótica (Figura 3.2).O hardware e firmware responsáveis por receber os valores de campo magnético são, em conjunto, chamados unidades de aquisição.

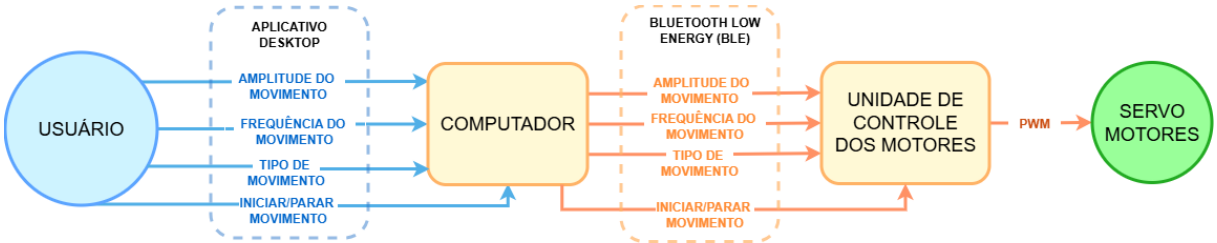


Figura 3.1 – Fluxograma do processo de emulação de uma interface miocinética. Representação da entrada de dados pelo usuário, processamento e conversão desses parâmetros em sinal PWM para controle dos motores.

A Figura 3.1 representa as etapas para o controle da plataforma que movimenta os motores. Essa plataforma tem como objetivo emular os movimentos dos ímãs implantados, possibilitando a realização de testes em ambiente controlado. O usuário insere os dados referentes ao movimento de cada ímã (amplitude, frequência, tipo de movimento e início

do experimento) por meio de uma aplicativo executado em um computador e esses dados são enviados para a placa de controle por meio de *Bluetooth Low Energy* (BLE). A placa de controle, processa esses parâmetros, modela o movimento com base neles e envia sinais de PWM para os motores se movimentarem conforme o usuário solicitou.

No momento da coleta para a construção da base de dados, é adicionado um fio de conexão entre a placa de controle dos motores e as placas de aquisição de sinais. Esse fio transportava um sinal utilizado como *trigger*: a cada movimento executado pelos motores, a placa de controle envia um pulso para que a placa de aquisição registre os dados. Dessa forma, é possível obter exatamente a mesma quantidade de passos por movimento e de amostras coletadas, permitindo relacionar cada conjunto de amostras à posição aproximada do motor. Note que esse recurso é utilizado somente para a construção da base de dados, já para testes e aplicações reais, não existe nenhum acoplamento físico entre os ímãs e as placas responsáveis pela medição da intensidade do campo magnético.

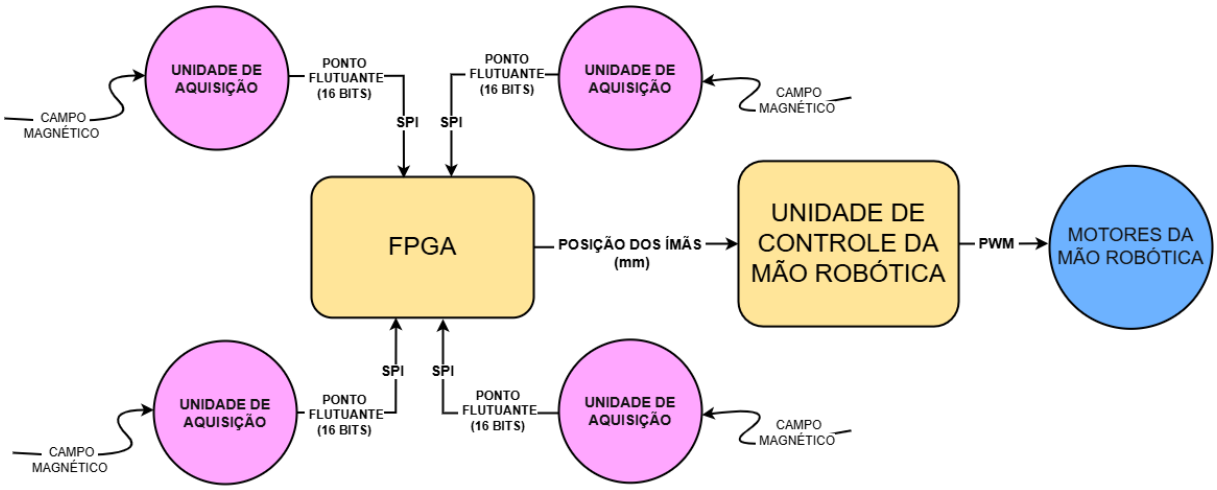


Figura 3.2 – Fluxograma do processamento dos dados para estimar a posição dos ímãs e controlar a mão robótica. Apresenta a comunicação das unidades de aquisição com FPGA e a comunicação da FPGA com a mão robótica.

A Figura 3.2 ilustra o fluxo de sinais na etapa de controle da mão robótica por parte do usuário. As placas de aquisição possuem sensores *hall*, que coletam os valores da intensidade do campo magnético e os enviam ao FPGA em ponto flutuante de 16 bits utilizando o protocolo SPI. São instanciados 4 barramentos SPI na FPGA, de forma que as placas de aquisição transmitam os dados de forma simultânea, garantindo paralelismo e agilidade ao processo de cálculo do deslocamento dos ímãs. O FPGA implementa uma SVR para cada ímã e calcula o deslocamento individualmente. Esses valores de deslocamento são enviados para a placa de controle da mão robótica que, a partir dessas informações, define a melhor estratégia e executa o controle dos motores para movimentar a mão artificial de acordo com a intenção do usuário. O controle da mão robótica em função do deslocamento dos ímas não está no escopo desta pesquisa.

3.2 Desenvolvimento da Interface Miocinética

A plataforma que emula uma interface miocinética apresenta uma estrutura em formato de um antebraço. Os softwares embarcados nas unidades de controle e de aquisição foram desenvolvidos nesta pesquisa, já a estrutura física e o hardware são resultados de outros estudos citados ao longo deste trabalho.

Na parte externa dessa estrutura estão as unidades de aquisição (Figura 3.3) e na parte interna os ímãs (Figura 3.4). Na base da estrutura estão localizados os servomotores MG995, cada um responsável por movimentar um ímã. Além do texto explicativo apresentado a seguir, uma visão geral sobre a estrutura e funcionamento da interface miocinética é apresentada no vídeo disponível neste link [1]



Figura 3.3 – Interface miocinética com estrutura em formato de antebraço e placas de aquisição de dados na parte externa.

As placas de aquisição foram desenvolvidas no trabalho apresentado em (Feitosa, 2023). Essas placas são compostas por 16 sensores LIS3MDL, sensores magnéticos do tipo *Hall*, sendo que cada um fornece ao microcontrolador ESP-WROOM-32 dados da intensidade do campo magnético nos eixos X, Y e Z. Devido à disponibilidade limitada de pinos GPIO, o trabalho em questão utilizou o protocolo SPI no modo *half-duplex*, o qual apresenta os sinais

[1] https://youtube.com/shorts/4BgI8RYp_CI?si=QzTXC5aBfv7niHNV

SDI e *SDO* no mesmo barramento. Durante a coleta de dados para treinamento dos modelos, as placas iniciam a captura de dados de forma sincronizada por meio de um sinal *trigger*.



Figura 3.4 – Interior da interface miocinética, apresentando os ímãs (marcadores magnéticos) e os servomotores responsáveis por emular o movimento muscular.

Algumas alterações foram necessárias para a utilização das placas propostas em (Feitosa, 2023). A modificação mais significativa foi a desabilitação de uma coluna de sensores, uma vez que um dos sensores foi conectado ao pino de *boot* do microcontrolador. Devido ao esquema de conexão implementado, não é possível desabilitar um sensor individualmente; assim, foram desabilitados quatro sensores, restando 12 sensores ativos em cada placa.

Outras observações relevantes para o uso das placas incluem a necessidade de ajustes na sensibilidade dos sensores, via firmware, para evitar saturação, além da escolha adequada da versão do *board support package* selecionada na IDE. Isso porque versões mais recentes descontinuaram o suporte ao protocolo SPI no modo *half-duplex*, de modo que o código compila mas não funciona como esperado.

O controle dos motores é realizado por um kit de desenvolvimento ESP32 com microcontrolador ESP32-WROOM-32, conforme apresentado na Figura 3.5. O usuário deve inserir na interface gráfica os parâmetros do movimento (amplitude, frequência, tipo de movimento e início/fim do movimento) e o computador envia através do BLE para o software embarcado no microcontrolador, que processa esses sinais e traduz sinal de PWM para os motores.

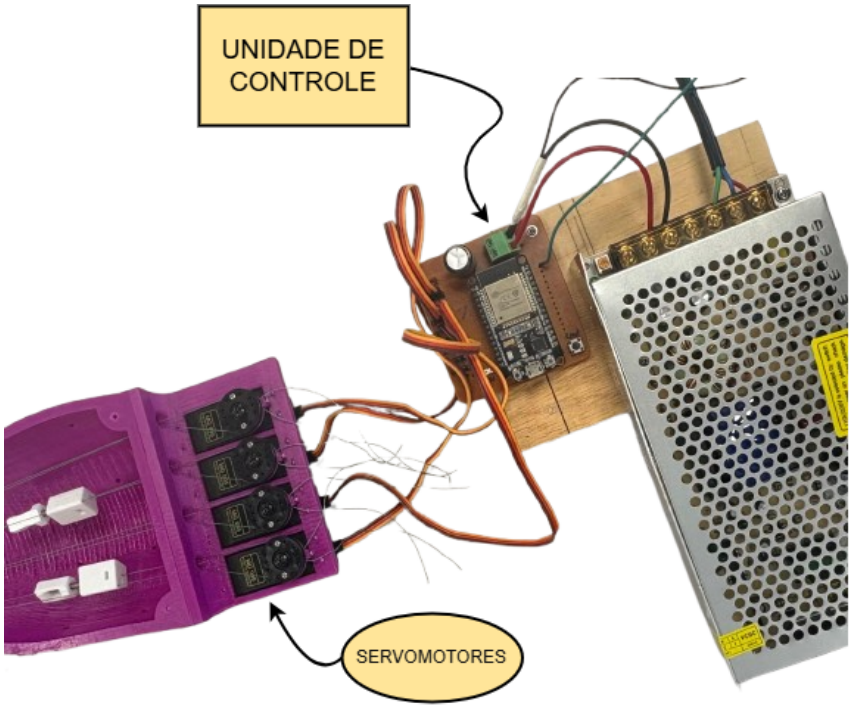


Figura 3.5 – Placa de controle do protótipo da interface miocinética, responsáveis por receber os parâmetros do movimento e converter em sinais de modulação por largura de pulso.



Figura 3.6 – À esquerda, as duas metades da plataforma de testes separadas; à direita, as duas metades parafusadas, configuração utilizada na realização dos testes.

Para uma melhor compreensão, foram apresentadas imagens da plataforma de testes desmontada, com as duas partes que a compõem separadas. No entanto, no momento da coleta de dados e na realização dos testes de validação, essas duas partes são parafusadas de modo a formar uma única estrutura, emulando um antebraço. A Figura 3.6 apresenta, à esquerda, as duas partes da plataforma separadas e, à direita, ambas já parafusadas.

A Figura 3.7 apresenta uma ilustração conceitual do sistema a ser implementado. Observa-se, na parte inferior, a plataforma que emula a interface miocinética, que é objeto de estudo desta pesquisa. Após o cálculo do deslocamento dos ímãs pelos regressores implementados em FPGA, esse valores são transmitidos para a placa de controle da mão robótica, conforme sugerido pelo fluxograma da Figura 3.1. Assim, a figura mostra a mão robótica desenvolvida no trabalho apresentado em (Mendez; Llanos; Muñoz, 2021) acoplada à interface miocinética.

Ressalta-se que a imagem foi editada com o objetivo de aprimorar a visualização e facilitar o entendimento do funcionamento proposto. Este trabalho tem como objetivo principal a estimativa de deslocamento dos ímãs, não contemplando o controle da mão robótica.



Figura 3.7 – Figura editada para ilustração da plataforma e mão robótica aplicadas em conjunto.

3.3 Modelagem dos movimentos dos servomotores

São abordadas duas formas de onda base para o movimento dos motores: rampa e senóide. Outros movimentos, como a representação de uma mão abrindo, fechando, realizando o movimento de pinça ou de agarre, podem ser obtidos pela combinação dessas formas de onda aplicadas a diferentes motores. Este trabalho não tem como objetivo estabelecer a relação anatômica entre os movimentos musculares e os movimentos da mão, tampouco definir qual forma de onda é mais adequada para representar o movimento humano. Busca-se, aqui, modelar os movimentos básicos e implementar uma estrutura funcional para que trabalhos futuros possam se valer desse desenvolvimento e estabelecer uma relação mais precisa com a anatomia do corpo humano.

3.3.1 Movimento de rampa

No movimento de rampa, a posição do motor é incrementada por um passo constante, definido em função da frequência e da amplitude do movimento, considerando também o tempo entre a execução de cada passo.

Nesse contexto, define-se um ciclo de movimento como o deslocamento do motor da posição inicial até amplitude desejada, ou no sentido inverso. Por exemplo, caso o movimento se inicie na posição 0 e o objetivo seja atingir a amplitude de 10 mm, o deslocamento de 0 até 10 mm corresponde a um ciclo, o deslocamento de 10 mm de volta para 0 mm corresponde a um novo ciclo. Embora a unidade de controle implemente esse movimento em *loop*, de modo que os ímãs permaneçam se deslocando da posição inicial até a amplitude desejada e retornando, nos movimentos naturais do corpo humano o deslocamento de cada ímã pode variar tanto em direção quanto em amplitude a cada ciclo, sem a necessidade de retornar à posição inicial do movimento.

Conforme as considerações realizadas, observa-se que um movimento de 1 Hz com amplitude de 5 mm indica que o ímã se desloca da posição inicial até 5 mm, ou de 5 mm de volta à posição inicial, uma vez por segundo.

Dessa forma, para calcular o incremento de posição, considera-se o período do ciclo, dado pela Equação 3.1:

$$T = \frac{1}{frequência} \tag{3.1}$$

Nomeando o tempo entre a execução de cada passo como Δt, dado em segundos, a quantidade de passos do movimento, denotada por N, é expressa pela Equação 3.2.

$$N = \frac{T}{\Delta t} \tag{3.2}$$

O valor do incremento, denominado Δs, é obtido pela razão entre a amplitude desejada e a quantidade de passos, apresentado na Equação 3.3.

$$\Delta s = \frac{\text{amplitude}}{N} \tag{3.3}$$

Substituindo as equações anteriores, obtém-se a fórmula final para o cálculo do incremento, observada na Equação 3.4.

$$\Delta s = amplitude \cdot frequência \cdot \Delta t \tag{3.4}$$

A tabela 3.1 apresenta um exemplo da posição do ímã no movimento de rampa, com uma frequência de 2 Hz, amplitude de 5 mm e Δt de 50 ms, valores que, conforme a Equação 3.4, resultam em um incremento constante de 0.5 mm.

Tabela 3.1 – Exemplo de posição do ímã para um movimento de 2 Hz, amplitude de 5 mm e $\Delta t = 0.05$ s.

Passo	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Posição (mm)	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5	4.0	4.5	5.0	4.5	4.0	3.5	3.0	2.5	2.0	1.5	1.0	0.5	0.0

A Figura 3.8 apresenta os dados da Tabela 3.1 de forma gráfica, evidênciando o movimento de rampa implementado.

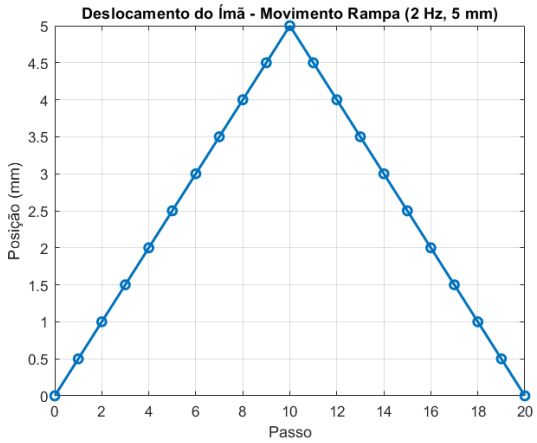


Figura 3.8 – Movimento do ímã com incrementos constantes.

3.3.2 Movimento senoidal

De forma semelhante, o incremento no movimento senoidal também se dá em função da amplitude, da frequência e do tempo de execução de cada passo. Nesse caso, é implementado somente o semiciclo positivo de uma senoide, considerando que o deslocamento do ímã da posição inicial até a amplitude máxima corresponde ao trecho de subida da senoide, e o retorno do pico para o zero da senoide representa o retorno do ímã da amplitude máxima para a posição inicial. Em outras palavras, um semiciclo positivo da senoide corresponde ao deslocamento do ímã 0 até a amplitude máxima e o movimento inverso, representando 2 ciclos do movimento do ímã.

Diferentemente do movimento de rampa, nesse caso o incremento não é constante. O movimento começa lento, com incrementos pequenos; na metade do percurso ocorre

uma aceleração, com incrementos maiores; e, ao final do movimento, há novamente uma desaceleração.

Derivando a expressão da posição no movimento de rampa, obtém-se uma velocidade constante. Já no caso do movimento senoidal, a derivada da posição produz uma velocidade variável, sendo maior na parte central do movimento.

O período do ciclo T é definido pela Equação 3.1, a variável t é o tempo atual do movimento, ou seja, a cada incremento esse valor também é incrementado com o valor de Δt. O valor do incremento Δs é dado pela Equação 3.5. Para valores de $t < T$, a equação representa o movimento do ímã da posição inicial até a amplitude máxima; para $t > T$, é o retorno do ímã para a posição inicial. A é o valor da amplitude, configurado pelo usuário.

$$\Delta s(t) = \begin{cases} \dfrac{A}{2}\left(1 - \cos\left(\pi \dfrac{t}{T}\right)\right), & 0 \leq t < T, \\ A - \dfrac{A}{2}\left(1 - \cos\left(\pi \dfrac{t}{T}\right)\right), & T \leq t \leq 2T. \end{cases} \tag{3.5}$$

A Tabela 3.2 apresenta um exemplo de senoide gerada para uma amplitude de 5 mm, frequência de 2 Hz e tempo de 0.05s para a realização de cada incremento. É possível observar nessa tabela que o valor do incremento não é constante.

Tabela 3.2 – Posição do ímã para A = 5 mm, f = 2 Hz e Δt = 0,05 s.

Passo	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
Posição (mm)	0.0	0.1	0.4	1.0	1.7	2.5	3.2	3.9	4.5	4.8	5.0	4.8	4.5	3.9	3.2	2.5	1.7	1.0	0.4	0.1	0.0

A Figura 3.9 representa de forma gráfica os dados apresentados na Tabela 3.2, apresentando uma movimento senoidal realizado pela trajetória do ímã.

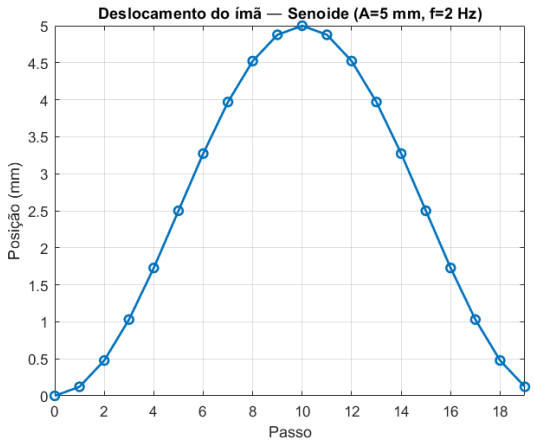


Figura 3.9 – Movimento do ímã com incrementos variáveis.

3.4 Arquitetura da implementação de SVR em hardware

Uma proposta de implementação de SVR em hardware é apresentada na Figura 3.10. Os dados de entrada são armazenados de forma sequencial em um *buffer* (FIFO), e as variáveis definidas como mais relevantes no processo de seleção univariada de variáveis, descrito no Capítulo 2, são direcionadas para a entrada do regressor correspondente. Cada bloco SVR_{mm} representa um regressor, sendo que cada um monitora o deslocamento de um ímã. A variável N no último regressor representa a quantidade de repressores implementados.

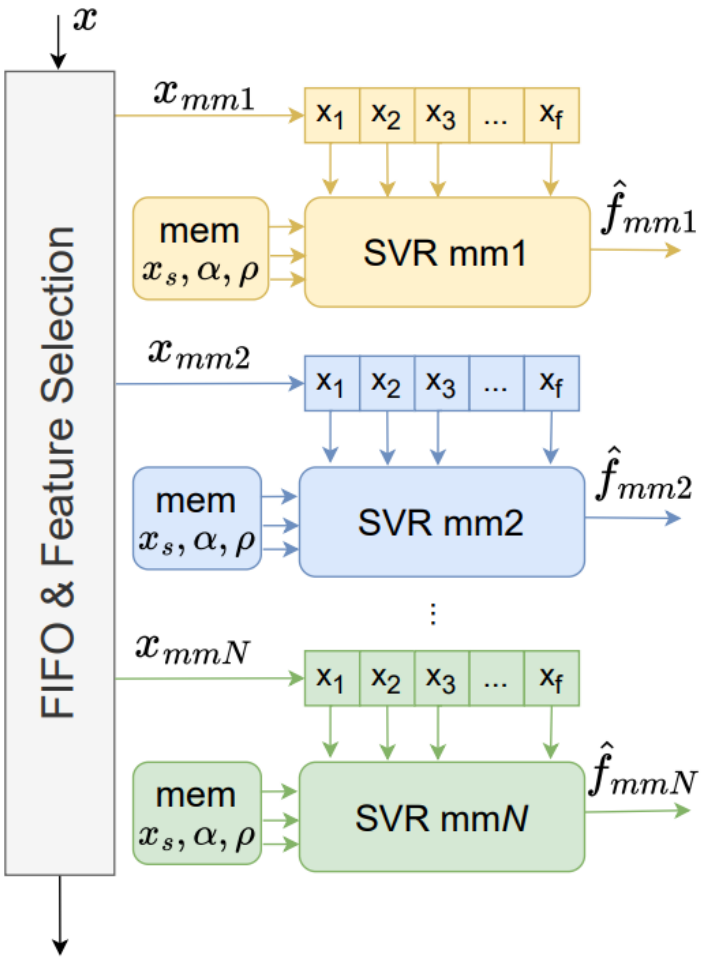


Figura 3.10 – Arquitetura da implementação de SVR em hardware

A quantidade total de dados de entrada corresponde aos três eixos dos 12 sensores de cada placa. Utilizando quatro placas de aquisição, obtêm-se 144 dados de entrada. No entanto, o processo de seleção de variáveis reduz a quantidade de entradas de cada regressor para uma quantidade x_f através do processo descrito na Seção 2.4, priorizando as entradas mais relevantes para a localização cada ímã . Inicialmente, o projetista determina a quantidade de variáveis x_f que serão processadas no cálculo do deslocamento de cada ímã. Posteriormente, análise estatística seleciona essas variáveis com base no cojunto de treinamento, podendo determinar, por exemplo, que o eixo X do primeiro sensor da placa 2 é um dos dados relevantes

para a localização do ímã 1 (SVR_{mm1}). A quantidade de variáveis de entrada (denominadas *features*) impacta diretamente a complexidade da implementação e os resultados obtidos.

Os regressores, portanto, recebem como entrada os valores das *features*, além dos parâmetros α, ρ e dos vetores de suporte x_s, que estão armazenados em memória. Esses dados são específicos para cada regressor. Com essas informações, o regressor calcula o deslocamento do ímã correspondente.

3.4.1 Arquitetura de SVR com *kernel* linear

A arquitetura proposta para a implementação do SVR com *kernel* linear é apresentada na Figura 3.11. O objetivo desses blocos é realizar as operações matemáticas da Equação (2.5), utilizando o *kernel* linear definido na Equação (2.1).

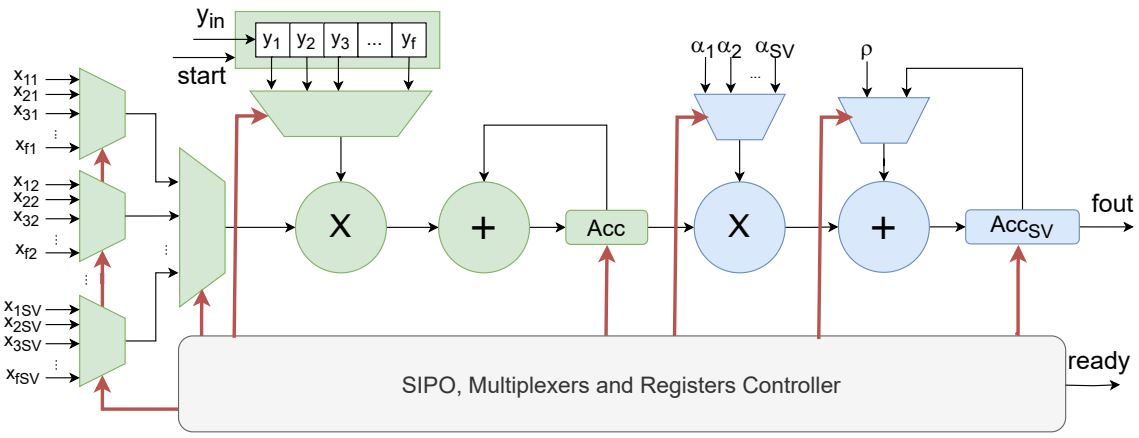


Figura 3.11 – Arquitetura SVR *kernel* Linear

Inicialmente, os elementos de cada vetor de suporte x_{ij} (o índice i representa o vetor de suporte e o índice j um elemento desse vetor) são multiplicados pelas respectivas entradas y_f, que representam as variáveis selecionadas entre os dados adquiridos pelos sensores. Os vetores de suporte estão armazenados em memória, enquanto as entradas y_f são fornecidas dinamicamente ao sistema.

Cada produto entre a uma entrada e um elemento do vetor de suporte é somado ao valor acumulado da operação anterior. Esse processo continua até que todos os índices do vetor de suporte tenham sido processados, implementando assim o cálculo do *kernel* linear (Equação (2.1)).

Em seguida, o valor do *kernel* é multiplicado pelo parâmetro $\alpha_i - \alpha_i^*$, específico de cada vetor de suporte, e o resultado é acumulado. Após o processamento de todos os vetores de suporte, soma-se o termo ρ, comum a todo o regressor.

A multiplexação dos parâmetros e o sincronismo entre os blocos são controlados por um controlador. A saída f_{out} representa o deslocamento calculado pelo regressor.

3.4.2 Arquitetura de SVR com *kernel* RBF

A arquitetura proposta para a SVR com *kernel* do tipo RBF é apresentada na Figura 3.12. Assim como no caso linear, as entradas são os elementos de cada vetor de suporte x_{ij} e y_f, que representam os dados dos sensores mais relevantes para o cálculo de cada ímã.

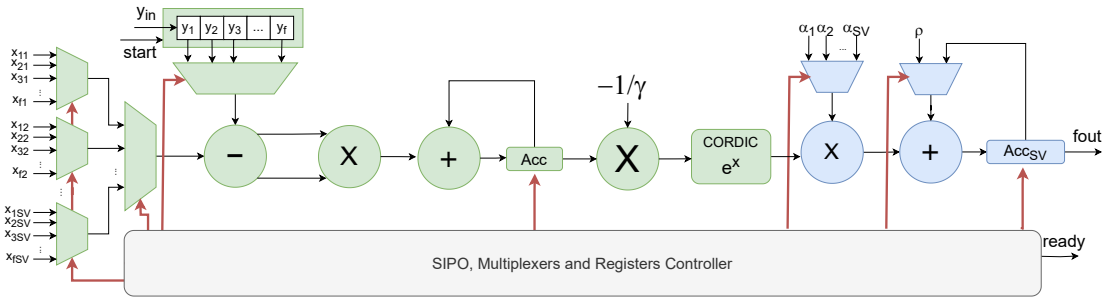


Figura 3.12 – Arquitetura SVR *kernel* RBF

As operações iniciais são utilizadas para a implementação do *kernel* RBF, conforme a Equação (2.2). Essa equação apresenta em seu numerador o quadrado da norma, resultando na acumulação do erro quadrático de todos as entradas. Inicialmente, um elemento do vetor de suporte é subtraído de uma entrada. Em seguida, esse resultado é direcionado ao multiplicador, resultando no erro quadrático. Essas operações são realizadas para cada entrada, acumulando até que o *kernel* tenha sido calculado para todos os elementos do vetor de suporte. Posteriormente, o valor acumulado é multiplicado por uma constante $\frac{1}{\gamma}$ com sinal negativo, conforme proposto na Equação (2.2). Em seguida, utiliza-se técnicas para cálculo de função exponencial, utilizando algoritmo CORDIC e a expansão em série de Taylor para calcular a função exponencial, como proposto em (Muñoz *et al.*, 2010a) e (Ayala *et al.*, 2017).

O resultado da exponencial é a implementação do *kernel* RBF. O *kernel* é multiplicado pelo $\alpha_i - \alpha_i^*$, específico de cada vetor de suporte. Após calcular todos os vetores de suporte e realizar a multiplicação por seus respectivos parâmetros $\alpha_i - \alpha_i^*$, o resultado final é acrescido de um termo constante ρ, compartilhado entre todos os vetores de suporte desse regressor. A saída do regressor corresponde ao deslocamento do ímã por ele monitorado.

3.5 Ferramentas auxiliares

Algumas ferramentas auxiliares são utilizadas no desenvolvimento deste trabalho. NIOTS é uma ferramenta desenvolvida nos estudos apresentados em (Santos, 2019), utilizadas para treinar as SVMs. Além disso, são utilizados IPs de ponto flutuante para implementação

dos regressores em hardware com maior precisão e algoritmo CORDIC para a implementação do *kernel* RBF.

3.5.1 NIOTS

Para obter uma SVM com ampla capacidade de generalização e baixa complexidade, é fundamental a escolha adequada dos hiperparâmetros do modelo. Para realizar essa tarefa, será utilizada a ferramenta NIOTS (*Nature-Inspired Optimization Tools for SVM*), desenvolvida em *Matlab* e baseada em algoritmos bioinspirados, com o objetivo de resolver o problema da otimização de hiperparâmetros (Santos, 2019).

Dentro da ferramenta NIOTS, é necessário configurar alguns parâmetros relacionados ao treinamento da SVM, como o algoritmo bioinspirado, o tipo de *kernel* e a quantidade de iterações. Em seguida, deve-se fornecer como entrada um arquivo contendo o conjunto de dados de treinamento. Como saída, a ferramenta apresenta diferentes combinações entre quantidade de vetores de suporte e erros quadráticos médios (MSE, sigla em inglês). Geralmente, a solução com maior quantidade de vetores de suporte apresenta o menor MSE, porém tende a ser mais complexa de implementar. Assim, o projetista deve selecionar, entre as opções apresentadas, aquela que melhor atende aos requisitos do projeto. Para cada solução, são fornecidos os parâmetros α e ρ. O ambiente gráfico do NIOTS é apresentado na Figura 3.13

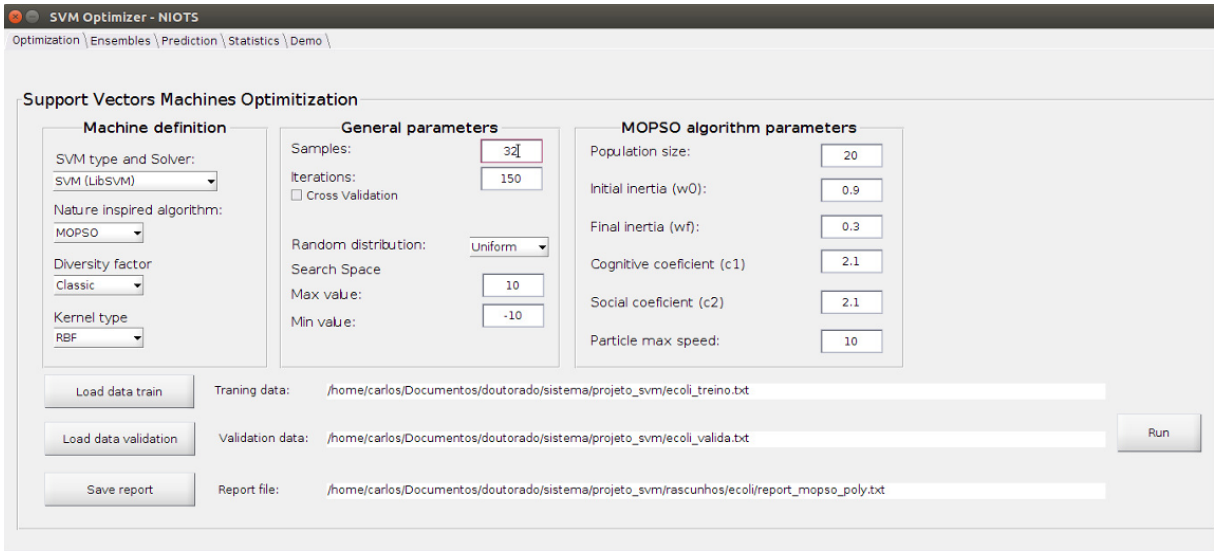


Figura 3.13 – Ambiente Gráfico NIOTS. Retirado de (Santos, 2019)

A cardinalidade, que representa a quantidade de vetores de suporte, está diretamente relacionada à complexidade da implementação do modelo. Quanto maior o número de vetores de suporte, mais complexa tende a ser a implementação. Por outro lado, modelos com maior cardinalidade costumam apresentar erros menores. Assim, o objetivo é selecionar a solução que ofereça um bom equilíbrio entre erro e complexidade, ou seja, que minimize o

erro utilizando uma quantidade reduzida de vetores de suporte, caracterizando um problema multiobjetivo.

3.5.2 Otimização multiobjetivo

O treinamento de SVRs para implementação em hardware configura um problema de otimização multiobjetivo, no qual se busca um compromisso entre dois objetivos principais:

- Reduzir o erro entre o valor estimado pelo regressor e o valor real, avaliado por meio do MSE (*Mean Square Error*);
- Reduzir a quantidade de vetores de suporte, uma vez que os recursos de hardware são limitados, e um número elevado de vetores de suporte implica maior tempo de execução, maior uso de recursos e maior consumo de energia.

Como aumentar o número de vetores de suporte tende a reduzir o erro, enquanto diminuir essa quantidade geralmente aumenta o MSE, os objetivos apresentam natureza conflitante.

Na ferramenta NIOTS, algoritmos selecionam combinações de hiperparâmetros (C, γ e ε), que são utilizadas para treinar diferentes modelos de SVR. O treinamento é realizado utilizando o algoritmo de otimização quadrática SMO (*Sequential Minimal Optimization*). Por se tratar de um problema convexo, uma vez fixados os hiperparâmetros, o treinamento de uma SVR produz uma solução única e determinística.

Diferentes combinações de hiperparâmetros resultam em diferentes modelos de SVR. As melhores soluções em termos de MSE e quantidade de vetores de suporte compõem a fronteira de Pareto, que representa os melhores compromissos possíveis entre ambos os objetivos. Ao longo dessa fronteira encontram-se soluções com MSE muito baixo e muitos vetores de suporte, soluções com poucos vetores de suporte e MSE moderadamente maior, e soluções intermediárias que equilibram esses critérios. Entre as soluções pertencentes à fronteira de Pareto aplica-se o conceito de não dominância, segundo o qual nenhuma solução supera outra simultaneamente em todos os objetivos.

O algoritmo utilizado neste trabalho para selecionar os hiperparâmetros é o MODE (*Multiobjective Differential Evolution*), uma extensão do algoritmo *Differential Evolution* (DE) aplicada a problemas multiobjetivo. Os algoritmos evolutivos constituem uma classe de métodos heurísticos inspirados na Teoria da Seleção Natural, na qual indivíduos mais adaptados possuem maior probabilidade de transmitir suas características às gerações seguintes.

O DE pertence a essa classe e realiza a adaptação da população por meio dos operadores de mutação, recombinação e seleção, aplicados iterativamente ao longo de diversas gerações. Enquanto mutação e recombinação introduzem diversidade ao modificar as características dos indivíduos, o operador de seleção preserva aqueles com melhor desempenho em relação aos objetivos definidos. Essa modelagem matemática do processo evolutivo per-

mite que o DE encontre soluções de alta qualidade para problemas de otimização complexos (Santos, 2019), justificando sua aplicação em cenários de ajuste de hiperparâmetros.

3.5.3 Bibliotecas para operações ariméticas em ponto flutuante e cálculo de funções transcendentais

Operações aritméticas implementadas em FPGA utilizando representação em ponto fixo apresentam alta precisão. No entanto, algumas aplicações, além de precisão, requerem faixa dinâmica. Os blocos somadores e multiplicadores utilizados na implementação da SVR utilizam representação em ponto flutuante por meio das bibliotecas apresentadas em (Muñoz *et al.*, 2010b). O uso da representação em ponto flutuante permite obter faixa dinâmica e alta precisão, possibilitando a representação de números reais tanto grandes quanto pequenos.

Para o cálculo da exponencial utilizada no *kernel* RBF, apresentado na Equação 2.2, é utilizada a biblioteca desenvolvida em (Muñoz *et al.*, 2010a), que implementa o cálculo de funções transcendentais em ponto flutuante de tamanho parametrizável. Nesse trabalho, o cálculo das funções seno, cosseno, arcotangente e exponencial é realizado utilizando o algoritmo CORDIC (*COordinate Rotation Digital Computer*), que permite o cálculo de funções transcendentais por meio de operações de adição, subtração, deslocamento de bits e acesso a tabelas de busca, reduzindo a complexidade de implementação. Implementações semelhantes para cálculo de exponencial com CORDIC são apresentadas em (Ayala *et al.*, 2017).

Para o cálculo da exponencial, a biblioteca alterna automaticamente entre o algoritmo CORDIC e a expansão em série de Taylor de acordo com o argumento de entrada, uma vez que a série de Taylor apresenta melhor desempenho do que o algoritmo CORDIC quando o argumento reduzido está no intervalo entre -0,05 e 0,05 (Muñoz *et al.*, 2010a).

3.6 Conclusão do capítulo

Este capítulo apresentou uma visão geral da solução proposta, detalhando os dois objetivos principais: emular uma interface miocinética e localizar ímãs por meio de uma SVR implementada em FPGA. Foi apresentado o hardware e a plataforma em formato de antebraço utilizada neste trabalho, além da modelagem dos movimentos de rampa e senoide a serem implementados no firmware de controle dos motores. As arquiteturas desenvolvidas para a implementação da SVR com *kernel* linear e com *kernel* RBF foram descritas em detalhes, assim como a arquitetura geral, que abrange o processo de direcionamento das variáveis mais relevantes para cada regressor. Também foram apresentadas as ferramentas auxiliares utilizadas: NIOTS, empregada no treinamento da SVR para obtenção dos vetores de suporte e demais parâmetros; e as bibliotecas responsáveis pela implementação de operações aritméticas em ponto flutuante e cálculos de funções transcendentais em hardware.

4 Resultados

Os principais resultados apresentados neste trabalho são: a construção de um *dataset* para o treinamento de modelos de aprendizado de máquina; o treinamento de modelos SVR para estimar a posição de ímãs utilizando a técnica de seleção univariada de variáveis; e a implementação desses modelos em hardware.

A etapa inicial consistiu na construção da base de dados, considerando a necessidade de uma plataforma funcional capaz de emular uma interface miocinética. Com os dados coletados, foram treinados modelos de SVR específicos para esse conjunto de amostras e, posteriormente, implementados em hardware. Além disso, com o objetivo de comparar o desempenho da técnica de seleção de *features* aplicada ao SVR com metodologias utilizadas em trabalhos anteriores — como os apresentados em (Clemente *et al.*, 2019), (Mendez *et al.*, 2024), (Mendez *et al.*, 2022) e (Mendes, 2024) — foram treinados modelos adicionais utilizando o mesmo *dataset* empregado nesses estudos. Esse *dataset* foi originalmente desenvolvido pelo Instituto de Bio-robótica da *Scuola Superiore Sant'Anna*.

Todo o desenvolvimento desse trabalho, com exeção do treinamento dos modelos por meio da ferramenta NIOTS, foi realizado no Sistema Operacional Windows 11 versão 25H2.

4.1 Construção da base de dados para treinamento e validação dos modelos

Durante a construção da base de dados, alguns resultados intermediários foram alcançados:

- Desenvolvimento de uma interface gráfica capaz de receber valores e comandos do usuário para a execução dos movimentos;
- Desenvolvimento do firmware da unidade de controle para receber esses parâmetros por BLE e executar os movimentos conforme solicitado;
- Adaptação do firmware da unidade de aquisição para enviar as amostras com base em um *trigger* para o desenvolvimento da base de dados e para transmitir os valores normalizados e representados em ponto flutuante de 16 bits, conforme o padrão IEEE 754, para a validação dos modelos;
- Implementação do protocolo SPI em VHDL, instanciando quatro barramentos independentes para a recepção simultânea dos dados das quatro placas;
- Desenvolvimento de códigos em *Matlab* para decodificar os valores de saída e anexar as saídas esperadas.

4.1.1 Interface gráfica para controle da interface miocinética

Como proposto no Capítulo 3, foi implementado um ambiente gráfico para que o usuário insira os valores de frequência, amplitude e tipo de movimento, além de um botão para iniciar ou parar o movimento. Uma demonstração completa da utilização deste recurso está disponível no link[1].

Para implementar esse ambiente gráfico, foi desenvolvido um aplicativo *desktop*. O *front-end* desse aplicativo foi criado utilizando o software Qt Designer, versão 5.11.1, enquanto o *back-end* foi desenvolvido em *Python* versão 3.13.8, utilizando principalmente a biblioteca PySide6.

A conexão entre o computador em que o usuário insere os dados e a unidade de controle é realizada por *Bluetooth Low Energy* (BLE), utilizando o protocolo GATT. Portanto, a tela inicial do aplicativo é destinada ao processo de conexão. O usuário deve digitar o nome do dispositivo BLE ao qual deseja se conectar, no caso deste firmware configurado como *"MYKI"*, e pressionar o botão *"CONECTAR"*. A tela inicial é apresentada na Figura 4.1.



Figura 4.1 – Tela inicial do aplicativo

Ao estabelecer a conexão, o usuário tem duas opções:

- Realizar movimentos personalizados, nos quais escolhe a frequência, a amplitude e a forma de onda (rampa ou senoide) de cada motor, podendo alterar esses parâmetros a qualquer momento e selecionar quais motores se movimentam e quais permanecem

[1] https://www.youtube.com/watch?v=23cnu7RmyKM

parados.

- Selecionar movimentos padronizados, como a emulação dos movimentos musculares que representam uma mão abrindo ou fechando, movimentos de pinça, agarre, entre outros. Ao selecionar esses movimentos, os parâmetros de cada motor já são pré-definidos no firmware para a execução do movimento desejado.

Cabe enfatizar que os movimentos implementados até o momento não possuem relação anatômica fidedigna. O objetivo principal é disponibilizar uma estrutura funcional de controle dos motores, permitindo que trabalhos futuros possam modelar de forma mais adequada os padrões musculares associados a cada tipo de movimento.

A Figura 4.2 apresenta a tela *"MOVIMENTOS PERSONALIZADOS"*. Nessa tela, observam-se oito campos destinados ao preenchimento dos parâmetros, um para cada motor. Em cada campo há um *checkbox*, cuja função é ativar o motor para participar do movimento, caso esteja marcado, ou mantê-lo parado independentemente dos valores de amplitude e frequência, caso esteja desmarcado. Há também duas barras de rolagem em cada uma das oito regiões, sendo uma destinada à configuração da amplitude e outra à configuração da frequência do movimento. Por fim, há uma lista de opções que permite alternar a forma de onda entre rampa e senoide.



Figura 4.2 – Tela de movimentos personalizados

A Figura 4.3 apresenta a tela *"MOVIMENTOS PADRONIZADOS"*. Nessa tela, a única ação necessária é selecionar o movimento desejado, que já possui os parâmetros implementados no firmware.

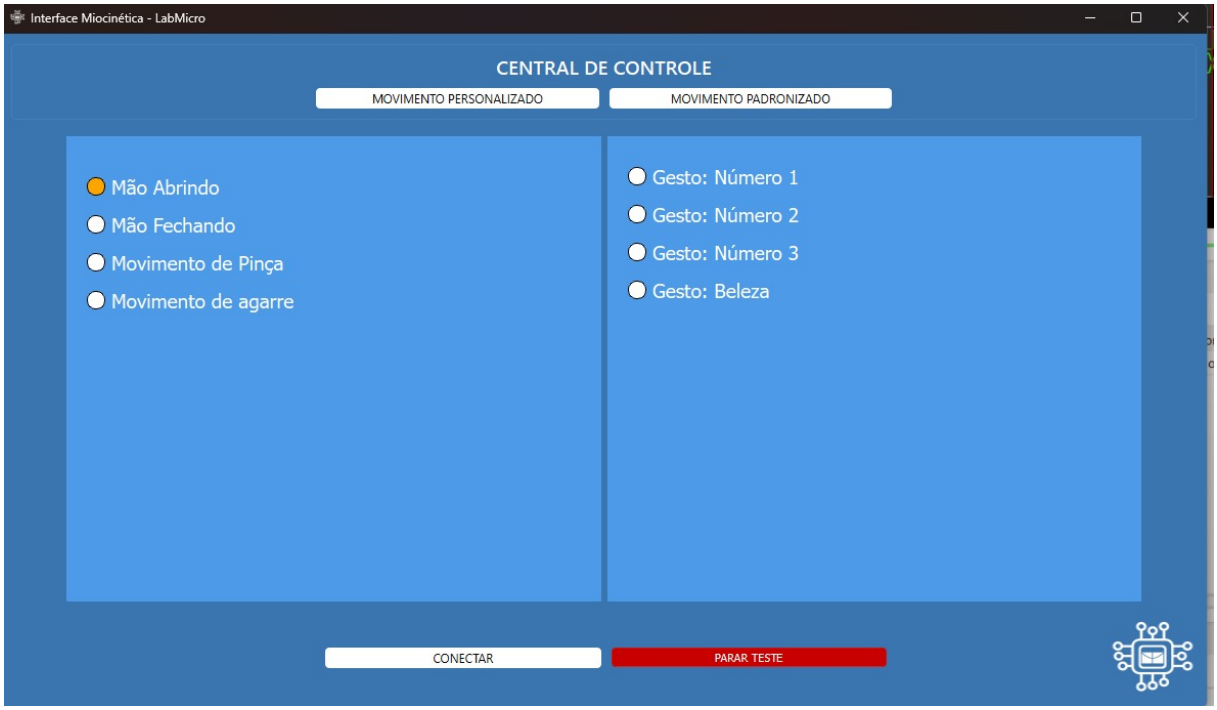


Figura 4.3 – Tela de movimentos padronizados

Em ambas as telas há dois botões na parte inferior. O botão *CONECTAR* serve para retornar para a tela inicial e reestabelecer a conexão *Bluetooth*, caso tenha sido perdida. O botão *INICIAR TESTE* inicia o movimento dos motores. No momento da coleta de dados, esse botão é uma condição para a aquisição dos dados, pois o *trigger* é ativado somente quando o teste está ativo. Ao pressionar esse botão, ele muda a cor para vermelho e o texto para *PARAR TESTE*, como apresentado na Figura 4.3, ao clicar nesse botão o movimento é pausado e o retorna ao texto e cor mostrados na Figura 4.2.

Esse software é disponibilizado na forma de um aplicativo executável, de modo que os pesquisadores precisam apenas clicar no ícone e ajustar os parâmetros conforme desejarem. Tanto o executável quanto o código fonte do aplicativo estão disponíveis no repositório deste link[2].

4.1.2 Firmware da unidade de controle

O firmware da Unidade de Controle foi desenvolvido na IDE Visual Studio Code, versão 1.104.3, utilizando o *framework* ESP-IDF na versão 4.4.7 e implementado em um kit de desenvolvimento ESP32, como mostrado na Figura 3.5. O código do firmware está disponível no repositório deste link[3].

Os principais objetivos do firmware são implementar os movimentos modelados

[2] https://github.com/fsidrone/Myokinetic-Interface-Emulator/tree/main/controlSoftware
[3] https://github.com/fsidrone/Myokinetic-Interface-Emulator/tree/main/controlUnit

pelas Equações 3.4 e 3.5, além de receber, por meio de Bluetooth Low Energy, os parâmetros configurados pelo usuário no software apresentado na Seção 4.1.1.

Para o recebimento dos dados, utilizou-se os recursos disponíveis no microcontrolador, que ja suportam a conexão BLE, sendo necessário somente a programação do sistema embarcado para implementar corretamente essa funcionalidade e interpretar os dados recebidos pelo usuário da maneira correta. Esse código foi desenvolvido em linguagem C.

Para que o recebimento dos dados ocorra de forma simultânea ao movimento dos motores, podendo o usuário interferir no movimento a qualquer momento, foram implementados os recursos de *Threads* do *FreeRTOS*.

Nesse firmware foi utilizada programação orientada à objetos, sendo criada uma classe em linguagem C++ para a representação dos motores. A classe *Motor* implementa a lógica de controle de um atuador acionado por PWM, encapsulando tanto os parâmetros de configuração quanto os métodos responsáveis pela geração dos diferentes tipos de movimentos. Entre seus atributos privados estão o pino utilizado pelo motor (*pin*), o canal configurado no temporizador LEDC (*channel*), o identificador da unidade (*id*), além dos parâmetros ajustáveis de controle, como *amplitude*, *frequency*, *waveform*, *direction* e o tipo de movimento desejado. A classe mantém ainda variáveis internas que representam a posição atual do motor (*position*).

Um método auxiliar privado realiza a conversão de um deslocamento linear para o ângulo correspondente. Entre os métodos públicos estão os *setter* e *getter* para configuração e leitura dos parâmetros de operação, além do método *pwmConfig()*, responsável por inicializar o PWM via driver LEDC do ESP32. Também são fornecidos métodos de movimento específicos, como *sineMovement()*, *rampMovement()*, *handOpening()* e *handClosing()*, permitindo executar desde movimentos livres de rampa e senoide até padrões definidos, como a representação de movimentos musculares que emulam uma mão abrindo ou fechando.

4.1.3 Implementação do protocolo SPI entre FPGA e unidades de aquisição

Para esta aplicação, foi necessária a implementação do protocolo SPI em VHDL. Os barramentos SPI disponíveis nativamente na placa *ZedBoard* são mapeados ao processador ARM, e este projeto utiliza exclusivamente os recursos de lógica programável, sem empregar o processador. Além disso, a quantidade de barramentos SPI disponível no *SoC* Zynq-7000 é insuficiente para atender às quatro placas utilizadas.

Foram instanciados quatro protocolos SPI em VHDL utilizando o software Vivado, versão 2019.2. Cada barramento SPI recebe 576 bits, pois cada placa possui 12 sensores ativos e cada sensor envia 16 bits para cada um dos seus três eixos (X, Y e Z). A frequência de amostragem utilizada foi de 40 kHz.

Para a programação das unidades de aquisição, utilizou-se a plataforma Arduino IDE. No *Gerenciador de Placas*, foi instalada a biblioteca ESP32, disponibilizada pela Espressif

Systems, na versão 2.0.4. Essa versão é fundamental para o correto funcionamento do projeto, pois algumas versões superiores deixaram de oferecer suporte ao modo SPI *half-duplex*, utilizado na comunicação entre os sensores e o microcontrolador de cada unidade de aquisição. Dessa forma, embora o código compile normalmente sem indicar erros, o firmware não funciona conforme esperado nas versões mais recentes, já que os sensores não apresentam qualquer resposta.

Cada unidade de aquisição opera como *controller* e o FPGA como *peripheral*. Ao receber 576 bits de cada uma das quatro placas de forma simultânea, o FPGA concatena esses dados, formando um *frame* de 2304 bits. Para a construção da base de dados, esses valores são direcionados ao ILA, sendo posteriormente salvos em formato *csv*, gerando os arquivos que compõem a base de dados e que são decodificados por códigos em *Matlab*, essse fluxo de dados é apresentado na Figura 4.4. Para a execução dos testes, esses 2304 bits são direcionados ao recurso de seleção de *features* (FIFO), apresentado na Figura 3.10, e as *features* de cada ímã são encaminhadas aos seus respectivos regressores para o cálculo do deslocamento.

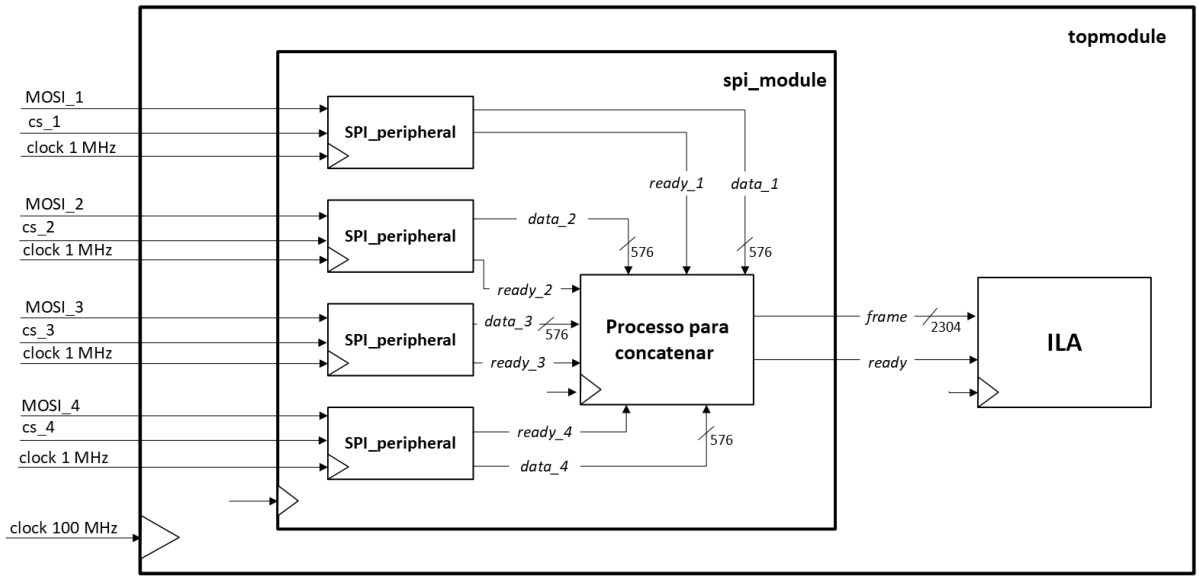


Figura 4.4 – Diagrama de blocos para implementação dos módulos SPI em VHDL.

Implementou-se, no FPGA e na placa de aquisição, o protocolo SPI MODE 0, que define configurações específicas de fase e polaridade do clock. A comunicação opera no modo *MSBFIRST*, no qual o bit mais significativo (MSB) de cada byte é transmitido primeiro.

Como observado na Figura 4.4, foram utilizados apenas três sinais em cada barramento, pois o sinal *MISO* do SPI não foi necessário, já que o FPGA não envia informações para as placas de aquisição. Quando o sinal *CS* é acionado em nível lógico baixo, a transmissão é iniciada. A cada ciclo de *clock*, que nesse experimento foi definido na frequência 1 MHz, as placas de aquisição enviam 1 bit para o FPGA. Após a transmissão dos 576 bits, o sinal *CS* retorna ao nível lógico alto, encerrando a comunicação. A Figura 4.5 apresenta a

conexão física entre as unidades de aquisição e a placa *ZedBoard*, como proposto na Figura 3.2.



Figura 4.5 – Conexão física entre a FPGA e as placas de aquisição como parte da implementação do protocolo SPI.

A cada transmissão completa, cada unidade de aquisição apresenta um sinal individual de *ready* em nível lógico alto. Quando as quatro placas sinalizam que a transmissão foi concluída, o *ready* geral vai para nível lógico alto, indicando que os 2304 bits (576 de cada unidade de aquisição) foram recebidos.

No contexto da construção da base de dados, esse sinal de *ready*, que indica a recepção dos 2304 bits, é configurado como *trigger* do ILA a cada borda de subida. Ao instanciar o *IP* do ILA, deve-se selecionar a opção de *controles avançados* para que seja possível dividir as amostras em janelas. Na opção *NUMBER OF WINDOWS*, configura-se a quantidade total de amostras que se deseja coletar. Na opção *WINDOW DATA DEPTH*, configura-se quantas amostras serão coletadas a cada *trigger*, sendo no mínimo duas. Na opção *TRIGGER POSITION IN WINDOW*, é possível realizar a configuração para obter amostras imediatamente antes do *trigger*, se necessário.

Caso a opção de configuração avançada não seja selecionada no momento da geração

do IP, essas opções não estarão disponíveis no ILA. Assim, após o primeiro *trigger*, o ILA passará a capturar uma amostra a cada ciclo de *clock*, resultando em várias amostras repetidas e exigindo mais recursos computacionais para a coleta dos dados necessários.

As configurações mencionadas, bem como a configuração do *trigger*, são apresentadas na Figura 4.6. Na parte inferior observa-se as configurações mencionadas e na parte superior um exemplo de coleta de dados realizada, sendo possível a visualização dos dados sendo capturados a cada *trigger*. Uma demonstração da coleta de dados das unidades de aquisição utilizando o protocolo SPI implementado em FPGA é apresentada neste link[4].

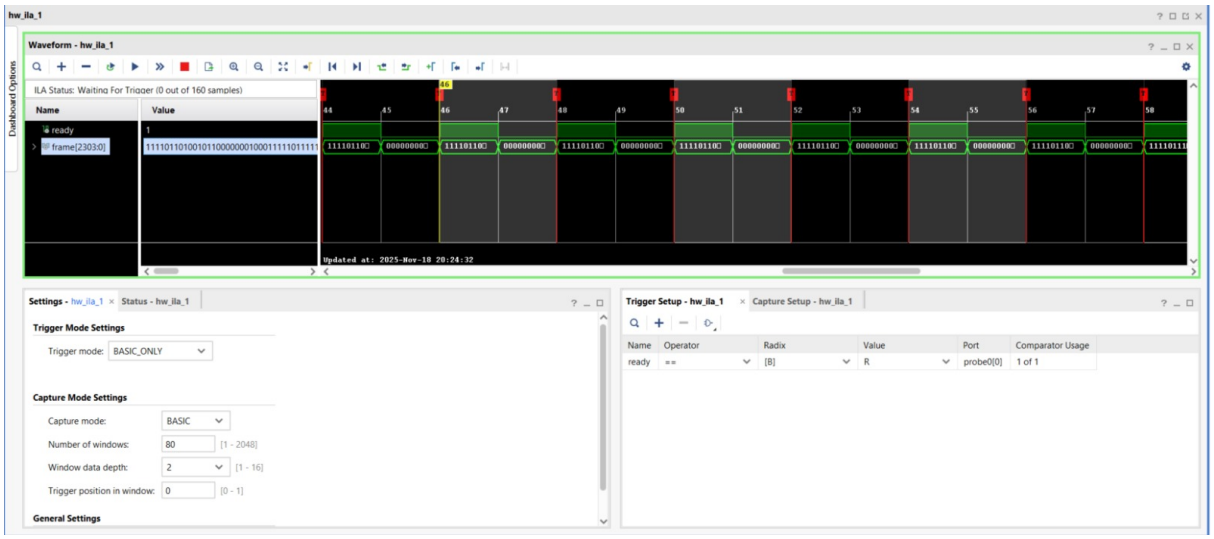


Figura 4.6 – Configuração do ILA e demonstração de uma coleta de dados

Como apresentado na Figura 4.4 na implementação do SPI há dois domínios de *clock*: o *clock* da FPGA de 100 MHz e o *clock* do microcontrolador, implementado na frequência de 1 MHz mas esse valor pode ser alterado. Os módulos SPI foram implementados como *peripheral*, logo, o módulo SPI na FPGA opera conforme a frequência de *clock* do microcontrolador que exerce a função de *controller*. A Figura 4.7 apresenta a conformidade dos tempos de *setup* e *hold* do RTL considerando os dois domínios mencionados.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,441 ns	Worst Hold Slack (WHS):	0,035 ns	Worst Pulse Width Slack (WPWS):	3,750 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	51034	Total Number of Endpoints:	51006	Total Number of Endpoints:	23182

All user specified timing constraints are met.

Figura 4.7 – Análise de timing do módulo SPI.

[4] https://youtu.be/0y4CcY7Hp54?si=9GTByT_5_-4lRsTQ

A Figura 4.8 apresenta o consumo de potência total para a implementação de coleta de dados com SPI, sendo 0.147 W para a parte dinâmica e 0.116W para a parte estática, totalizando 0.263 W.

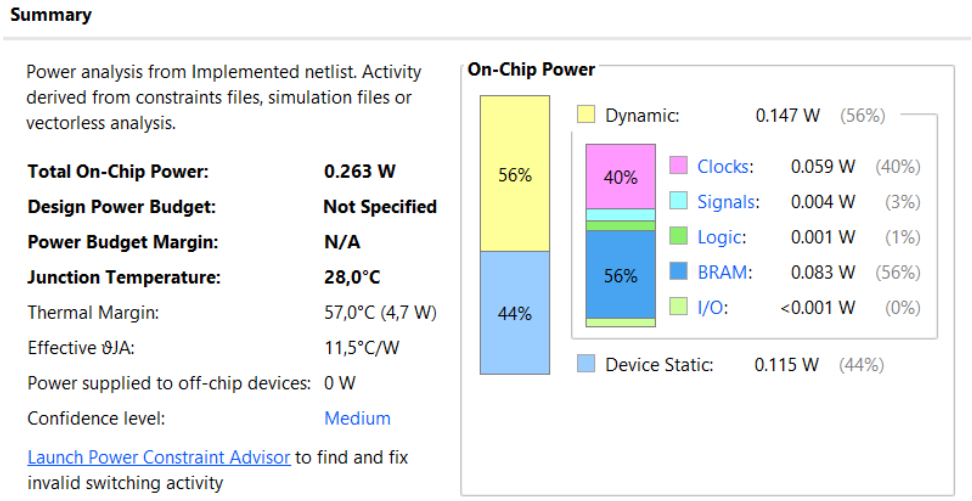


Figura 4.8 – Consumo de potência módulo SPI implementado em conjunto com o ILA.

A Figura 4.9 considera o ILA, que foi um recurso utilizado durante a coleta de dados. No entanto, para uma implementação dos regressores é necessário desconsiderar o ILA pois a saída dos módulos SPI é direcionada para a entrada dos regressores, de modo que o ILA não estára presente em uma implementação final. Dessa forma, a Figura 4.9 apresenta uma análise de potência de cada módulo do projeto. Observa-se que cada módulo SPI consumiu 0.002 W de potência, totalizando 0.008 W para as quatro unidades de aquisição.

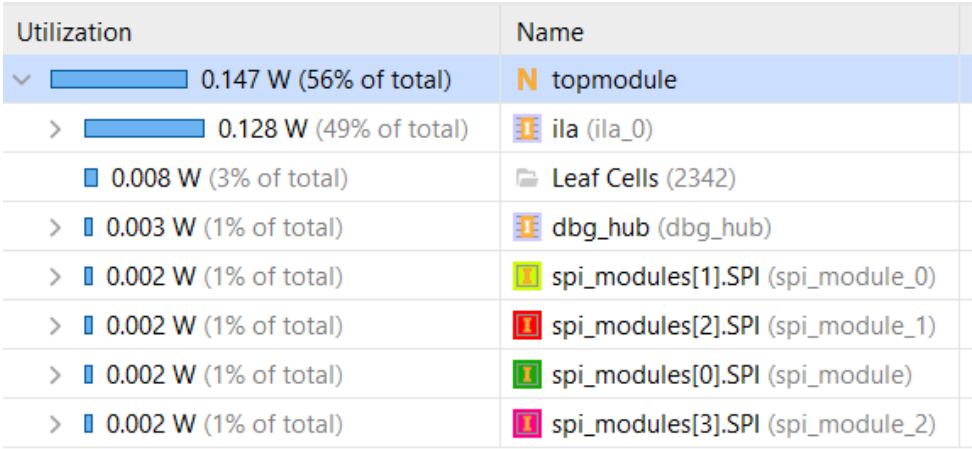


Figura 4.9 – Consumo de potência de cada módulo SPI individualmente e do ILA.

A estratégia de receber os 576 bits de cada placa de aquisição foi adequada para a fase de coleta de dados, pois nesse caso o FPGA foi utilizado exclusivamente para essa função.

Entretanto, em uma implementação final, na qual o módulo SPI opere em conjunto com os regressores, pode ser mais vantajoso receber os dados em blocos de 16 bits e encaminhá-los sequencialmente aos regressores. Módulos que manipulam 576 bits de uma só vez consomem uma quantidade considerável de recursos do FPGA, o que é um fator crítico no projeto de hardware. Os códigos HDL foram implementados de forma parametrizável, sendo necessário alterar somente a quantidade de bits recebidas para obter um módulo mais compacto.

Na Tabela 2.3 foi avaliado o tempo de transmissão entre as unidades de aquisição e a unidade de processamento, que nesse caso é o FPGA. Observa-se que, no trabalho apresentado em (Clemente *et al.*, 2019), que utiliza um processador ARM, o tempo de transmissão foi maior do que nos casos em que se utiliza FPGA. Isso ocorre porque o FPGA permite a recepção dos dados de todas as placas de aquisição em paralelo, de modo que a quantidade de placas de aquisição não influencia no tempo de transmissão.

Os trabalhos apresentados em (Mendez *et al.*, 2022), (Mendez *et al.*, 2024) e (Mendes, 2024) consideram a recepção paralela, pois suas implementações são realizadas em FPGA considerando o protocolo I2C. Entretanto, este trabalho apresenta a implementação do protocolo SPI em FPGA para a recepção de quatro unidades de aquisição em paralelo, fornecendo uma estimativa mais precisa do consumo de potência dessa implementação.

O tempo de transferência entre as unidades de aquisição e a unidade de coleta utilizando o protocolo I2C é de 23,9 ms (Clemente *et al.*, 2019). Com a implementação do protocolo SPI operando a 1 MHz, obtém-se uma taxa de transmissão de 1000000 bits/s. Dessa forma, o tempo necessário para que cada placa envie os seus 576 bits é de 576 μs. Caso a recepção seja realizada em paralelo, esse tempo permanece constante independentemente do número de unidades de aquisição; caso contrário, a cada unidade adicional soma-se um atraso de 576 μs.

4.1.4 Dataset gerado a partir dos dados de variação de campo magnético

O *dataset* gerado é utilizado neste trabalho para aplicar a técnica de seleção de *features* e, posteriormente, treinar os modelos de SVR. No entanto, esse *dataset* pode ser empregado em outros trabalhos que explorem técnicas distintas para estimar o deslocamento de ímãs.

O *dataset* é dividido em dois grupos: o conjunto de treinamento, utilizado para treinar os modelos; e o conjunto de validação, utilizado para validar os modelos treinados, apresentando dados que não estão contidos no treinamento, de modo a verificar a capacidade de generalização. Uma explicação completa da estrutura e organização da base de dados gerada é apresentada neste link[5].

Os conjuntos de treinamento e validação possuem os mesmos tipos de movimento para todos os ímãs, mas cada um com suas próprias amostras. Por exemplo, no conjunto de

[5] https://youtu.be/CaDbMZc421M?si=3O-cMoUSHLR4raCR

treinamento do ímã 1 existem dados relacionados ao movimento desse ímã em forma de rampa, com amplitude de 10 mm e frequência de 1 Hz. No conjunto de treinamento do ímã 2, há esse mesmo movimento em rampa, mas, nesse caso, quem está em movimento é o ímã 2, de modo que os dados de variação do campo magnético são distintos. Após a validação os modelos podem ser testados, implementando os modelos treinados em hardware e verificando as saídas para diferentes movimentos que podem ser configurados na plataforma que emula um antebraço, apresentada anteriormente.

Foram construídos um conjunto de treinamento e um de validação para cada ímã. Sendo 8 ímãs, foram gerados 16 conjuntos de dados no total.

Cada conjunto de treinamento é composto por:

- Todos os motores parados para coletar amostras na posição inicial;
- Movimento de rampa com amplitude 3 mm e frequência 1 Hz;
- Movimento de rampa com amplitude 3 mm e frequência 1 Hz (amostra 2);
- Movimento de rampa com amplitude 6 mm e frequência 2 Hz;
- Movimento de rampa com amplitude 6 mm e frequência 2 Hz (amostra 2);
- Movimento de rampa com amplitude 10 mm e frequência 1 Hz;
- Movimento de rampa com amplitude 10 mm e frequência 1 Hz (amostra 2);
- Todos os motores parados (amostra 2);
- Movimento de senoide com amplitude 2 mm e frequência 1 Hz;
- Movimento de senoide com amplitude 2 mm e frequência 1 Hz (amostra 2);
- Movimento de senoide com amplitude 6 mm e frequência 2 Hz;
- Movimento de senoide com amplitude 6 mm e frequência 2 Hz (amostra 2);
- Movimento de senoide com amplitude 9 mm e frequência 1 Hz;
- Movimento de senoide com amplitude 9 mm e frequência 1 Hz (amostra 2);
- Outros motores se movimentando, enquanto o motor em questão permanece parado.

Nos momentos em que há um movimento de rampa ou senoide, apenas o ímã em questão é movimentado. Imagine que está sendo construído o conjunto de treinamento para localizar o ímã 1, somente esse ímã executa os movimentos mencionados enquanto os outros permanecem parados. A última amostra corresponde ao movimento de todos os outros ímãs, com exceção do ímã 1, nesse exemplo, para que o modelo entenda que, enquanto outros ímãs se movimentam, o ímã em questão pode permanecer parado. A Figura 4.10 apresenta de forma gráfica a posição do ímã durante as coletas de dados para o conjunto de treinamento.

Cada conjunto de validação é composto por:

- Todos os motores parados, posição inicial;

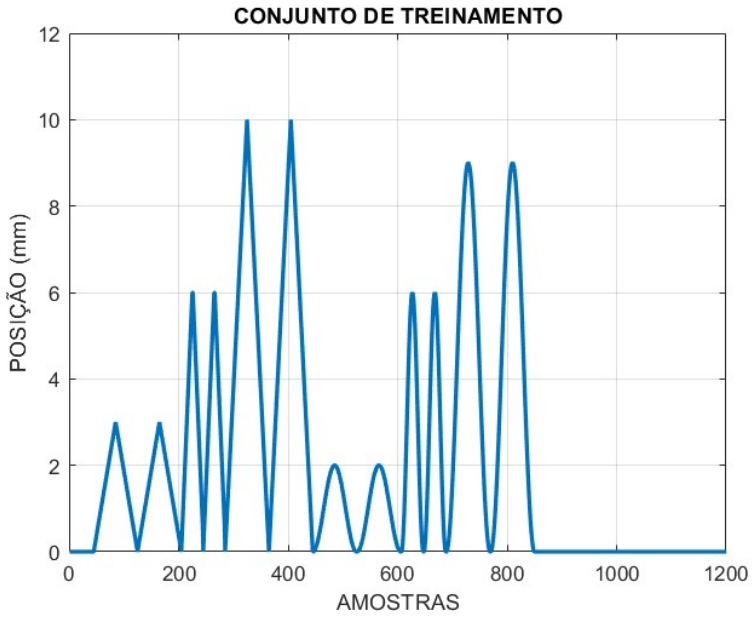


Figura 4.10 – Exemplo de conjunto de treinamento para o ímã 1.

- Movimento de rampa com amplitude 4 mm e frequência 2 Hz;
- Movimento de rampa com amplitude 4 mm e frequência 2 Hz (amostra 2);
- Movimento de rampa com amplitude 5 mm e frequência 1 Hz;
- Movimento de rampa com amplitude 5 mm e frequência 1 Hz (amostra 2);
- Movimento de rampa com amplitude 8 mm e frequência 2 Hz;
- Movimento de rampa com amplitude 8 mm e frequência 2 Hz (amostra 2);
- Movimento de senoide com amplitude 4 mm e frequência 1 Hz;
- Movimento de senoide com amplitude 4 mm e frequência 1 Hz (amostra 2);
- Movimento de senoide com amplitude 8 mm e frequência 2 Hz;
- Movimento de senoide com amplitude 8 mm e frequência 2 Hz (amostra 2);
- Movimento de senoide com amplitude 10 mm e frequência 1 Hz;
- Movimento de senoide com amplitude 10 mm e frequência 1 Hz (amostra 2);
- Outros motores se movimentando, enquanto o motor em questão permanece parado.

A Figura 4.11 apresenta de forma gráfica a posição do ímã durante as coletas de dados para o conjunto de validação.

Além dos cenários em que os ímãs se movimentam individualmente, também foram coletadas amostras em que dois ou mais ímãs se deslocam simultaneamente. Embora, no presente estudo, o treinamento dos modelos considere apenas a movimentação de um ímã por vez, essa base de dados pode ser utilizada em trabalhos futuros que proponham melhorias

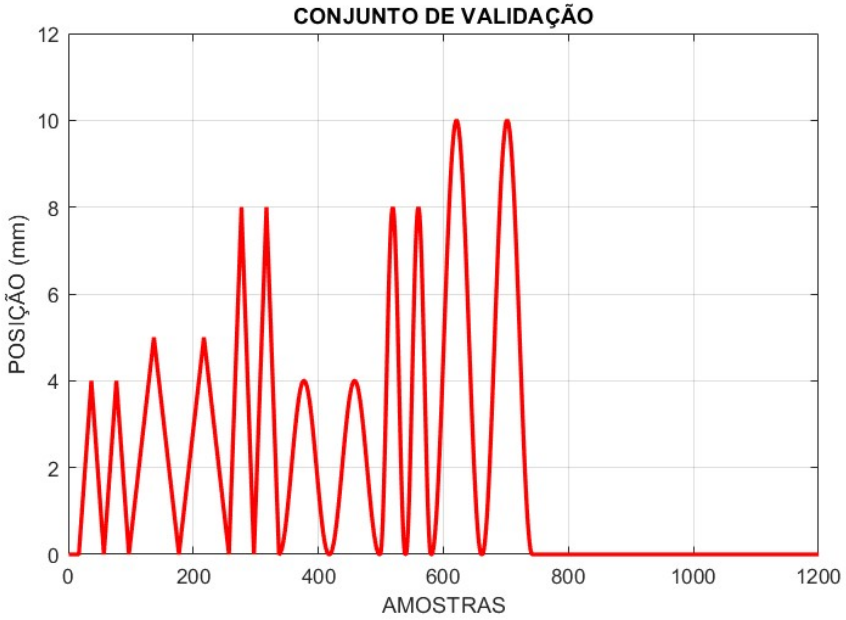


Figura 4.11 – Exemplo de conjunto de validação para o ímã 1.

ou novas abordagens. A seguir, apresentam-se alguns exemplos de movimentos em que foram obtidas amostras de deslocamento simultâneo:

- Dois ou mais motores com movimento de rampa;
- Dois ou mais motores com movimento de senoide;
- Alguns motores em movimento de rampa e outros com movimento de senoide.

Além desses movimentos aleatórios utilizados para validar os algoritmos implementados verificando o desempenho com mais de um ímã se movendo simultaneamente, também foram definidos alguns padrões de movimento, como o movimento de mão abrindo, em que todos os ímãs se deslocam em movimento de senoide da posição inicial até a amplitude máxima, e o movimento de mão fechando, considerando o mesmo deslocamento no sentido contrário. O objetivo deste estudo não é estabelecer uma correlação exata entre o deslocamento dos ímãs e o movimento anatômico, mas deixar a estrutura preparada para que trabalhos futuros possam realizar essa etapa.

Na Figura 4.3 observam-se alguns dos padrões de movimento implementados na interface gráfica. As modelagens de rampa e senoide também estão integradas ao firmware apresentado na Seção 4.1.2, com parametrização de amplitude e frequência de modo a possibilitar uma fácil inserção de novos movimentos.

Para inserir padrões de movimento anatomicamente coerentes, será necessário posicionar os ímãs nas angulações corretas referentes a cada músculo estudado e obter informações sobre quais músculos são ativados em determinados movimentos, como, por exemplo, o movimento de pinça. No firmware, basta ativar os motores relacionados ao movimento desejado, definir se eles se moverão em rampa, senoide ou em uma combinação de ambas as

formas, e configurar a frequência e amplitude de cada motor. Dessa maneira, o padrão de movimento estará implementado.

4.2 Treinamento dos modelos SVR

O treinamento dos modelos foi realizado por meio da ferramenta NIOTS. Dentro dessa ferramenta é realizada uma otimização multiobjetivo baseada em meta-heurística. Os hiperparâmetros C e ε são definidos antes do treinamento da SVR, que é a etapa em que os pesos α são calculados. Esses valores não são modificados durante o treinamento, mas influenciam diretamente o resultado final.

A meta-heurística consiste em testar combinações de C e ε, treinar a SVR com cada combinação e avaliar o desempenho obtido. Como resultado, a ferramenta NIOTS gera uma fronteira de Pareto contendo diferentes soluções possíveis, cada uma com um conjunto de valores ótimos de α e seu respectivo MSE.

Para a obtenção dos hiperparâmetros foi utilizado o algoritmo MODE (Multi-Objective Differential Evolution), um método bioinspirado aplicado à otimização multiobjetivo. Nesse processo, foram empregadas 20 partículas, fator de escala igual a 0.7, taxa de cruzamento igual a 0.4 e 150 iterações. O treinamento das SVRs é realizado no NIOTS por meio da biblioteca *libSVM*, que utiliza internamente o algoritmo SMO (Sequential Minimal Optimization) para calcular os pesos dos regressores. A Figura 4.12 apresenta a configuração dos parâmetros e algoritmos utilizados para o treinamento dos modelos.

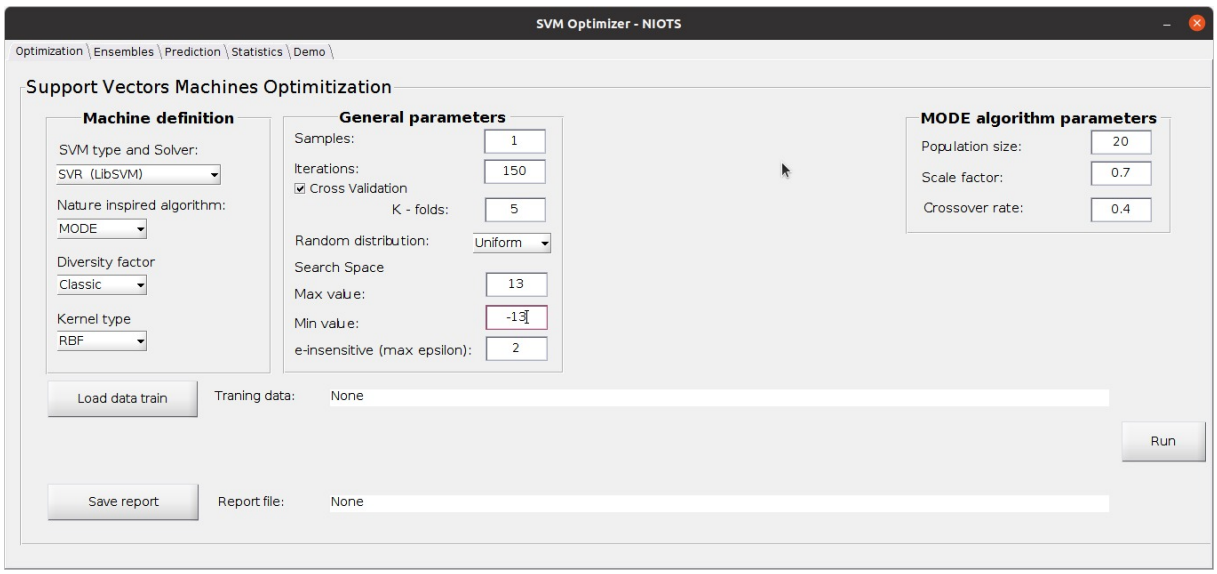


Figura 4.12 – Configuração do NIOTS para treinamento dos modelos.

A Figura 4.13 apresenta um exemplo de resultado de treinamento apresentado pelo NIOTS. Na coluna da esquerda observa-se o índice, útil para encontrar o arquivo que contém os vetores de suporte relacionado a essa linha. Em seguida observa-se o MSE em mm^2 e o

MSE em decibéis. Na quarta coluna observa-se a correlação, que mede quanto o modelo consegue acompanhar a variação dos valores reais e a última coluna é a quantidade de vetores de suporte. A linha amarela é uma escolha de projeto, levando em conta o MSE, a correlação e a quantidade de vetores de suporte, feita pelo projetista após analisar os resultados gerados pelo NIOTS, levando em conta a disponibilidade de recursos e os requisitos da aplicação

Index	MSE	MSE_dB	R	SV
2	1,701189162	5,313275133	0,935316176	3
3	1,102591648	0,976634527	0,939215316	4
15	1,047890454	0,467790517	0,939594369	4
14	0,966057483	-0,345319402	0,940264752	5
5	0,911978738	-0,921386031	0,94079743	5
6	0,780992738	-2,471894272	0,944283046	7
10	0,682621659	-3,818145126	0,948772425	9
28	0,73748483	-3,045097607	0,94629187	10
13	0,571895766	-5,587985316	0,95459873	11
11	0,528964037	-6,368348322	0,956493253	13
31	0,388475257	-9,455258008	0,964427931	13
19	0,42060741	-8,660553989	0,962595999	14
30	0,326520575	-11,19262315	0,967786715	15
40	0,270821043	-13,06297035	0,970860646	18
35	0,254550087	-13,68257656	0,972483963	21
20	0,199370187	-16,12591947	0,976198984	25
8	0,150730374	-18,92262644	0,975929922	47

Figura 4.13 – Exemplo de resultados do treinamento apresentado pelo NIOTS.

4.2.1 Treinamento dos modelos sem seleção de *features*

Os modelos RBF e linear foram treinados inicialmente sem utilizar a técnica de seleção de *features*, isso significa que o regressor treinado processa todas as 144 entradas enviadas das unidades de aquisição, consequentemente, cada vetor de suporte possui 144 elementos. Esse treinamento foi realizado com o conjunto de treinamento apresentado na Figura 4.10. A Figura 4.14 apresenta o resultados dos testes para os ímãs de 1 a 4 e a Figura 4.15 para os ímãs de 1 a 8.

No eixo vertical de cada figura observa-se a posição em milímetros e no eixo horizontal as amostras. Representado pela linha vermelha está o conjunto de validação apresentado na Figura 4.11, em preto o SVR com *kernel* RBF e em azul o SVR com *kernel* linear. Em cima de cada gráfico está identificado qual regressor representa.

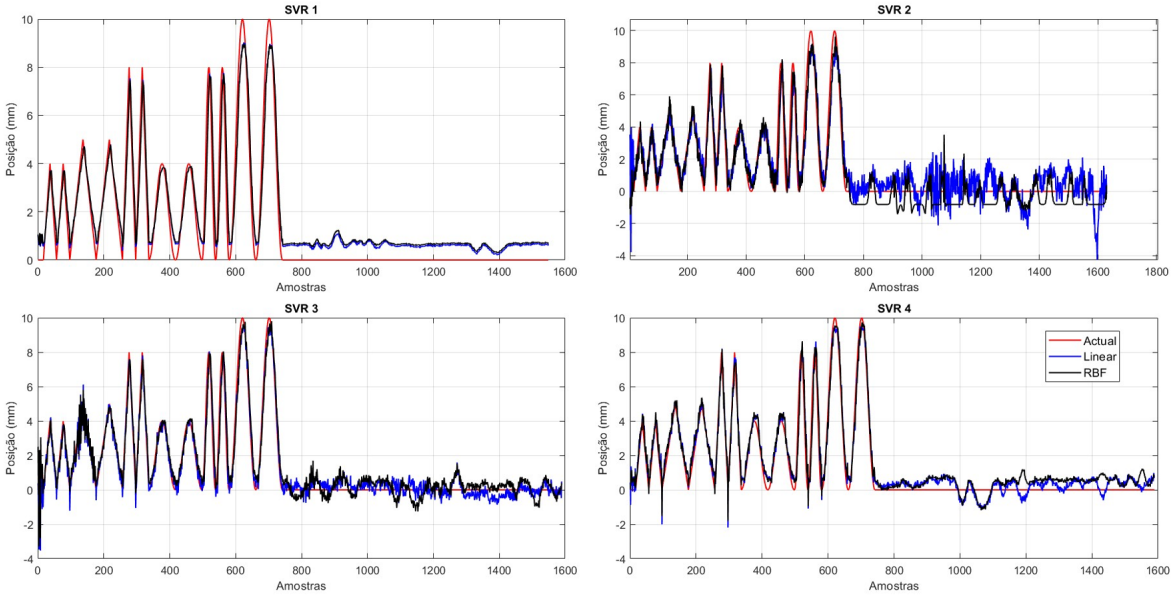


Figura 4.14 – Resultados do treinamento com 144 entradas para os regressores 1 ao 4. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF.

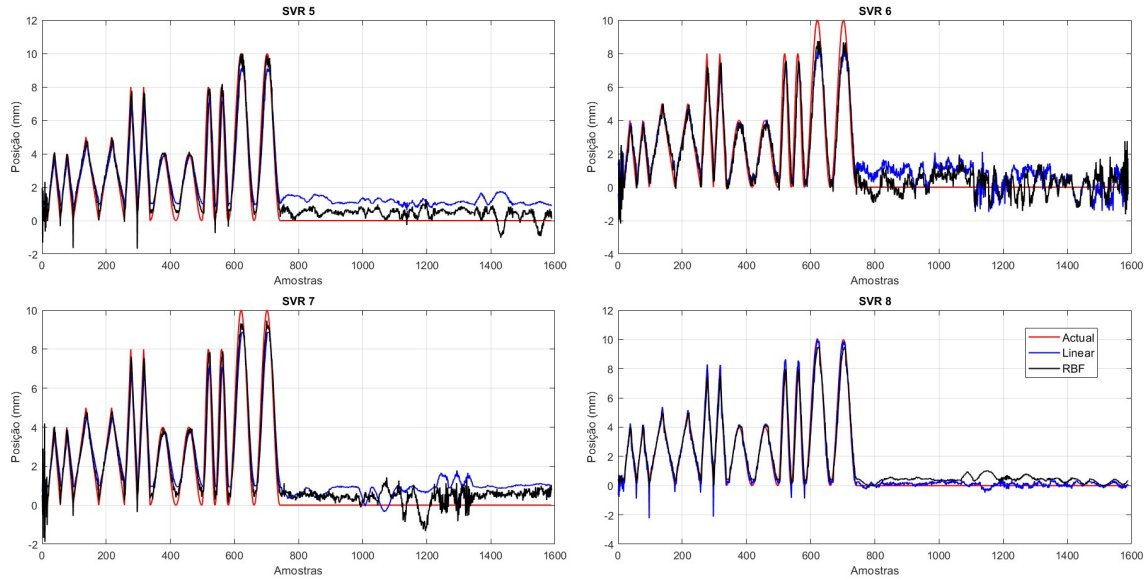


Figura 4.15 – Resultados do treinamento com 144 entradas para os regressores 5 ao 8. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF.

4.2.2 Treinamento dos modelos aplicando seleção de *features*

Os modelos Linear e RBF foram treinados aplicando a técnica de seleção de *features*, técnica aplicada antes do treinamento para definir as entradas mais relvantes no cálculo do deslocamento dos ímãs, excluindo as outras. Por meio da Equação 2.8 foram escolhidos 10 dados de entrada para cada regressor, de modo que os vetores de suporte resultantes passaram a possuir apenas 10 elementos, em vez dos 144 originais. Os modelos foram treinados com o

conjunto de treinamento mostrado na Figura 4.10 e testados com o conjunto de validação mostrado na Figura 4.11. A Figura 4.16 apresenta o resultados dos testes para os ímãs de 1 a 4 e a Figura 4.17 para os ímãs de 1 a 8.

A Tabela 4.1 apresenta uma comparação entre os resultados de treinamento utilizando e não utilizando a técnica de seleção de *features* para o *kernel* Linear. A Tabela 4.2 apresenta uma comparação semelhante para o caso do *kernel* RBF.

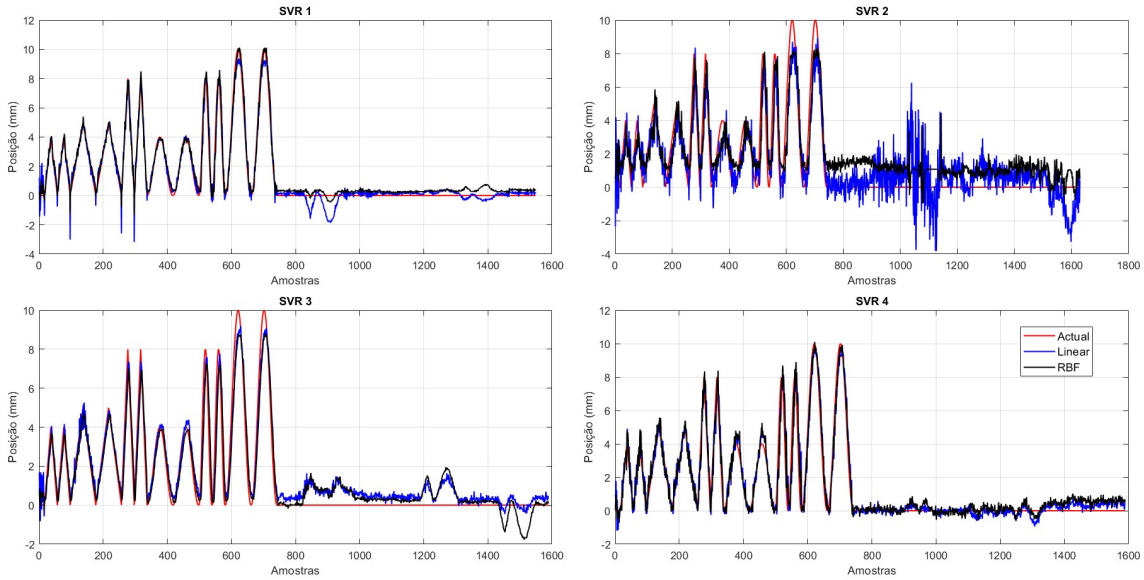


Figura 4.16 – Resultados do treinamento com 10 *features* para os regressores 1 ao 4. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF.

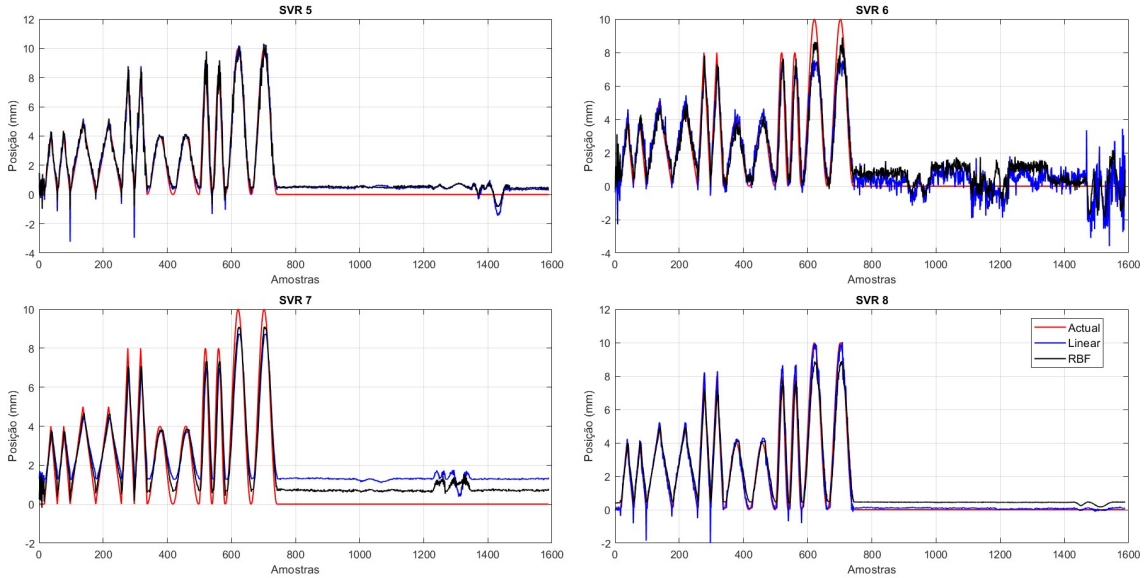


Figura 4.17 – Resultados do treinamento com 10 *features* para os regressores 1 ao 8. Em vermelho a saída real, em azul o valor aproximado pelo SVR com *kernel* linear e em preto o valor aproximado pelo SVR com *kernel* RBF.

Tabela 4.1 – Resultado de MSE e quantidade de vetores de suporte para 144 entradas e 10 entradas com kernel linear.

	Sem seleção de features (144 entradas)		**Aplicando seleção de features (10 entradas)**	
Regressor	**MSE**	**Quantidade de vetores de suporte**	**MSE**	**Quantidade de vetores de suporte**
SVR1	0.08	592	0.06	594
SVR2	0.50	1570	1.04	1500
SVR3	0.25	991	0.23	786
SVR4	0.22	718	0.27	713
SVR5	0.12	707	0.12	619
SVR6	0.39	1229	0.30	1203
SVR7	0.08	815	0.06	586
SVR8	0.13	606	0.09	565

Tabela 4.2 – Resultado de MSE e quantidade de vetores de suporte para 144 entradas e 10 entradas com kernel RBF.

	Sem seleção de features (144 entradas)		**Aplicando seleção de features (10 entradas)**	
Regressor	**MSE**	**Quantidade de vetores de suporte**	**MSE**	**Quantidade de vetores de suporte**
SVR1	0.38	11	0.06	599
SVR2	0.63	64	1.22	1685
SVR3	0.36	29	0.23	788
SVR4	0.35	35	0.29	695
SVR5	0.32	42	0.13	619
SVR6	0.60	67	0.34	1200
SVR7	0.37	30	0.06	583
SVR8	0.27	18	0.10	564

4.2.3 Treinamento com o dataset produzido na *Scuola Superiore Sant'Anna*

O *dataset* produzido pela *Scuola Superiore Sant'Anna* possui uma estrutura distinta do *dataset* desenvolvido neste trabalho. O hardware utilizado é descrito em (Clemente *et al.*, 2019), nesse caso, utilizam-se unidades de aquisição diferentes das apresentadas neste trabalho e, portanto, em vez de 144 entradas (soma de 36 saídas de cada uma das 4 unidades de aquisição), têm-se 384 entradas. Além disso, esse *dataset* considera 5 ímãs, e não 8, de modo que são treinados apenas 5 regressores.

O treinamento também foi realizado através da ferramenta NIOTS pelo mesmo procedimento descrito. Pelas características do *dataset* os regressores resultaram em modelos menos complexos, isto é, com menos vetores de suporte. A técnica de seleção de *features* permitiu reduzir o número de entradas de 384 para 10, mantendo um valor reduzido de

vetores de suporte e um MSE aceitável para a aplicação.

A Tabela 4.3 apresenta os resultados do treinamento para o *kernel* linear e a Tabela 4.4 para o *kernel* RBF.

Tabela 4.3 – Resultado de MSE e quantidade de vetores de suporte para 384 entradas e 10 entradas com kernel linear.

	Sem seleção de features (384 entradas)		**Aplicando seleção de features (10 entradas)**	
Regressor	**MSE**	**Quantidade de vetores de suporte**	**MSE**	**Quantidade de vetores de suporte**
SVR1	0.0281	8	0.0012	3
SVR2	0.0075	11	0.0001	5
SVR3	0.0173	11	0.0019	5
SVR4	0.0027	9	0.0022	5
SVR5	0.0030	20	0.008	4

Tabela 4.4 – Resultado de MSE e quantidade de vetores de suporte para 384 entradas e 10 entradas com kernel RBF.

	Sem seleção de features (384 entradas)		**Aplicando seleção de features (10 entradas)**	
Regressor	**MSE**	**Quantidade de vetores de suporte**	**MSE**	**Quantidade de vetores de suporte**
SVR1	0.0236	9	0.0085	5
SVR2	0.0062	12	0.0081	2
SVR3	0.0177	16	0.0222	3
SVR4	0.0027	49	0.0021	4
SVR5	0.0024	17	0.0171	3

A Seção 4.3 apresenta a implementação dos modelos, seus respectivos *kernels* e uma tabela apresentando o MSE entre o valor real e o valor calculado pelo hardware com os regressores implementados.

4.3 Implementação dos modelos SVR em FPGA

Como explicado no início deste Capítulo, foram treinados dois modelos distintos de SVR: um modelo considerando a base de dados desenvolvida neste trabalho e um modelo considerando a base de dados disponibilizada no trabalho de (Clemente *et al.*, 2019), pois outros estudos obtiveram seus resultados com base nesse *dataset*, dessa forma, é possível uma comparação mais adequada para avaliar a eficiência das técnicas aplicadas.

As principais diferenças entre os dois *datasets* são que o conjunto de dados construído no trabalho de (Clemente *et al.*, 2019) utiliza cinco ímãs, possui 384 entradas e contempla somente o movimento de rampa com uma única amplitude e frequência, o que limita a sua eficácia em aplicações reais. Já o conjunto de dados desenvolvido neste trabalho utiliza oito ímãs, possui 144 entradas e contempla tanto movimentos de rampa quanto movimentos senoidais, com amplitude e frequência variáveis.

Os modelos treinados utilizando SVRs para localizar os 8 ímãs utilizando o *dataset* desenvolvido neste trabalho resultaram em modelos complexos pela quantidade de vetores de suporte necessária. Os pesos dos vetores de suporte são armazenados em memórias na FPGA, isto é, blocos BRAM. Além disso, cada vetor de suporte adiciona tempo de cálculo, um fator que é limitante nessa aplicação. Portanto, os modelos Linear e RBF foram implementados considerando os modelos treinados a partir do *dataset* desenvolvido pela *Scuola Superiore Sant'Anna*, pois, como apresentado nas Tabelas 4.3 e 4.4, resultaram em modelos com poucos vetores de suporte, sendo implementáveis em hardware.

Entretanto, é importante salientar que, uma vez estabelecidas estratégias para reduzir a quantidade de vetores de suporte para a localização de ímãs considerando o *dataset* da Figura 4.11, o código VHDL dos modelos pode ser facilmente adaptado, alterando-se somente os pesos da função preditora e a quantidade de vetores de suporte.

4.3.1 Implementação de SVR com *kernel* linear

A arquitetura proposta na Figura 3.11 foi implementada em VHDL no software Vivado 2019.2, utilizando representação aritmética de ponto flutuante com 27 bits. O regressor foi implementado com base em dados de treinamento disponibilizados no trabalho de (Clemente *et al.*, 2019), no qual utilizaram-se quatro placas de aquisição, resultando em 384 entradas para rastrear 5 ímãs. Nesse trabalho, cada placa envia o conjunto de 96 dados (32 sensores de três eixos) utilizando o protocolo I2C com uma latência de 23.9 ms.

Dessa forma, foram implementados cinco regressores com *kernel* linear em paralelo no *chip* AMD-Xilinx 7z020clg484-1, operando a 100 MHz (período de *clock* de 10 ns) e utilizando os resultados da Tabela 4.3. O *chip* foi escolhido baseado nos trabalhos apresentados em (Mendez *et al.*, 2022) e (Mendez *et al.*, 2024) que também utilizaram o AMD-Xilinx 7z020clg484-1, permitindo uma comparação adequada entre as implementações. A implementação física do SVR Linear é apresentada na Figura 4.18.

Nessa implementação o ILA-core foi utilizado para obter o valor da saída dos regressores e o sinal de *ready*, o qual pode ser configurado como *trigger*, sendo assertado no nível lógico alto quando todos os regressores finalizam o cálculo. No entanto, essa ferramenta deve ser desconsiderada na análise do consumo de recursos e de energia, uma vez que não estará presente na implementação final.

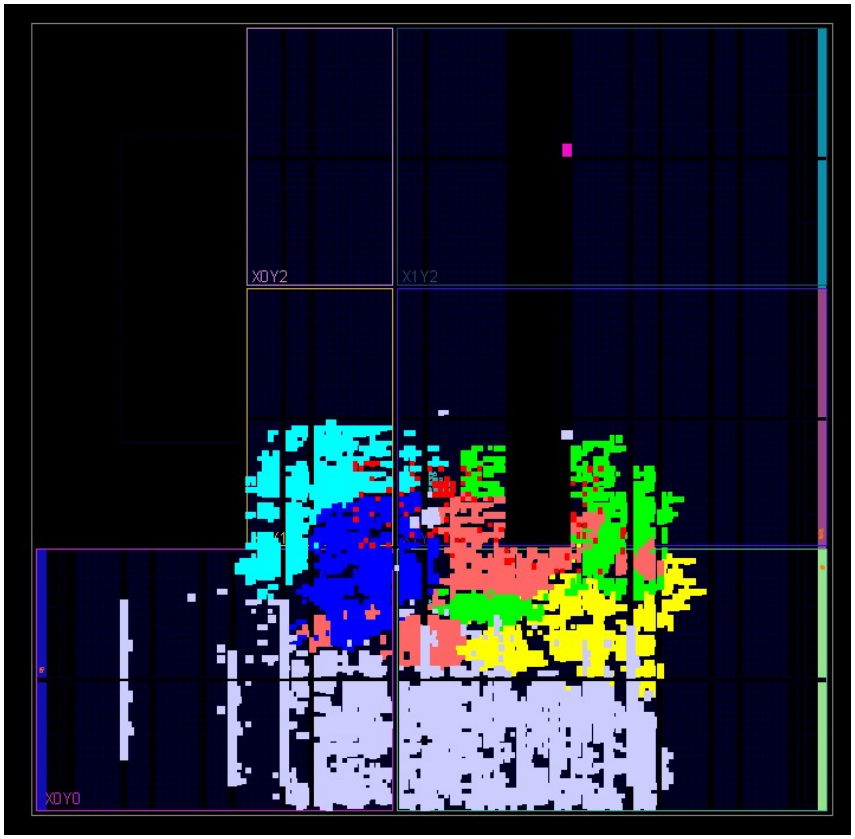


Figura 4.18 – Layout da implementação SVR linear. Vermelho: FIFO e o recurso de seleção de features; Amarelo: SVR para o ímã 1; Verde: SVR para o ímã 2; Rosa: SVR para o ímã 3; Azul: SVR para o ímã 4; Ciano: SVR para o ímã 5; Cinza: ocupação do ILA-core.

A Tabela 4.5 apresenta o consumo de recursos de cada módulo em relação ao total de recursos disponíveis no *chip* em termos de LUTs, *flip-flops*, blocos DSPs, e blocos de RAM (BRAM).

Módulos	LUTs (53200)	FFs (106400)	DSP (220)	BRAM (140)
FIFO & Multiplexador de Variáveis	37	140	0	0.5
Linear SVR 1	979	841	2	0
Linear SVR 2	1003	854	2	0
Linear SVR 3	1008	852	2	0
Linear SVR 4	1008	853	2	0
Linear SVR 5	985	848	2	0
Total	5020 (9.43%)	4388 (4.12%)	10 (4.54%)	0 (0%)

Tabela 4.5 – Ocupação de recursos de cada módulo dos regressores de vetores de suporte com *Kernel* linear implementados em um SoC FPGA Zynq 7020 da AMD-Xilinx.

Os regressores 1 e 4 utilizam 3 vetores de suporte; os regressores 2 e 3 utilizam 5 vetores de suporte e o regressor 5 utiliza 4 vetores de suporte, esses parâmetros foram obtidos por meio do treinamento no framework NIOTS. Em simulação, observou-se que os regressores

precisaram de 98, 128 e 158 ciclos de clock configurados com 3, 4 e 5 vetores de suporte, respectivamente, apresentando uma performance computacional de 1.02, 0.78 e 0.63 milhões de estimações de deslocamento por segundo. A simulação é apresentada na Figura 4.19.

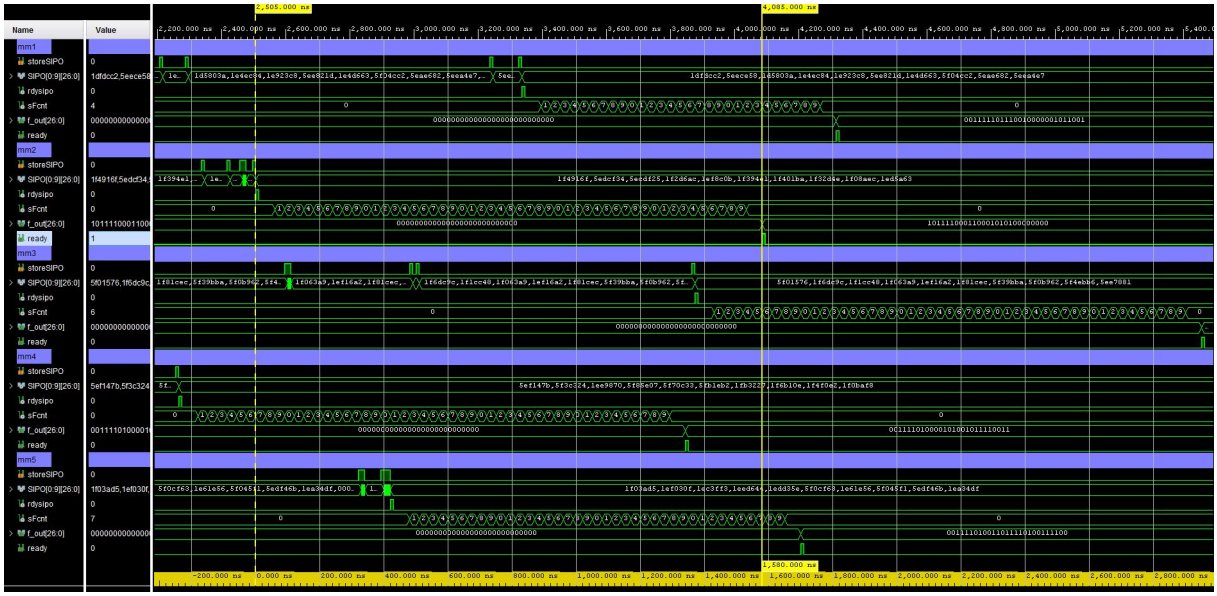


Figura 4.19 – Simulação de 5 regressores de vetores de suporte com *kernel* linear

Considerando o pior cenário, em que o cálculo requer 158 ciclos, somado ao tempo de transmissão das unidades de aquisição apresentado em (Clemente *et al.*, 2019), de 23.9 ms, e multiplicando esse valor por 5, uma vez que o cálculo dos cinco regressores é realizado em paralelo, obtém-se a quantidade de localizações de ímãs por segundo dada por:

$$\text{MLPS} = 5 \times \left(\frac{1}{23.9 \text{ ms} + 1.58\ \mu\text{s}} \right) = 209.2 \tag{4.1}$$

A Figura 4.20 apresenta um gráfico referente ao consumo de energia total da implementação, considerando o ILA. A potência total consumida foi de aproximadamente 202 mW, sendo aproximadamente 106 mW para a parte estática e 96 mW para a parte dinâmica. Da parte dinâmica 28% da energia foi consumida pela implementação do *clock*, 27% para os sinais, 23% para a lógica implementada, 12% para os blocos RAM, 10% para DSP e uma pequena parte para os pinos de saída, mapeados ao ILA.

O consumo de energia de cada SVR foi aproximadamente 13 mW. Considerando a arquitetura proposta na Figura 3.10, o consumo total de energia para a implementação de 5 regressores foi de 65 mW, somados a 1 mW do recurso de seleção de *features* e 8 mW das 4 unidades de SPI, a implementação da arquitetura consome 74 mW. Esse relatório de consumo de energia é apresentado na Figura 4.21 e o consumo de cada SPI na Figura 4.9.

Calculando a eficiência computacional em MLPS/W, o consumo de cada unidade de aquisição é de 550 mW (Clemente *et al.*, 2019). Considerando quatro unidades de aquisição, o consumo total é de 4×550 mW. Somando-se o consumo dos cinco regressores, de 74 mW,

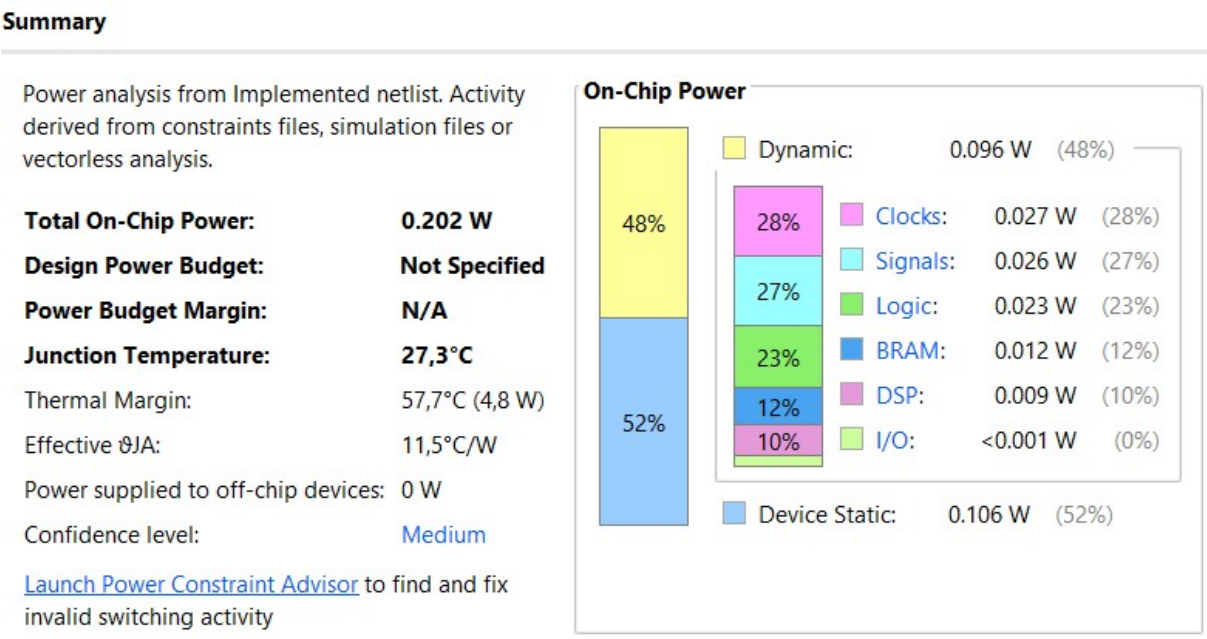


Figura 4.20 – Relatório de consumo energia total para o *Kernel* linear.

obtém-se um consumo total de 2.274 W. Dessa forma, é possível calcular uma eficiência computacional de 91.9 MLPS/W.

$$\text{MLPS/W} = \frac{\text{Localização de ímãs por segundo}}{\text{Consumo de Energia}} \tag{4.2}$$

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	DSP (W)	I/O (W)
0.096 W (48% of total)	topmodule									
0.024 W (12% of total)	my_ila (ila_0)	0.012	0.002	0.002	<0.001	<0.001	0.001	0.01	<0.001	<0.001
0.013 W (7% of total)	svr1 (linearSVM3)	0.002	0.005	0.005	<0.001	<0.001	0.004	<0.001	0.002	<0.001
0.013 W (7% of total)	svr3 (linearSVM5_2)	0.002	0.005	0.005	<0.001	<0.001	0.004	<0.001	0.002	<0.001
0.013 W (7% of total)	svr2 (linearSVM5_1)	0.002	0.005	0.005	<0.001	<0.001	0.004	<0.001	0.002	<0.001
0.013 W (6% of total)	svr4 (linearSVM5)	0.002	0.005	0.005	<0.001	<0.001	0.004	<0.001	0.002	<0.001
0.013 W (6% of total)	svr5 (linearSVM4)	0.002	0.004	0.004	<0.001	<0.001	0.004	<0.001	0.002	<0.001
0.003 W (1% of total)	dbg_hub (dbg_hub)	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.002 W (1% of total)	fifo_in (datasrc)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.002	<0.001	<0.001
0.001 W (1% of total)	features (feature_selection)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (80)									

Figura 4.21 – Relatório de consumo de energia discriminado por módulos

A implementação também atendeu as restrições de *timing*. A Figura 4.22 apresenta o relatório de *timing* avaliado para uma frequência de *clock* de 100 MHz, indicando que não existem violações na temporização de *setup* e de *hold*.

Tabela 4.6 – Diferença entre o valor calculado por cada regressor implementado no Vivado e em MATLAB. Esse diferença foi calculada para uma amostra do conjunto de dados

Regressor	Diferença entre MATLAB (64 bits) e Hardware (27 bits)
SVR1	$1{,}6 \times 10^{-6}$
SVR2	$1{,}08 \times 10^{-6}$
SVR3	$3{,}5 \times 10^{-6}$
SVR4	6×10^{-6}
SVR5	$6{,}76 \times 10^{-6}$



Figura 4.22 – Análise de timing - Clock em 100 MHz

Os cinco regressores implementados em hardware também foram reproduzidos no MATLAB, utilizando os mesmos vetores de suporte e os mesmos parâmetros α e ρ, com o objetivo de validar a implementação em relação aos resultados esperados. Para demonstrar o funcionamento dos regressores, uma amostra do conjunto de dados foi processada tanto na simulação da implementação em VHDL, apresentada na Figura 4.19, quanto no ambiente MATLAB. A Tabela 4.6 apresenta a diferença entre os valores calculados em ambos os casos.

Os valores das entradas foram mapeados em memória no FPGA para validação do modelo. A saída dos regressores foram mapeadas para o ILA, sendo possível a decodificação desses valores para cálculo do MSE entre a saída esperada e o valor calculado por cada regressor. A Tabela 4.7 apresenta os valores do erro quadrático médio.

Tabela 4.7 – MSE entre o valor real e o valor calculado pelo regressor.

Regressor	MSE	Quantidade de vetores de suporte
SVR1	$1{,}26 \times 10^{-3}$	3
SVR2	$0{,}18 \times 10^{-3}$	5
SVR3	$1{,}95 \times 10^{-3}$	5
SVR4	$0{,}30 \times 10^{-3}$	5
SVR5	$0{,}89 \times 10^{-3}$	4

A Figura 4.23 apresenta a saída dos cinco regressores implementados para este dataset. No conjunto de validação, cada ímã foi movimentado individualmente em um determinado instante com um sinal em rampa, enquanto os demais permaneceram parados. O primeiro

gráfico corresponde ao ímã 1, o segundo ao ímã 2, e assim sucessivamente. A linha vermelha representa a saída real; a linha preta, o valor calculado no MATLAB (64 bits); e a linha azul, o valor calculado na FPGA (27 bits).

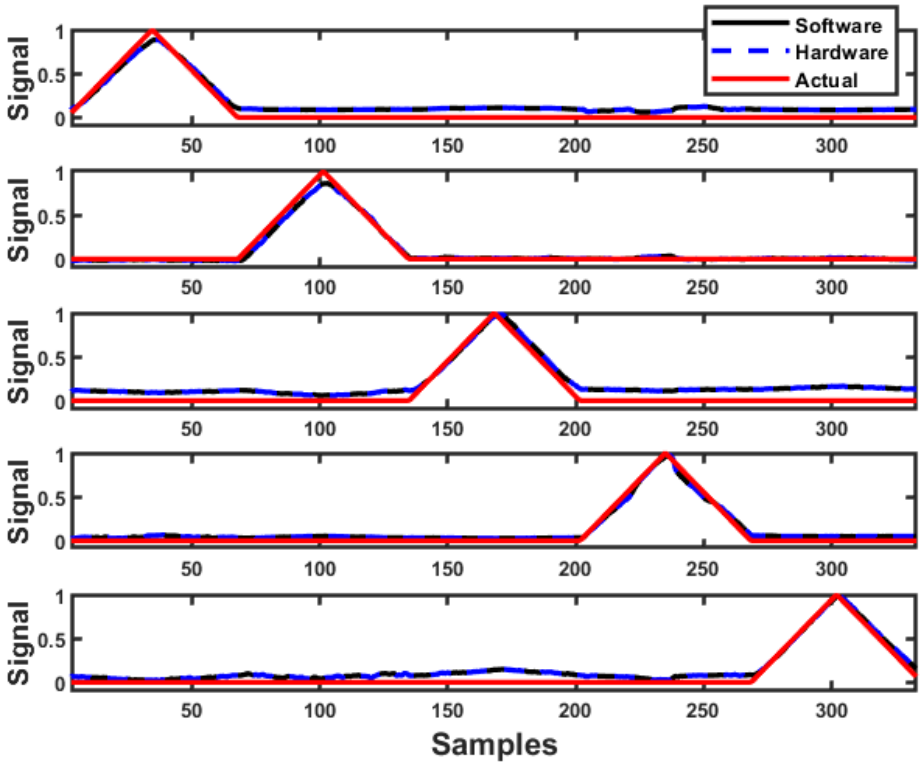


Figura 4.23 – Comparação entre o valor real e a saída de cada regressor. Em vermelho a saída real, em preto o valor calculado pelo MATLAB (64 bits) e em azul o valor calculado pelo FPGA (27 bits)

4.3.2 Implementação de SVR com *kernel* RBF

O modelo RBF foi implementado utilizando as mesmas considerações e procedimentos descritos na Seção 4.3.1 para o modelo linear. Foi utilizada a arquitetura proposta na Figura 3.12 para implementação de SVR com *kernel* RBF com base na Equação 2.2 e os resultados apresentados na Tabela 4.4 para definição dos hiperparâmetros e pesos da função preditora apresentada na Equação 2.5.

Foram implementados cinco regressores com *kernel* RBF em paralelo no *chip* AMD-Xilinx 7z020clg484-1, operando a 100 MHz. A implementação física do SVR RBF é apresentada na Figura 4.24. É possível observar uma maior ocupação de área em relação ao modelo linear, fator esperado, considerando que o modelo RBF que exige operações matemáticas complexas.

O ILA também foi utilizado para mapear as saídas. A Tabela 4.8 apresenta o consumo de recursos de cada módulo em termos de LUTs, *flip-flops*, blocos DSPs, e blocos de RAM

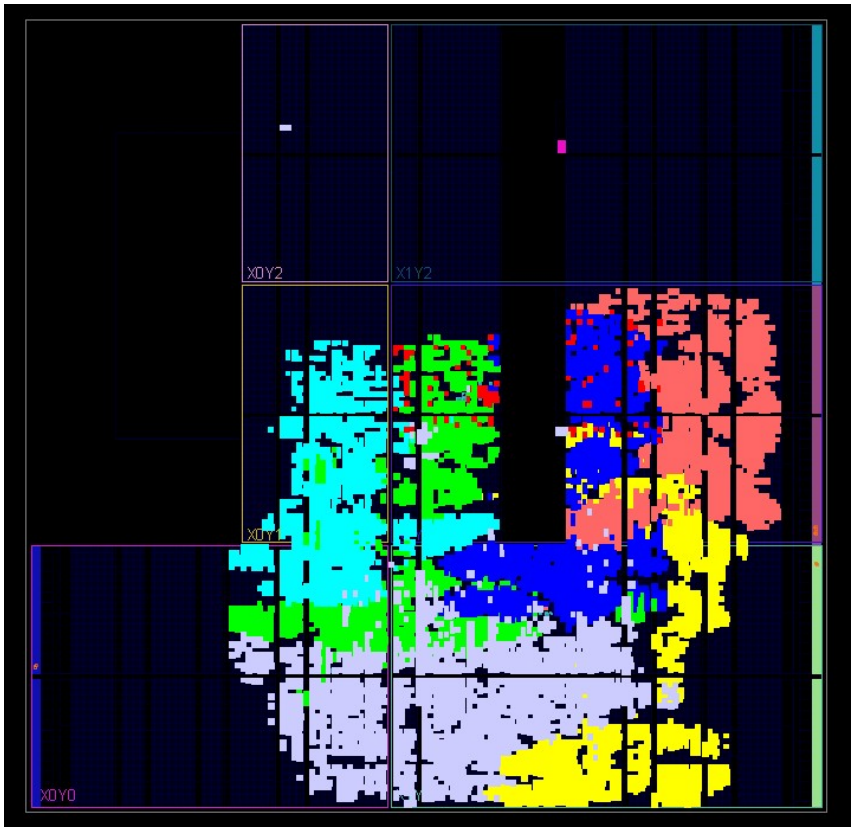


Figura 4.24 – Layout da implementação SVR RBF. Vermelho: FIFO e o recurso de seleção de features; Amarelo: SVR para o ímã 1; Verde: SVR para o ímã 2; Rosa: SVR para o ímã 3; Azul: SVR para o ímã 4; Ciano: SVR para o ímã 5; Cinza: ocupação do ILA-core.

(BRAM). Como esperado, observa-se um maior consumo de recurso se comparado ao modelo linear, isso ocorre pois a implementação da exponencial exige operações que demandam mais recursos de hardware.

Módulos	LUTs (53200)	FFs (106400)	DSP (220)	BRAM (140)
FIFO & Multiplexador de Variáveis	35	140	0	0.5
Linear SVR 1	2362	1410	4	0
Linear SVR 2	2245	1389	4	0
Linear SVR 3	2328	1394	4	0
Linear SVR 4	2325	1401	4	0
Linear SVR 5	2320	1392	4	0
Total	11615 (21.83%)	6986 (6.56%)	20 (9.09%)	0 (0%)

Tabela 4.8 – Ocupação de recursos de cada módulo dos regressores de vetores de suporte com *Kernel* RBF implementados em um SoC FPGA Zynq 7020 da AMD-Xilinx.

A partir do momento em que ocorre o primeiro *start*, o SVR apresenta uma latência de 6.13 μs. Em seguida, ocorre um *pipeline* após a FIFO estar completa, aumentando a taxa de saída. A etapa de cálculo da exponencial demanda tempo; enquanto esse processo é

realizado, os vetores de suporte continuam a ser multiplicados pelas entradas, produzindo, após o primeiro cálculo, uma saída a cada 3.85 μs. A Figura 4.25 apresenta a simulação em que esses valores foram observados.

Alguns regressores finalizam o cálculo antes dos demais, a depender da quantidade de vetores de suporte e do algoritmo selecionado no cálculo da exponencial. o *IP* utilizado para cálculo da exponencial, apresentado em (Ayala *et al.*, 2017), seleciona o cálculo por meio do algoritmo CORDIC ou expansão em série de Taylor, a depender das caracteríscas do argumento de entrada. Esse fator também influência o tempo de execução conforme o valor de entrada.

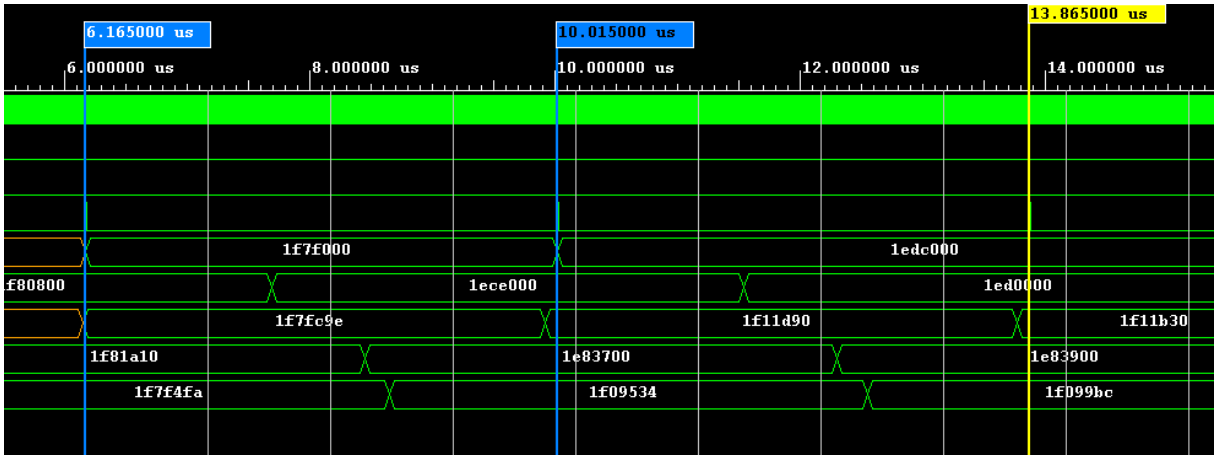


Figura 4.25 – Simulação de 5 regressores de vetores de suporte com *kernel* RBF com marcadores de tempo nas bordas de subida dos sinais de *ready*, que indicam uma saída válida.

A Figura 4.26 apresenta um gráfico referente ao consumo de energia total da implementação, considerando o ILA. A potência total consumida foi de aproximadamente 249 mW, sendo aproximadamente 108 mW para a parte estática e 140 mW para a parte dinâmica. Da parte dinâmica 25% da energia foi consumida pela implementação do *clock*, 31% para os sinais, 30% para a lógica implementada, 5% para os blocos RAM, 9% para DSP e uma pequena parte para os pinos de saída, mapeados ao ILA.

O consumo de energia de cada SVR foi em média 23 mW. Considerando a arquitetura proposta na Figura 3.10, o consumo total de energia para a implementação de 5 regressores foi de 115 mW, somados a 1 mW do recurso de seleção de *features* e 8 mW das 4 unidades de SPI, a implementação da arquitetura consome 124 mW. Esse relatório de consumo de energia é apresentado na Figura 4.27 e o consumo de cada SPI na Figura 4.9.

Conforme o procedimento já descrito para cálculo da eficiência computacional utilizando as Equações 4.1 e 4.2, e observando os dados de consumo de energia e tempo de processamento, é obtida uma eficiência computacional de 90 MLPS/W.

Summary

Power analysis from Implemented netlist. Activity derived from constraints files, simulation files or vectorless analysis.

Total On-Chip Power:	**0.249 W**
Design Power Budget:	**Not Specified**
Power Budget Margin:	**N/A**
Junction Temperature:	**27,9°C**
Thermal Margin:	57,1°C (4,8 W)
Effective ϑJA:	11,5°C/W
Power supplied to off-chip devices:	0 W
Confidence level:	Medium

Launch Power Constraint Advisor to find and fix invalid switching activity

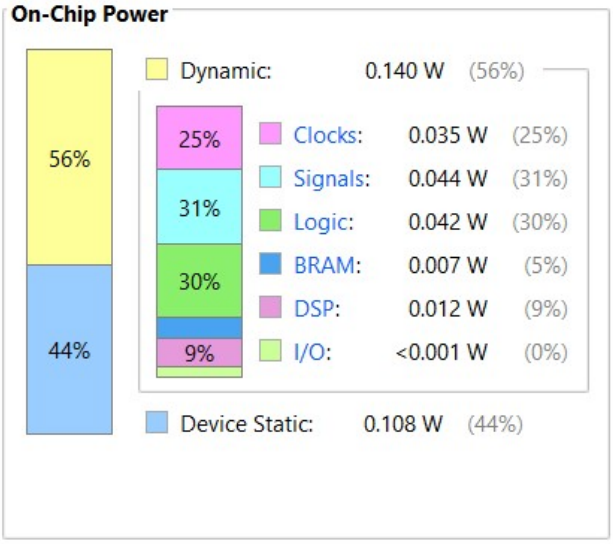


Figura 4.26 – Relatório de consumo energia total para o *kernel* RBF

Utilization	Name	Clocks (W)	Signals (W)	Data (W)	Clock Enable (W)	Set/Reset (W)	Logic (W)	BRAM (W)	DSP (W)	I/O (W)
0.14 W (56% of total)	topmodule									
0.024 W (10% of total)	svr4 (RBF_SVM4)	0.004	0.009	0.009	<0.001	<0.001	0.008	<0.001	0.003	<0.001
0.023 W (9% of total)	svr1 (RBF_SVM5)	0.004	0.008	0.008	<0.001	<0.001	0.008	<0.001	0.003	<0.001
0.023 W (9% of total)	svr3 (RBF_SVM3)	0.004	0.008	0.008	<0.001	<0.001	0.008	<0.001	0.002	<0.001
0.023 W (9% of total)	svr5 (RBF_SVM3__1)	0.004	0.008	0.008	<0.001	<0.001	0.008	<0.001	0.002	<0.001
0.022 W (9% of total)	svr2 (RBF_SVM2)	0.004	0.008	0.008	<0.001	<0.001	0.007	<0.001	0.002	<0.001
0.019 W (8% of total)	my_ila (ila_0)	0.012	0.001	0.001	<0.001	<0.001	0.001	0.005	<0.001	<0.001
0.003 W (1% of total)	dbg_hub (dbg_hub)	0.002	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
0.002 W (1% of total)	fifo_in (datasrc)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	0.002	<0.001	<0.001
0.001 W (1% of total)	features (feature_selection)	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001	<0.001
<0.001 W (<1% of total)	Leaf Cells (83)									

Figura 4.27 – Relatório de consumo de energia discriminado por módulos para a implementação com *kernel* RBF

A implementação atendeu as restrições de *timing*, sendo necessário ativar a opção *retiming* do Vivado, em que se permite que a ferramenta realize estratégias para atender as restrições de timing, podendo adicionar latência ao processo. A Figura 4.28 apresenta o relatório de *timing* avaliado para uma frequência de *clock* de 100 MHz,indicando que não existem violações na temporização de *setup* e de *hold*.

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	0,171 ns	Worst Hold Slack (WHS):	0,028 ns	Worst Pulse Width Slack (WPWS):	3,750 ns
Total Negative Slack (TNS):	0,000 ns	Total Hold Slack (THS):	0,000 ns	Total Pulse Width Negative Slack (TPWS):	0,000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	23721	Total Number of Endpoints:	23705	Total Number of Endpoints:	12086

All user specified timing constraints are met.

Figura 4.28 – Análise de timing para regressor com *kernel* RBF - Clock em 100 MHz

Os resultados da implementação do *kernel* RBF foram coletados por meio do ILA e decodificados em MATLAB. A Figura 4.29 apresenta os resultados para os 5 regressores e a Tabela 4.9 o MSE entre o valor calulado pela implementação e o valor real.

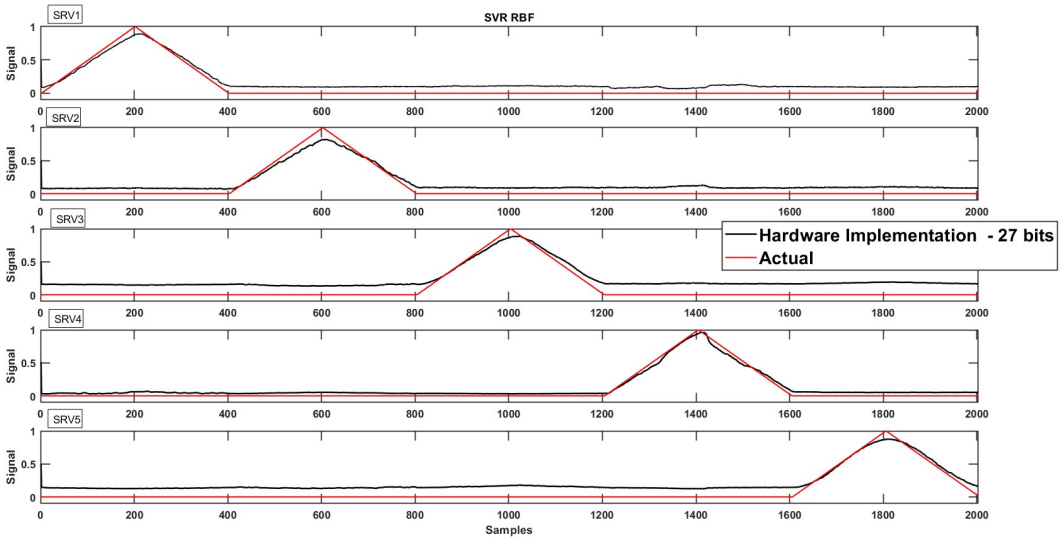


Figura 4.29 – Comparação entre o valor real e a saída de cada regressor RBF. Em vermelho a saída real, em preto o valor calculado pelo FPGA (27 bits)

Tabela 4.9 – MSE entre o valor real e o valor calculado pelo regressor para o *kernel* RBF.

Regressor	MSE	Quantidade de vetores de suporte
SVR1	8.5×10^{-3}	5
SVR2	8.1×10^{-3}	2
SVR3	22×10^{-3}	3
SVR4	$2{,}2 \times 10^{-3}$	4
SVR5	$1{,}71 \times 10^{-3}$	3

4.4 Comparação dos resultados obtidos com trabalhos correlatos

A Tabela 4.10 apresenta uma comparação dos resultados obtidos neste trabalho com aqueles reportados em estudos correlatos, em termos de consumo de energia, tempo de processamento para a localização dos ímãs, tempo de transmissão entre as unidades de aquisição e a unidade de processamento (neste trabalho corresponde ao FPGA) e eficiência computacional, expressa em localizações de ímãs por segundo (MLPS, sigla em inglês). A tabela também indica qual *dataset* foi utilizado para o treinamento dos modelos em cada estudo.

Tabela 4.10 – Comparação entre diferentes implementações de modelos de aprendizado de máquina para estimar deslocamento de ímãs, incluindo os resultados obtidos neste trabalho.

	Descrição	**Consumo de energia (W)**	**Tempo de processamento para localização de ímã**	**Tempo de transmissão (ms)**	**Eficiência computacional (MLPS/W)**	**Base de dados**
(Clemente *et al.*, 2019)	ARM Cortex M7 @600 MHz/ LMA	2.63	0.8 ms (1 mm)	95.6	3.94	(Clemente *et al.*, 2019)
(Mendez *et al.*, 2022)	SoC FPGA / regressor linear	2.409	4.8 μs (1 mm)	23.9	17.37	(Clemente *et al.*, 2019)
(Mendez *et al.*, 2022)	SoC FPGA / regressor RBFNN	2.745	12.07 μs (1 mm)	23.9	15.26	(Clemente *et al.*, 2019)
(Mendez *et al.*, 2024)	SoC FPGA / Reconfiguração Dinâmica	3.919	9.52 ms (5 mms)	23.9	38.18	(Clemente *et al.*, 2019)
Este Trabalho	SoC FPGA / SVR Linear	2.274	1.58 μs (5 mms)	23.9	91.9	(Clemente *et al.*, 2019)
Este Trabalho	SoC FPGA / SVR RBF	2.324	3.85 μs (5 mms)	23.9	90.0	(Clemente *et al.*, 2019)

O cálculo do tempo de localização para todos os casos foi realizado de acordo com o raciocínio apresentado na Equação 4.1, enquanto a eficiência computacional foi determinada com base na Equação 4.2.

Para o regressor com *kernel* linear, a taxa de localização dos ímãs, considerando a amostragem e a transmissão dos sensores, foi de 41,838 Hz, enquanto para o *kernel* RBF obteve-se 41,834 Hz. Em ambos os casos, os requisitos de tempo real do sistema foram atendidos, assim como o Teorema da Amostragem de Nyquist–Shannon, considerando uma frequência máxima de movimento da mão humana de 10,5 Hz (Mendez, 2021).

5 Conclusões

Este trabalho contribuiu para os avanços na viabilização das interfaces miocinéticas por meio de várias frentes: a construção de uma base de dados para o treinamento de modelos de aprendizado de máquina; a implementação do protocolo SPI em FPGA, dado o impacto do tempo de transmissão na eficiência computacional observado na Tabela 2.3, possibilitando que outros trabalhos reduzam o tempo de transmissão que anteriormente considerava o protocolo I2C (23.9 ms) para o tempo de transmissão do protocolo SPI (576 μs); o desenvolvimento do firmware e da interface gráfica para controle dos motores, possibilitando a implementação de movimentos coerentes com os movimentos musculares humanos e permitindo testes dos modelos em tempo de execução; e o desenvolvimento de códigos em VHDL para a implementação de máquinas de vetores de suporte com *kernel* linear e RBF em FPGA.

A técnica de seleção de variáveis univariadas, denominada seleção de *features*, apresentou uma redução de complexidade considerável. Em relação ao *dataset* apresentado em (Clemente *et al.*, 2019), a aplicação dessa técnica, além de reduzir a quantidade de elementos nos vetores de entrada, reduziu também a quantidade de vetores de suporte, diminuindo em quase todos os casos o MSE e, em outros casos, mantendo-o em uma faixa de valores aceitável para a aplicação. Essa técnica impactou positivamente os modelos de SVR linear e RBF.

No caso do treinamento dos modelos utilizando o *dataset* desenvolvido neste trabalho, a técnica de seleção de *features* apresentou um bom desempenho para a maioria dos regressores, reduzindo a dimensionalidade dos vetores de entrada, a quantidade de vetores de suporte e reduzindo ou mantendo o MSE dentro de uma faixa aceitável em relação ao treinamento sem a aplicação da técnica. Para o modelo de SVR com *kernel* RBF, porém, a técnica de seleção de *features* não apresentou bom desempenho. Isso ocorreu porque, embora tenha reduzido a dimensionalidade dos vetores de entrada, aumentou drasticamente a quantidade de vetores de suporte. Para o modelo RBF, a quantidade de vetores de suporte é um fator mais crítico que a dimensionalidade dos vetores de entrada, pois para cada vetor de suporte é necessário calcular uma exponencial, que é uma operação complexa em hardware, enquanto cada elemento adicional no vetor de entrada representa apenas uma multiplicação, uma subtração e uma adição.

Para o treinamento utilizando o *dataset* deste trabalho, o modelo linear apresentou um menor erro, porém com uma maior quantidade de vetores de suporte. O modelo com *kernel* RBF sem a aplicação da técnica de seleção de *features*, apresentou a menor quantidade de vetores de suporte, mas com um erro considerável. Para o treinamento utilizando o *dataset* apresentado em (Clemente *et al.*, 2019), o modelo com *kernel* RBF apresentou um menor

erro e uma quantidade de vetores de suporte comparável. Entretanto, o modelo com *kernel* linear também apresentou um erro aceitável e um tempo de localização de 1.58 μs, enquanto o modelo com *kernel* RBF apresentou 3.85 μs e demandou maior consumo de recursos. Dessa forma, é possível concluir que o modelo com *kernel* linear apresenta um desempenho melhor nesse cenário.

Alguns fatores explicam a diferença de performance no treinamento das SVRs para os diferentes *datasets*. Um deles utiliza 128 sensores para localizar 5 ímãs (Clemente *et al.*, 2019), enquanto a base de dados desenvolvida neste trabalho utiliza 48 sensores para localizar 8 ímãs. Além disso, o *dataset* apresentado em (Clemente *et al.*, 2019) utiliza um único padrão de movimento, com parâmetros de amplitude e frequência fixos. Dessa forma, os modelos de SVR treinados aprendem especificamente esse padrão, não sendo possível avaliar seu desempenho em cenários distintos, com diferentes formas de onda, frequências ou amplitudes de movimento.

Essas diferenças contribuem para explicar por que o treinamento com o *dataset* de (Clemente *et al.*, 2019) resultou em modelos menos complexos e mais implementáveis, devido às características intrínsecas dos respectivos dados, fator que também justifica a diferença de desempenho entre o *kernel* linear e o *kernel* RBF em cada um desses casos.

Outro fator que impactou os resultados do treinamento para a base de dados deste trabalho é a disposição das unidades de aquisição. Observou-se experimentalmente que os regressores 2 e 6 apresentaram maiores perturbações nas saídas que os demais, pois o posicionamento desses ímãs está em uma região menos favorecida no experimento realizado, estando ambos mais distantes das unidades de aquisição quando comparados aos outros.

Por fim, os resultados demonstraram o potencial do uso de máquinas de vetores de suporte em conjunto com a técnica de seleção de *features*. Em comparação com os trabalhos apresentados na Tabela 4.10, os modelos de SVR implementados tanto com *kernel* linear quanto com *kernel* RBF apresentaram menor tempo de localização e menor consumo de energia, resultando em maior eficiência computacional quando comparados aos trabalhos citados.

5.0.1 Trabalhos futuros

Esta seção apresenta sugestões para trabalhos futuros. Em relação à base de dados, os movimentos dos ímãs foram modelados como rampa e senoide. Trabalhos futuros podem explorar formas de onda distintas para representar esses movimentos e investigar quais delas melhor se assemelham ao comportamento muscular. Além disso, podem ser implementados padrões de movimento que reproduzam gestos musculares reais utilizados em atividades específicas, bem como estratégias para definir o posicionamento adequado dos ímãs, respondendo às seguintes questões: qual é o posicionamento e angulação ideais dos ímãs na plataforma para emular de forma fidedigna uma interface miocinética? Como movimentar os

ímãs em conjunto de modo que o movimento implementado seja anatomicamente coerente com movimentos da mão humana? Como representar a variação de força presente em cada movimento executado?

No que se refere ao treinamento dos modelos, trabalhos futuros podem explorar diferentes disposições das unidades de aquisição, bem como treinar máquinas de vetores de suporte utilizando configurações diversas dos algoritmos bioinspirado para otimização dos hiperparâmetros, além de empregar algoritmos bioinspirados alternativos na obtenção dos pesos da função preditora e comparar os resultados obtidos. Outras técnicas de aprendizado de máquina também podem ser investigadas como alternativas às máquinas de vetores de suporte, assim como novas abordagens de redução de dimensionalidade em substituição à técnica univariada de seleção de variáveis. Observou-se que as máquinas de vetores de suporte apresentaram desempenho superior em relação aos outros trabalhos baseados no dataset de (Clemente *et al.*, 2019); entretanto, para o dataset desenvolvido nesta pesquisa, outras técnicas podem se mostrar mais eficazes, dada a diferença nas características dos conjuntos de dados.

O treinamento de modelos que considerem movimentos simultâneos presentes na base de dados representa um desafio significativo, pois o campo magnético gerado pelos ímãs interfere nas unidades de aquisição, exigindo que o modelo seja capaz de distinguir corretamente a posição individual de cada ímã.

Por fim, é necessário correlacionar a posição de cada ímã com o movimento executado pela mão robótica, traduzindo tais posições em estratégias de controle que gerem sinais PWM para os motores responsáveis pela movimentação da mão, de forma a produzir o gesto desejado.

Referências

AMD. **CLB Architecture**. 2025. Disponível em: https://docs.amd.com/r/en-US/am005-versal-clb/CLB-Architecture. Acesso em: 11/07/2025. Citado na p. 31.

AMD. **Integrated Logic Analyzer**. 2025. Disponível em: https://docs.amd.com/v/u/en-US/pg172-ila. Acesso em: 11/07/2025. Citado na p. 32.

AMD. **Zynq 7000 SoC Technical Reference Manual**. 2025. Disponível em: https://docs.amd.com/r/en-US/ug585-zynq-7000-SoC-TRM/Zynq-7000-SoC-Technical-Reference-Manual. Acesso em: 15/07/2025. Citado nas pp. 7 e 32.

AYALA, H. V. H.; MUÑOZ, D. M.; LLANOS, C. H.; COELHO, L. dos S. Efficient hardware implementation of radial basis function neural network with customized-precision floating-point operations. **Control Engineering Practice**, v. 60, p. 124–132, 2017. Citado nas pp. 48, 51 e 78.

BRUNTON, S. L.; KUTZ, J. N. **Data-Driven Science and Engineering: Machine Learning, Dynamical Systems, and Control**. 1. ed. United Kingdom: Cambridge University Press, 2019. Disponível em: https://assets.cambridge.org/97811084/22093/frontmatter/9781108422093_frontmatter.pdf. Citado na p. 29.

BURGES, C. J. A tutorial on support vector machines for pattern recognition. **Data mining and knowledge discovery**, Springer, v. 2, n. 2, p. 121–167, 1998. Citado nas pp. 24, 26 e 27.

CIPRIANI, C.; GHERARDINI, M.; IANNICIELLO, V.; MASIERO, F.; PAGGETTI, F.; D'ACCOLTI, D.; FRAZIA, E. L.; MANI, O.; DALISE, S.; FRAGAPANE, N. **The Myo-Kinetic prosthetic hand: Implanted magnets restore grasping in humans with upper limb amputation**. 2023. Citado nas pp. 14 e 17.

CLEMENTE, F.; IANNICIELLO, V.; GHERARDINI, M.; CIPRIANI, C. Development of an embedded myokinetic prosthetic hand controller. **Sensors**, v. 19, n. 3137, 2019. Citado nas pp. 15, 34, 35, 52, 61, 69, 70, 71, 73, 81, 82, 83 e 84.

CORTES, C.; VAPNIK, V. Support-vector networks. **Machine learning**, Springer, v. 20, p. 273–297, 1995. Citado na p. 18.

ESPOSITO, D.; CENTRACCHIO, J.; ANDREOZZI, E.; GARGIULO, G. D.; NAIK, G. R.; BIFULCO, P. Biosignal-based human–machine interfaces for assistance and rehabilitation: A survey. **Sensors**, v. 21, n. 6863, 2021. Citado na p. 14.

FEITOSA, E. T. **Interface Miocinética Para Aquisição de Dados de Ações Motoras do Antebraço**. Monografia ((Trabalho de Conclusão de Curso)) — Universidade de Brasília, Brasília, DF, 2023. Citado nas pp. 16, 34, 39 e 40.

IANNICIELLO, V.; GHERARDINI, M.; CIPRIANI, C. Transcutaneous magnet localizer for a self-contained myokinetic prosthetic hand. **IEEE Transactions on biomedical engineering**, v. 71, n. 3, p. 1068–1075, 2024. Citado nas pp. 15, 17 e 18.

KALTENBACH, M. **The Levenberg-Marquardt Method and its Implementation in Python**. Dissertação ((Tese de mestrado)) — Universität Konstanz, 2022. Citado na p. 15.

LUCA, C. J. de. The use of surface electromyography in biomechanics. **Journal of Applied Biomechanics**, v. 13, n. 2, p. 135–163, 1997. Disponível em: https://www.delsys.com/downloads/TUTORIAL/the-use-of-semg-in-biomechanics.pdf. Citado na p. 21.

MENDES, D. de A. **Enhanced Run-Time Reconfigurable Myokinetic Interface for Prosthetic Control of Artificial Hands**. Dissertação ((Tese de mestrado)) — Universidade de Brasília, 2024. Citado nas pp. 16, 18, 29, 52 e 61.

MENDEZ, S. A. P. **Run-time reconfiguration for efficient tracking of implanted magnets with a myokinetic control interface applied to robotic hands**. Tese (Doutorado) — Universidade de Brasília, Brasília, DF, 2021. Citado nas pp. 18 e 81.

MENDEZ, S. A. P.; GHERARDINI, M.; SANTOS, G. V. de P.; MUÑOZ, D. M.; AYALA, H. V. H.; CIPRIANI, C. Data-driven real-time magnetic tracking applied to myokinetic interfaces. **IEEE Transactions on biomedical circuits and systems**, v. 16, n. 2, p. 266–274, 2022. Citado nas pp. 10, 15, 16, 17, 22, 35, 52, 61, 71 e 81.

MENDEZ, S. A. P.; LLANOS, C. H.; MUÑOZ, D. M. Development of a robotic hand using bioinspired optimization for mechanical and control design: UnB-Hand. **IEEE Access**, v. 9, p. 61010–61023, 2021. Citado na p. 42.

MENDEZ, S. A. P.; MENDES, D. de A.; GHERARDINI, M.; MUÑOZ, D. M.; AYALA, H. V. H.; CIPRIANI, C. Dynamic reconfiguration for multi-magnet tracking in myokinetic prosthetic interfaces. **IEEE Transactions on medical robotics and bionics**, v. 6, n. 4, p. 1679–1687, 2024. Citado nas pp. 15, 16, 17, 18, 29, 35, 52, 61, 71 e 81.

MICERA, S.; CARPANETO, J.; RASPOPOVIC, S. Control of hand prostheses using peripheral information. **IEEE Reviews in Biomedical Engineering**, v. 3, p. 48–68, 2010. Citado na p. 21.

MORADI, A.; RAFIEI, H.; DALIRI, M.; AKBARZADEH, M. A. A.; NADDAF-SH., A.; NADDAF-SH., S. Clinical implementation of a bionic hand controlled with kineticomyographic signals. **Scientific reports**, v. 12, n. 14805, 2022. Citado nas pp. 14 e 16.

MUÑOZ, D. M.; SANCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCóN, M. FPGA based floating-point library for cordic algorithms. **VI Southern Programmable Logic Conference (SPL)**, p. 55–60, 2010. Citado nas pp. 48 e 51.

MUÑOZ, D. M.; SANCHEZ, D. F.; LLANOS, C. H.; AYALA-RINCóN, M. Tradeoff of FPGA design of a floating-point libraryfor arithmetic operators. **Journal Integrated Circuits and Systems**, v. 5, n. 1, p. 42–52, 2010. Citado na p. 51.

PEDRONI, V. A. **Circuit Design with VHDL**. Londres, Inglaterra: MIT Press, 2004. ISBN 9780262162241. Citado nas pp. 7 e 31.

SANTOS, C. E. da S. **Seleção de parâmetros de máquinas de vetores de suporte usando otimização multiobjetivo baseada em meta-heurísticas**. Tese (Doutorado) — Universidade de Brasília, Brasília, DF, 2019. Citado nas pp. 7, 24, 25, 26, 27, 28, 48, 49 e 51.

STEINWART, I.; CHRISTMANN, A. Sparsity of SVMs that use the epsilon-insensitive loss. **Advances in Neural Information Processing Systems**, v. 21, 2008. Citado na p. 27.

TARANTINO, S.; CLEMENTE, F.; BARONE, D.; CONTROZZI, M.; CIPRIANI, C. The myokinetic control interface: tracking implanted magnets as a means for prosthetic control. **Scientific Reports**, v. 7, n. 17149, 2017. Citado nas pp. 7, 14, 15, 22 e 23.

WEIR, R. F.; TROYK, P. R.; A., M. D. G.; KERNS, D. A.; SCHORSCH, J. F.; MAAS, H. Implantable myoelectric sensors (IMESs) for intramuscular electromyogram recording. **IEEE Transactions on Biomedical Engineering**, v. 56, n. 1, p. 159–171, 2009. Citado na p. 21.

ZUO, S.; SCHMALZ, J.; ÖZDEN, M. Ömür; GERKEN, M.; SU, J.; NIEKIEL, F.; LOFINK, F.; NAZARPOUR, K.; HEIDARI, H. Ultrasensitive magnetoelectric sensing system for pico-tesla magnetomyography. **IEEE Transactions on biomedical circuits and systems**, v. 14, n. 5, p. 971–984, 2020. Citado na p. 21.