UNIVERSIDADE ESTADUAL DE CAMPINAS

Faculdade de Engenharia Mecânica

FERNANDO ORTOLANO

Proposta de uma arquitetura de processamento paralelo usando FPGA para o controle de vibrações em estruturas inteligentes sujeitas a danos

Campinas

2021

FERNANDO ORTOLANO

Proposta de uma arquitetura de processamento paralelo usando FPGA para o controle de vibrações em estruturas inteligentes sujeitas a danos

Dissertação de Mestrado apresentada à Faculdade de Engenharia Mecânica da Universidade Estadual de Campinas como parte dos requisitos exigidos para a obtenção do título de Mestre em Engenharia Mecânica, na Área de Mecatrônica.

Orientador: Prof. Dr. Helói Francisco Gentil Genari

Coorientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

ESTE TRABALHO CORRESPONDE À VERSÃO FINAL DA DISSERTAÇÃO DE MESTRADO DEFENDIDA PELO ALUNO FERNANDO ORTOLANO, E ORIENTADA PELO PROF. DR. HELÓI FRANCISCO GENTIL GENARI.

Campinas

2021

Ficha catalográfica
Universidade Estadual de Campinas
Biblioteca da Área de Engenharia e Arquitetura
Rose Meire da Silva - CRB 8/5974

Or8p

Ortolano, Fernando, 1986-
Proposta de uma arquitetura de processamento paralelo usando FPGA para o controle de vibrações em estruturas inteligentes sujeitas a danos / Fernando Ortolano. – Campinas, SP : [s.n.], 2021.

Orientador: Helói Francisco Gentil Genari.
Coorientador: Eurípedes Guilherme de Oliveira Nóbrega.
Dissertação (mestrado) – Universidade Estadual de Campinas, Faculdade de Engenharia Mecânica.

1. Vibração-controle. 2. Identificação de sistemas. 3. Processamento paralelo (Computadores). 4. FPGA (Arranjo de lógica programável em campo). 5. Materiais inteligentes. I. Genari, Helói Francisco Gentil, 1985-. II. Nóbrega, Eurípedes Guilherme de Oliveira, 1950-. III. Universidade Estadual de Campinas. Faculdade de Engenharia Mecânica. IV. Título.

Informações para Biblioteca Digital

Título em outro idioma: Proposal for a parallel processing architecture using FPGA for the vibration control of smart structures subject to damage
Palavras-chave em inglês:
Vibration - Control
System Identification
Parallel processing
FPGA (Field programmable gate array)
Smart materials
Área de concentração: Mecatrônica
Titulação: Mestre em Engenharia Mecânica
Banca examinadora:
Helói Francisco Gentil Genari [Orientador]
André Ricardo Fioravanti
Daniel Rodrigues Pipa
Data de defesa: 17-03-2021
Programa de Pós-Graduação: Engenharia Mecânica

Identificação e informações acadêmicas do(a) aluno(a)
- ORCID do autor: https://orcid.org/0000-0001-9591-6449
- Currículo Lattes do autor: http://lattes.cnpq.br/9421369073428558

UNIVERSIDADE ESTADUAL DE CAMPINAS
FACULDADE DE ENGENHARIA MECÂNICA

DISSERTAÇÃO DE MESTRADO

Proposta de uma arquitetura de processamento paralelo usando FPGA para o controle de vibrações em estruturas inteligentes sujeitas a danos

Autor: Fernando Ortolano

Orientador: Prof. Dr. Helói Francisco Gentil Genari
Coorientador: Prof. Dr. Eurípedes Guilherme de Oliveira Nóbrega

A Banca Examinadora composta pelos membros abaixo aprovou esta Dissertação de Mestrado:

Prof. Dr. Helói Francisco Gentil Genari
CECS/UFABC

Dr. André Ricardo Fioravanti
DMC/FEM/UNICAMP

Prof. Dr. Daniel Rodrigues Pipa
DAELN/UTFPR

A Ata de Defesa com as respectivas assinaturas dos membros encontra-se no SIGA/Sistema de Fluxo de Dissertação/Tese e na Secretaria do Programa da Unidade.

Campinas, 17 de março de 2021

Dedico este trabalho à minha família, em especial à minha mãe e minha esposa.

AGRADECIMENTOS

Agradeço aos Professores Helói Francisco Gentil Genari e Eurípedes Guilherme de Oliveira Nóbrega pelos ensinamentos e pela enorme paciência na orientação deste trabalho.

Agradeço aos Professores do Departamento de Mecânica Computacional da FEM pelos ensinamentos.

Sou grato à minha família pelo apoio incondicional em momentos difíceis.

Sou grato ao meus colegas de UNICAMP pela amizade e convívio.

RESUMO

As estruturas constituídas por materiais inteligentes são uma realidade em diversos campos do conhecimento, em específico nas áreas de robótica, automobilismo, aeroespacial e na engenharia civil. A utilização de materiais inteligentes nessas estruturas permite que elas respondam a diferentes estímulos do meio ambiente de forma automática, monitorando a própria integridade e agindo para atenuar vibrações. Nesse contexto, este trabalho propõe o desenvolvimento de um sistema para controle de vibrações em estruturas inteligentes por meio de uma arquitetura de processamento paralelo que monitora periodicamente a estrutura e, em caso de dano, reconfigura o controlador para mitigar os efeitos do dano no desempenho. O monitoramento é feito utilizando a técnica por subespaços que compara dois modelos. O primeiro é considerado como modelo de referência e representa a estrutura saudável, enquanto, o segundo é identificado continuamente, utilizando uma técnica de realização de autossistema (ERA, de *Eigen System Realization Algoritm*). A partir da distância Euclidiana entre o modelo de referência e o modelo identificado, é possível acompanhar a severidade do dano e, quando necessário, atualizar os ganhos do controlador. Essa metodologia é implementada utilizando uma arquitetura em rede compreendida por um microcontrolador Raspberry Pi atuando como *gateway* local, comandando os sensores e atuadores da estrutura. Além disso, utiliza-se um kit SoC como servidor, denominado DE10-Nano, que contém um processador ARM e um FPGA encapsulado no mesmo chip. O Raspberry Pi envia ao DE10-Nano os parâmetros de Markov da estrutura, a partir do processamento dos sinais de vibrações. Em seguida, o kit DE10-Nano identifica o modelo da estrutura flexível, calcula a distância entre os vetores pertencentes aos subespaços do modelo identificado e do modelo de referência, reprojeta os ganhos do controlador e, caso necessário, os envia ao Raspberry Pi, tornando-o habilitado a realizar o controle da vibração estrutural. Com o intuito de acelerar o cálculo numérico presente na técnica de identificação ERA, o processador ARM requisita o FPGA, programado em OpenCL, para executar o cálculo de algumas funções em paralelo, como a construção das matrizes de Hankel, a decomposição em valores singulares e a multiplicação matricial. Para a validação da proposta de arquitetura e a metodologia de controle tolerante a danos, foi utilizado uma estrutura vertical flexível que simula um prédio alto constituído de uma massa ativa em seu topo. Um motor de corrente contínua é utilizado para mover a base da estrutura e um outro motor similar é aplicado para deslocar a massa ativa. O sistema de controle é projetado utilizando a realimentação de estados, estimados com um observador de Luenberger. Assim, o controlador gera uma trajetória especí-

fica para a massa ativa, visando minimizar a influência do distúrbio na vibração estrutural. Em caso de danos provocados por fadiga, por exemplo, o sistema de controle identifica a mudança na dinâmica estrutural e atualiza automaticamente o controlador e o observador, garantindo a estabilidade e mitigando os efeitos do dano no desempenho. Os resultados experimentais mostram a eficácia da abordagem proposta em identificar o modelo estrutural, detectar a presença de danos e reduzir a vibração quando a estrutura está sujeita a perturbações e danos.

Palavras–chave: Vibração-controle, Identificação de Sistemas, Processamento Paralelo, FPGA, Materiais inteligentes.

ABSTRACT

Structures made up of smart materials are a reality in several fields of knowledge, specifically in the areas of robotics, automobile industry, aerospace, and civil engineering. The use of smart materials in these structures allows them to respond automatically to different stimuli from the environment, monitoring their own integrity and acting to attenuate vibrations. In this context, this work proposes the development of a framework to control vibrations in smart structures using a parallel processing architecture that periodically monitors the structure and, in the case of damage, reconfigures the controller to mitigate damage effects on the performance. The monitoring process is done using a subspace technique that compares two models. The first model is the reference model and represents the healthy structure while the second model is identified continuously, employing the eigen system realization algorithm (ERA). From the Euclidean distance between the reference model and the identified one, it is possible to monitor the damage severity and, when necessary, reconfigure the controller's gains. This methodology is implemented using a network architecture comprised of a Raspberry Pi microcontroller acting as a local gateway, controlling the structure's sensors and actuators. Additionally, a system-on-a-chip kit is used as a server, called DE10-Nano, which contains an ARM processor and an FPGA, both encapsulated in the same chip. The Raspberry Pi sends the structure Markov parameters to the DE10-Nano, obtained from the processing of the vibration signals. Then, the DE10-Nano kit identifies the flexible structure model, computing the Euclidean distance between the vectors belonging to the subspaces of the identified model and of the reference model, redesigning the controller and observer gains, and, if necessary, sends them to the Raspberry Pi, enabling it to perform the structural vibration control. In order to speed up the numerical calculation present in the ERA identification technique, the ARM processor requests the FPGA, programmed in OpenCL, to execute the calculation of some functions in parallel such as the construction of Hankel matrices, the decomposition into singular values, and matrix multiplication. For the validation of the proposed architecture and damage-tolerant active control methodology, a flexible vertical structure is used to simulate a tall building with an active mass on it. A direct current motor is used to move the structure base and another similar motor is used to move the active mass. The control system is designed using the state feedback strategy, estimated with a Luenberger observer. Thus, the controller generates a specific trajectory for the active mass, aiming to minimize the disturbance influence on the structural vibration. At the damage occurrence caused by fatigue, for example, the control system identifies the change in the structural

dynamics and automatically reconfigures the controller and the observer, ensuring stability and mitigating the damage effects on the performance. The experimental results show the effectiveness of the proposed approach in identifying the structure model, detecting the presence of damage and reducing vibration when the structure is subject to disturbances and damage.

Keywords: Vibration - Control, System Identification, Parallel processing, FPGA (Field programable gate array), Smart materials.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

AMD	Amortecedor de Massa Ativa, do inglês *Active Mass Driver*
AOC	Compilador Offline do Altera, do inglês *Altera Offline Compiler*
API	Programas de Interface de Aplicação, do inglês *Application Programming Interface*
ARM	Máquina RISC Avançada, do inglês *Advanced RISC Machine*
ATAC	Controle Ativo Tolerante Adaptativo, do inglês *Adaptive Tolerant Active Control*
AVC	Controle Ativo de Vibração, do inglês *Active Vibration Control*
BSP	Pacote de Suporte de Placa, do inglês *Board Support Package*
CPU	Unidade Central de Processamento, do inglês *Central Processing Unit*
DDR SDRAM	Memória de Acesso Aleatório Dinâmica Síncrona com taxa de dados Dupla, do inglês *Double Data Rate Synchronous Dynamic Random-Access Memory*
DSP	Processador de Sinal Digital, do inglês *Digital Signal Processor*
DTAC	Controle Ativo Tolerante a Danos, do inglês *Damage-Tolerant Active Control*
EDAC	Controle Ativo de Danos em Evolução, do inglês *Evolving Damage Active Control*
ERA	Algoritmo de Realização de Autossistemas, do inglês *Eigensystem Realization Algorithm*
FPGA	Arranjo de Portas Programáveis em Campo, do inglês *Field-Programmable Gate Array*
FDI	Detecção e Isolamento de Falhas, do inglês *Fault Detection and Isolation*

FTC	Controle Tolerante a Falha, do inglês *Faut-Tolerant Control*
FRF	Função de Resposta em Frequência, do inglês *Frequency Response Function*
GPU	Unidade de Processamento Gráfica, do inglês *Graphics Processing Unit*
MIMD	Múltiplas-Instruções Múltiplos-Dados, do inglês *Multiple-Instructions Multiple-Data*
MIMO	Múltiplas Entradas e Múltiplas Saídas, do inglês *Multiple-Input-Multiple-Output*
NExT	Técnica de Excitação Natural, do inglês *Natural Excitation Technique*
OpenCL	Linguagem de Computação Aberta, do inglês *Open Computing Language*
PAC	Controle Ativo Preventivo, do inglês *Preventive Active Control*
PZT	Titanato Zirconato de Chumbo, do inglês *Lead Zirconate Titanate*
PE	Elemento de Processamento, do inglês *Processor Element*
PWM	Modulação por Largura de Pulso, do inglês *Pulse-Width Modulation*
RISC	Cálculo de Conjunto de Instruções Reduzido, do inglês *Reduced Instruction Set Compute*
RAM	Memória de Acesso Aleatório, do inglês *Random-Access Memory*
SDK	Kit de Desenvolvimento de Software, do inglês *Software Development Kit*
SIMD	Instrução-Única Múltiplos-Dados, do inglês *Single-Instructions Multiple-Data*
SoC	Sistema no Chip, do inglês *System-on-a-Chip*
SHM	Monitoramento da Integridade Estrutural, do inglês *Structural Health Monitoring*
STAC	Controle Ativo Estritamente Tolerante, do inglês *Strictly-Tolerant Active Control*

SVD	Decomposição em Valores Singulares, do inglês *Singular Value Decomposition*
UDP	Protocolo de Transmissão de Dados via Internet, do inglês *User Datagram Protocol*
VHDL	Linguagem de Descrição de Hardware VHSIC, do inglês *VHSIC Hardware Description Language*
VHSIC	Circuitos Integrados de Altíssima Velocidade, do inglês *Very High Speed Integrated Circuits*

LISTA DE SÍMBOLOS

$\mathbf{A}^{-1}$	inversa da matriz $\mathbf{A}$
$\mathbf{A}^{T}$	transposta da matriz $\mathbf{A}$
$\mathbf{A}'$	matriz modificada após transformação ortogonal de $\mathbf{A}$
$\mathbf{v}'$	vetor modificado após transformação ortogonal de $\mathbf{v}$
$\mathbf{x}_k$	k-ésimo elemento do vetor $\mathbf{x}$
$x_{k,y}$	elemento pertencente k-ésima linha e y-ésima coluna de uma matriz $\mathbf{X}$
$\mathbf{u}^{\perp}$	vetor perpendicular de $\mathbf{u}$
$\|\| \cdot \|\|$	norma euclidiana
$\| \cdot \|$	valor absoluto de um escalar
$\hat{\mathbf{i}}$	versor do vetor $\mathbf{i}$
$\cos$	cosseno
$\sin$	seno
$\tan$	tangente
$\mathbf{H}_h$	matriz ortogonal de reflexão de Householder
$\mathbf{G}_v$	matriz ortogonal de rotação de Givens
$\mathbf{U}, \mathbf{V}$	matrizes que formam bases ortogonais
$\mathbf{\Sigma}$	matriz diagonal
$\mathbf{B}_{idiag}$	matriz bidiagonal obtida de sussessivas aplicações da reflexão de Householder em ambos os lados de uma determinada matiz
$\mathbf{P}$	matriz de reflexão de Householder aplicada à direita de uma matriz
$\mathbf{Q}$	matriz de reflexão de Householder aplicada à esquerda de uma matriz

$\mathbf{G}_l$	matriz de rotação de Givens aplicada à esquerda de uma matriz
$\mathbf{G}_r$	matriz de rotação de Givens aplicada à direita de uma matriz
ν	posição do vetor constituído pelos elementos da diagonal superior de uma matriz bidiagonal
δ	valor de tolerância para considerar um elemento nulo no procedimento de diagonalização de uma matriz bidiagonal
ι	numero máximo de iterações permitido para diagonalizar uma matriz bidiagonal
$\mathbf{O}$	base ortogonal obtida por meio de sussessivas aplicações da rotação de Jacobi sobre uma determinada matiz
$\mathbf{J}$	matriz de rotação de Jacobi aplicada à esquerda de uma matriz
$\mathbf{a}_p^T$	vetor referente a p-ésima coluna de uma matriz $\mathbf{A}$
$\mathbf{a}_p^{T'}$	vetor modificado da p-ésima coluna de uma matriz $\mathbf{A}$
υ	quantidade de varreduras necessárias para obter uma matriz ortogonal
$\aleph$	quantidade de rotações de Jacobi aplicadas
$\mathbf{W}$	vetor cujos elementos indicam as colunas nas quais serão aplicadas as rotações de Jacobi
tol	valor de tolerância para considerar a matriz ortogonal
!	fatorial de um número
$\mathrm{sign}(\cdot)$	sinal de um número, indicando se é positivo ou negativo
$\mathrm{qr}(\cdot)$	função que calcula a decomposição QR de uma matriz
$\mathrm{eye}(n)$	matriz identidade de ordem n
$\star$	valor modificado
$\clubsuit$	valor inserido em uma iteração e aniquilado na próxima iteração

$\mathbf{M}$	matriz de massa
$\mathbf{E}$	matriz de amortecimento
$\mathbf{K}$	matriz de rigidez
t	tempo
$\mathbf{w}(t)$	entrada de perturbação
$\mathbf{u}(t)$	entrada de controle
$\mathbf{y}(t)$	resposta às entradas
$\mathbf{p}(t)$	deslocamento nodal
$\mathbf{B}_w$	matriz relativa às entradas $\mathbf{w}(t)$
$\mathbf{B}_u$	matriz relativa às entradas $\mathbf{u}(t)$
$\mathbf{C}_p$	matriz de saída relativa à $\mathbf{p}(t)$
$\mathbf{C}_v$	matriz de saída relativa à $\dot{\mathbf{p}}(t)$
$\mathbf{C}_w$	matriz de saída relativa à $\mathbf{w}(t)$
$\mathbf{C}_u$	matriz de saída relativa à $\mathbf{u}(t)$
$\mathbf{x}(t)$	vetor de estado
$\mathbf{A}$	matriz de transmissão dos estados $\mathbf{x}(t)$
$\mathbf{B}_1$	matriz das entradas de perturbação $\mathbf{w}(t)$
$\mathbf{B}_2$	matriz das entradas de controle $\mathbf{u}(t)$
$\mathbf{C}$	matriz de saídas
$\mathbf{D}_1$	matriz de transmissão direta relativa às entradas $\mathbf{w}(t)$
$\mathbf{D}_2$	matriz de transmissão direta relativa às entradas $\mathbf{u}(t)$
$\mathbf{x}[k]$	representação do vetor $\mathbf{x}(t)$ no instante discreto k
$\mathbf{A}_d$	representação da matriz $\mathbf{A}$ no tempo discreto

$\mathbf{B}_{1d}$	representação da matriz $\mathbf{B}_1$ no tempo discreto
$\mathbf{B}_{2d}$	representação da matriz $\mathbf{B}_2$ no tempo discreto
$\mathbf{C_d}$	representação da matriz $\mathbf{C}$ no tempo discreto
$\mathbf{D}_{1d}$	representação da matriz $\mathbf{D}_1$ no tempo discreto
$\mathbf{D}_{2d}$	representação da matriz $\mathbf{D}_2$ no tempo discreto
ξ_i	fator de amortecimento pertencente ao i-ésimo modo de vibração
ω_{n_i}	frequência natural relativa ao i-ésimo modo de vibração
$\mathbf{T}_{yw}$	função de transferência entre a saída y e a entrada w
$\mathbf{T}_{yu}$	função de transferência entre a saída y e a entrada u
$\mathbf{P}_{yw}$	função de resposta em frequência entre a saída y e a entrada w
$\mathbf{P}_{yu}$	função de resposta em frequência entre a saída y e a entrada u
$\mathbf{G}_1[k]$	matriz com parâmetros de Markov relativos à entrada $w[k]$
$\mathbf{G}_2[k]$	matriz com parâmetros de Markov relativos à entrada $u[k]$
$\mathbf{M}_{kv}[l]$	matriz com parâmetros de Markov relativos às entradas do sistema
$\mathbf{H}p, q[l]$	matriz de Hankel com dimensão dada em função dos parâmetros p e q, cujo primeiro elemento é matriz Markov $\mathbf{M}_{kv}[l]$
$\mathbf{\Delta}_q$	matriz de controlabilidade, dada em função de q
$\mathbf{\Gamma}_p$	matriz de observabilidade, dada em função de p
$\hat{\mathbf{x}}[k]$	vetor de estados estimados no instante discreto k
$\hat{\mathbf{y}}[k]$	saídas estimadas no instante discreto k
$\mathbf{K}_g$	vetor de ganhos do controlador
$\mathbf{L}_g$	vetor de ganhos do observador
M_{odel}	modelo identificado pela técnica ERA

$\mathcal{O}_\infty(M_{odel})$ matriz intinifa de observavilidade de M_{odel}

$D_s(M_{odel}^{(1)}, M_{odel}^{(2)})$ distância entre os subespaços formados por dois modelos $M_{odel}^{(1)}$ e $M_{odel}^{(2)}$

SUMÁRIO

1 INTRODUÇÃO

1.1 Motivações

A última revolução no campo da ciência dos materiais permitiu o desenvolvimento de estruturas inteligentes que são amplamente utilizadas em aplicações de engenharia, como em peças de aeronaves, antenas, sistemas robóticos, equipamentos eletrônicos e na construção civil. A inteligência consiste na capacidade que os sensores e atuadores incorporados a essas estruturas possuem de alterar as suas próprias propriedades estruturais em resposta a estímulos (Fisco; Adeli, 2011). Dessa forma, é possível realizar o controle da suspensão em veículos automotores para promover segurança e conforto ao passageiro (Du *et al.*, 2005), construir edifícios que conseguem reagir às condições climáticas (Sun *et al.*, 2018), entre outras aplicações. Diferentes materiais multifuncionais podem ser acoplados a essas estruturas como atuadores e sensores responsivos. Por exemplo, no controle de vibrações estruturais são utilizados atualmente fluidos magneto-reológicos em amortecedores para atenuar os efeitos das vibrações devido a eventos sísmicos, ventos, movimentos rítmicos de pessoas e/ou tráfego em pontes ou arranha-céus. O estado físico desses fluidos é modificado na presença de um campo magnético alterando abruptamente sua viscosidade (K Gholami M, 2017). Além dessa classe de fluidos, os materiais piezelétricos são largamente utilizados em estruturas inteligentes, pois eles convertem energia mecânica em energia elétrica e vice-versa, podendo o mesmo componente ser utilizado como atuador ou sensor, dependendo do objetivo da estrutura (Daraji *et al.*, 2018; Chen *et al.*, 2019). Através da análise dos valores medidos e enviados a esses transdutores, é possível monitorar a saúde de estruturas inteligentes, diagnosticando e reagindo a estados anormais e, assim, preservando a sua funcionalidade original.

O monitoramento da integridade estrutural (SHM, de *Structural Health Monitoring*) tem como objetivo fornecer um diagnóstico da condição física das estruturas e detectar a existência de possíveis danos estruturais. Danos são definidos como alterações nas propriedades materiais e/ou geométricas da estrutura, que afetam o seu estado atual e desempenho futuro (Chen; Ni, 2018). Além da possibilidade de prolongar a vida útil de uma estrutura por meio da detecção e manutenção precoce de danos, o monitoramento contínuo do estado de saúde estrutural permite construir estruturas com *design* mais sofisticado e mais seguras. Os métodos de SHM envolvem a integração de sensores, os quais possivelmente são materiais

inteligentes, transmissão de dados, potência computacional e capacidade de processamento embarcado (Balageas *et al.*, 2010). Encontra-se na literatura muitas aplicações, com diferentes abordagens, que realizam o monitoramento da saúde estrutural em estruturas inteligentes tais como: impedância (Martowicz *et al.*, 2016; Na; Baek, 2018), ondas *Lamb* (Souza; Nóbrega, 2012; Stepinski *et al.*, 2017), vibração estrutural (Tsogka *et al.*, 2017; Tang *et al.*, 2020), entre outros. Os métodos baseados em vibração estrutural fornecem informações valiosas sobre os impactos dos danos nas vibrações estruturais. Eles permitem detectar, localizar e analisar a gravidade dos danos causados por mudanças nas características de vibração (Genari, 2016). Neste trabalho, este tipo de estratégia SHM é adotada para gerar informações de danos utilizadas no projeto e/ou reconfiguração do controlador.

Com esse desenvolvimento tecnológico está sendo possível construir estruturas cada vez maiores e mais leves. Consequentemente, as estruturas estão cada vez mais flexíveis e suscetíveis a vibrações causadas por diferentes agentes. Dessa forma, o interesse no controle de vibrações estruturais vem crescendo nas últimas décadas (Wang *et al.*, 1983; Abreu; Lopes Jr., 2010; Genari *et al.*, 2017a; Ortolano *et al.*, 2018), tendo como destaque o controle ativo devido ao melhor desempenho em relação ao semi-ativo e passivo (Preumont, 2011). Nos métodos de controle ativo, os componentes do sistema de controle utilizam energia externa e a implementação do controlador é geralmente baseada para operar em uma banda de frequência de interesse. Existem diversas técnicas de controle de vibrações ativo (AVC, de *Active Vibration Control*) para atenuar as vibrações estruturais. Por exemplo, Yu *et al.*, 2014 apresentaram o uso dos controladores Proporcional-Derivativo e Proporcional-Integral-Derivativo, sintonizados segundo a teoria de estabilidade de Lyapunov, para controlar as vibrações do protótipo de um prédio com dois andares. Abdulameer; Wasmi,2015 realizaram o controle simulado e experimental da vibração na asa de uma aeronave usando transdutores piezelétricos. Kim *et al.*, 2016 utilizaram um controlador Regulador Quadrático Linear para atenuar os efeitos do vento sobre uma estrutura vertical alta. Ali *et al.*, 2016 projetaram um controlador ativo de vibração para a asa de uma aeronave, baseado na realimentação de velocidade. He *et al.*, 2019 analisaram o posicionamento de sensores para determinar a vibração tolerável em uma asa flexível de uma aeronave, sob um determinada faixa de velocidade de vôo, no qual a vibração da asa é avaliada pelo desempenho da norma H_2, modelada por um controlador linear com parâmetros variantes. Entretanto, nessa abordagem tradicional de AVC, os efeitos do dano na dinâmica da estrutura são geralmente desconsiderados no projeto dos controladores, podendo levar o sistema de con-

trole a instabilidade ou desempenho insatisfatório quando um dano vier a ocorrer. Dessa forma, é fundamental incluir a possibilidade de danos no projeto de sistema de controle (Genari *et al.*, 2017b).

Os sistemas complexos de engenharia tais como estruturas aeronáuticas, dinâmica veicular, processos químicos, turbinas eólicas de geração de energia e equipamentos eletrônicos industriais possuem critérios críticos de segurança que exigem uma demanda cada vez maior de confiabilidade. A implementação de controladores clássicos nessas aplicações considera que todos os componentes funcionem adequadamente. No entanto, nas situações práticas, esses componentes podem sofrer falhas na operação. Dessa forma, o projeto de controle convencional pode resultar na perda de estabilidade e de desempenho satisfatório e, assim, novas abordagens têm sido implementadas com o intuito de tolerar o mau funcionamento dos componentes e acomodar algumas falhas automaticamente, mantendo a estabilidade e um adequado nível de desempenho (Zhang; Jiang, 2008; Abbaspour *et al.*, 2020). Estes tipos de sistemas são denominados como Controle Tolerante a Falhas (FTC, de *Faut-Tolerant Control*). Tendo em vista a dependência de informações das possíveis falhas, os sistemas FTC podem ser classificados em duas classes principais: passivo e ativo. Vale ressaltar que esta definição é completamente diferente da área de controle de vibração, cujos termos ativos e passivos são relativos ao uso ou não de uma fonte externa de energia (Genari, 2016).

Na abordagem FTC passiva, não existe dependência de informações sobre as falhas no sistema e o controle é projetado para ser fixo e robusto, considerando uma classe de falhas pré-estabelecidas (Alwi *et al.*, 2011; Moor, 2016). Em geral, a redundância é incorporada ao projeto do FTC passivo para torná-lo robusto (Yu; Jiang, 2015). Entretanto, este procedimento não garante a estabilidade e um desempenho satisfatório na ocorrência de falhas não previstas. Em contrapartida, os sistemas com FTC ativos realizam um procedimento específico denominado detecção e isolamento de falhas (FDI, de *Fault Detection and Isolation*) para encontrar a localização da falha e medir sua intensidade (Klimkhieo, 2009). Em seguida, um módulo de supervisão decide a estratégia para reconfigurar o controlador, visando compensar ou mitigar a falha (Tabbache *et al.*, 2012; Zhang *et al.*, 2016).

Recentemente, após intensa investigação multidisciplinar entre SHM e FTC, os conceitos de Controle Ativo Tolerante a Danos (DTAC, de *Damage-Tolerant Active Control*) foram propostos por Mechbal; Nóbrega,2012, visando trazer operações mais seguras e eficientes para estruturas flexíveis inteligentes. O método DTAC pode ser considerado uma extensão do

FTC em conjunto com AVC, no controle de vibrações de estruturas inteligentes (Genari *et al.*, 2017c). Para isso, da mesma forma que na abordagem FTC, o método DTAC é baseado em SHM para projetar controladores que apresentam tolerância a danos (Mechbal; Nóbrega, 2012). No entanto, a adaptação dos métodos FTC no controle de vibração de estruturas flexíveis danificadas levam a novos desafios e, consequentemente, soluções específicas da área DTAC (Mechbal; Nóbrega, 2015b).

As estratégias DTAC são baseadas na condição de estado da estrutura a fim de alcançar desempenho satisfatório do controle de vibração na ocorrência de danos, controlando fluxo de energia ou isolando ativamente a vibração em regiões danificadas. Considerando a estrutura saudável, Mechbal; Nóbrega,2012 apresentaram duas estratégias possíveis. A primeira é denominada Controle Ativo Estritamente Tolerante (STAC, de *Strictly-Tolerant Active Control*), o qual se baseia na implementação de um controlador clássico não reconfigurável, tendo robustez suficiente para garantir um desempenho aceitável sob certos tipos de danos previsíveis. Nesta configuração de robustez, existe a possibilidade que o sistema de controle tenha um desempenho insatisfatório na ausência de danos, devido a oposição dessas duas características, robustez e desempenho. Com o intuito de contornar essa deficiência, Genari *et al.*, 2017a apresentaram uma abordagem modal e robusta, cuja estrutura saudável e danificada possuem um nível de desempenho adequado com o mesmo controlador. A segunda estratégia do DTAC consiste em realizar o Controle Ativo Preventivo (PAC, de *Preventive Active Control*) a fim de evitar ou retardar a ocorrência de danos. Ambrosio *et al.* 2014 apresentam um exemplo similar dessa estratégia, no qual o controlador preventivo é obtido mediante a minimização de uma função de custo que possui como parâmetro a fadiga do material. Seguindo a estratégia PAC, Genari *et al.*, 2017b desenvolveram um controlador H_∞ modal para reduzir a vibração global da estrutura sujeita a danos. Para isso, utilizaram as informações dos danos preexistentes, inseridas na norma H_∞ modal, para projetar um controlador robusto e tolerante.

Caso ocorra a identificação, localização e quantificação de danos estruturais, uma das estratégias pospostas por Mechbal; Nóbrega, 2012 consiste em implementar o Controle Ativo de Danos em Evolução (EDAC, de *Evolving Damage Active Control*). Esta abordagem compreende em reduzir a energia de vibração na região danificada, com o intuito de evitar a progressão dos danos, conforme apresentado em Mechbal; Nóbrega, 2015b. Genari *et al.*, 2017b utilizaram também a informação de dano, inserida como requisito de projeto, para projetar um controlador com desempenho adequado para a estrutura saudável e, em caso de dano, reduzisse

o fluxo de energia na região afetada, objetivando evitar e/ou reduzir a propagação do dano. Outra estratégia DTAC é denominada Controle Ativo Tolerante Adaptável (ATAC, de *Adaptive Tolerant Active Control*), a qual detecta automaticamente e, em seguida, acomoda os danos com o intuito de fornecer um desempenho aceitável para o sistema controlado em todas as circunstâncias. Esta estratégia foi implementada em Mechbal; Nóbrega, 2015a, no qual foi utilizado um módulo SHM baseado em ondas *Lamb* para localizar os danos na estrutura e, na sequência, implementar a reconfiguração de um controlador H_∞ espacial, a fim de reduzir a energia de vibração na região detectada e danificada. Genari *et al.*, 2017c também realizaram a estratégia, baseado em um sistema de controle modal de dupla malha. O primeiro controlador é projetado para cumprir os requisitos de desempenho e robustez para a estrutura saudável e o segundo controlador é adaptativo, utilizado para mitigar os efeitos dos danos no desempenho estrutural.

Vale mencionar que na literatura atual, os sistemas de controle são projetados de tal forma que o controlador e a identificação da planta do sistema ocorrem simultaneamente, sob um processo de otimização. Ao contrário do sistema de controle adotado neste trabalho, o qual é considerado invariante no tempo, existem técnicas de controle adaptativo sobre parâmetros variantes (Sun *et al.*, 2016; Wu; Shao, 2017; Liu *et al.*, 2020).

1.2 Objetivos

O objetivo central dessa dissertação é desenvolver um sistema embarcado de controle para atenuar vibrações em estruturas inteligentes flexíveis sujeitas a danos. Como um primeiro objetivo parcial, a estratégia proposta consiste em automatizar um sistema de controle que monitore a estrutura periodicamente. Caso seja detectado um dano na estrutura, o sistema de controle é reconfigurado de forma automática, garantindo estabilidade e desempenho. A Figura 1.1 apresenta o fluxograma da estratégia DTAC desenvolvida na dissertação. Inicialmente, o modelo de referência que representa a estrutura saudável e os parâmetros do sistema de controle são carregados com o intuito de atenuar a vibração quando a estrutura for sujeita a perturbações. Em seguida, é necessário avaliar a presença de perturbações para decidir entre realizar o monitoramento da integridade ou o controle estrutural. Essa tarefa tem prioridade máxima de execução, ou seja, uma vez que exista perturbação, o sistema de controle atua para minimizar a vibração estrutural, de acordo com o modelo de referência atual. Caso contrário, o sistema deve realizará o monitoramento da integridade mediante a identificação periódica do modelo que representa a estrutura. Na hipótese da estrutura estar saudável, a estratégia consiste

em retornar a função que avalia a presença de distúrbio. Se o algoritmo detectar danos, o sistema de controle deve ser reconfigurado. Por fim, o modelo de referência é atualizado e o ciclo recomeça automaticamente.

Como segundo objetivo, a estratégia de controle presente na Figura 1.1 é implementada utilizando dois dispositivos: um Raspberry Pi® e um Terasic DE10-Nano®, que contem um processador ARM e um FPGA, encapsulados no mesmo chip. O Raspberry Pi é utilizado como um *gateway*, comandando os sensores, observando os estados, gerando o sinal de controle e selecionando o modo de operação. No monitoramento, a métrica de detecção de danos e a identificação da estrutura, com os dados dos sensores e atuadores enviados pelo Raspberry Pi para o Terasic DE10-Nano, são implementadas no processador ARM, entretanto, as funções mais custosas, como montar a matriz de Hankel, calcular determinados passos da SVD e algumas multiplicações matriciais, são executadas de forma paralela pelo FPGA. No caso de ocorrência de um dano ou aumento de sua severidade, o processador ARM é responsável por reconfigurar o controlador e o observador, enviando as matrizes atualizadas do sistema de controle para o Raspberry Pi.

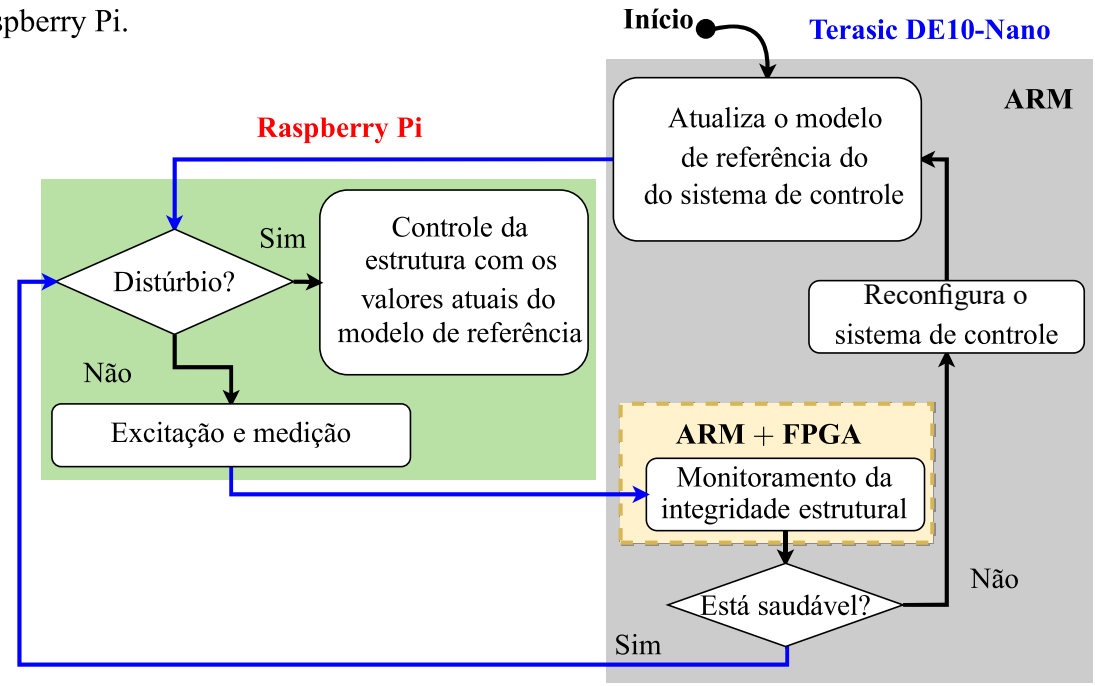


Figura 1.1 – Fluxograma de funcionamento do sistema embarcado

Por fim, o último objetivo é validar experimentalmente a estratégia de DTAC e a implementação do sistema embarcado em uma estrutura inteligente. Para isso, é utilizado uma estrutura flexível vertical que simula um prédio alto, sendo montada sobre uma base móvel, utilizada para simular o efeitos provocados por um evento sísmico. As oscilações laterais da estrutura são medidas por um sensor piezoelétrico denominado PZT, colocado em uma das co-

lunas. A redução da vibração é então obtida pelo movimento da massa ativa no topo da estrutura vertical, cuja trajetória é determinada pelo controlador. Para comandar os movimentos da estrutura, do amortecedor de massa ativa e adquirir o sinal do sensor piezoelétrico, o Raspberry Pi é utilizado como interface.

1.3 Estrutura do trabalho

O segundo capítulo apresenta os conceitos fundamentais de álgebra linear utilizados para realizar a decomposição em valores singulares de uma matriz e calcular a distância entre os subespaços de cada modelo identificado, utilizado na detecção de danos. Além disso, o capítulo descreve duas implementações práticas diferentes para realizar o cálculo da SVD, nas quais alguns de seus trechos são implementados em paralelo no sistema embarcado.

O terceiro capítulo apresenta a modelagem matemática de uma estrutura flexível no espaço de estados, a discretização de sistemas contínuos, o método de identificação ERA, o projeto do controlador via realimentação dos estados, estimados pelo observador de Luenberger, para reduzir a vibração estrutural. O capítulo é finalizado com a descrição de um método para detecção de danos, que utiliza a distância entre os subespaços dos modelos identificados.

O quarto capítulo descreve os principais programas implementados no sistema embarcado. Inicialmente, a arquitetura de programação paralela em OpenCL é apresentada e, em seguida, a compilação dos arquivos embarcados no servidor. Na sequência, são mostrados os principais programas implementados no processador ARM, o qual comanda a execução do DE10-Nano. Na intenção de acelerar o tempo de execução da identificação estrutural pela técnica ERA, as estratégias de paralelização do cálculo da matriz de Hankel e de duas implementações práticas, que calculam a decomposição em valores singulares de uma matriz, são apresentadas através dos programas enviados ao FPGA, para que ele execute determinadas etapas desses cálculos. Por último, são descritos os principais códigos implementados no cliente Raspberry Pi.

No quinto capítulo é apresentado a validação experimental do método de identificação e controle de vibrações em estruturas inteligentes sujeitas a danos. Para isso, o protótipo de um edifício alto é apresentado, bem como sua instrumentação. Em seguida, são apresentadas as formas dos sinais utilizados para identificação da estrutura e validação do sistema de controle. São induzidos no protótipo dois danos diferentes com severidades distintas, os quais são identificados pelo métodos de detecção utilizando subespaços. Quando o dano é identificado, o

sistema de controle é reprojetado automaticamente. Por fim, é realizado a comparação do tempo de execução da identificação ERA, utilizando os métodos disponíveis na biblioteca Eigen em relação a implementação paralela no FPGA.

O sexto capítulo apresenta as conclusões e as perspectivas futuras.

2 FUNDAMENTOS DE ÁLGEBRA LINEAR

A utilização e implementação da decomposição em valores singulares utilizada no algoritmo de identificação deste trabalho requer conhecer algumas propriedades fundamentais da álgebra de uma matriz, como o seu polinômio característico, seus autovalores e autovetores, bem como, a ortogonalidade entre vetores e bases ortonormais que formam seu espaço vetorial e transformações de similaridade. Esses atributos são a base para reduzir qualquer matriz em um produtos de três matrizes no qual uma delas é diagonal e composta pelos valores denominados singulares da matriz de entrada. As outras duas são matrizes formadas por diferentes base ortonormais relacionadas a esses valores. Os conceitos de álgebra linear descritos neste capítulo servem também de base para o entendimento de alguns conceitos descritos no capítulo seguinte, tais como a noção geométrica dos ângulos principais entre dois subespaços a partir das suas respectivas direções e o cálculo da distância utilizando esses subespaços.

2.1 Matrizes ortogonais

São matrizes cujas inversas podem ser obtidas por transposição, ou seja, uma matriz quadrada $\mathbf{P}_{quad}$ de ordem n é ortogonal se

$$\mathbf{P}_{quad}^{-1} = \mathbf{P}_{quad}^{T} \quad \text{ou} \quad \mathbf{P}_{quad}\mathbf{P}_{quad}^{T} = \mathbf{P}_{quad}^{T}\mathbf{P}_{quad} = \mathbf{I}, \tag{2.1}$$

em que $\mathbf{P}_{quad}$ é formada por n vetores linha e colunas ortonormais e, portanto, linearmente independentes. Consequentemente, o determinante de uma matriz ortogonal é igual a ± 1. Dessa forma, a multiplicação por esse tipo de matriz é dita como uma transformação ortogonal que mantêm inalterado a norma euclidiana de todos os vetores por ela multiplicados, o que equivale a rotacioná-los ou refleti-los no espaço vetorial (Anton; Rorres, 2017).

As transformações ortogonais são utilizadas por grande parte dos algoritmos de álgebra linear devido à estabilidade numérica que ela produz (Datta, 2009; Press *et al.*, 2007a). As mais empregadas para fatoração de matrizes são a reflexão de Householder e a rotação de Givens. Elas permitem anular determinadas componentes vetoriais de uma matriz, decompondo-a em uma forma compacta e similar a original. Esse processo facilita a resolução de vários problemas algébricos.

2.1.1 Reflexão de Householder

Com o intuito de anular as componentes (exceto uma) de um determinado vetor $\mathbf{v}$, pertencente a uma coluna ou linha da matriz $\mathbf{A} \in \mathbb{R}^{m \times n}$, é necessário encontrar primeiro o vetor $\mathbf{u}$ tal que a reflexão de $\mathbf{v}$ no hiperplano (subespaço de dimensão $n-1$) perpendicular a $\mathbf{u}$, resulte em um vetor $\mathbf{v}'$ que possua componente não-nula apenas na coordenada de base. De posse do vetor $\mathbf{u}$, calcula-se a matriz de Householder $\mathbf{H}_h$, cuja transformação ortogonal reflete todos os vetores de $\mathbf{A}$ em relação a $\mathbf{u}^{\perp}$. Para que ela ocorra, é necessário definir $\mathbf{u} = \mathbf{v} + \|\mathbf{v}\|\hat{\mathbf{i}}$ em que $\hat{\mathbf{i}}$ é o versor da coordenada de base, como mostra a Figura 2.1, para o caso em que o vetor $\mathbf{v}$ é descrito em função do par de coordenadas $(\mathbf{i}, \mathbf{j})$.

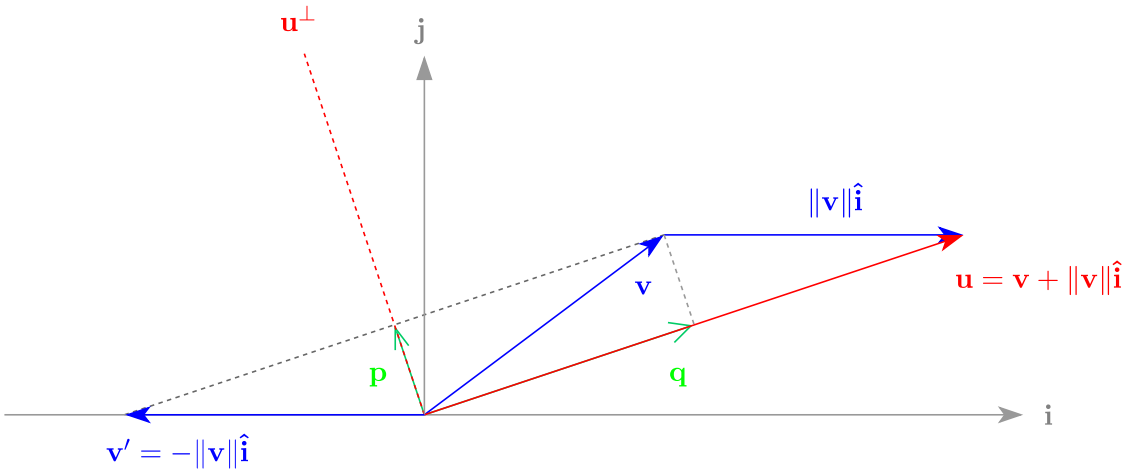


Figura 2.1 – Reflexão de $\mathbf{v}$ em relação a $\mathbf{u}^{\perp}$
Fonte: Adaptado de Barreto,2002.

As projeções ortogonais de $\mathbf{v}$ em $\mathbf{u}$ e $\mathbf{u}^{\perp}$ são dadas respectivamente por $\mathbf{q}$ e $\mathbf{p}$. Assim, o vetor refletido $\mathbf{v}'$ é definido da seguinte maneira

$$\mathbf{v}' = \mathbf{p} - \mathbf{q} = \mathbf{v} - proj_{\mathbf{u}}\mathbf{v} - proj_{\mathbf{u}}\mathbf{v} = \mathbf{v} - 2proj_{\mathbf{u}}\mathbf{v}, \tag{2.2}$$

em que

$$proj_{\mathbf{u}}\mathbf{v} = \frac{\langle \mathbf{u}, \mathbf{v} \rangle}{\|\mathbf{u}\|^2}\mathbf{u}. \tag{2.3}$$

Dessa forma, $\mathbf{v}'$ pode ser reescrito como

$$\mathbf{v}' = \mathbf{v} - 2\mathbf{u}\frac{\mathbf{u}^T\mathbf{x}}{\mathbf{u}^T\mathbf{u}}. \tag{2.4}$$

Colocando o vetor $\mathbf{v}$ em evidência no lado direito da Equação 2.4, chega-se a seguinte relação

$$\mathbf{v}' = \left(\mathbf{I} - 2\frac{\mathbf{u}\mathbf{u}^T}{\mathbf{u}^T\mathbf{u}}\right)\mathbf{v} = \mathbf{H}_h\mathbf{v}, \tag{2.5}$$

da qual é obtida $\mathbf{H}_h$. A transformação ortogonal provoca uma reflexão que preserva a norma $\|\mathbf{v}\| = \|\mathbf{v}'\|$.

2.1.2 Rotação de Givens

Outra transformação de similaridade frequentemente utilizada para fatoração de matrizes, consiste em rotacionar, por um certo ângulo θ, os vetores de uma matriz em um determinado plano de coordenadas (p, q), de modo que em uma dessas coordenadas, o vetor passa ter valor nulo (Demmel, 1997). Na Figura 2.2, o vetor $\mathbf{v}$ é rotacionado em relação ao eixo de coordenada $\mathbf{k}$, resultando no vetor $\mathbf{v}'$ com valor nulo em $\mathbf{j}$. O vetor $\mathbf{x}$ é a projeção de $\mathbf{v}$ no plano $p \times q$.

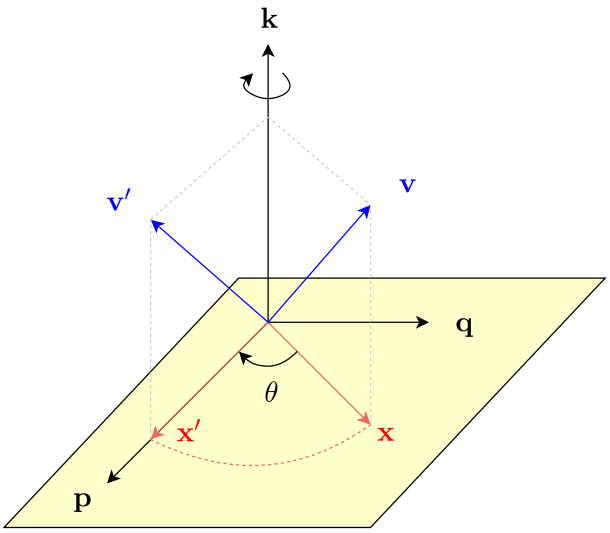


Figura 2.2 – Rotação de $\mathbf{v}$ em relação ao plano $p \times q$

O operador ortogonal $\mathbf{G}_v$ dessa transição é formado por uma matriz identidade de ordem n, exceto por quatro valores posicionados em relação ao plano de coordenadas no qual a rotação ocorre. Dessa maneira, eles são definidos por

$$\mathbf{G}_v(p,p) = \cos(\theta),\ \mathbf{G}_v(p,q) = -\sin(\theta),\ \mathbf{G}_v(q,p) = \sin(\theta) \text{ e } \mathbf{G}_v(q,q) = \cos(\theta), \tag{2.6}$$

em que

$$\cos(\theta) = \frac{x_p}{\sqrt{x_p^2 + x_q^2}},\ \sin(\theta) = \frac{x_q}{\sqrt{x_p^2 + x_q^2}}. \tag{2.7}$$

A matriz $\mathbf{G}_v$ é descrita em função dos valores $c = \cos(\theta)$ e $s = \sin(\theta)$, contidos nas interseções

das suas p-ésima e q-ésima linhas e colunas como

$$\mathbf{G}_v(p,q,\theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & -s & & & \\ & & s & c & & & \\ & & & & \ddots & & \\ & & & & & 1 & \end{bmatrix} \begin{matrix} \\ \\ p \\ q \\ \\ \\ \end{matrix} \quad , \tag{2.8}$$
$$\begin{matrix} & & p & q & & & \end{matrix}$$

sendo que p e q descritos abaixo de $\mathbf{G}_v$ se referem as suas colunas, enquanto que os elementos descritos à direita da matriz $\mathbf{G}_v$, as suas linhas. Isto é, os elementos não-nulos da matriz de Givens são dados por: $g_{k,k} = 1$ para $k \neq p, q$; $g_{p,p} = c$; $g_{k,k} = c$; $g_{q,q} = -s$ e $g_{p,q} = s$ para $p > q$. Tendo em vista que $c^2 + s^2 = 1$, a condição de ortogonalidade da Equação 2.1 é satisfeita e essa transformação provoca uma rotação que preserva a norma $\|\mathbf{v}\| = \|\mathbf{v}'\|$, alterando somente os valores das coordenadas p e q selecionadas.

2.2 Diagonalização ortogonal

Uma alternativa para simplificar uma matriz quadrada $\mathbf{A}$, de dimensão n e com autovalores reais, é utilizar a fatoração de Schur, a qual reduz $\mathbf{A}$ em uma matriz semelhante e triangular superior, cujos elementos da diagonal principal são os mesmos autovalores de $\mathbf{A}$ denominados λ. A decomposição ocorre mediante a multiplicação, em ambos os lados, de $\mathbf{A}$ por uma determinada matriz ortogonal $\mathbf{P}_{quad}$, tal que

$$\mathbf{P}_{quad}^T \mathbf{A} \mathbf{P}_{quad} = \mathbf{S}_h = \begin{bmatrix} \lambda_1 & \times & \cdots & \times \\ 0 & \lambda_2 & \cdots & \times \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}, \tag{2.9}$$

sendo possível converter $\mathbf{A}$ em uma matriz de Hessenberg superior por meio de outra matriz $\mathbf{P}$ como:

$$\mathbf{P}_{quad}^{T}\mathbf{A}\mathbf{P}_{quad} = \mathbf{H}_{e} = \begin{bmatrix} \times & \times & \cdots & \times & \times & \times \\ \times & \times & \cdots & \times & \times & \times \\ 0 & \times & \ddots & \times & \times & \times \\ \vdots & \vdots & \ddots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & \times & \times & \times \\ 0 & 0 & \cdots & 0 & \times & \times \end{bmatrix}. \tag{2.10}$$

Em particular, se a matriz $\mathbf{A}$ for também simétrica, a decomposição ortogonal reduz a matriz $\mathbf{A}$ em uma matriz diagonal $\mathbf{D}_{iag}$ da seguinte maneira (Anton; Rorres, 2017):

$$\mathbf{P}_{\mathbf{quad}}^{\mathbf{T}}\mathbf{A}\mathbf{P}_{quad} = \mathbf{D}_{iag} = \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}, \tag{2.11}$$

em que $\mathbf{A} = \mathbf{P}_{quad}\mathbf{D}_{iag}\mathbf{P}_{quad}^{T}$. Assim, os n autovalores de $\mathbf{A}$ correspondem aos elementos diagonais da matriz $\mathbf{D}_{iag}$. Cada autovalor λ é associado a um autovetor ortonormal $\mathbf{v}$. Então, se $\mathbf{A}$ for ortogonalmente diagonalizada por

$$\mathbf{P}_{quad} = \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix}, \tag{2.12}$$

ela pode ser escrita como

$$\begin{aligned} \mathbf{A} &= \mathbf{P}_{quad}\mathbf{D}_{iag}\mathbf{P}_{quad}^{T} \qquad (2.13) \\ &= \begin{bmatrix} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_n \end{bmatrix} \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} \\ &= \begin{bmatrix} \lambda_1\mathbf{v}_1 & \lambda_2\mathbf{v}_2 & \cdots & \lambda_n\mathbf{v}_n \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_n^T \end{bmatrix} \\ &= \sum_{i=1}^{n} \lambda_i \mathbf{v}_i \mathbf{v}_i^T, \qquad (2.14) \end{aligned}$$

no qual a Equação 2.14 é denominada decomposição espectral de $\mathbf{A}$. Dessa forma, tanto a fatoração de Schur como a de Hessenberg são importantes em alguns algoritmos numéricos, pois além das matrizes resultantes estarem em um formato mais simplificado, a transformação não aumenta os erros de arredondamento mediante as operações com matrizes ortogonais.

2.3 Decomposição QR

Seja agora $\mathbf{A} \in \mathbb{R}^{m \times n}$, em que $m \geq n$, com vetores coluna linearmente independentes, então $\mathbf{A}$ pode ser fatorada como

$$\mathbf{A} = \mathbf{Q}_r \begin{pmatrix} \mathbf{R} \\ \mathbf{0} \end{pmatrix}, \tag{2.15}$$

sendo $\mathbf{Q}_r$ uma matriz $m \times m$ de vetores coluna que formam uma base ortonormal e $\mathbf{R}$ uma matriz $n \times n$ triangular superior invertível. O conjunto dos vetores coluna $\{\mathbf{q}_1, \mathbf{q}_2, \cdots, \mathbf{q}_n\}$ e $\{\mathbf{q}_{n+1}, \cdots, \mathbf{q}_m\}$ formam um base ortonormal do espaço coluna e linha de $\mathbf{A}$, respectivamente.

Em muitos algoritmos numéricos, a decomposição QR ocorre ao aplicar sucessivamente as reflexões de Householder sobre $\mathbf{A}$ com o intuito de anular os elementos abaixo da sua diagonal principal e, assim, obter $\mathbf{R}$. Concomitantemente a cada iteração i, a matriz ortogonal de Householder $\mathbf{H}_h$ é calculada utilizando a Equação 2.5 para formar a matriz $\mathbf{Q}_r$. Cada transformação aplicada anula os elementos de uma coluna da matriz $\mathbf{A}$ e então são necessárias $n-1$ iterações para obter $\mathbf{R}$. Dessa maneira, $\mathbf{Q}_r$ é dada por

$$\mathbf{Q}_r = \prod_{i=0}^{n-2} \mathbf{H}_{h_i}. \tag{2.16}$$

2.4 Autovalores e autovetores

O problema de encontrar os autovalores e autovetores surge em muitas situações práticas e devido a utilizações dos autovalores em aplicações de múltiplos domínios, eles ocupam um lugar importante na álgebra linear (Datta, 2009). A vibração dos sistemas mecânicos, por exemplo, é modelada por equações diferenciais, cujas soluções levam ao problema dos autovalores. Considerando novamente $\mathbf{A} \in \mathbb{R}^{m \times n}$, no qual $m = n$, então um vetor não-nulo $\mathbf{v} \in \mathbb{R}^n$ é denominado autovetor de $\mathbf{A}$, caso o produto $\mathbf{Av}$ seja um múltiplo escalar de $\mathbf{v}$, isto é,

$$\mathbf{Av} = \lambda \mathbf{v} \tag{2.17}$$

em que λ é um múltiplo escalar denominado autovalor de $\mathbf{A}$ e $\mathbf{v}$ um autovetor associado a λ. Considerando que $\mathbf{A}$ tenha posto n, os autovetores são a solução não nula do sistema homogêneo $(\mathbf{A} - \mathbf{I}\lambda)\mathbf{v}$, sendo $\mathbf{I}$ uma matriz identidade de ordem n. Os autovalores são as soluções da equação característica de $\mathbf{A}$, dada por $\det(\mathbf{A} - \lambda\mathbf{I}) = 0$.

Como os autovalores são os zeros da equação polinomial característica $\det(\mathbf{A}-\lambda\mathbf{I})$, poderia-se pensar que o cálculo do determinante forneceria a equação característica e, então, os autovalores seriam obtidos por um método de localização das raízes. Entretanto, esta não é uma abordagem prática, pois o cálculo explícito do determinante é numericamente instável (Trefethen; Bau, 1997). Além disso, caso os coeficientes da equação característica não sejam calculados com precisão, os autovalores não são obtidos corretamente. Dessa forma, utiliza-se frequentemente algoritmos iterativos que calculam os autovalores e autovetores. Nesta dissertação, o método de iteração QR com deslocamento simples μ será utilizado. Este procedimento reduz para a forma diagonal, uma determinada matriz simétrica e tridiagonal $\mathbf{T}$ dada por

$$\mathbf{T} = \begin{bmatrix} a_1 & b_1 & & \\ b_1 & \ddots & \ddots & \\ & \ddots & \ddots & b_{n-1} \\ & & b_{n-1} & a_n \end{bmatrix}. \tag{2.18}$$

Nessa transformação, os valores na diagonal principal da matriz reduzida convergem para os autovalores de $\mathbf{T}$ (Golub; Loan, 2013). O procedimento explícito demonstrado pelo Algoritmo 1 consiste em aplicar sucessivas transformações QR até que a convergência seja atingida, i.e., quando os elementos da diagonal principal forem suficientemente próximos aos autovalores de $\mathbf{T}$ e os elementos da diagonal inferior e superior, suficientemente pequenos em comparação a um valor de tolerância.

No Algoritmo 1, a cada iteração k, o valor de μ_k mais eficiente é denominado deslocamento de Wilkinson, que possibilita uma taxa de convergência cúbica, sendo calculado como

$$\mu_k = \text{autovalor de } \begin{bmatrix} a_{n-1} & b_{n-1} \\ b_{n-1} & a_n \end{bmatrix} \text{ mais próximo de } a_n \tag{2.19}$$

$$= a_n + d - sign(d)\sqrt{d^2 + b_{n-1}^2}, \tag{2.20}$$

em que $d = (a_{n-1} - a_n)/2$.

Algoritmo 1: Iteração QR (Golub; Loan, 2013, p.385)

Entrada: Matriz tridiagonal $\mathbf{T}$
Saída: Matriz diagonal com autovalores aproximados $\mathbf{T}$

início
  para $k = 1, 2, \cdots$ **faça**
    Calcule o deslocamento μ_k
    $[\mathbf{Q}_{r_k}, \mathbf{R}_k] = qr(\mathbf{T}_k - \mu_k \mathbf{I})$
    $\mathbf{T}_{k+1} = \mathbf{R}_k \mathbf{Q}_{r_k} + \mu_k \mathbf{I}$
  fim
fim

2.5 Decomposição em valores singulares

A decomposição em valores singulares é uma técnica que fatora uma matriz $\mathbf{A} \in \mathbb{R}^{m \times n}$, sendo $m \geq n$, em um produto de três matrizes

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T, \tag{2.21}$$

em que $\mathbf{U} \in \mathbb{R}^{m \times m}$ e $\mathbf{V} \in \mathbb{R}^{n \times n}$ são ortogonais e $\mathbf{\Sigma} \in \mathbb{R}^{m \times n}$ é diagonal, se $m = n$. Caso contrário, adiciona-se $m - n$ linhas de zeros em $\mathbf{\Sigma}$. Os valores $\sigma_1, \sigma_2, \cdots, \sigma_n$ contidos na diagonal principal de $\mathbf{\Sigma}$ são chamados valores singulares de $\mathbf{A}$. As colunas de $\mathbf{U}$ e $\mathbf{V}$ são vetores ortonormais que formam duas bases ortogonais diferentes e são denominados, respectivamente, vetores singulares à esquerda e à direita de $\mathbf{A}$.

O produto $\mathbf{A}^T\mathbf{A}$ resulta em uma matriz simétrica, no qual sua decomposição em autovalores é dada por

$$\mathbf{A}^T\mathbf{A} = \mathbf{V}\mathbf{D}_{iag}\mathbf{V}^T, \tag{2.22}$$

em que as colunas de $\mathbf{V} = [\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_n]$ são os autovetores unitários de $\mathbf{A}^T\mathbf{A}$ associados aos autovalores dispostos na diagonal principal da matiz $\mathbf{D}_{iag}$, tal que $\lambda_1, \lambda_2, \cdots, \lambda_n \geq 0$. Além disso, $\mathbf{A}^T\mathbf{A}$ tem o mesmo espaço nulo, linha, coluna e posto de $\mathbf{A}$ (Anton; Rorres, 2017). Em particular, se $\mathbf{A}$ possuir posto $k < n$, então o posto de $\mathbf{A}^T\mathbf{A}$ também será igual a k. Ordenando os autovalores de tal modo que $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_k > 0$, $\mathbf{D}_{iag}$ e $\mathbf{V}$ são determinadas como

$$\mathbf{D}_{iag} = diag(\lambda_1, \lambda_2, \cdots, \lambda_k, 0, \cdots 0) \quad \text{e} \quad \mathbf{V} = \left[\begin{array}{cccc|ccc} \mathbf{v}_1 & \mathbf{v}_2 & \cdots & \mathbf{v}_k & \mathbf{v}_{k+1} & \cdots & \mathbf{v}_n \end{array}\right],$$

em que o conjunto das colunas $\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\}$ formam uma base ortonormal para o espaço linha de $\mathbf{A}^T\mathbf{A}$ e, portanto, de $\mathbf{A}$. Devido a ortogonalidade de $\mathbf{V}$, a matriz $\mathbf{S} = \mathbf{A}\mathbf{V}$ possui k colunas ortogonais e linearmente independentes. Em consequência, $\mathbf{S}$ pode ser considerada

como uma base ortogonal do espaço coluna de $\mathbf{A}$. Ao normalizar os vetores em $\mathbf{S}$, obtém-se uma base ortonormal $\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k\}$ do espaço coluna de $\mathbf{A}$, cujos vetores unitários são dados por

$$\mathbf{u}_i = \frac{\mathbf{A}\mathbf{v}_i}{\|\mathbf{A}\mathbf{v}_i\|} = \frac{1}{\sqrt{\lambda_i}}\mathbf{A}\mathbf{v}_i \quad \text{para} \quad 1 \leq i \leq k, \tag{2.23}$$

em que

$$\sigma_1 = \sqrt{\lambda_1}, \quad \sigma_2 = \sqrt{\lambda_2}, \quad \cdots \quad , \quad \sigma_k = \sqrt{\lambda_k}. \tag{2.24}$$

Substituindo os valores singulares da Equação 2.24 na Equação 2.23, obtém-se

$$\mathbf{U}\mathbf{\Sigma} = \mathbf{A}\mathbf{V}, \tag{2.25}$$

sendo

$$\mathbf{\Sigma} = \begin{bmatrix} \sigma_1 & & & & \\ & \ddots & & & \\ & & \sigma_k & & \\ & & & 0 & \\ & & & & \ddots \\ & & & & & 0 \end{bmatrix}. \tag{2.26}$$

Desse modo, a SVD fornece bases ortogonais dos quatro subespaços fundamentais de uma matriz (Olver; Shakiban, 2018, p.114):

Conjunto	**Espaço Vetorial**	**Dimensão**
$\{\mathbf{u}_1, \mathbf{u}_2, \cdots, \mathbf{u}_k\}$	$col(\mathbf{A}) = imag(\mathbf{A})$	k
$\{\mathbf{u}_{k+1}, \cdots, \mathbf{u}_m\}$	$null(\mathbf{A}^T)$	$m-k$
$\{\mathbf{v}_1, \mathbf{v}_2, \cdots, \mathbf{v}_k\}$	$lin(\mathbf{A})$	k
$\{\mathbf{v}_{k+1}, \cdots, \mathbf{v}_n\}$	$null(\mathbf{A})$	$n-k$

Tabela 2.1 – Quatro espaços fundamentais de uma matriz

Devido a ortogonalidade de $\mathbf{V}$, a Equação 2.25 pode ser escrita como a Equação 2.21, que em forma detalhada é dada por:

$$\mathbf{A} = \left[\begin{array}{cccc|ccc} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k & \mathbf{u}_{k+1} & \cdots & \mathbf{u}_m \end{array}\right] \left[\begin{array}{cccc|c} \sigma_1 & 0 & \cdots & 0 & \\ 0 & \sigma_2 & \cdots & 0 & \\ \vdots & \vdots & \ddots & \vdots & 0_{k\times(n-k)} \\ 0 & 0 & \cdots & \sigma_k & \\ \hline \multicolumn{4}{c|}{0_{(m-k)\times k}} & 0_{(m-k)\times(n-k)} \end{array}\right] \left[\begin{array}{c} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_k^T \\ \hline \\ \mathbf{v}_{k+1}^T \\ \vdots \\ \mathbf{v}_n^T \end{array}\right]. \tag{2.27}$$

Eliminando as linhas e colunas nulas de $\mathbf{\Sigma}$, a SVD reduzida de $\mathbf{A}$ fica na forma

$$\mathbf{A} = \begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \cdots & \mathbf{u}_k \end{bmatrix} \begin{bmatrix} \sigma_1 & 0 & \cdots & 0 \\ 0 & \sigma_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \sigma_k \end{bmatrix} \begin{bmatrix} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \vdots \\ \mathbf{v}_k^T \end{bmatrix}, \tag{2.28}$$

que pode ser escrita como uma soma de k matrizes com posto 1. Como os valores singulares estão dispostos em ordem decrescente, a parte dominante torna-se visível e, por isso, a SVD é utilizada para redução de dados em muitas aplicações

$$\mathbf{A} = \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T. \tag{2.29}$$

Levando em conta agora que $\mathbf{A}$ tenha posto incompleto e contenha ruídos, ou seja, $\mathbf{A} = \mathbf{A}_0 + \mathbf{N}$, no qual a matriz de ruído $\mathbf{N}$ é pequena comparada com $\mathbf{A}$, então, o posto de $\mathbf{A}$ pode ser estimado por inspeção dos seus valores singulares. Ao considerar os mais significativos, os ruídos podem ser removidos e, assim, obtém-se uma boa aproximação de $\mathbf{A}$ (Eldén, 2007), cujo truncamento é dado por

$$\mathbf{A} = \sum_{i=1}^{n} \sigma_i \mathbf{u}_i \mathbf{v}_i^T \approx \sum_{i=1}^{k} \sigma_i \mathbf{u}_i \mathbf{v}_i^T. \tag{2.30}$$

2.6 Cálculo da SVD

Os valores singulares de $\mathbf{A} \in \mathbb{R}^{m\times n}$, para $m \geq n$, podem ser facilmente obtidos via decomposição espectral de $\mathbf{A}^T\mathbf{A}$ e $\mathbf{A}\mathbf{A}^T$ tal como (Golub; Loan, 2013, 8.6)

$$\mathbf{V}^T(\mathbf{A}^T\mathbf{A})\mathbf{V} = diag(\sigma_1^2, \cdots, \sigma_n^2) \quad \text{e} \quad \mathbf{U}^T(\mathbf{A}\mathbf{A}^T)\mathbf{U} = diag(\sigma_1^2, \cdots, \sigma_n^2, \underbrace{0, \cdots, 0}_{m-n}). \tag{2.31}$$

Considerando a Equação 2.25, um dos procedimentos para calcular a decomposição em valores singulares da matriz $\mathbf{A}$ de forma serial consiste em:

1. Formar a matriz simétrica $\mathbf{C}_{sim} = \mathbf{A}^T\mathbf{A}$
2. Calcular os valores singulares mediante a decomposição espectral $\mathbf{V}_1^T\mathbf{C}_{sim}\mathbf{V}_1 = diag(\sigma_i^2)$
3. Aplicar a fatoração QR com pivotamento das colunas em $\mathbf{AV}_1$ para obter $\mathbf{U}^T(\mathbf{AV}_1)\mathbf{\Pi} = \mathbf{R}$

Uma vez que as colunas de $\mathbf{R}$ são ortogonais, o produto $\mathbf{U}^T\mathbf{A}(\mathbf{V}_1\mathbf{\Pi})$ é diagonal e as matrizes $\mathbf{\Sigma} = \mathbf{R}$ e $\mathbf{V} = \mathbf{V}_1\mathbf{\Pi}$ completam a SVD. Entretanto, o cálculo explícito de $\mathbf{A}^T\mathbf{A}$ pode levar à perda de informações (Golub; Loan, 2013, 5.3.2), sendo recomendado a utilização de outras técnicas para obter a SVD de $\mathbf{A}$.

2.6.1 SVD via Iteração QR

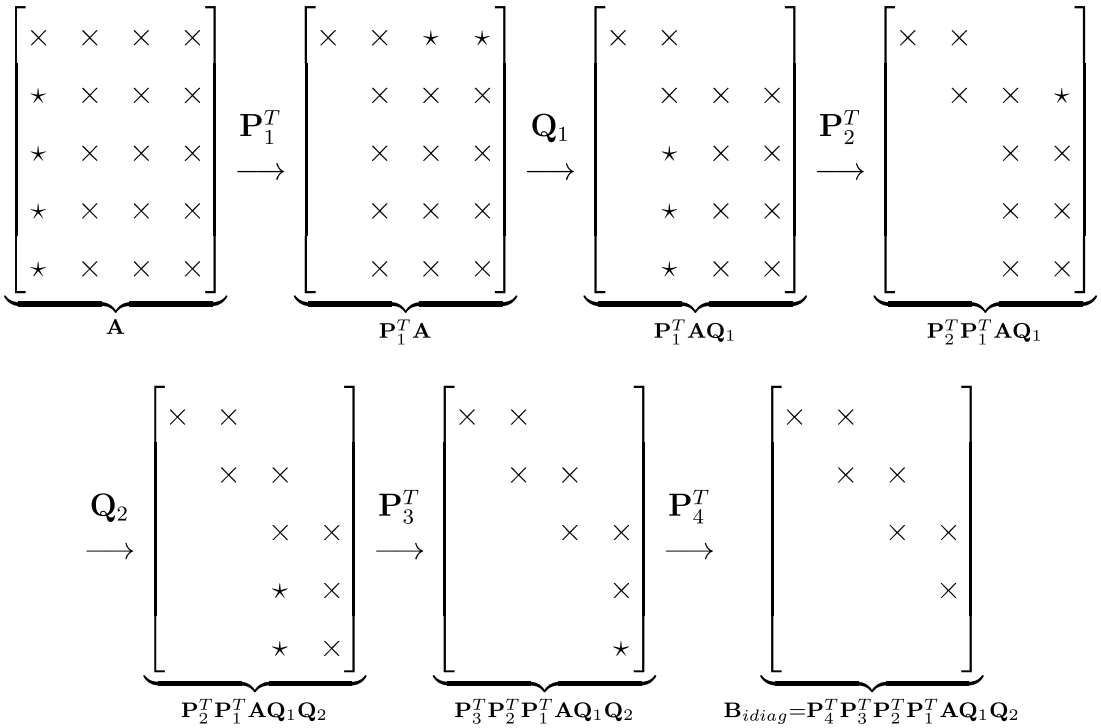
O método proposto por Golub; Kahan,1965 realiza decomposição em valores singulares da matriz de entrada $\mathbf{A} \in \mathbb{R}^{m\times n}$ $(m \geq n)$ em duas etapas.

Etapa 1 - Redução para a forma bidiagonal: a matriz $\mathbf{A}$ é reduzida em uma matriz bidiagonal superior e similar $\mathbf{B}_{idiag}$, mediante sucessivas transformações de Householder sobre suas linhas e colunas. Nesse processo, duas bases ortogonais diferentes $\mathbf{P} \in \mathbb{R}^{m\times m}$ e $\mathbf{Q} \in \mathbb{R}^{n\times n}$ são geradas tal que

$$\mathbf{P}^T\mathbf{AQ} = \begin{pmatrix} \mathbf{B}_{idiag} \\ \mathbf{0} \end{pmatrix}. \tag{2.32}$$

Neste procedimento, a quantidade de transformações aplicadas em cada lado de $\mathbf{A}$ é finita e depende do número de colunas e linhas que ela possui. Serão aplicadas $n-2$ reflexões $\mathbf{Q}$ à direita de $\mathbf{A}$. Na hipótese de m ser maior que n, o número de transformações $\mathbf{P}^T$ à esquerda de $\mathbf{A}$ será igual a n e no caso em que $m = n$, $\mathbf{A}$ será pré-multiplicada por $\mathbf{P}^T$ $(n-1)$ vezes (Golub; Loan, 2013, 5.4.8). Considerando, por exemplo, $m = 5$ e $n = 4$, o primeiro passo consiste em pré-multiplicar $\mathbf{A}$ por $\mathbf{P}_1^T$ com o intuito de anular os elementos da primeira coluna de $\mathbf{A}$, exceto o primeiro. Em seguida, multiplica-se este resultado à direita por $\mathbf{Q}_1$ a fim de anular os elementos da primeira linha de $\mathbf{A}$, representados por $\star$, que estão fora da diagonal principal e superior. O processo é repetido até obter a matriz na forma bidiagonal. As

transformações em cada lado são aplicadas sucessivamente sobre blocos da matriz resultante que possui tamanho específico a cada iteração como



Os valores singulares de $\mathbf{B}_{idiag}$ são iguais aos de $\mathbf{A}$. Considerando que a SVD de $\mathbf{B}_{idiag}$ seja dada por

$$\mathbf{B}_{idiag} = \mathbf{G}_l \mathbf{\Sigma} \mathbf{H}_r^T, \tag{2.33}$$

então

$$\mathbf{A} = \underbrace{\mathbf{P}\mathbf{G}_l}_{\mathbf{U}} \mathbf{\Sigma} \underbrace{\mathbf{H}_r^T\mathbf{Q}^T}_{\mathbf{V}^T}. \tag{2.34}$$

O Algoritmo 2 descreve o procedimento sequencial de aplicação das reflexões de Householder $\mathbf{P}$ e $\mathbf{Q}$ sobre a matriz de entrada $\mathbf{A}$. Em cada iteração i, as matrizes de reflexão $\mathbf{P}$ e $\mathbf{Q}$ são calculadas (linhas $7-9$ e $15-17$, respectivamente) conforme a Equação 2.5, ou seja, com base nos vetores de Householder $\mathbf{u}$ e nos valores $\eta = \frac{\mathbf{u}^T\mathbf{u}}{2}$ obtidos nas linhas 6 e 14. A função que calcula $\mathbf{u}$ e η é descrita pelo Algoritmo 3. O seu parâmetro de entrada é o vetor linha ou coluna $\mathbf{v}$ de $\mathbf{B}_{idiag}$ (linhas 5 e 13, do Algoritmo 2), que possui tamanho específico a cada iteração. Além disso, na linha 6 do Algoritmo 3 é descrito o cálculo de $\mathbf{u} = \mathbf{v} + \|\mathbf{v}\|\hat{\mathbf{i}}$, conforme definido em 2.1.1.

Algoritmo 2: Bidiagonalização (Golub; Loan, 2013, p.284)

Entrada: $\mathbf{A}_{m\times n}$
Saída: $\mathbf{P}_{m\times n}, \mathbf{B}_{idiag_{n\times n}}, \mathbf{Q}_{n\times n}$

início
  $\mathbf{B}_{idiag} = \mathbf{A}$, $[m, n] = size(\mathbf{A})$;
  $\mathbf{V}_{acc} = eye(n)$,$\mathbf{U}_{acc} = eye(m)$;
  para $i = 1, 2, \cdots, n$ **faça**
    $\mathbf{v} = \mathbf{B}_{idiag}(i : m, i)$
    calcule: $[\mathbf{u}, \eta] = house(\mathbf{v})$
    $\mathbf{P}_{temp} = eye(n - i + 1) - \frac{1}{\eta} * \mathbf{u} * \mathbf{u}^T$
    $\mathbf{P} = eye(m)$
    $\mathbf{P}(i : m, i : m) = \mathbf{P}_{temp}$
    $\mathbf{U}_{acc} = \mathbf{U}_{acc}\mathbf{P}$
    $\mathbf{B}_{idiag} = \mathbf{P}\mathbf{B}_{idiag}$
    se $n - i > 0$ **então**
      $\mathbf{v} = \mathbf{B}_{idiag}(i, i + 1 : n)$
      calcule: $[\mathbf{u}, \eta] = house(\mathbf{v}^T)$
      $\mathbf{Q}_{temp} = eye(n - i) - \frac{1}{\eta} * \mathbf{u} * \mathbf{u}^T$
      $\mathbf{Q} = eye(n)$
      $\mathbf{Q}(i + 1 : n, i + 1 : n) = \mathbf{Q}_{temp}$
      $\mathbf{V}_{acc} = \mathbf{V}_{acc}\mathbf{Q}$
      $\mathbf{B}_{idiag} = \mathbf{B}_{idiag}\mathbf{Q}$
    fim
  fim
  $\mathbf{B}_{idiag} = \mathbf{B}_{idiag}(1 : n, :)$
  $\mathbf{Q} = \mathbf{V}_{acc}$
  $\mathbf{P} = \mathbf{U}_{acc}(:, 1 : n)$
fim

Algoritmo 3: Cálculo de $\mathbf{u}$ e η (Demmel, 1997, 3.4.1)

Entrada: $\mathbf{v}$
Saída: $\mathbf{u}, \eta$

início
  $\eta = 0$;
  $\mathbf{u} = \mathbf{v}$;
  $s = \sqrt{\mathbf{u}^T\mathbf{u}}$;
  $s = s * sign(u[0])$;
  $u[0] = u[0] + s$;
  se $s \neq 0$ **então**
    $\eta = \frac{\mathbf{u}^T\mathbf{u}}{2}$
  fim
fim

Etapa 2 - Decomposição em valores singulares da matriz bidiagonal: esta etapa consiste em diagonalizar a matriz reduzida $\mathbf{B}_{idiag}$ iterativamente como

$$\mathbf{B}_{idiag}[0] \rightarrow \mathbf{B}_{idiag}[1] \rightarrow \cdots \rightarrow \mathbf{\Sigma}, \tag{2.35}$$

em que para cada iteração k, $\mathbf{B}_{idiag}[k]$ é multiplicada em ambos os lados pelas matrizes ortogonais de Givens $\mathbf{G}_l[k]$ e $\mathbf{H}_r[k]$, como

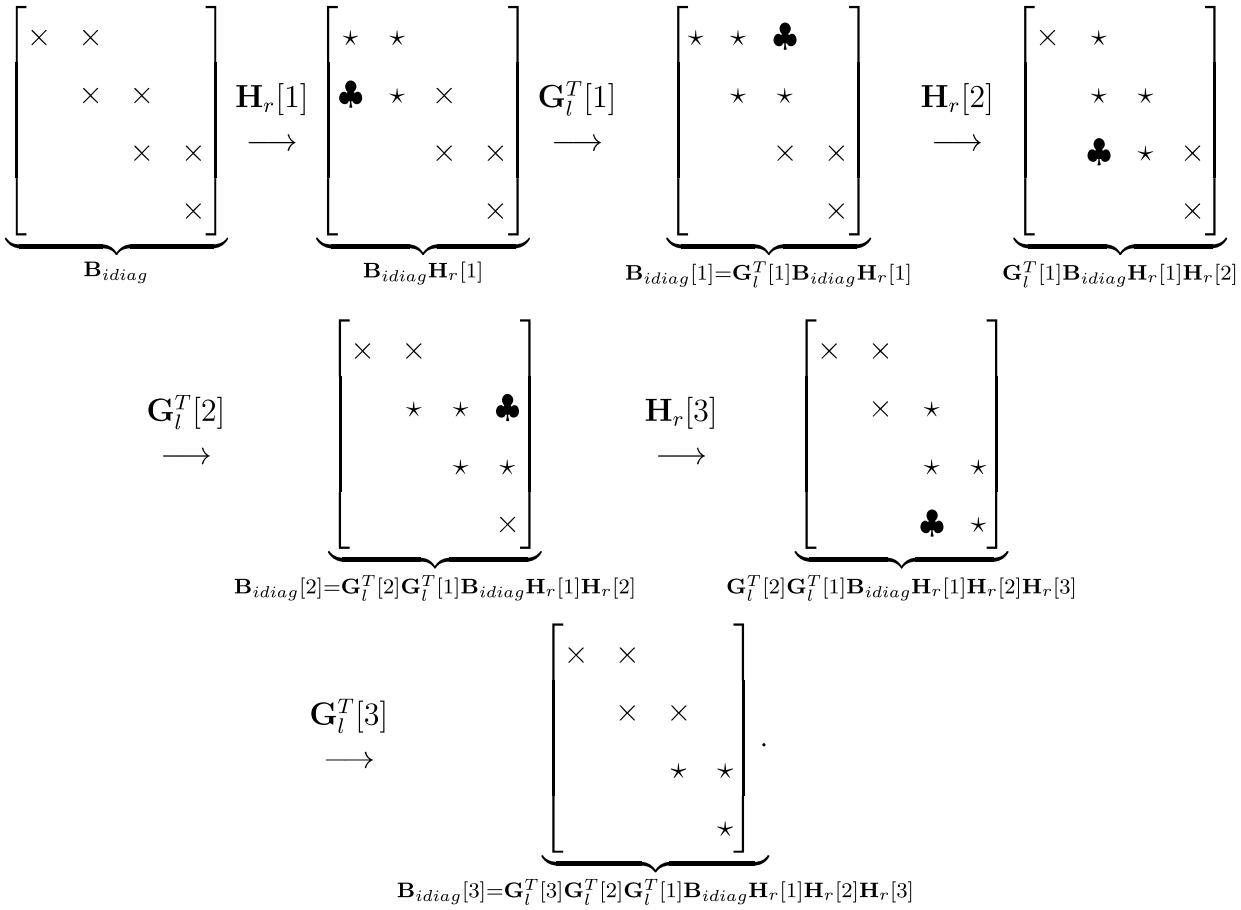
$$\mathbf{B}_{idiag}[k+1] = \mathbf{G}_l^T[k]\mathbf{B}_{idiag}[k]\mathbf{H}_r[k], \tag{2.36}$$

sendo

$$\mathbf{H}_r[k](i,\theta) = \begin{bmatrix} 1 & & & & & & & \\ & \ddots & & & & & & \\ & & 1 & & & & & \\ & & & \cos\theta & -\sin\theta & & & \\ & & & \sin\theta & \cos\theta & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{bmatrix} \begin{matrix} \\ \\ \\ i-1 \\ i \\ \\ \\ \\ \end{matrix} \tag{2.37}$$

$$\begin{matrix} & & & i-1 & i & & & \end{matrix}$$

e $\mathbf{G}_l[k]$ é definida de modo análogo a $\mathbf{H}_r[k]$, exceto pelo ângulo ϕ ao invés de θ. $\mathbf{H}_r[k]$ é uma matriz identidade, exceto por quatro valores descritos em função do ângulo θ e posicionados em relação a um certo valor i. Na Equação 2.37, os valores i e $i-1$ descritos abaixo de $\mathbf{H}_r[k]$ representam as colunas nas quais são aplicados esses valores específicos, enquanto que os descritos à direita de $\mathbf{H}_r[k]$ identificam as linhas. Devido ao formato dessas matrizes, a transformação aplicada em cada lado altera os valores de duas colunas (pós-multiplicação por $\mathbf{H}_r[k]$) ou linhas (pré-multiplicação por $\mathbf{G}_l[k]$) de $\mathbf{B}_{idiag}[k]$. Assim, os parâmetros definidos em cada matriz de rotação são escolhidos de tal forma que uma sequência de transformações seja capaz de introduzir e anular novos valores ♣ adjacentes às diagonais. Para diagonalizar um intervalo da matriz $\mathbf{B}_{diag}[k]$, com n elementos na diagonal principal, é preciso então $k = n-1$ iterações sequenciais definidas como



A cada diagonalização, os valores da superdiagonal de $\mathbf{B}_{idiag}$ tendem a diminuir. Essa operação equivale em aplicar, de forma implícita, uma variação da transformação QR explícita sobre a matriz tridiagonal $\mathbf{B}_{idiag}^T\mathbf{B}_{idiag}$, com deslocamento de Wilkinson descrita pelo Algoritmo 1 (Golub; Kahan, 1965). Em outras palavras, o produto $\mathbf{B}_{idiag}^T\mathbf{B}_{idiag}$ nunca é calculado. Em geral, após alguns ciclos, os elementos da diagonal superior de $\mathbf{B}_{idiag}$ tornam-se insignificantes, i.e, menores ou iguais a um valor de tolerância prescrito δ e, então, os elementos da diagonal principal são considerados os valores singulares de $\mathbf{B}_{idiag}$. Considerando que a matriz bidiagonal seja dada por

$$\mathbf{B}_{idiag} = \begin{bmatrix} q_0 & e_1 & & & \\ & q_1 & e_2 & & \\ & & \ddots & \ddots & \\ & & & \ddots & e_{n-1} \\ & & & & q_{n-1} \end{bmatrix}, \tag{2.38}$$

o algoritmo percorre os elementos diagonais de $\mathbf{B}_{idiag}$ com início na superdiagonal e_n de posição $\nu = n - 1$ até encontrar um valor menor ou igual a δ. Nesse ponto, o intervalo de diagonalização é definido e as transformações QR são aplicadas em $\mathbf{B}_{idiag}$ implicitamente. Após

nova análise dos elementos diagonais, se $|e_{n-1}| \leq \delta$, a convergência é atingida e $|q_{n-1}|$ é aceito como valor singular. Caso contrário, as transformações QR são aplicadas novamente dentro do novo intervalo de diagonalização e o ciclo se repete até convergir. Quando isso ocorre, a ordem de $\mathbf{B}_{idiag}$ é reduzida em uma unidade e a próxima análise começa na diagonal superior de posição $n-2$. A função que avalia os valores diagonais é apresentada no método de referência adotado neste trabalho (Golub; Kahan, 1965, p.143). Press *et al.*,2007b disponibiliza também um código escrito em C que realiza o cálculo da SVD e se baseia também nesta referência. O diagrama que realiza esta avaliação é representado na Figura 2.3.

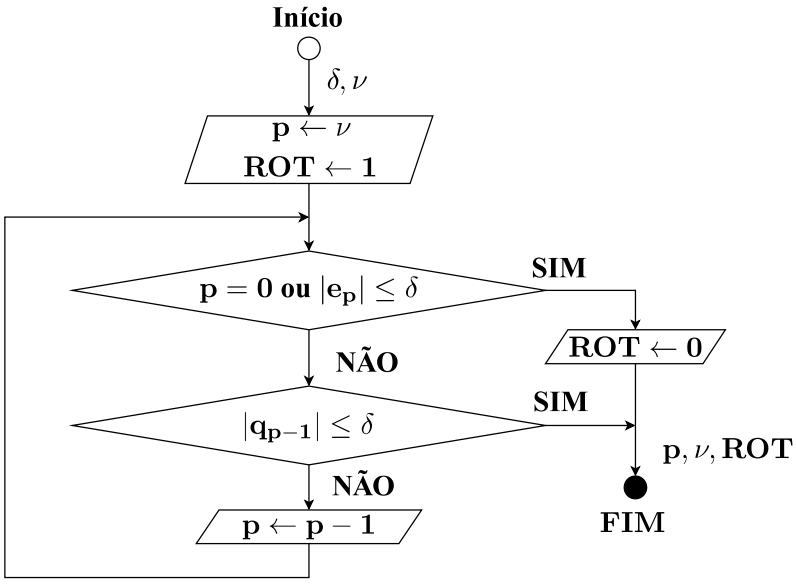


Figura 2.3 – Teste para divisão da Matriz

Durante análise dos valores de $\mathbf{B}_{idiag}$, caso $|e_p| \leq \delta$ para $p \neq \nu$, a matriz $\mathbf{B}_{idiag}$ é divida em duas e os valores singulares de cada bloco são calculados de forma independente. Além disso, caso $|q_p| \leq \delta$, então uma transformação de Givens adicional deve ser aplicada à esquerda ($ROT = 1$) com o intuito de zerar e_{k+1}. Essa função retorna os parâmetros p, ν e ROT, sendo o intervalo de diagonalização em cada análise dado por $\nu - p$. O método completo para diagonalizar a matriz $\mathbf{B}_{idiag}$ é representado na Figura 2.4, sendo descrito por quatro funções básicas:

1. **Teste para Divisão:** conforme descrita anteriormente, esta função consiste em avaliar os elementos diagonais e retornar o intervalo de diagonalização e a variável de rotação adicional ROT;

2. **Cancelamento:** esta função aplica, à esquerda, a transformação de Givens que anula o valor de uma determinada superdiagonal;

3. **Convergência:** após a convergência ser atingida, essa função impõe que o valor singular seja positivo. Caso não seja, essa variável e sua coluna respectiva em $\mathbf{V}$, são multiplicadas por -1;

4. **Transformação QR:** definido o intervalo de diagonalização, a multiplicação em ambos os lados diagonaliza iterativamente $\mathbf{B}_{idiag}$.

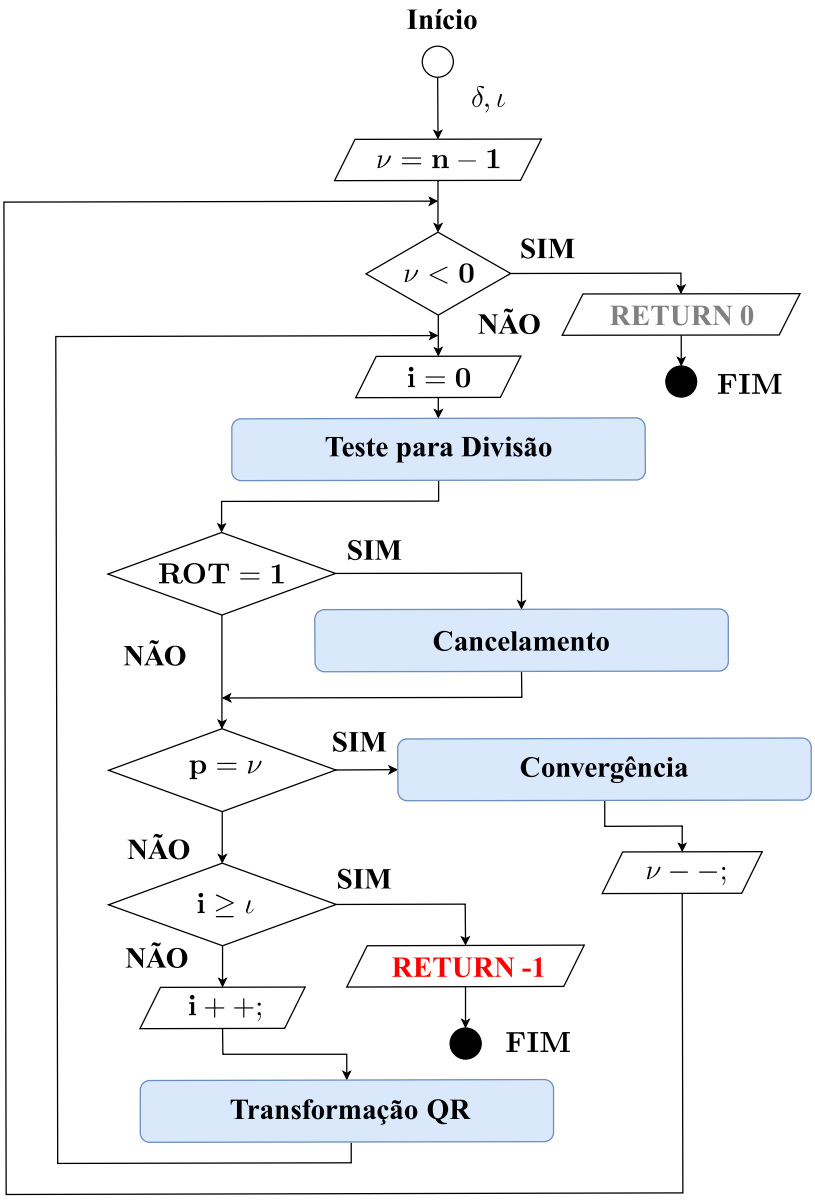


Figura 2.4 – Diagonalização iterativa da matriz bidiagonal

Na hipótese da convergência não ser atingida em ι iterações sobre cada coluna de $\mathbf{B}_{idiag}$, então o método retorna -1 e a SVD não é calculada. Caso contrário, o valor de retorno é nulo e a SVD é calculada. Para obter finalmente a forma padrão da decomposição, o último passo é ordenar os valores singulares e os respectivos autovetores em uma função específica.

2.6.2 SVD via Hestenes-Jacobi

Diferentemente do método anterior que calcula a SVD de $\mathbf{A} \in \mathbb{R}^{m \times n}$ $(m \geq n)$, o método de Hestenes-Jacobi necessita que as colunas da matriz de entrada sejam maiores ou iguais as linhas e, assim, este procedimento é realizado sobre $\mathbf{A}^T \in \mathbb{R}^{n \times m}$ $(m \geq n)$. As matrizes ortonormais de Givens pós-multiplicam, de forma iterativa, a matriz de entrada $\mathbf{A}^T$ até obter uma base ortogonal $\mathbf{O}$, o que equivale em fazer com que todas as suas colunas sejam perpendiculares entre si. Seja a Equação 2.23 reescrita como

$$\mathbf{O} = \mathbf{A}^T V. \tag{2.39}$$

Conforme definido anteriormente, $\mathbf{O}$ é uma base ortogonal obtida do produto $\mathbf{AV}$, no qual a matriz $\mathbf{V}$ é ortonormal e, portanto, possível de ser desmembrada em uma série de transformações ortogonais relativas a cada iteração como

$$\mathbf{V} = \mathbf{J}_1 \mathbf{J}_2 \mathbf{J}_3 \cdots \mathbf{J}_{\aleph}, \tag{2.40}$$

em que $\aleph$ é o número total das rotações de Givens aplicadas. Essas transformações são definidas por

$$\mathbf{J}(p, q, \theta) = \begin{bmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & c & -s & & & p \\ & & s & c & & & q \\ & & & & \ddots & & \\ & & & & & 1 & \\ & & p & q & & & \end{bmatrix} \tag{2.41}$$

e são também denominadas rotações de Jacobi. Vale ressaltar que a Equação 2.41 é idêntica a Equação 2.8. Dessa forma, escolhendo corretamente os parâmetros de rotação em cada transformação, duas colunas p e q de $\mathbf{A}^T$ podem ser ortogonalizadas através da multiplicação à direita pela matriz $\mathbf{J}$, que devido a sua forma, modifica somente os valores dessas colunas. Assim, representando somente os vetores-coluna $\mathbf{a}_p^{T'}$ e $\mathbf{a}_q^{T'}$ que serão modificados, a operação é descrita como

$$\begin{bmatrix} \mathbf{a}_p^{T'} & \mathbf{a}_q^{T'} \end{bmatrix} = \begin{bmatrix} \mathbf{a}_p^{T} & \mathbf{a}_q^{T} \end{bmatrix} \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \text{ em que } \quad \langle \overrightarrow{\mathbf{a}}_p^{T'}, \overrightarrow{\mathbf{a}}_q^{T'} \rangle = \cos\theta. \tag{2.42}$$

A primeira igualdade da Equação 2.42 é calculada por

$$\mathbf{a}_p^{T'} = c * \mathbf{a}_p^{T'} + s * \mathbf{a}_q^{T'} \quad \text{e} \quad \mathbf{a}_q^{T'} = c * \mathbf{a}_q^{T'} - s * \mathbf{A}_p^{T'}. \tag{2.43}$$

Quando as colunas $\mathbf{a}_p^{T'}$ e $\mathbf{a}_q^{T'}$ são perpendiculares entre si, o produto escalar entre esses vetores é nulo. Assim, é possível escrever a segunda igualdade da Equação 2.42 como

$$\begin{aligned}
0 =& \sum_{i=0}^{n} (c\mathbf{a}_p^T[i] + s\mathbf{a}_q^T[i])(c\mathbf{a}_q^T[i] - s\mathbf{a}_p^T[i]) \\
=& \sum_{i=0}^{n} \Big(c^2\mathbf{a}_p^T[i]\mathbf{a}_q^T[i] - cs\big(\mathbf{a}_p^T[i]\big)^2 + cs\big(\mathbf{a}_q^T[i]\big)^2 - s^2\mathbf{a}_p^T[i]\mathbf{a}_q^T[i] \Big) \\
=& \sum_{i=0}^{n} \Big((c^2 - s^2)\mathbf{a}_p^T[i]\mathbf{a}_q^T[i] + cs\Big[\big(\mathbf{a}_q^T[i]\big)^2 - \big(\mathbf{a}_p^T[i]\big)^2\Big] \Big) \\
=& (c^2 - s^2)\underbrace{\mathbf{a}_p\mathbf{a}_q^T}_{cov} + cs\big(\underbrace{\|\mathbf{a}_q^T\|^2}_{n_q} - \underbrace{\|\mathbf{A}_p^T\|^2}_{n_p}\big),
\end{aligned} \tag{2.44}$$

então

$$0 = cov(c^2 - s^2) + (n_q - n_p)cs. \tag{2.45}$$

Se $cov = 0$, então $c = 1$ e $s = 0$. Caso contrário, definem-se

$$\tau = \frac{n_q - n_p}{2cov} \quad \text{e} \quad t = s/c, \tag{2.46}$$

em que $t = \tan(\theta)$ é solução de

$$t^2 + 2\tau t - 1 = 0. \tag{2.47}$$

Com o intuito de maximizar o valor de c, escolhe-se a menor raiz

$$t = \frac{sign(\tau)}{|\tau| + \sqrt{1 + \tau^2}}, \tag{2.48}$$

cujos parâmetros de rotação são definidos como

$$c = \frac{1}{\sqrt{1 + \tau^2}} \quad \text{e} \quad s = ct. \tag{2.49}$$

A matriz **O** é uma base ortogonal caso seus vetores coluna sejam ortogonais entre si. Deste modo, o procedimento de varredura busca ortogonalizar todas as ρ combinações possíveis de duas colunas diferentes da matriz **O**, representadas por

$$\rho = C_2^n = \frac{n!}{2!(n-2)!} = \frac{n(n-1)}{2} = \alpha\beta, \tag{2.50}$$

em que $\alpha = m/2$ e $\beta = m - 1$. Após repetir o processo de varredura algumas vezes, as colunas de $\mathbf{O}$ serão praticamente ortogonais entre si. A quantidade v de varreduras necessárias depende do valor de tolerância definido para considerar a ortogonalidade entre os vetores-coluna de $\mathbf{O}$. Quando todas as colunas de $\mathbf{O}$ forem ortogonais entre si (menor que o tolerância estipulado), a convergência é atingida e o número total de rotações $\aleph$ aplicadas será igual a

$$\aleph = v\rho = v\alpha\beta. \tag{2.51}$$

Dessa forma, as matrizes $\mathbf{U}$ e $\mathbf{\Sigma}$ podem ser obtidas como

$$\mathbf{\Sigma} = \sqrt{\mathbf{O}^T\mathbf{O}} \quad \text{e} \quad \mathbf{U} = \mathbf{O}/\mathbf{\Sigma}. \tag{2.52}$$

O paralelismo deste algoritmo é implementado quando se realiza uma varredura. O procedimento pode ser parcialmente paralelizado caso execute α combinações em paralelo, uma vez que as m colunas envolvidas nesse conjunto são independentes. Como exemplo das ρ combinações de uma varredura, caso m seja igual a 8, é possível implementar $\alpha = 4$ execuções paralelas entre as colunas de $\mathbf{O}$, em um total de $\beta = 7$ vezes, conforme apresentado na Figura 2.5. Em cada iteração β da varredura, as colunas da matriz são reposicionadas de tal forma que se obtenha todas as combinações entre elas.

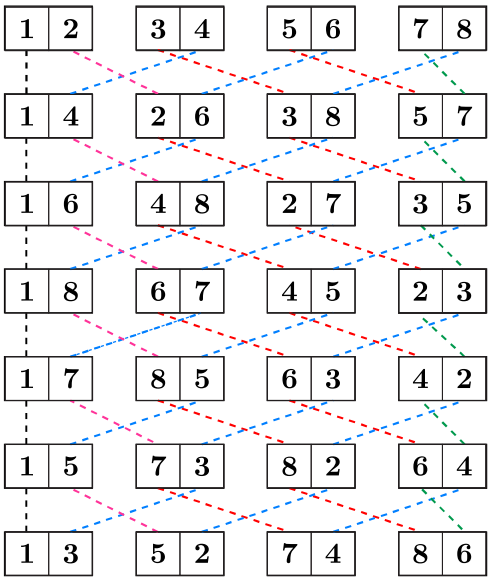


Figura 2.5 – Diagonalização iterativa da matriz bidiagonal para $m = 8$
Fonte: Adaptado de Brent; Luk,1985.

Para coordenar a escolha das colunas de $\mathbf{O}$, os valores p e q são armazenados em um vetor $\mathbf{W}$, o qual é atualizado a cada iteração. Dessa forma, todas as normas n_p e n_q e a

covariância entre as possíveis combinações de colunas serão calculadas. O Algoritmo 4 apresenta como é realizado o cálculo da SVD pelo método Hestenes-Jacobi, sem a classificação em ordem decrescente dos valores singulares e dos respectivos autovetores. O valor inicial da variável $erro$, na linha 5 do algoritmo, é maior que o valor de tolerância para ser possível executar as linhas $6-19$. Uma vez que a execução do algoritmo entre nesse laço de repetição, a variável $erro$ é atribuída por um valor menor que a tolerância na linha 7. A execução da varredura (linhas $8-17$) atualiza a variável $erro$ na linha 12 pelo máximo valor entre $erro$ e $|cov|$. O valor final da variável $erro$ é idealmente igual a 0, o que representa a ortogonalidade perfeita entre todas as colunas dessa matriz. Em situações práticas, no entanto, quando essa variável for menor que um valor de tolerância pré-estabelecido, considera-se que $\mathbf{O}$ é ortogonal e a SVD de $\mathbf{A}$ pode ser obtida.

Algoritmo 4: Hestenes Jacobi (Wang *et al.*, 2020)

Entrada: $\mathbf{A}_{m\times n}, tol$
Saída: $\mathbf{U}_{m\times n}, \mathbf{\Sigma}_{n\times n}, \mathbf{V}_{n\times n}$

1 **início**
2 $\quad \mathbf{W} = \{0, 1, \cdots, n-1\}$;
3 $\quad \mathbf{O} = \mathbf{A}$;
4 $\quad v = 0$;
5 $\quad erro = 2 * tol$;
6 $\quad$ **repita**
7 $\quad\quad erro = tol/2$;
8 $\quad\quad$ **para** $\beta = 1, 2, \cdots, m-1$ **faça**
9 $\quad\quad\quad$ **para** $\alpha = 0, 1, \cdots, m/2-1$ **faça**
10 $\quad\quad\quad\quad p = \mathbf{W}[2\alpha], q = \mathbf{W}[2\alpha+1]$
11 $\quad\quad\quad\quad$ **calcule:** n_p, n_q, cov
12 $\quad\quad\quad\quad erro = \max(erro, |cov|)$;
13 $\quad\quad\quad\quad$ **calcule:** c, s
14 $\quad\quad\quad\quad$ **atualize:** $\mathbf{O}_p, \mathbf{O}_q, \mathbf{V}_p, \mathbf{V}_q$
15 $\quad\quad\quad$ **fim**
16 $\quad\quad\quad$ **atualize:** $\mathbf{W}$
17 $\quad\quad$ **fim**
18 $\quad\quad v \leftarrow v + 1$;
19 $\quad$ **até** $erro > tol$;
20 $\quad$ **para** $i = 1, 2, \cdots, n$ **faça**
21 $\quad\quad \mathbf{\Sigma}[i] = \sqrt{\mathbf{O}[:,i]^T \mathbf{O}[:,i]}$;
22 $\quad\quad \mathbf{U}[:,i] = \frac{\mathbf{O}[:,i]}{\mathbf{\Sigma}[i]}$;
23 $\quad$ **fim**
24 **fim**

3 IDENTIFICAÇÃO, CONTROLE E DETECÇÃO DE DANOS

Este capítulo descreve a modelagem de uma estrutura no espaço de estados em tempo contínuo e a abordagem para discretizar esse modelo de representação. Encontram-se na literatura diversos trabalhos sobre identificação de sistemas, que realizam a estimativa dos parâmetros importantes, a partir dos dados de medição. Entre os métodos de identificação que utilizam como ferramenta a SVD e um algoritmo de realização mínima, Juang; Pappa,1985 proporam o método ERA (*Eigensystem Realization Algorithm*), o qual é descrito na sequência deste capítulo. Em seguida, é apresentado o projeto de um controlador discreto no espaço de estados por meio da alocação dos polos em malha fechada visando atenuar a vibração estrutural. Esta implementação é condicionada a medição de todas as variáveis e assim foi utilizado um observador para estimar todos os possíveis estados do sistema. Por último, é descrito nesse capítulo a métrica de subespaços com a qual é possível analisar a severidade de danos que a estrutura possa conter por meio da comparação de um modelo identificado periodicamente com outro modelo de referência considerado como saudável.

3.1 Representação de uma estrutura vertical flexível

A dinâmica de uma estrutura flexível, com n_d graus de liberdade e que possui s entradas e r saídas, é geralmente descrita em coordenadas nodais pelo par de equações diferenciais

$$\begin{aligned} \mathbf{M}\ddot{\mathbf{p}}(t) + \mathbf{E}\dot{\mathbf{p}}(t) + \mathbf{K}\mathbf{p}(t) &= \mathbf{B}_w\mathbf{w}(t) + \mathbf{B}_u\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}_p\mathbf{p}(t) + \mathbf{C}_v\dot{\mathbf{p}}(t) + \mathbf{C}_w\mathbf{w}(t) + \mathbf{C}_u\mathbf{u}(t), \end{aligned} \tag{3.1}$$

$\mathbf{p}(t) \in \mathbb{R}^{n_d \times 1}$ é o vetor de deslocamento. As entradas $\mathbf{w}(t) \in \mathbb{R}^{s_w \times 1}$ e $\mathbf{u}(t) \in \mathbb{R}^{s_u \times 1}$ são as forças de perturbação e controle agindo na estrutura, respectivamente. A influência dessas entradas na dinâmica do sistema são modeladas pelas matrizes $\mathbf{B_w} \in \mathbb{R}^{n_d \times s_w}$ e $\mathbf{B_u} \in \mathbb{R}^{n_d \times s_u}$, em que $s = s_w + s_u$. O vetor de saídas do sistema é representado por $\mathbf{y}(t) \in \mathbb{R}^{r \times 1}$, modelado pelas matrizes de saída $\mathbf{C}_p \in \mathbb{R}^{r \times n_d}$, $\mathbf{C}_v \in \mathbb{R}^{r \times n_d}$, $\mathbf{C}_w \in \mathbb{R}^{r \times s_w}$ e $\mathbf{C}_u \in \mathbb{R}^{r \times s_u}$. Por fim, $\mathbf{M}$, $\mathbf{E}$ e $\mathbf{K}$ são, respectivamente, as matrizes de massa, amortecimento e rigidez. Dessa forma, a Equação 3.1 representa um sistema acoplado de equações, pois os valores das matrizes $\mathbf{E}$ e $\mathbf{K}$ possuem parâmetros que influenciam concomitantemente ao menos duas variáveis independentes (graus

de liberdade). A equação homogênea da primeira igualdade da Equação 3.1 é definida por:

$$\mathbf{M}\ddot{\mathbf{p}}(t) + \mathbf{E}\dot{\mathbf{p}}(t) + \mathbf{K}\mathbf{p}(t) = 0. \tag{3.2}$$

Substituindo $\mathbf{p}(t) = \boldsymbol{\phi} e^{\lambda t}$ como solução da Equação 3.2, obtém-se o problema de auto valor quadrático:

$$(\lambda^2\mathbf{M} + \lambda\mathbf{E} + \mathbf{K})\boldsymbol{\phi} = 0. \tag{3.3}$$

Em aplicações práticas de sistemas mecânicos, as matrizes $\mathbf{M}$, $\mathbf{E}$ e $\mathbf{K}$ são consideradas simétricas (Adhikari, 2000). Além disso, $\mathbf{M}$ é uma matriz diagonal definida positiva e, portanto, não-singular. A matriz $\mathbf{K}$ é semi-definida positiva e a matriz $\mathbf{E}$ é considerada como sendo combinação linear das matrizes de massa e rigidez, em que $\mathbf{E} = \alpha\mathbf{M} + \beta\mathbf{K}$ para $\alpha, \beta \geq 0$ (Genari *et al.*, 2017b).

Para o projeto de controle da estrutura é conveniente representar a Equação 3.1 na forma de espaço de estados. Dessa maneira, define-se o vetor de estados $\mathbf{x}(t)$ como:

$$\mathbf{x}(t) = \begin{bmatrix} \mathbf{x}_1(t) \\ \mathbf{x}_2(t) \end{bmatrix} = \begin{bmatrix} \mathbf{p}(t) \\ \dot{\mathbf{p}}(t) \end{bmatrix}, \tag{3.4}$$

logo,

$$\dot{\mathbf{x}}_1(t) = \mathbf{x}_2(t) \tag{3.5}$$

$$\dot{\mathbf{x}}_2(t) = -\mathbf{M}^{-1}\mathbf{K}\mathbf{x}_1(t) - \mathbf{M}^{-1}\mathbf{E}\mathbf{x}_2(t) + \mathbf{B}_w\mathbf{w}(t) + \mathbf{B}_u\mathbf{u}(t) \tag{3.6}$$

$$\mathbf{y}(t) = \mathbf{C}_p\mathbf{x}_1(t) + \mathbf{C}_v\mathbf{x}_2(t) + \mathbf{C}_w\mathbf{w}(t) + \mathbf{C}_u\mathbf{u}(t). \tag{3.7}$$

Essas relações levam a seguinte representação:

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\mathbf{x}(t) + \mathbf{B}_1\mathbf{w}(t) + \mathbf{B}_2\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\mathbf{x}(t) + \mathbf{D}_1\mathbf{w}(t) + \mathbf{D}_2\mathbf{u}(t), \end{aligned} \tag{3.8}$$

em que $\mathbf{C} = [\mathbf{C}_p \quad \mathbf{C}_v]$, $\mathbf{D} = [\mathbf{D}_1 \quad \mathbf{D}_2] = [\mathbf{C}_w \quad \mathbf{C}_u]$ e as matrizes $\mathbf{A}$, $\mathbf{B}_1$ e $\mathbf{B}_2$ são obtidas como

$$\mathbf{A} = \begin{bmatrix} \mathbf{0} & \mathbf{I} \\ -\mathbf{M}^{-1}\mathbf{K} & -\mathbf{M}^{-1}\mathbf{E} \end{bmatrix}, \quad \mathbf{B}_1 = \begin{bmatrix} \mathbf{0} \\ \mathbf{B}_w \end{bmatrix} \quad \text{e} \quad \mathbf{B}_2 = \begin{bmatrix} \mathbf{0} \\ \mathbf{B}_u \end{bmatrix} \tag{3.9}$$

Vale a pena salientar que o modelo no espaço de estados em coordenadas nodais descrito pela Equação 3.8 pode ser facilmente representado nas formas canônicas modais usando matrizes de transformação específicas (Gawronski, 2004).

3.2 Solução de equações de estado

Uma vez que o sistema representado pela Equação 3.8 é linear invariante no tempo e que as matrizes $\mathbf{A}$, $\mathbf{B}_1$, $\mathbf{B}_2$, $\mathbf{C}$, $\mathbf{D}_1$ e $\mathbf{D}_2$ são constantes, a resposta temporal $\mathbf{y}(t)$ será baseada na condição inicial $x(0)$ e nas entradas $\mathbf{w}(t)$ e $\mathbf{u}(t)$. Para encontrar a solução geral, leva-se em conta a seguinte propriedade (Chen, 1995, 4.2):

$$\frac{d}{dt}e^{\mathbf{A}t} = \mathbf{A}e^{\mathbf{A}t} = e^{\mathbf{A}t}\mathbf{A}. \tag{3.10}$$

Pré-multiplicando ambos os lados da primeira igualdade da Equação 3.8 por $e^{-\mathbf{A}t}$, têm-se

$$e^{-\mathbf{A}t}\dot{\mathbf{x}}(t) - e^{-\mathbf{A}t}\mathbf{A}\mathbf{x}(t) = e^{-\mathbf{A}t}\mathbf{B}_1\mathbf{w}(t) + e^{-\mathbf{A}t}\mathbf{B}_2\mathbf{u}(t), \tag{3.11}$$

que pode ser reescrito como

$$\frac{d}{dt}\Big(e^{-\mathbf{A}t}\mathbf{x}(t)\Big) = e^{-\mathbf{A}t}\mathbf{B}_1\mathbf{w}(t) + e^{-\mathbf{A}t}\mathbf{B}_2\mathbf{u}(t). \tag{3.12}$$

Integrando ambos os lados da Equação 3.12 com limitante de 0 a t, resultando em

$$e^{-\mathbf{A}\tau}\mathbf{x}(\tau)|_{\tau=0}^{t} = \int_0^t e^{-\mathbf{A}\tau}\mathbf{B}_1\mathbf{w}(\tau)d\tau + \int_0^t e^{-\mathbf{A}\tau}\mathbf{B}_2\mathbf{u}(\tau)d\tau, \tag{3.13}$$

que produz

$$e^{-\mathbf{A}t}\mathbf{x}(t) - e^{-\mathbf{0}}\mathbf{x}(0) = \int_0^t e^{-\mathbf{A}\tau}\mathbf{B}_1\mathbf{w}(\tau)d\tau + \int_0^t e^{-\mathbf{A}\tau}\mathbf{B}_2\mathbf{u}(\tau)d\tau. \tag{3.14}$$

Tendo em vista que a inversa de $e^{-\mathbf{A}t}$ é igual a $e^{\mathbf{A}t}$ e $e^{\mathbf{0}} = \mathbf{I}$, têm-se a solução da primeira igualdade da Equação 3.8:

$$\mathbf{x}(t) = e^{\mathbf{A}t}\mathbf{x}(0) + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}_1\mathbf{w}(\tau)d\tau + \int_0^t e^{\mathbf{A}(t-\tau)}\mathbf{B}_2\mathbf{u}(\tau)d\tau. \tag{3.15}$$

Em sistemas de controle discreto, considera-se que as entradas $\mathbf{u}(t)$ e $\mathbf{w}(t)$ sejam constantes entre cada período de amostragem T, ou seja, os valores das entradas e saídas serão modificados apenas nos instantes de tempo discreto, utilizando um amostrador de ordem zero, por exemplo. O erro entre o valor real e o amostrado pode ser controlado a partir de uma escolha adequada de T. Dessa forma,

$$\mathbf{u}(t)_{t=kT} =: \mathbf{u}[k] \quad \text{e} \quad \mathbf{w}(t)_{t=kT} =: \mathbf{w}[k], \tag{3.16}$$

em que

$$kT \leq t < (k+1)T \quad \text{para} \quad k = 0, 1, ...$$

Assim, o valor da variável de estado nos instantes $t = kT$ e $t = (k+1)T$ são dados por:

$$\begin{aligned}\mathbf{x}[k] :=& \mathbf{x}(kT) \\ =& e^{\mathbf{A}kT}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}(\mathbf{B}_1\mathbf{w}(\tau)d\tau + \mathbf{B}_2\mathbf{u}(\tau))d\tau \end{aligned} \tag{3.17}$$

e

$$\begin{aligned}\mathbf{x}[k+1] :=& \mathbf{x}((k+1)T) \\ =& e^{\mathbf{A}(k+1)T}\mathbf{x}(0) + \int_0^{(k+1)T} e^{\mathbf{A}((k+1)T-\tau)}(\mathbf{B}_1\mathbf{w}(\tau)d\tau + \mathbf{B}_2\mathbf{u}(\tau))d\tau. \end{aligned} \tag{3.18}$$

Rearranjando os termos, a Equação 3.18 pode ser reescrita como

$$\begin{aligned}\mathbf{x}[k+1] =& e^{\mathbf{A}T}\left[e^{\mathbf{A}kT}\mathbf{x}(0) + \int_0^{kT} e^{\mathbf{A}(kT-\tau)}(\mathbf{B}_1\mathbf{w}(\tau)d\tau + \mathbf{B}_2\mathbf{u}(\tau))d\tau\right] \\ &+ \int_{kT}^{(k+1)T} e^{\mathbf{A}(kT+T-\tau)}(\mathbf{B}_1\mathbf{w}(\tau)d\tau + \mathbf{B}_2\mathbf{u}(\tau))d\tau. \end{aligned} \tag{3.19}$$

Introduzindo uma nova variável $\alpha = kT + T - \tau$ e substituindo as Equações 3.16 e 3.18 na Equação 3.19, obtém-se:

$$\mathbf{x}[k+1] = e^{\mathbf{A}T}\mathbf{x}[k] + \left(\int_0^T e^{\mathbf{A}\alpha}d\alpha\right)\mathbf{B}_1\mathbf{w}[k] + \left(\int_0^T e^{\mathbf{A}\alpha}d\alpha\right)\mathbf{B}_2\mathbf{u}[k]. \tag{3.20}$$

No caso em que a resposta do sistema é adquirida somente nos instantes de amostragem $t = kT$, a Equação 3.8 têm sua representação no tempo discreto dada por:

$$\mathbf{x}[k+1] = \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_{1d}\mathbf{w}[k] + \mathbf{B}_{2d}\mathbf{u}[k] \tag{3.21}$$

$$\mathbf{y}[k] = \mathbf{C}_d\mathbf{x}[k] + \mathbf{D}_{1d}\mathbf{w}[k] + \mathbf{D}_{2d}\mathbf{u}[k], \tag{3.22}$$

com

$$\mathbf{A}_d = e^{\mathbf{A}T}, \quad \mathbf{B}_{1d} = \left(\int_0^T e^{\mathbf{A}\alpha}d\alpha\right)\mathbf{B}_1, \quad \mathbf{B}_{2d} = \left(\int_0^T e^{\mathbf{A}\alpha}d\alpha\right)\mathbf{B}_2, \tag{3.23}$$

$$\mathbf{C}_d = \mathbf{C}, \quad \mathbf{D}_{1d} = \mathbf{D}_1 \quad \text{e} \quad \mathbf{D}_{2d} = \mathbf{D}_2. \tag{3.24}$$

Levando em conta que que a matriz $\mathbf{A}$ é não-singular, então as matrizes de entrada são obtidas simplesmente como

$$\mathbf{B}_{1d} = \mathbf{A}^{-1}(\mathbf{A}_d - \mathbf{I})\mathbf{B}_1 \quad \text{e} \quad \mathbf{B}_{2d} = \mathbf{A}^{-1}(\mathbf{A}_d - \mathbf{I})\mathbf{B}_2. \tag{3.25}$$

Dessa forma, é possível obter as matrizes que representam as equações de um sistema a tempo discreto como resultado da discretização do mesmo em tempo contínuo.

3.3 Método de identificação ERA

O método de identificação ERA foi originalmente desenvolvido para estimar o modelo numérico de sistemas a partir das vibrações livres causadas por um impulso como excitação, sendo frequentemente utilizado para estimar os parâmetros modais de estruturas (Caicedo *et al.*, 2004; Ma *et al.*, 2010; Chiang *et al.*, 2010). No contexto da estrutura vertical flexível utilizada neste trabalho, as matrizes $\mathbf{A}_d$, $\mathbf{B}_{1d}$, $\mathbf{B}_{2d}$, $\mathbf{C}_d$, $\mathbf{D}_{1d}$ e $\mathbf{D}_{2d}$ são obtidas utilizando princípios de realização mínima. Os polos do modelo identificado são os autovalores da matriz $\mathbf{A}_d$, sendo dados por

$$polo_i = -\xi_i \omega_{n_i} \pm j\left(\omega_{n_i}\sqrt{1-\xi_i^2}\right), \tag{3.26}$$

no qual cada par de polos complexos conjugados $polo_i$, corresponde ao i-ésimo modo de vibração e pode ser utilizado para determinar a frequência natural ω_{n_i} correspondente ao respectivo amortecimento ξ_i. A parte imaginária representa a frequência natural amortecida pertencente ao i-ésimo modo.

Considerando a estrutura flexível representada pelas Equações 3.21 e 3.22, sua representação pode ser decomposta em duas plantas diferentes T_{yw} e T_{yu}. A planta T_{yw} é formada pelas matrizes $\mathbf{A}_d$, $\mathbf{B}_{1d}$, $\mathbf{C}_d$ e $\mathbf{D}_{1d}$, enquanto, a planta T_{yu}, é formada pelas matrizes $\mathbf{A}_d$, $\mathbf{B}_{2d}$, $\mathbf{C}_d$ e $\mathbf{D}_{2d}$. A saída $\mathbf{y}[k]$ é dada através da superposição das respostas de cada planta às entradas $\mathbf{w}[k]$ e $\mathbf{u}[k]$.

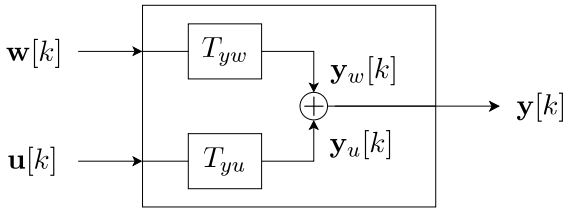


Figura 3.1 – Representação da estrutura flexível

Com o intuito de apresentar a técnica ERA para sistemas MIMO (de *Multiple-Input-Multiple-Output*) de forma compacta, inicialmente é considerado que exista apenas uma força de distúrbio $\mathbf{w}[k]$ e apenas uma força de controle $\mathbf{u}[k]$ atuando na estrutura. O desenvolvimento do ERA MIMO é expandido a partir dessa condição ao longo do texto. Dessa forma, supondo que exista condições iniciais nulas, i.e., $\mathbf{x}[0] = 0$ e que os sinais aplicados nas entradas são $\mathbf{u}[k] = 0$ e $\mathbf{w}[k]$ é um impulso unitário, ou seja, $\mathbf{w}[k] = 1$ somente para $k = 0$, a saída $\mathbf{y}[k]$ é dada por $\mathbf{y}_w[k]$ em termos das matrizes $\mathbf{A}_d, \mathbf{B}_{1d}, \mathbf{C}_d$ e $\mathbf{D}_{1d}$. Dessa forma, propagando as

Equações 3.21 e 3.22 para essas condições e para $k \geq 0$, tem-se:

$$\begin{aligned}
x[0] &= \mathbf{0} & y_w[0] &= \underbrace{\mathbf{D}_{1d}}_{\mathbf{G}_1[0]} \\
x[1] &= \mathbf{B}_{1d} & y_w[1] &= \underbrace{\mathbf{C}_d\mathbf{B}_{1d}}_{\mathbf{G}_1[1]} \\
x[2] &= \mathbf{A}_d\mathbf{B}_{1d} & y_w[2] &= \mathbf{C}_d\mathbf{A}_d\mathbf{B}_{1d} \\
x[3] &= \mathbf{A}_d^2\mathbf{B}_{1d} & y_w[3] &= \mathbf{C}_d\mathbf{A}_d^2\mathbf{B}_{1d} \\
&\vdots & &\vdots \\
\mathbf{x}[k] &= \mathbf{A}_d^{k-1}\mathbf{B}_{1d} & \mathbf{y}_w[k] &= \underbrace{\mathbf{C}_d\mathbf{A}_d^{k-1}\mathbf{B}_{1d}}_{\mathbf{G}_1[k]},
\end{aligned} \tag{3.27}$$

no qual os valores de $\mathbf{G}_1[k]$ são definidos como

$$\mathbf{G}_1[k] = \begin{cases} 0, & k < 0 \\ \mathbf{D}_{1d}, & k = 0 \\ \mathbf{C}_d\mathbf{A}_d^{k-1}\mathbf{B}_{1d}, & k > 0. \end{cases} \tag{3.28}$$

Os valores do conjunto $\mathbf{G}_1[k]$ são denominados parâmetros de Markov relativos a entrada $\mathbf{w}[k]$. Neste formato, é possível obter a resposta do sistema $\mathbf{y}[k]$ a uma excitação $\mathbf{w}[k]$ qualquer, utilizando a convolução deste sinal com a resposta impulsiva da Equação 3.28 na seguinte forma matricial:

$$\begin{bmatrix} y_w[0] \\ y_w[1] \\ y_w[2] \\ \vdots \end{bmatrix} = \begin{bmatrix} \mathbf{G}_1[0] & 0 & 0 & \cdots & 0 \\ \mathbf{G}_1[1] & \mathbf{G}_1[0] & 0 & \cdots & 0 \\ \mathbf{G}_1[2] & \mathbf{G}_1[1] & \mathbf{G}_1[0] & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \end{bmatrix} \begin{bmatrix} \mathbf{w}[0] \\ \mathbf{w}[1] \\ \mathbf{w}[2] \\ \vdots \end{bmatrix}, \tag{3.29}$$

em que

$$\mathbf{y}_w[k] = \sum_{j=0}^{\infty} \mathbf{G}_1[k-j]\mathbf{w}[j]. \tag{3.30}$$

Analogamente, levando em conta que $\mathbf{x}[0] = 0$ e que as entradas sejam $\mathbf{w}[k] = 0$ e $\mathbf{u}[k]$ um impulso unitário, a resposta em $\mathbf{y}[k]$ é dada por $\mathbf{y}_u[k]$, obtida em termos das matrizes $\mathbf{A}_d, \mathbf{B}_{2d}, \mathbf{C}_d$ e $\mathbf{D}_{2d}$. Portanto, a resposta do sistema a uma excitação $\mathbf{u}[k]$ qualquer é dada por

$$\mathbf{y}_u[k] = \sum_{j=0}^{\infty} \mathbf{G}_2[k-j]\mathbf{u}[j]. \tag{3.31}$$

Considerando que o sistema seja causal, adota-se o conceito de que as saídas futuras $\mathbf{Y}_+$ foram medidas para $k \geq 0$ e que as entradas passadas $\mathbf{E}_-$ foram aplicadas anteriormente

ao instante $k = 0$, ou seja, elas possuem valores nulos para $k \geq 0$ e não nulos para $k \leq -1$. Assim, é possível obter a saída $\mathbf{y}[k]$ em função da soma das Equações 3.30 e 3.31 como

$$\begin{aligned}\mathbf{y}[k] &= \mathbf{y}_w[k] + \mathbf{y}_u[k] \\ &= \mathbf{G}_1[k] * \mathbf{w}[k] + \mathbf{G}_2[k] * \mathbf{u}[k] \\ &= \sum_{j=-1}^{-\infty} \mathbf{G}_1[k-j]w[j] + \sum_{j=-1}^{-\infty} \mathbf{G}_2[k-j]u[j], \quad k \in \mathbb{Z}\end{aligned} \tag{3.32}$$

no qual sua representação na forma matricial é dada por

$$\underbrace{\begin{bmatrix} y[0] \\ y[1] \\ y[2] \\ \vdots \end{bmatrix}}_{\mathbf{Y}_+} = \underbrace{\begin{bmatrix} \mathbf{M}_{kv}[1] & \mathbf{M}_{kv}[2] & \mathbf{M}_{kv}[3] & \cdots \\ \mathbf{M}_{kv}[2] & \mathbf{M}_{kv}[3] & \mathbf{M}_{kv}[4] & \cdots \\ \mathbf{M}_{kv}[3] & \mathbf{M}_{kv}[4] & \mathbf{M}_{kv}[5] & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}}_{\mathbf{H}[1]} \underbrace{\begin{bmatrix} \mathbf{e}[-1] \\ \mathbf{e}[-2] \\ \mathbf{e}[-3] \\ \vdots \end{bmatrix}}_{\mathbf{E}_-}, \tag{3.33}$$

em que $\mathbf{M}_{kv}[l] = \begin{bmatrix} \mathbf{G}_1[l] & \mathbf{G}_2[l] \end{bmatrix}$, $\mathbf{e}[-l] = \begin{bmatrix} \mathbf{w}[-l] & \mathbf{u}[-l] \end{bmatrix}^T$ para $l = 1, 2, \cdots$ e $\mathbf{H}[1]$ é uma matriz de Hankel com dimensão infinita. A realização do sistema pelo algoritmo ERA ocorre mediante a construção da matriz de Hankel agrupando os parâmetros de Markov a partir de um número finito de medidas N. Por essa razão, a matriz de Hankel infinita é truncada escolhendo adequadamente os parâmetros p e q na seguinte maneira:

$$\mathbf{H}[1]_{p,q} = \begin{bmatrix} \mathbf{M}_{kv}[1] & \mathbf{M}_{kv}[2] & \mathbf{M}_{kv}[3] & \cdots & \mathbf{M}_{kv}[q] \\ \mathbf{M}_{kv}[2] & \mathbf{M}_{kv}[3] & \mathbf{M}_{kv}[4] & \cdots & \mathbf{M}_{kv}[q+1] \\ \mathbf{M}_{kv}[3] & \mathbf{M}_{kv}[4] & \mathbf{M}_{kv}[5] & \cdots & \mathbf{M}_{kv}[q+2] \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{M}_{kv}[p] & \mathbf{M}_{kv}[p+1] & \mathbf{M}_{kv}[p+2] & \cdots & \mathbf{M}_{kv}[p+q-1] \end{bmatrix}. \tag{3.34}$$

Para obter a matriz de Hankel na Equação 3.34, foi considerado uma entrada de distúrbio e outra entrada de controle. Entretanto, a extensão do desenvolvimento para um sistema genérico MIMO é realizado construindo os sinais na Equação 3.32 com os sinais de entradas e de saídas genéricos. Dessa forma, a matriz $\mathbf{M}_{kv}[l]$ e o vetor $\mathbf{e}$ presentes na Equação 3.33 terão as respectivas ordens expandidas em função do número de entradas e saídas. Por fim, matriz de Hankel é obtida de forma experimental, através da obtenção das respostas ao impulso utilizando técnica de análise espectral (Hoagg *et al.*, 2006). Dessa forma, é possível agrupar os parâmetros de Markov $\mathbf{M}_{kv}[l]$, formar a matriz de Hankel $\mathbf{H}[1]_{p,q}$ e a matriz de Hankel deslocada $\mathbf{H}[2]_{p,q}$ como

$$\mathbf{H}[1]_{p,q} = \begin{bmatrix} \mathbf{M}_{kv}[1] & \mathbf{M}_{kv}[2] & \cdots & \mathbf{M}_{kv}[q] \\ \mathbf{M}_{kv}[2] & \mathbf{M}_{kv}[3] & \cdots & \mathbf{M}_{kv}[q+1] \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{M}_{kv}[p] & \mathbf{M}_{kv}[p+1] & \cdots & \mathbf{M}_{kv}[p+q-1] \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{C}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{q-1}\mathbf{B}_d \\ \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{2}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{q}\mathbf{B}_d \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_d\mathbf{A}_d^{p-1}\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{p}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{p+q-2}\mathbf{B}_d \end{bmatrix}, \tag{3.35}$$

$$\mathbf{H}[2]_{p,q} = \begin{bmatrix} \mathbf{M}_{kv}[2] & \mathbf{M}_{kv}[3] & \cdots & \mathbf{M}_{kv}[q+1] \\ \mathbf{M}_{kv}[3] & \mathbf{M}_{kv}[4] & \cdots & \mathbf{M}_{kv}[q+2] \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{M}_{kv}[p+1] & \mathbf{M}_{kv}[p+2] & \cdots & \mathbf{M}_{kv}[p+q] \end{bmatrix}$$
$$= \begin{bmatrix} \mathbf{C}_d\mathbf{A}_d\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{2}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{q}\mathbf{B}_d \\ \mathbf{C}_d\mathbf{A}_d^{2}\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{3}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{q+1}\mathbf{B}_d \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{C}_d\mathbf{A}_d^{p}\mathbf{B}_d & \mathbf{C}_d\mathbf{A}_d^{p+1}\mathbf{B}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{p+q-1}\mathbf{B}_d \end{bmatrix}. \tag{3.36}$$

sendo $\mathbf{B}_d = [\mathbf{B}_{1d} \quad \mathbf{B}_{2d}]$ e $\mathbf{M}_{kv}[l] \in \mathbb{R}^{r\times s}$ é uma matriz que representa o l-ésimo valor de resposta ao impulso. Nessa configuração, cada matriz de Hankel possui dimensão $pr \times qs$.

A matriz de Hankel $\mathbf{H}[1]_{p,q}$ pode ser decomposta pelas matrizes de controlabilidade $\mathbf{\Delta}_q$ e observabilidade $\mathbf{\Gamma}_p$ como

$$\mathbf{H}[1]_{p,q} = \mathbf{\Gamma}_p\mathbf{\Delta}_q, \tag{3.37}$$

em que

$$\mathbf{\Gamma}_p = \begin{bmatrix} \mathbf{C}_d \\ \mathbf{C}_d\mathbf{A}_d \\ \vdots \\ \mathbf{C}_d\mathbf{A}_d^{p-1} \end{bmatrix} \quad \text{e} \quad \mathbf{\Delta}_q = \begin{bmatrix} \mathbf{B}_d & \mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{A}_d^{q-1}\mathbf{B}_d \end{bmatrix}. \tag{3.38}$$

Substituindo a Equação 3.37 na Equação 3.36, têm-se

$$\mathbf{H}[2]_{p,q} = \mathbf{\Gamma}_p\mathbf{A}_d\mathbf{\Delta}_q. \tag{3.39}$$

O segundo passo do algoritmo ERA é utilizar as propriedades de subespaços a fim de obter as informações sobre os parâmetros do sistema através da decomposição em valores singulares (SVD) da matriz de Hankel $\mathbf{H}[1]_{p,q}$. Assim, a matriz $\mathbf{H}[1]_{p,q}$ pode ser decomposta como

$$\mathbf{H}[1]_{p,q} = \mathbf{U}\boldsymbol{\Sigma}^{1/2}\boldsymbol{\Sigma}^{1/2}\mathbf{V}^T, \tag{3.40}$$

em que $\mathbf{U} \in \mathbb{R}^{r*p \times r*p}$ e $\mathbf{V} \in \mathbb{R}^{s*q \times s*q}$ são matrizes ortonormais e $\boldsymbol{\Sigma} \in \mathbb{R}^{r*p \times s*q}$ é uma matriz com números não negativos, em ordem decrescente, na diagonal. Comparando a Equação3.37 com a Equação 3.40, tem-se:

$$\boldsymbol{\Gamma}_p = \mathbf{U}\boldsymbol{\Sigma}^{1/2} \qquad \text{e} \qquad \boldsymbol{\Delta}_q = \boldsymbol{\Sigma}^{1/2}\mathbf{V}^T. \tag{3.41}$$

Na realização mínima, os modos de menor energia da estrutura são desconsiderados através da inspeção dos valores mais significativos e da seleção dos primeiros m valores singulares não nulos e preponderantes de $\mathbf{H}[1]_{p,q}$ para representar a dinâmica estrutural. Dessa maneira, a matriz $\boldsymbol{\Sigma}$ é truncada na forma

$$\boldsymbol{\Sigma} = \begin{bmatrix} \boldsymbol{\Sigma}_m & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \end{bmatrix}, \tag{3.42}$$

sendo $\boldsymbol{\Sigma}_m \in \mathbb{R}^{m \times m}$. A eliminação dos valores singulares de menor intensidade, resulta em um sistema com a menor ordem possível e que representa satisfatoriamente a dinâmica da estrutura. Logo,

$$\mathbf{U}_m = \mathbf{U}(1 : r * p, 1 : m) \qquad \mathbf{V}_m = \mathbf{V}(1 : s * q, 1 : m). \tag{3.43}$$

A decomposição em valores singulares da Equação 3.40 não é única. Dessa forma, usando a matriz de Hankel $\mathbf{H}[2]_{p,q}$, obtém-se

$$\mathbf{H}[2]_{p,q} = \underbrace{\mathbf{U}_m\boldsymbol{\Sigma}_m^{1/2}}_{\boldsymbol{\Gamma}_m}\mathbf{A_d}\underbrace{\boldsymbol{\Sigma}_m^{1/2}\mathbf{V}_m^T}_{\boldsymbol{\Delta}_m}, \tag{3.44}$$

e, dessa forma,

$$\mathbf{A}_d = \boldsymbol{\Sigma}_m^{-1/2}\mathbf{U}_m^T\mathbf{H}[2]_{p,q}\mathbf{V}_m\boldsymbol{\Sigma}_m^{-1/2}. \tag{3.45}$$

As matrizes $\mathbf{B}_d$ e $\mathbf{C}_d$ são obtidas de forma direta a partir de $\boldsymbol{\Delta}_m$ e $\boldsymbol{\Gamma}_m$:

$$\mathbf{B}_d = \boldsymbol{\Delta}_m(1 : m, 1 : s) \quad \text{e} \quad \mathbf{C}_d = \boldsymbol{\Gamma}_m(1 : r, 1 : m), \tag{3.46}$$

por fim

$$\mathbf{D}_{1d} = \big[0\big]_{r \times s_w} \quad \text{e} \quad \mathbf{D}_{2d} = \big[0\big]_{r \times s_u} \tag{3.47}$$

3.4 Projeto do controlador

Das Equações 3.21 e 3.22, o par $(\mathbf{A}_d, \mathbf{B}_{2d})$ é considerado completamente controlável e o par $(\mathbf{A}_d, \mathbf{C}_d)$ é considerado completamente observável. Levando em conta que $\mathbf{D}_d = 0$ e atribuindo a lei de controle

$$\mathbf{u}[k] = -\mathbf{K}_g \mathbf{x}[k], \tag{3.48}$$

esta equação pode ser escrita como

$$\mathbf{x}[k+1] = (\mathbf{A}_d - \mathbf{B}_{2d}\mathbf{K}_g)\mathbf{x}[k] + \mathbf{B}_{1d}\mathbf{w}[k] \tag{3.49}$$

$$\mathbf{y}[k] = \mathbf{C}_d \mathbf{x}[k], \tag{3.50}$$

em que o ganho $\mathbf{K}_g$ leva os polos do sistema em malha fechada para a localização desejada, conduzindo ao desempenho desejado.

No controle por realimentação de estados é necessário que todos os estados estejam disponíveis. Entretanto, frequentemente não é possível medir todos eles. Dessa forma, uma alternativa é estimá-los a partir da saída do sistema por meio um observador com a seguinte forma:

$$\hat{\mathbf{x}}[k+1] = \mathbf{A}_d \hat{\mathbf{x}}[k] + \mathbf{B}_{2d}\mathbf{u}[k] + \mathbf{L}_g(\mathbf{y}[k] - \hat{\mathbf{y}}[k]) \tag{3.51}$$

$$\hat{\mathbf{y}}[k] = \mathbf{C}_d \hat{\mathbf{x}}[k], \tag{3.52}$$

em que $\hat{\mathbf{x}}[k]$ e $\hat{\mathbf{y}}[k]$ são as estimativas do vetor de estados e da saída. O ganho $\mathbf{L}_g$ é escolhido para minimizar o erro de estimação $\mathbf{e}[k] = \hat{\mathbf{x}}[k] - \mathbf{x}[k]$. Substituinto a Equação 3.52 na Equação 3.51, tem-se

$$\hat{\mathbf{x}}[k+1] = \mathbf{A}_d \hat{\mathbf{x}}[k] + \mathbf{B}_{2d}\mathbf{u}[k] + \mathbf{L}_g(\mathbf{y}[k] - \mathbf{C}_d\hat{\mathbf{x}}[k]) \tag{3.53}$$

$$= \big(\mathbf{A}_d - \mathbf{L}_g\mathbf{C}_d\big)\hat{\mathbf{x}}[k] + \mathbf{B}_{2d}\mathbf{u}[k] + \mathbf{L}_g\mathbf{y}[k]. \tag{3.54}$$

Usando as Equações 3.21, 3.22 e a Equação 3.54, a estimação da dinâmica do erro é dada por

$$\begin{aligned}
\mathbf{e}[k+1] &= \mathbf{x}[k+1] - \hat{\mathbf{x}}[k+1] \\
&= \mathbf{A}_d\mathbf{x}[k] + \mathbf{B}_{1d}\mathbf{w}[k] + \mathbf{B}_{2d}\mathbf{u}[k] - \big(\mathbf{A}_d - \mathbf{L}_g\mathbf{C}_d\big)\hat{\mathbf{x}}[k] - \mathbf{B}_{2d}\mathbf{u}[k] - \mathbf{L}_g\mathbf{y}[k] \qquad (3.55) \\
&= \big(\mathbf{A}_d - \mathbf{L}_g\mathbf{C}_d\big)\mathbf{e}[k] + \mathbf{B}_{1d}\mathbf{w}[k], \qquad (3.56)
\end{aligned}$$

em que o vetor de ganho $\mathbf{L}_g$ é escolhido para minimizar o erro de estimação de estados. O princípio da separação estabelece que os projetos do controlador e do observador podem ser realizados de modo independente, i.e., a ordem do cálculo não afeta o resultado do sistema controlado (Ogata, 2009). A Figura 3.2 ilustra as relações entre a planta, o observador e a lei de controle.

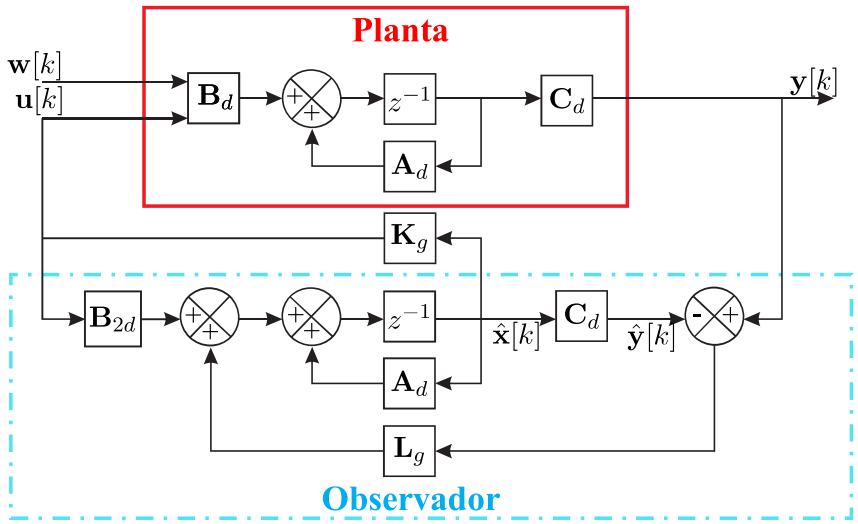


Figura 3.2 – Controle por realimentação de estados

3.4.1 Cálculo dos ganhos do sistema de controle

Uma abordagem frequentemente utilizada para calcular os ganhos do sistema de controle a partir da alocação dos polos, tanto para o controlador quanto para o observador, consiste em aplicar a fórmula de Ackermann (Ogata, 1994). Para um sistema de ordem m, o vetor de ganhos do controlador é determinado como

$$\mathbf{K}_g = \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix} \begin{bmatrix} \mathbf{B}_d & \mathbf{A}_d\mathbf{B}_d & \cdots & \mathbf{A}_d^{m-1}\mathbf{B}_d \end{bmatrix}^{-1} \phi_c(\mathbf{A}_d), \tag{3.57}$$

em que $\phi_c(\mathbf{A}_d) = \mathbf{A}_d^m + c_1\mathbf{A}_d^{m-1} + \cdots + c_n\mathbf{I}$, cujos coeficientes $c_1 \cdots c_m$, são dados pela equação característica dos polos desejados para o controlador. O cálculo do vetor de ganhos do observador, pela fórmula de Ackermann, é dado por

$$\mathbf{L}_g = \phi_o(\mathbf{A}_d) \left(\begin{bmatrix} \mathbf{C}_d & \mathbf{C}_d\mathbf{A}_d & \cdots & \mathbf{C}_d\mathbf{A}_d^{m-1} \end{bmatrix}^T \right)^{-1} \begin{bmatrix} 0 & 0 & \cdots & 1 \end{bmatrix}^T, \tag{3.58}$$

em que $\phi_o(\mathbf{A}_d) = \mathbf{A}_d^m + o_1\mathbf{A}_d^{m-1} + \cdots + o_n\mathbf{I}$, cujos coeficientes $o_1 \cdots o_m$, são dados pela equação característica dos polos desejados para o observador.

3.5 Indicador de dano

O cepstro é definido como a transformada inversa de Fourier do espectro de um sinal na forma logarítmica. Além disso, a métrica cepstral pode ser utilizada como ferramenta para calcular a distância entre dois modelos (Martin, 2000). Nesse contexto, Cock, 2002 demonstra que a norma cepstral correspondente a um modelo de n-ésima ordem é dada em função dos n ângulos principais θ_n entre os espaços-linha das matrizes de controlabilidade desse modelo e o modelo inverso (ou os espaços de coluna de suas matrizes de observabilidade).

Com o intuito de permitir a inclusão de danos como um dos requisitos do projeto de controle, utiliza-se a métrica de subespaços. A abordagem regular deste procedimento consiste em avaliar a distância entre os subespaços que representam um modelo saudável e outro possivelmente danificado, cuja mudança da resposta dinâmica é utilizada como indicador de dano. Na literatura, Zheng; Mita 2008 aplicaram a métrica de subespaço para monitorar a saúde estrutural de um edifício simulado com cinco andares e sujeito a excitações ambientais e sísmicas. Genari *et al.*, 2013 investigaram experimentalmente a métrica para detecção, quantificação da severidade e localização de danos. Genari *et al.*, 2017b modificaram a métrica para determinar o efeito do dano em cada modo de vibração. Nessa dissertação, essa métrica é utilizada para quantificar o danos e sinalizar ao sistema de controle a necessidade de reconfiguração para manter um desempenho adequado e sobretudo garantir a estabilidade. A matriz infinita de observabilidade do sistema descrito pelas Equações 3.21 3.22 é definida como

$$\mathcal{O}_\infty(M_{odel}) = \begin{bmatrix} \mathbf{C}_d & \mathbf{C}_d\mathbf{A}_d & \mathbf{C}_d\mathbf{A}_d^2 & \cdots \end{bmatrix}^T. \tag{3.59}$$

Considerando dois modelos estáveis $M_{odel}^{(1)} = \left(\mathbf{A}_d^{(1)}, \mathbf{C}_d^{(1)}\right)$ e $M_{odel}^{(2)} = \left(\mathbf{A}_d^{(2)}, \mathbf{C}_d^{(2)}\right)$, de ordem m, suas respectivas matrizes de observabilidade infinita são $\mathcal{O}_\infty(M_{odel}^{(1)})$ e $\mathcal{O}_\infty(M_{odel}^{(2)})$. Um dos modelos é considerado como sendo a referência saudável e o outro é um modelo para comparação. A distância D_s entre $M_{odel}^{(1)}$ e $M_{odel}^{(2)}$ pode ser calculada em termos dos ângulos principais entre os subespaços colunas de $\mathcal{O}_\infty(M_{odel}^{(1)})$ e $\mathcal{O}_\infty(M_{odel}^{(2)})$, sendo dada dada por (Genari *et al.*, 2013):

$$D_s(M_{odel}^{(1)}, M_{odel}^{(2)})^2 = \log\left(\prod_{i=1}^{n} \frac{1}{\cos^2\theta_i}\right), \tag{3.60}$$

em que θ_i são os ângulos principais entre os subespaços colunas de $\mathcal{O}_\infty(M_{odel}^{(1)})$ e $\mathcal{O}_\infty(M_{odel}^{(2)})$. Em Genari, 2012 é possível obter mais detalhes sobre a métrica por subespaços, incluindo uma rotina para calcular a distância entre dois modelos.

4 PROGRAMAÇÃO DO SISTEMA EMBARCADO

Este capítulo descreve o ambiente de aplicação da linguagem OpenCL e sua arquitetura, mediante uma abstração de baixo nível do hardware no qual ela é inserida. Em seguida, são apresentados os procedimentos de compilação dos programas do servidor DE-Nano. Os códigos desses programas, consistem em uma aplicação executada no processador ARM e outra executada no FPGA, sendo apresentados em detalhes, nas seguintes seções. Por fim, são descritos os principais códigos implementados no cliente Raspberry Pi.

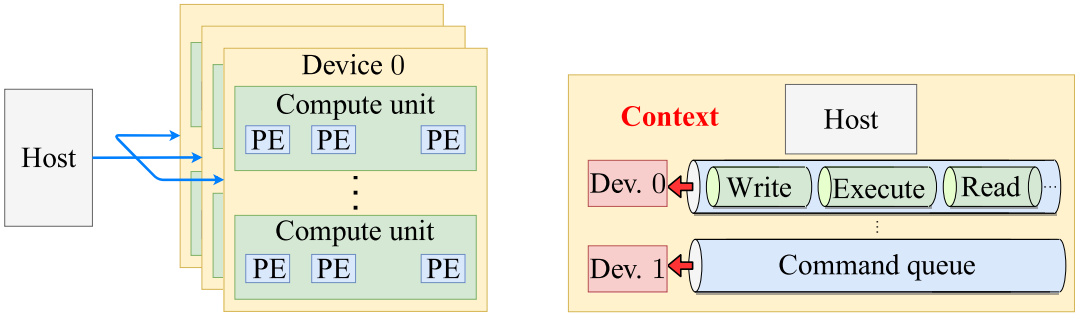
4.1 Arquitetura OpenCL

O padrão de linguagem OpenCL (*Open Computing Language*) foi desenvolvido pela Apple® e atualmente é mantido pelo Khronos Group (Khronosgroup, 2010). O OpenCL é um *framework* para escrever programas que são executados em plataformas heterogêneas constituídas por CPUs, GPUs, DSPs, FPGAs e outros tipos de processadores. Sua característica principal é a portabilidade, ou seja, ele pode ser executado em várias plataformas diferentes, sendo capaz de extrair alto desempenho de cada uma. O OpenCL fornece mecanismos para computação paralela mediante ambos os paralelismos de dados e tarefas.

A arquitetura OpenCL é descrita pelos modelos plataforma, execução, memória e programação, bem como por uma série de conceitos associados a cada representação. No ambiente que envolve uma aplicação OpenCL, o modelo de plataforma mostrado na Figura4.1a é classificado em dois aspectos. De um lado o programa principal é executado na unidade central de processamento (CPU), denominada *host*. Ele comanda e se comunica com os dispositivos aceleradores através das API's (interfaces de aplicação), que são fornecidas pelo SDK (kit de desenvolvimento do software) para OpenCL de um determinado fabricante. Do outro lado, o dispositivo acelerador (*target*) é formado por múltiplas unidades de computação, que são divididas em um ou mais elementos de processamento (PE's) e de memória local.

A interação *host-target* é regida sob um modelo de execução demonstrado pela Figura 4.1b, que descreve o modo como os *kernels* no dispositivos são executados pela CPU. Nesse contexto, o *host* usa um mecanismo de comunicação denominado fila para enviar comandos e coordenar a execução dos *kernels* nos dispositivos. Dessa forma, ao menos uma fila de comando por dispositivo deve ser criada. Os comandos se baseiam na execução dos *kernels*

e no acesso de memória. Cada comando gera um objeto de evento que pode ser submetido a comandos de sincronização pelo *host*.

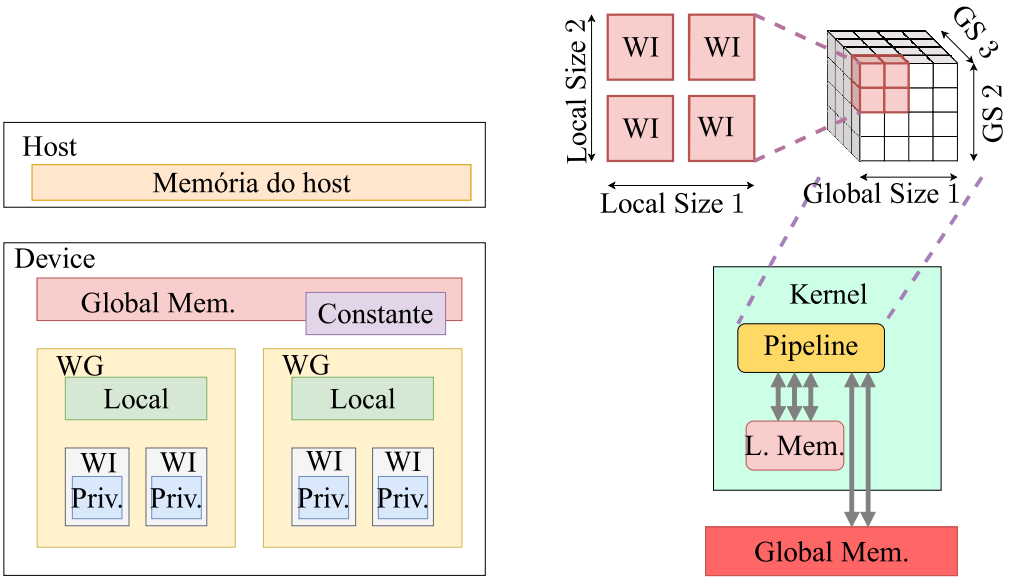


(a) Modelo de platafoma (b) Modelo de execução

Figura 4.1 – Representação OpenCL dos modelos de platafoma e de execução
Fonte: Adaptado de Waidyasooriya *et al.*,2018.

Os *kernels* são funções declaradas em um programa pelo qualificador `__kernel` e executadas nos dispositivos OpenCL através de comandos específicos. A unidade de execução em um ou mais PE é denominada *work-item*. Cada *work-item* possui uma memória privada (composta por RAM ou registrador) que é dedicada para armazenar as variáveis internas de um *kernel*. O *work-group* por sua vez é uma coleção de *work-itens* que são executados em uma única unidade de computação. Os *work-itens*, pertencentes a um *work-group*, executam o mesmo *kernel* e podem compartilhar a memória RAM local de rápido acesso dentro do chip, desde que a operação seja sincronizada por meio de barreiras. No entanto, o compartilhamento de dados entre diferentes *work-groups* requer o acesso à memória global externa, o que leva um tempo consideravelmente maior. Existe também um espaço constante e somente de leitura que reside na memória global. Esses dados são carregados automaticamente na cache do dispositivo na execução, com intuito de agilizar o seu acesso. Dessa maneira, o modelo de memória OpenCL do acelerador está representado na Figura 4.2a.

O número de *work-itens* em um *work-group* é chamado de *local work size*, e o número total de *work-itens* necessários para executar completamente uma aplicação é denominado *global work size*. Um *work-item* é diferenciado de outro por sua identificação global na aplicação ou pela posição local dentro de um *work-group*, uma vez que eles podem ser organizados em até três dimensões no espaço de índice denominado NDRange, de acordo com o modelo de programação adotado. Dessa forma, esse modelo de programação é representado conforme a Figura 4.2b.



(a) Modelo de memória (b) Modelo de programação

Figura 4.2 – Representação OpenCL dos modelos de memória e de programação

4.2 Geração dos arquivos embarcados no servidor DE10-Nano

Neste trabalho foi utilizado como servidor o kit de desenvolvimento Terasic DE10-Nano, o qual é baseado em um FPGA Cyclone®V encapsulado no mesmo SoC (*System-on-a-chip*) que um processador ARM Cortex-A9®. Os códigos dos *kernels*, escritos em OpenCL, são executados no *target* FPGA, enquanto o código do *host*, escrito em C, é executado no processador ARM. O programa do *host* é compilado pelo pacote de desenvolvimento embarcado (EDS, de *Embedded Development Suite*) da Intel®, instalado em um computador Windows ou Linux, conectado com as bibliotecas do SDK OpenCL para FPGA, que implementam as chamadas da API para OpenCL. O arquivo binário gerado nesse procedimento é enviado ao DE10-Nano para ser executado no processador ARM. Dessa forma, o SDK fornece ao *host* meios para executar e gerenciar os *kernels* escritos em OpenCL e executados no acelerador FPGA, sem a necessidade de conhecimento das linguagens VHDL ou Verilog, frequentemente utilizadas na implementação de aplicações com FPGA. A compilação de um *kernel* OpenCL para o FPGA é realizada em modo *offline* pelo compilador AOC (*Altera Offline Compiler*) incluso no SDK, pois o tempo gasto no roteamento do circuito a ser implementado torna inviável efetuar a compilação em tempo de execução. Dessa forma, o processo de compilação do código de um *kernel* consiste em duas etapas. Inicialmente, o código OpenCL é convertido em um arquivo objeto intermediário com extensão .aoco, que representa o sistema personalizado do hardware FPGA capaz de executar os *kernels*. Em seguida, ele é utilizado para gerar o arquivo executável de exten-

são .aocx. Este último é uma sequência binária que o *host* utiliza para programar o FPGA em tempo de execução. O compilador AOC cria a lógica necessária para executar os algoritmos do *kernel* e também constrói as estruturas de memória apropriadas e caminhos de conexão entre os *kernels*, o *host*, a memória interna e o controlador de acesso à DDR SDRAM externa. Por isso, para implementar a aplicação em OpenCL no FPGA, é necessário que o dispositivo contenha um BSP (pacote de suporte de placa) pré-compilado. Dessa forma, os *kernels* e sua interface dentro do chip são mesclados com o BSP e, então, eles podem acessar os recursos da plataforma. Na Figura 4.3, as conexões contidas externamente à linha tracejada são constituídas pelo BSP, enquanto que o conteúdo interno à linha tracejada é gerado pela compilação *offline* do *kernel* e programado posteriormente no FPGA.

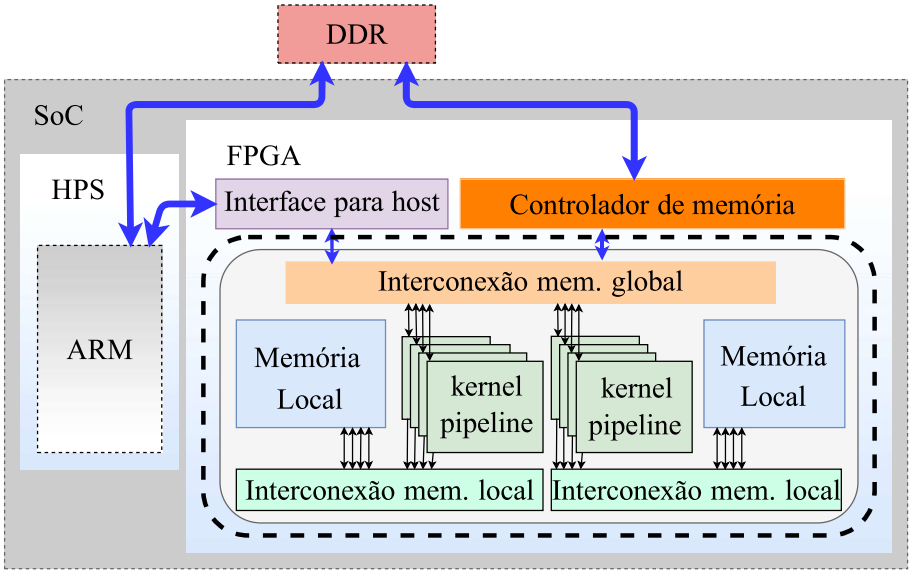


Figura 4.3 – Compilação OpenCL no chip FPGA
Fonte: Adaptado de Altera,2013.

A implementação de um algoritmo no acelerador FPGA descrito em OpenCL não é suficiente para garantir a aceleração no tempo de execução. O desempenho do algoritmo ainda é sujeito às restrições da plataforma. Além disso, o algoritmo deve ser inerentemente paralelizável. Existem diversas estratégias de otimização que podem ser adotadas para melhorar o desempenho dos programa OpenCL (Altera, 2020). Por exemplo, a vetorização do kernel e o aumento das unidades de computação, descritas nas Seções 4.4.1 e 4.4.2. Waidyasooriya *et al.*, 2018 demonstram algumas abordagens para melhorar o desempenho de execução, as quais consistem em definir o número máximo de *work-groups*, eliminar a dependência de dados entre *loops* aninhados, a fim de reduzir o intervalo de iniciação entre cada interação, entre outras.

4.3 Programação do Host embarcada no processador ARM do DE10-Nano

O programa embarcado no processador ARM calcula o modelo numérico estrutural, requisitando o FPGA para executar algumas funções desse cálculo e, em seguida, projeta os ganhos de controle do sistema. Dessa forma, o processador ARM deve se comunicar com o FPGA e com o cliente Raspberry Pi, recebendo deste último, os sinais processados de vibração via comunicação UDP (de *User Datagram Protocol*). Assim, o servidor DE10-Nano envia ao cliente Raspberry Pi a condição da integridade estrutural e as matrizes do modelo da estrutura e de controle do sistema. A Figura 4.4 apresenta o programa principal embarcado no ARM. Nas linhas 2 a 6 do código, são inicializadas as configurações iniciais e, na linha 8, o programa entra em um *loop* para receber os comandos do Raspberry Pi e realizar as tarefas anteriormente mencionadas.

```
1  // Funcao principal do processador ARM
2  int main() {
3    // Inicializa OpenCL
4    if(!init_opencl()){
5      return -1;
6    }
7    init_UDP();      // Inicializa conexao UDP
8    init_problem(); // Inicializa as variaveis
9    get_Markov();    // Aquire os parametros iniciais
10   ERA_OCL();       // Calcula as matrizes de referencia
11   char cmd[16];
12   while(1){
13     printf("Waiting for command...\n");
14     // Recebe comando do Raspberry Pi
15     strcpy(cmd,getCMD(s, recv_len, si_other, sendsize));
16     if(strcmp(cmd,"A")==0){
17       update_Markov(); // Atualiza os parametros de Markov
18       printf("Rebuild Model and Controller...\n");
19       ERA_OCL();  // Identificc a estrutura utilizando a tecnica ERA
20       Kgain = poleAlocation(A_ocl,B_ocl,C_ocl,D_ocl,k,dt,0.12);
21       Kc << Kgain.block(0,0,1,k); // Vetor de ganhos do controlador
22       Ke << Kgain.block(1,0,1,k); // Vetor de ganhos do observador
23       // Envia a condicao de integridade
24       sendHealth(s, recv_len, slen, si_other, sendsize);
25     }
26     else if(strcmp(cmd,"B")==0){
27       int iter_send = 0;
28       cout << "Sendind Matrix values..." << "\n";
29       while(iter_send<16){ // Envia as matrizes e os ganhos
30         sendToRasp(s, recv_len, slen, si_other, sendsize);
31         iter_send++;
32       }
33     }
34   }
35   cleanup();
36   return 0;
37 }
```

Figura 4.4 – Função principal no programa do ARM

4.3.1 Comunicação com cliente Raspberry Pi

A Figura 4.5 ilustra o diagrama sequencial das etapas da comunicação em rede UDP entre o servidor DE10-Nano e o cliente Raspberry Pi. Inicialmente, os *sockets*, que realizam a interface de comunicação entre servidor e cliente, são criados para prover a transmissão de dados entre eles e, na sequência, são definidos os respectivos endereços de IP (de *Internet Protocol*) através da função `bind()`. Todo cliente deve conhecer o *socket* do servidor (conjunto ip e porta) para se comunicar, mas o servidor só irá conhecer o *socket* do cliente quando este realizar uma conexão com ele, ou seja, a conexão no modelo cliente-servidor é sempre iniciada pelo cliente. Assim, o servidor recebe o pacote enviado do endereço informado, mediante o uso da função `recvfrom()`. Em seguida, o servidor responde ao cliente enviando um pacote para o seu endereço através da função `sendto()` e, por fim, ambos os processos finalizam a conexão pela execução da função `close()`. A Figura 4.6 ilustra os códigos de configuração do dispositivo DE10-Nano como servidor UDP, utilizando as funções `socket()` e `bind()` e ilustra os códigos de leitura e envio ao cliente Raspberry Pi, utilizando as funções `recvfrom()` e `sendto()`.

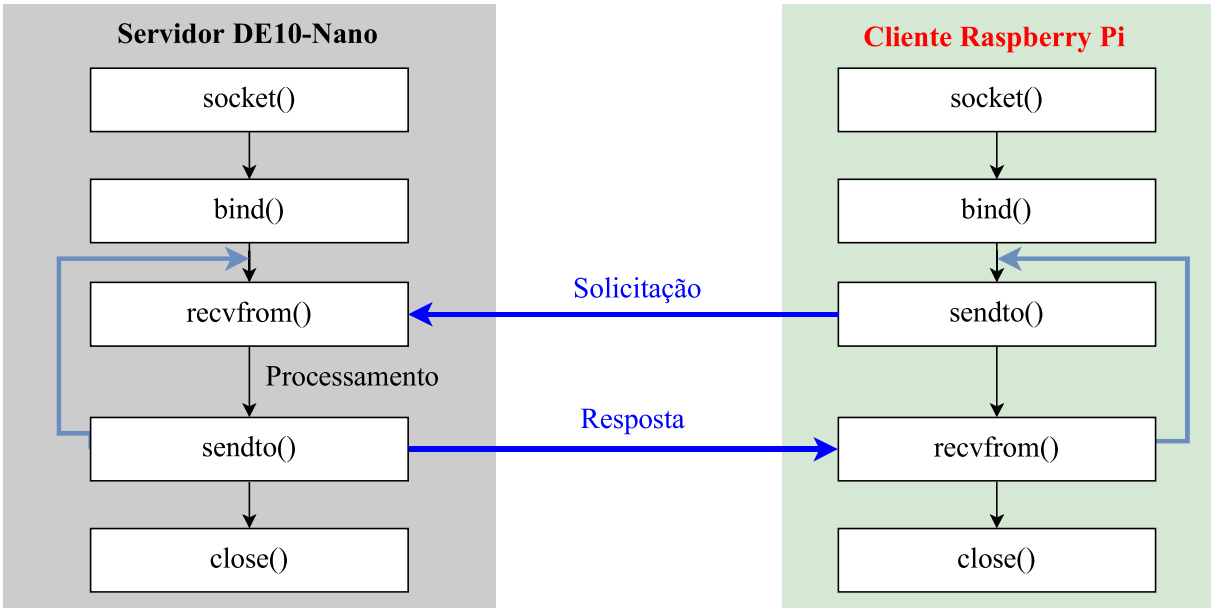


Figura 4.5 – Diagrama da comunicação via UDP
Fonte: Adaptado de Fernandes,2020.

```
// Dados no processador ARM.
struct sockaddr_in si_me, si_other;
socklen_t sendsize = sizeof(si_other);
int s, s_i, slen = sizeof(si_other) , recv_len;
// Funcao que configura a comunicacao UDP
void init_UDP() {
  //create a UDP socket
  if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1){
    perror("socket");
    exit(1);
  }
  else printf("Successfully created socket!\n");
  // Zera a estrutura de dados
  memset((char *) &si_me, 0, sizeof(si_me));
  si_me.sin_family = AF_INET;
  si_me.sin_port = SERVER_PORT;
  si_me.sin_addr.s_addr = htonl(INADDR_ANY);
  //Conecta o socket na porta
  if(bind(s,(struct sockaddr*)&si_me, sizeof(si_me))==-1){
    perror("bind ");
    exit(1);
  }
}
// Funcao que recebe dados do cliente do tipo char
char* getCMD(int s_fun, int recv_len_fun, sockaddr_in si_other_fun,
   socklen_t sendsize_fun){
  static char str_leitura[16];
  memset( str_leitura, '\0', sizeof(str_leitura));
  // recebe dado
  if ((recv_len_fun = recvfrom(s_fun, str_leitura, BUFLEN, 0, (struct
      sockaddr *) &si_other_fun, &sendsize_fun)) == -1)
    perror("recvfrom()");
  return str_leitura;
}
// Funcao que recebe dados do cliente e converte-os para tipo float
float captura(int s_fun, int recv_len_fun, sockaddr_in si_other_fun,
   socklen_t sendsize_fun){
  char leitura_buf[16];
  float leitura;
  fflush(stdout);
  memset( leitura_buf, '\0', sizeof(leitura_buf));
  if ((recv_len_fun = recvfrom(s_fun, leitura_buf, BUFLEN, 0, (struct
      sockaddr *) &si_other_fun, &sendsize_fun)) == -1)
    perror("recvfrom()");
  leitura = strtod(leitura_buf,NULL); // String para double
  return leitura;
}
// Converte a variavel de entrada float para char e envia ao cliente
float envia(int s_fun, int slen_fun, sockaddr_in si_other_fun, float
   envioValue){
  char envio_buf[8];
  sprintf(envio_buf, "%4.9f", envioValue); // Float para char
  // envia ao cliente Raspberry Pi
  if (sendto(s_fun, envio_buf, sizeof(envio_buf), 0, (struct sockaddr
     *) &si_other_fun, slen_fun) == -1)
    perror("sendto()");
}
```

Figura 4.6 – Configuração do DE10-Nano como servidor UDP

4.3.2 Atualização dos parâmetros de Markov

Com a configuração da conexão UDP no servidor e no cliente é possível, através da utilização das funções `getCMD()`, `captura()` e `envia()`, realizar a transferência de dados entre o DE10-Nano e o Raspberry Pi. Dessa forma, a Figura 4.7 apresenta a função na qual o DE10-Nano recebe os parâmetros de Markov enviados pelo Raspberry Pi, armazenando-os nas variáveis `ysup_vec` (linha 10) e `yinfe_vec`(linha 16). Para facilitar a construção das matrizes de Hankel, esses dados são posteriormente transferidos para a variável `yIn`, conforme descrito nas linhas 20 e 21 do algoritmo.

```
1  void update_Markov(){ //Funcao que atualiza os parametros de Markov
2    if(num_devices == 0) { checkError(-1, "No devices"); }
3    outHu.reset(HnkSize); outHz.reset(HnkSize);
4    ysup_vec.clear();  yinfe_vec.clear();
5    yIn.reset(yaSize);
6    printf("Waiting for identification data...\n\n");
7    int acqsup = 0;
8    while(acqsup<n){ // parametros Markov da entrada de coltrole
9      in_data =  captura(s, recv_len, si_other, sendsize);
10     ysup_vec.push_back(in_data);
11     acqsup++;
12   }
13   int acqinf = 0;
14   while(acqinf<n){// parametros Markov da entrada de perturbacao
15     in_data =  captura(s, recv_len, si_other, sendsize);
16     yinfe_vec.push_back(in_data);
17     acqinf++;
18   }// Agrupa os parametros de Markov no vetor yIn
19   for (unsigned j = 0; j < n; j++){
20     yIn[j*l] =  ysup_vec[j];
21     yIn[j*l+1] =  yinfe_vec[j];
22   }
23 }
```

Figura 4.7 – Atualização dos parâmetros de Markov

4.3.3 Cálculo dos ganhos do sistema de controle

Conforme apresentado na Seção 3.4, alocando o polos do sistema de malha fechada, o cálculo do vetor de ganhos do controlador e do observador são implementados através da fórmula de Ackermann. Assim, a Figura 4.8 apresenta o código que realiza esse procedimento. Esta função foi implementa com o auxílio da biblioteca Eigen, cujas variáveis declaradas na cor cinza-escuro pertencem a estrutura desta biblioteca, assim como as funções inseridas com a cor azul. Os polos do sistema identificado são os autovalores da matriz $\mathbf{A}_d$, obtidos na linha 11 do algoritmo. No intervalo das linhas $13-16$ e $28-31$ são alocados, respectivamente, os polos do controlador e do observador, utilizando o fator de amortecimento `qsi` previamente

definido. Levando em conta que o sistema flexível esteja na forma modal e desacoplada, o projeto do observador e do controlador pode ser feito independentemente para cada modo. Por exemplo, considerando um sistema identificado de segunda ordem, as matrizes de controlabilidade e observabilidade das Equações 3.57 e 3.58, para cada modo de vibração, são calculadas no intervalo das linhas $18 - 23$.

```
1  //Fuancao que preojeta os ganhos do sistema de controle
2  MatrixXf poleAlocation(MatrixXf MA, MatrixXf MB, MatrixXf MC,
       MatrixXf MD, int k, double dt, float qsi)
3  {
4    Kcnovo = MatrixXf::Zero(1,k); Kenovo = MatrixXf::Zero(1,k);
5    Knovo = MatrixXf::Zero(2,k); MatrixXf MBu (k,1);
6    ctrlb = MatrixXf::Zero(k,k); obsrv = MatrixXf::Zero(k,k);
7    MatrixXf DelctrlA (k,k); MatrixXf DelobsvA (k,k);
8    VectorXcf AUTOVALOR(k); VectorXcf pole_ctrl(k);
9    VectorXcf pole_obsv(k); MatrixXf operador(1,k);
10
11   AUTOVALOR = MA.eigenvalues(); //Calcula os autovalores de Ad
12   float Wncomplexa = (abs(log(AUTOVALOR(0))))/dt;
13   complex<double> polea_ctrl(-qsi*Wncomplexa*dt, Wncomplexa*sqrt(1-(
         qsi*qsi))*dt);
14   complex<double> poleb_ctrl(-qsi*Wncomplexa*dt, -Wncomplexa*sqrt(1-(
         qsi*qsi))*dt);
15   pole_ctrl(0) = exp(polea_ctrl);
16   pole_ctrl(1) = exp(poleb_ctrl);
17   MBu << MB.block(0,0,k,1);
18   ctrlb.block(0,0,k,1) = MBu;
19   obsrv.block(0,0,1,k) = MC;
20   for (int i = 1; i < k; i++){
21     ctrlb.block(0,i,k,1) = (MA.pow(i))*MBu;
22     obsrv.block(i,0,1,k) = MC*(MA.pow(i));
23   }
24   float ca = real(pole_ctrl(0)*pole_ctrl(1));
25   float cb = (-1)*real(pole_ctrl(0) + pole_ctrl(1));
26   float cc = 1;
27   DelctrlA = cc*MA*MA + cb*MA + ca*MatrixXf::Identity(k, k);
28   complex<double> polea_obsv(-4*qsi*Wncomplexa*dt, 4*Wncomplexa*sqrt(
         1-(qsi*qsi))*dt);
29   complex<double> poleb_obsv(-4*qsi*Wncomplexa*dt, -4*Wncomplexa*sqrt
         (1-(qsi*qsi))*dt);
30   pole_obsv(0) = exp(polea_obsv); //polos discretizados
31   pole_obsv(1) = exp(poleb_obsv);
32   float oa = real(pole_obsv(0)*pole_obsv(1));
33   float ob = (-1)*real(pole_obsv(0) + pole_obsv(1));
34   float oc = 1;
35   DelobsvA = oc*MA*MA + ob*MA + oa*MatrixXf::Identity(k, k);
36   operador.row(0) << 0, 1;
37
38   // Ganhos do controlador e observador
39   Kcnovo = operador*(ctrlb.inverse())*DelctrlA;
40   Kenovo = DelobsvA*(obsrv.inverse())*operador.transpose();
41   Knovo.block(0,0,1,k) = Kcnovo;
42   Knovo.block(1,0,1,k) = Kenovo.transpose();
43   return Knovo;
44 }
```

Figura 4.8 – Alocação dos polos e cálculo dos ganhos do sistema de controle

Para ilustrar o cálculo dos ganhos do sistema de controle, considerando-o de ordem 2, os polos do controlador são dados por $p(1) = -\alpha + j\beta$ e $p(2) = -\alpha - j\beta$. Desa forma, os coeficientes $c_1 = p(1) * p(2)$ e $c_2 = p(1) + p(2)$ de $\phi_c(\mathbf{A}_d)$, presentes nas linhas 24 e 25, são obtidos de

$$\big(s - (\alpha + j\beta)\big)\big(s - (\alpha - j\beta)\big) = s^2 - \underbrace{(2\alpha)}_{p(1)+p(2)}\ s + \underbrace{\alpha^2 + \beta^2}_{p(1)*p(2)}. \tag{4.1}$$

Assim, as funções $\phi(\mathbf{A}_d)$ referentes ao controlador e observador são calculadas, respectivamente, no intervalo de linhas $24 - 27$ e $32 - 35$. Por fim, os ganhos $\mathbf{K}_g$ e $\mathbf{L}_g$ são calculados nas linhas 39 e 40 do algoritmo.

4.3.4 Cálculo da integridade estrutural

As funções do programa que realizam o cálculo da integridade estrutural também utilizam a biblioteca Eigen. A Figura 4.9 apresenta a função que realiza o cálculo das matrizes infinitas de observabilidade relativas ao modelo de referência e ao modelo de comparação, cujas dimensões dependem da variável `tam` préviamente definida. De posse das matrizes de observabilidade infinita pertencente a cada modelo, o próximo passo consiste em calcular a distância entre os subespaços de cada um. A Figura 4.10 exibe o código para o cálculo da distância entre os subespaços desses dois modelos. Dessa forma, a função `sendHealth` da Figura 4.4, baseia-se nas funções `ObsI` e `dSubs` para enviar o sinal de integridade ao Raspberry Pi.

```
1  // Funcao que calcula a matriz de observabilidade infinita
2  MatrixXf ObsI(MatrixXf A,MatrixXf C,MatrixXf AA,MatrixXf CC,int tam)
3  {
4          int k = A.cols();
5          int LIN = tamanho+1;
6
7          MatrixXf Ob(LIN,2*k);
8          Ob = MatrixXf::Zero(LIN,2*k);
9
10         Ob.block(0,0,1,k) = C;
11         Ob.block(0,2,1,k) = CC;
12         //Incrementa a matriz de acordo com a varivavel tam
13         for (int i = 1; i <= tamanho; i++){
14           Ob.block(i,0,1,k) =  C*(A.pow(i));
15           Ob.block(i,k,1,k) =  CC*(AA.pow(i));
16         }
17         return Ob;
18 }
```

Figura 4.9 – Função utilizada para o cálculo da matriz de observabilidade infinita

```
// Funcao que calcula a distancia entre os sobespacos
float dSubs(MatrixXf A,MatrixXf C,MatrixXf AA, MatrixXf CC, int tam)
{
  MatrixXf Obs; MatrixXf F; MatrixXf G;
  MatrixXf MA; MatrixXf MB; MatrixXf OB;

  Obs = calcula_matriz_obs(A, C,AA,CC, tamanho);
  F = Obs.block(0,0,Obs.rows(),2);
  G = Obs.block(0,2,Obs.rows(),2);

  FullPivLU<MatrixXf> lu_decompa(F);
  FullPivLU<MatrixXf> lu_decompb(G);

  int na = lu_decompa.rank();
  int nb = lu_decompb.rank();

  if(nb>=na){MA = F; MB = G;}
  else{MA = G; MB = F;}

  OB = MatrixXf::Zero(MA.rows(),2*MA.cols());
  OB.block(0,0,MA.rows(),MA.cols()) = MA;
  OB.block(0,MA.cols(),MB.rows(),MB.cols()) = MB;

  int p = MA.cols();
  int q = MB.cols();

  HouseholderQR<MatrixXf> qra(OB);
  MatrixXf L = qra.matrixQR().template  triangularView<Upper>();
  MatrixXf Ln = L.block(0,p,p+q,q);
  HouseholderQR<MatrixXf> qrb(Ln);

  MatrixXf S = qrb.matrixQR().template  triangularView<Upper>();
  MatrixXf Sf = S.block(0,0,q,S.cols());
  MatrixXf Lf = Ln.block(0,0,q,Ln.cols());
  MatrixXf MT = Lf*(Sf.inverse());

  JacobiSVD<MatrixXf> svd(MT);

  VectorXf sing =  svd.singularValues();
  VectorXf cosines_quad = sing.cwiseProduct(sing);
  float distancia = 1.0;

  for (int i = 0; i < max(na,nb); i++)
      distancia *=cosines_quad(i);

  float distance = sqrtf(log(1/distancia));
  return distance;
}
```

Figura 4.10 – Função utilizada para o cálculo da distância entre dois modelos

4.3.5 Interação com dispositivos OpenCL

Além dos códigos de comunicação com o Raspberry Pi, a programação C/C++ embarcada no processador ARM (*host*) precisa conter as funções para interação com o FPGA (dispositivo OpenCL acelerador). Essas funções devem executar, essencialmente, as tarefas ilustradas na Figura 4.11. Na configuração inicial, são criadas seis estruturas de dados utiliza-

das na aplicação OpenCL: plataformas, dispositivos, contextos, programas, *kernels* e filas de comando. Dessa forma, é possível realizar a transmissão de dados entre o ARM e o FPGA, executar os *kernels* no dispositivo acelerador e como última tarefa, liberar os recursos OpenCL utilizados na aplicação.

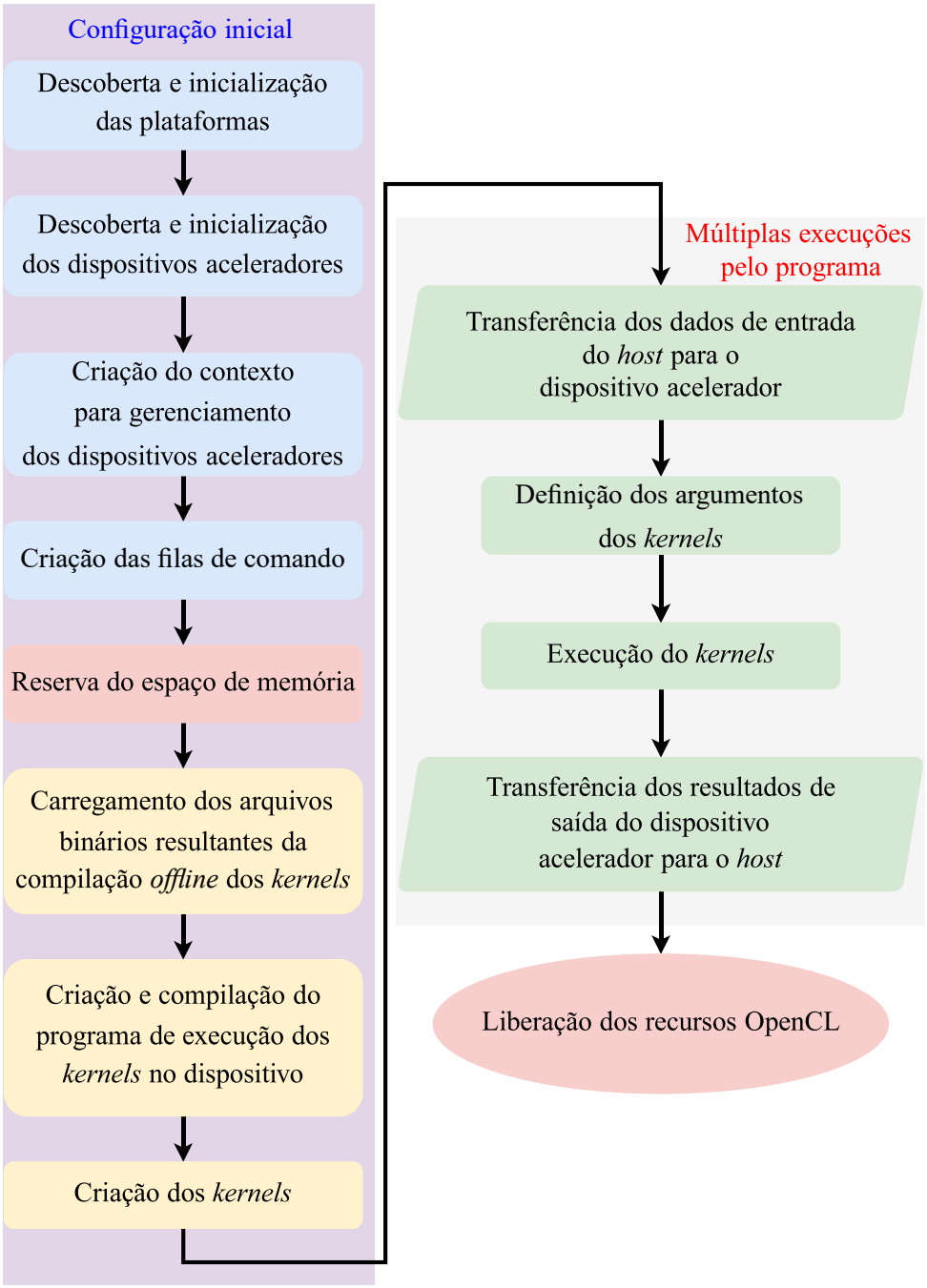


Figura 4.11 – Passos da implementação OpenCL na CPU ARM

A Figura 4.12 apresenta a função `init_opencl`, utilizada na configuração inicial do ambiente OpenCL. São empregadas algumas funções específicas tais como: `findPlatform`, `getDevices`, `clCreateContext`, `clCreateCommandQueue`, `clCreateBuffer` e

as funções `getBoardBinaryFile`, `createProgramFromBinary`, `clBuildProgram` e `clCreateKernel`, incluídas no SDK OpenCL.

```
// Funcao que inicia a configuracao OpenCL
bool init_opencl() {
   cl_int status; // Valor de retorno
   printf("Initializing OpenCL\n");
   if(!setCwdToExeDir()) {return false;}
// --> Descoberta e inicializacao das plataformas
   platform = findPlatform("Intel(R) FPGA");
   if(platform == NULL){
      printf("ERROR: Unable to find Intel FPGA OpenCL platform.\n");
      return false;}
// --> Descoberta e inicializacao dos dispositivos aceleradores
   scoped_array<cl_device_id> devices;
   devices.reset(getDevices(platform, CL_DEVICE_TYPE_ALL, &numDevices
      ));
   device = devices[0]; // Considera somente um dispositivo.
// --> Criacao do contexto para gerenciamento dos dispositivos.
   context = clCreateContext(NULL, 1, &device, NULL, NULL, &status);
   checkError(status, "Failed to create context");
// --> Criacao das filas de comando
   queue = clCreateCommandQueue(context, device,
      CL_QUEUE_PROFILING_ENABLE, &status);
   checkError(status, "Failed to create command Queue");
// --> Reserva do espaco de memoria.
   yBuf = clCreateBuffer(context, CL_MEM_READ_ONLY, yaSize * sizeof(
      float), NULL, &status);
   checkError(status, "Failed to create buffer for input ya");
   HzBuf = clCreateBuffer(context, CL_MEM_READ_WRITE, HnkSize *
      sizeof(float), NULL, &status);
   checkError(status, "Failed to create buffer for output Hzero");
   HuBuf = clCreateBuffer(context, CL_MEM_READ_WRITE, HnkSize *
      sizeof(float), NULL, &status);
   checkError(status, "Failed to create buffer for output Hum");
// --> Carregamento do arquivo binario do kernel
   std::string binary_file = getBoardBinaryFile("kernel_id", device);
   printf("Using AOCX: %s\n", binary_file.c_str());
// --> Criacao e compilacao do programa de execucao
   program = createProgramFromBinary(context, binary_file.c_str(), &
      device, 1);
   status = clBuildProgram(program, 0, NULL, "", NULL, NULL);
   checkError(status, "Failed to build program");
// --> Criacao do kernel
   const char *kernel_name = "hankel";
   kernel_Hankel = clCreateKernel(program, kernel_name, &status);
   checkError(status, "Failed to create kernel_Hankel");
   return true;
}
```

Figura 4.12 – Configuração inicial OpenCL no programa do ARM

Vale ressaltar que a Figura 4.12 apresenta a configuração inicial para implementar o *kernel* que calcula as matrizes de Hankel, omitindo os atributos dos *kernels* que realizam as outras funções do sistema embarcado. Além disso, não é necessário executar essas tarefas estritamente na ordem em que elas estão distribuídas nas Figuras 4.11 e 4.12. Entretanto, algumas tarefas possuem dependência de outras. Por exemplo, a reserva do espaço de memória pode ser

realizada antes da criação das filas de comando que, por sua vez, deve ser executada somente após a criação do contexto. Para enfileirar, no programa do *host*, o comando de execução do *kernel* que calcula as matrizes de Hankel, a função `clenqueueNDRangeKernel`, definida no SDK OpenCL, deve ser utilizada, conforme apresentado na linha 21 da Figura 4.13.

```
1  // Funcao que o ARM (host) utiliza para requisitar que o FPGA (
       acelerador) calcule as matrizes de Hankel
2  void funHankel(){
3     cl_int status;    // Valor de retorno
4     cl_event Wev[1]; // Eventos de escrita
5     cl_event ev[1];  // Eventos de execucao
6     cl_event Rev[2]; // Eventos de leitura
7  // --> Transferencia dos dados do host para a memoria global
8     status = clEnqueueWriteBuffer(queue, yBuf, CL_FALSE, 0, MrkvSize *
          sizeof(float), yIn, 0, NULL, &W_ev[0]);
9  // --> Definicao dos argumentos do kernel
10    status = clSetKernelArg(kernel_Hankel, 0, sizeof(cl_mem), &yBuf);
11    status = clSetKernelArg(kernel_Hankel, 1, sizeof(cl_mem), &HzBuf);
12    status = clSetKernelArg(kernel_Hankel, 2, sizeof(cl_mem), &HuBuf);
13    status = clSetKernelArg(kernel_Hankel, 3, sizeof(cl_int), &l);
14    status = clSetKernelArg(kernel_Hankel, 4, sizeof(cl_int), &Q_era);
15    status = clSetKernelArg(kernel_Hankel, 5, sizeof(cl_int), &P_era);
16    //Define o tamanho do work-size global
17    const size_t GWS[2] = {nrows, ncols};
18 // --> Executa o kernel no formato NDRange
19    status = clEnqueueNDRangeKernel(queue, kernel_Hankel, 2, NULL, GWS
          , NULL, 1, &Wev[0], &ev[0]);
20    // Espera a flag de retorno do evento de execucao ev
21    clWaitForEvents(numDevices, ev);
22 // --> Executa o kernel no formato NDRange
23    status = clEnqueueReadBuffer(queue, HzBuf, CL_FALSE, 0, HnkSize *
          sizeof(float), outHz, 0, NULL, &Rev[0]);
24    status = clEnqueueReadBuffer(queue, HuBuf, CL_FALSE, 0, HnkSize *
          sizeof(float), outHu, 0, NULL, &Rev[1]);
25 // --> Transferencia dos dados da memoria global para o host
26    clWaitForEvents(numDevices, Rev);
27 }
```

Figura 4.13 – Execução do *kernel* comandada pelo *host* para calcular $\mathbf{H}[1]_{p,q}$ e $\mathbf{H}[2]_{p,q}$

Antes disso, é necessário carregar os parâmetros de Markov `yIn` na memória global `yBuf` através da função `clenqueueWriteBuffer`. Em seguida, é preciso atribuir os argumentos do *kernel* (`clsetKernelArg`) e definir o tamanho global do espaço NDRange (`GWS`). As operações de escrita na memória global e execução do *kernel* são controladas pelos eventos `ev` e `Wev`, ambos definidos no *host*. O valor `1` definido no penúltimo argumento da função `clEnqueueNDRangeKernel` indica que, antes de executar o *kernel* e gerar o objeto de evento `ev`, é necessário esperar que ocorra a escrita na memória global (evento `Wev`). Por fim, a Figura 4.14 apresenta a tarefa para liberação dos recursos OpenCL, mediante a utilização de funções específicas.

```
1  void cleanup() {
2  // --> Liberacao dos recursos OpenCL
3     if(kernel_Hankel) clReleaseKernel(kernel_Hankel);
4     if(queue) clReleaseCommandQueue(queue);
5     if(yBuf) clReleaseMemObject(yBuf);
6     if(HzBuf) clReleaseMemObject(HzBuf);
7     if(HuBuf) clReleaseMemObject(HuBuf);
8     if(program) clReleaseProgram(program);
9     if(context) clReleaseContext(context);
10 }
```

Figura 4.14 – Liberação dos recursos OpenCL

4.4 Programação dos Kernels embarcados no FPGA do DE10-Nano

No FPGA é possivel implementar dois tipos de *kernel* em OpenCL: *NDRange* e *Single-Work-Item*. Os *kernels NDRange* são constituídos de múltiplos *work-itens* que implementam o paralelismo de dados, compartilhando esses dados entre os *work-itens* através da memória local. Os *kernels Single-Work-Item* são constituídos de um único *work-item*, que compartilha dados entre várias iterações de *loop*, utilizando a memória privada. A Figura 4.15 ilustra o exemplo de código para um *kernel NDRange* escrito na linguagem OpenCL. A programação deve ser iniciada pelo qualificador `__kernel`. Esta função realiza a soma de dois vetores `inputA` e `inputB`, armazenado o resultado no vetor `outputC`. Essas variáveis são alocadas na memória global, conforme descrito pelo qualificador `__global`. O *kernel* acessa as posições desses vetores com base no espaço de índice `j`.

```
1          // Kernel generico de soma de vetores
2          __kernel void vecAdd(__global float* restrict outputC,
3                               __global float* restrict inputA,
4                               __global float* restrict inputB)
5          {  //posicao global no indice ND-Range
6              int j = get_global_id(0);
7              outputC[j] = inputA[j] + inputB[j];
8          }
```

Figura 4.15 – Kernel OpenCL para soma de 2 vetores

4.4.1 Paralelismo de dados no FPGA

Conforme apresentado na Figura 4.15, a programação do *kernel* em OpenCL para esse modelo é semelhante a desenvolvida para GPUs (Unidades de Processamento Gráfico). Entretanto, ao contrário das GPUs, nas quais o paralelismo de dados se dá pela execução simultânea de múltiplos *work-itens* sob diferentes dados, nos FPGAs, o compilador gera por padrão

uma única unidade de computação implementada em *pipeline*, no qual os *work-itens* são acionados de forma ordenada a cada ciclo de *clock*. O compilador executa automaticamente também a ordenação em *pipeline* dos *work-groups*, permitindo que eles sejam executados na mesma unidade de computação. O intervalo de iniciação é o número de ciclos de *clock* necessários antes de uma nova iteração vir a ser iniciada no *pipeline*. Na eficiencie máxima, o intervalo de iniciação é igual a um, i.e, cada iteração no *pipeline* leva um ciclo de *clock* para ser executada. Nesta forma de paralelismo, o intervalo de iniciação entre os *work-itens* é ajustado em tempo de execução pelo escalonador de hardware, que o compilador implementa no FPGA para minimizar as paralisações do *pipeline* e maximizar a sua eficiência. Essa configuração é semelhante à arquitetura MIMD (*Multiple-Instructions Multiple-Data*).

Para alcançar rendimento maior, várias unidades de computação podem ser utilizadas através de um atributo fornecido pelo SDK. O compilador implementa essas unidades como um *pipeline* exclusivo e, então, escalonador de hardware separa automaticamente os *work-groups* para serem executados de forma simultânea em cada unidade de computação. Além disso, com a vetorização do kernel é possível obter uma taxa de transferência maior, na qual múltiplos *work-items* podem ser executados em modo SIMD (*Single-Instructions Multiple-Data*). Uma vez que as operações de leitura e escrita acessam locais consecutivos na memória global, o compilador *offline* reúne essas operações de carregamento em um único processo vetorial, reduzindo o número de acessos à memória. Entretanto, esta otimização é limitada, pois o custo de se realizar mais operações simultâneas no modo SIMD é a necessidade de mais recursos como o aumento de dados por ciclo de *clock* e largura de banda de memória, por exemplo (Waidyasooriya *et al.*, 2018). Quando os *work-itens* de um *kernel* são explicitamente divididos em *work-groups* e não há dependência de dados, este modelo de programação pode ser utilizado como uma arquitetura SIMD. No caso em que o acesso à memória é dependente de dados ou desconhecido na compilação, o compilador *offline* não aglutina os acessos à ela. Além disso, caso exista dependência entre os dados em um *work-group*, o tempo total de execução do *kernel* no FPGA é ligeiramente maior comparado ao mesma implementação em GPU, pois é necessário esperar até que todos os *work-itens* em *pipeline* atinjam a barreira de sincronização.

4.4.2 Paralelismo de tarefas no FPGA

Nas situações em que existir dependência de dados ou não ser possível separar facilmente as operações em *work-itens*, é recomendado utilizar esse modelo de programação no

qual todo o *kernel* é executado por um único *work-item*. A programação em OpenCL do *kernel* a ser implementação é similar a um programa em linguagem C e diversas iterações de um *loop* são calculadas em diferentes estágios de *pipeline* e separados pelo compilador, o que caracteriza a paralelização em *pipeline* no nível de iteração. Dessa forma, o *work-item* compartilha dados entre várias iterações de *loop*, usando a memória privada ao invés de implementar um escalonador de hardware no FPGA. O escalonamento de iteração é estático e o intervalo de inicialização é determinado pelo compilador, que identifica automaticamente as dependências entre as iterações e minimiza o intervalo de iniciação. Além disso, a vetorização do *kernel* neste modelo de programação pode ser alcançada mediante o uso do comando específico denominado `pragma unroll`, o qual separa os *loops*.

4.4.3 Kernel de paralelização do cálculo da matriz de Hankel

Conforme descrito no capítulo anterior, um passo fundamental para a identificação do sistemas por meio da técnica ERA é a construção da matriz de Hankel. Tendo em vista que não há dependência de dados para atribuir os valores das matrizes $\mathbf{H}[1]_{p,q}$ e $\mathbf{H}[2]_{p,q}$, o *kernel* OpenCL pode ser implementado por um modelo de programação NDRange, como apresentado na Figura 4.16. Os *work-itens* acessam a memória global externa, que contém os parâmetros de Markov e monta as matrizes de Hankel conforme os identificadores de posição `j` e `i` no espaço NDRange e pelos parâmetros de entrada `l`, `q` e `p`. O tempo de execução desse *kernel*, no acelerador FPGA, foi menor comparado ao tempo de execução da função que calcula as matrizes de Hankel implementada em C, dento do programa principal (no *host* ARM). Entretanto, vale salientar que o tempo necessário para executar esta função, tanto com a implementação em OpenCL quanto na forma serial, é pequeno comparado ao tempo total do cálculo da SVD.

```
1 // Kernel utilizado para o calculo das matrizes de Hankel
2 __kernel void hankel(__global float* restrict Markov,
3                       __global float* restrict H_zero,
4                       __global float* restrict H_one,
5                       int l,int q, int p){
6   int j = get_global_id(0);  //posicao global no indice ND-Range
7   int i = get_global_id(1);
8   H_zero[i + j*q] = Markov[i*l + j];
9   H_one[j + i*p*l] = Markov[i*l + j + l];}
```

Figura 4.16 – Kernel utilizado para acelerar o cálculo da matriz de Hankel

4.4.4 Kernels de paralelização do método de redução para a forma Bidiagonal

Conforme exposto no capítulo 2, a SVD via iterações QR é precedida pela redução da matriz de entrada para a forma bidiagonal e, então, o algoritmo 2 que realiza este procedimento calcula explicitamente, a cada iteração i, as matrizes de Householder $\mathbf{P}_i$ e $\mathbf{Q}_i$. Dessa forma, ele bidiagonaliza $\mathbf{A}$ e, simultaneamente, a cada passo, atualiza $\mathbf{P}$ e $\mathbf{Q}$. Na implementação em OpenCL no FPGA, inicialmente, a cada iteração i, calcula-se o respectivo par de valores da matriz que repesentam o elemento da diagonal principal e superior. As colunas e linhas de $\mathbf{A}$, que representam $\mathbf{v}$, são substituídas pelos vetores de Householder $\mathbf{u}$ correspondentes e todos elementos da submatriz inferior direita $\mathbf{A}_{(m-i)\times(n-i)}$ são atualizados (devido à reflexão) para serem utilizados no cálculo da próxima iteração. Ao fim desse processo, $\mathbf{A}$ resulta na matriz $\mathbf{A}'$ e a matriz $\mathbf{B}$ é obtida. O próximo passo consiste em calcular iterativamente as matrizes $\mathbf{P}$ e $\mathbf{Q}$ por meio dos valores de $\mathbf{A}'$ e $\mathbf{B}$. Com essa abordagem, não é preciso calcular o produto vetorial $\mathbf{u}\mathbf{u}^T$ e, consequentemente, não é preciso calcular a matriz de Householder, explicitamente. Ao invés disso, $\mathbf{v}'$ é calculado conforme a Equação 2.4, que pode ser reescrita como

$$\mathbf{v}' = \mathbf{v} - \underbrace{\left(\frac{2}{\mathbf{u}^T\mathbf{u}}\right)}_{1/\eta}\underbrace{\left(\mathbf{u}^T\mathbf{x}\right)}_{s_i}\mathbf{u} \tag{4.2}$$

$$\mathbf{v}' = \mathbf{v} - \beta\mathbf{u}, \quad \text{em que} \quad \beta = s_i/\eta. \tag{4.3}$$

Tendo em vista que a cada operação linha e coluna correspondentes a uma iteração, tem-se

$$u_0 = v_0 + \|\mathbf{V}\|, \tag{4.4}$$

em que o produto escalar $\mathbf{u}^T\mathbf{u}$ também não necessita ser calculado, pois

$$\begin{aligned}
\eta = \frac{\mathbf{u}^T\mathbf{u}}{2} &= \frac{\left(v_0 + \|\mathbf{V}\|\right)^2 + v_1^2 + \cdots + v_n^2}{2} \\
&= \frac{v_0^2 + 2v_0\|\mathbf{V}\| + v_0^2 + v_1^2 + \cdots + v_n^2 v_1^2 + \cdots + v_n^2}{2} \\
&= \frac{2\|\mathbf{V}\|^2 + 2v_0\|\mathbf{V}\|}{2} \\
&= \underbrace{\|\mathbf{V}\|}_{s}\left(v_0 + \|\mathbf{V}\|\right)
\end{aligned}$$

e, portanto,

$$u_0 = v_0 + s * sign(v_0), \quad \eta = |s * u_0|, \quad q_i = -s * sign(u_0) \quad \text{e} \quad e_i = -s * sign(u_0).$$

Na prática são necessários quatro passos consecutivos em uma única iteração para calcular um par de valores de $\mathbf{B}$ e os respectivos vetores $\mathbf{u}$. O par de valores corresponde a

um termo específico da diagonal principal q e outro da superior e. Vale ressaltar que na última iteração somente o valor da diagonal principal será calculado. Especificamente, caso $\mathbf{A}$ possua $m = 5$ linhas e $n = 3$ colunas, esta iteração é representada na Figura 4.17. Como nos passos 1 e 3 existem dependência de dados no *kernel* para o cálculo de s, é recomendado utilizar o *kernel* sob o paralelismo de tarefas como mostrado na Figura 4.18.

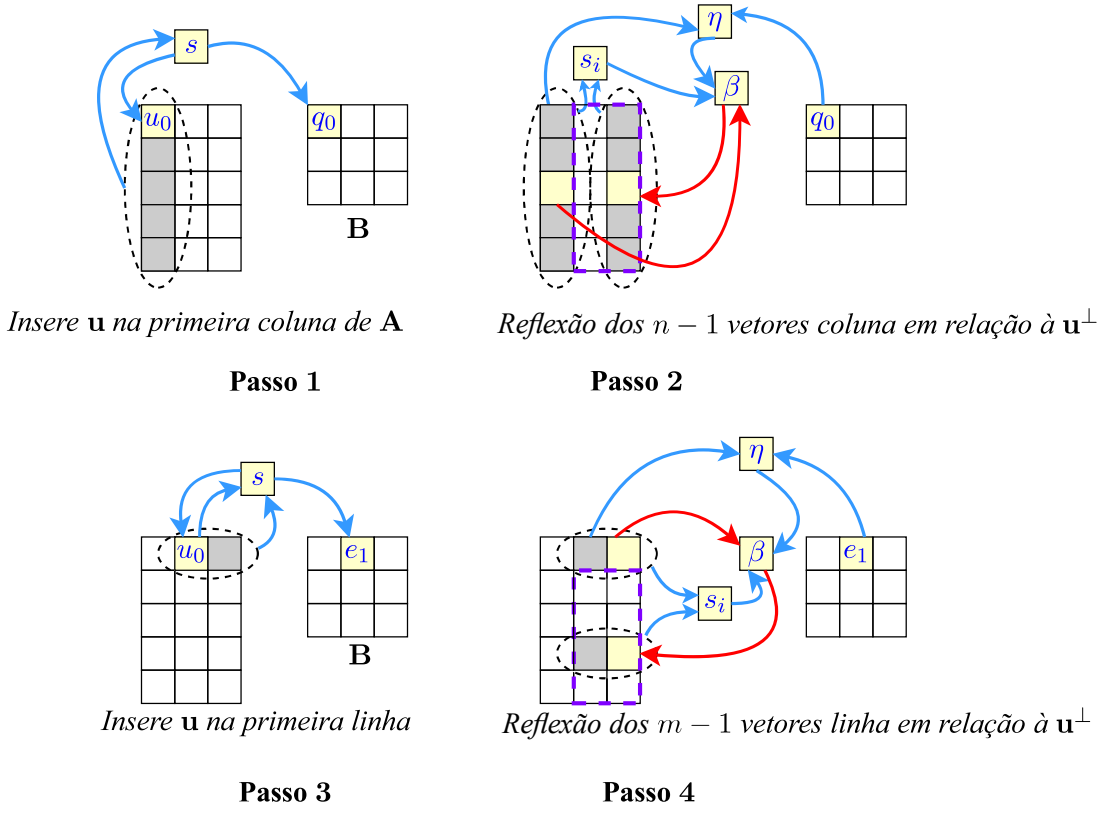


Figura 4.17 – Pré-etapa do processo de bidiagonalização em uma iteração

```
1  // Kernel utilizado para o calculo dos vetores de householder e dos
       valores da matriz bidiagonal
2  __attribute__((max_global_work_dim(0)))
3  __kernel void get_s(__global float* restrict U,__global float*
       restrict diagonal, int iplus, int SIZE,int posDiag, int mul){
4       float s, norm_squared, value, signal;
5       norm_squared = 0.0;
6       for (int j = 0;  j < SIZE; j++){ // Calcula s = |U|
7           value = U[posDiag + mul*j];
8           norm_squared += value*value;}
9       s = sqrt(norm_squared);
10      if(s > 0.0){
11          signal = ( U[posDiag] < 0.0 ) ? -s : s;
12          U[posDiag] += signal;}// Soma s na primeira posicao do vetor
13      signal = ( U[posDiag] < 0.0 ) ? -s : s;
14      diagonal[iplus] = -signal; //Calcula q[iplus] ou e[iplus]
15 }
```

Figura 4.18 – Passos 1 e 3 referentes a primeira etapa de bidiagonalização

Um obstáculo para a otimização desse *kernel* é a dificuldade em realizar a sua vetorização manual, que aceleraria os acessos à memoria global. O valor do atributo `posDiag` presente na linha 10 da Figura 4.18 é desconhecido pelo AOC em tempo de compilação e, assim, ele não aglutina os acessos à memória. Nos passos 2 e 4 também existe a dependência de dados no *kernel* para o cálculo de s_i e, portanto, a atualização das linhas e colunas da sub-matriz de $\mathbf{A}'$ pode ser realizada utilizando múltiplos *work-itens* executados em pipeline, sendo possível escrever o programa em trechos que representam o paralelismo de dados e tarefas. Dessa forma, o identificador `j` da Figura 4.19 refere-se a posição no índice NDRange.

```
1  // Kernel utilizado para aplicar as reflexoes de householder na
       matriz de entrada
2  __kernel void reflectHh(__global float* restrict diagonal,
3                          __global float* restrict U,
4                          __global float* restrict X,
5                          int SIZE_B, int SIZE_C, int iplus, int pos,
6                          int offsetA, int offsetC, int mulA,
7                          int offsetB, int offsetD, int mulB, int mulC)
8  {
9     int j = get_global_id(0); //posicao global
10    float half_norm_squared, si, s, beta;
11
12    s = fabs(diagonal[iplus]); //Obtem s da matriz Bidiagonal
13    half_norm_squared =  fabs(s*U[pos]); //Calcula eta
14
15    if (s > 0.0){
16       si = 0.0;
17
18       // Calcula si = u^T*X
19       for (int k = 0; k < SIZE_B; k++)
20          si += U[offsetA + mulA*k] * X[offsetB + mulB*k + mulC*j];
21
22      //Calcula beta
23       beta = si/half_norm_squared;
24
25       // Atualiza os vetores-linha ou colunas
26       for (int k = 0; k < SIZE_C; k++)
27          X[offsetD + mulB*k + mulC*j] -= beta * U[offsetC + mulA*k];
28    }
29 }
```

Figura 4.19 – Passos 2 e 4 referentes a primeira etapa de bidiagonalização

Logo após obter $\mathbf{A}'$ e $\mathbf{B}$ simultâneamente (processo de bidiagonalização), as matrizes $\mathbf{Q}$ e $\mathbf{P}$ são calculadas mediante sucessivas transformações constituídas pelos valores bidiagonais de $\mathbf{B}$ e pelos vetores $\mathbf{u}$ de $\mathbf{A}'$. Considerando novamente $m = 5$ e $n = 3$, o procedimento para obter $\mathbf{Q}$ e $\mathbf{P}$ é apresentado pelas Figuras 4.20 e 4.21, respectivamente.

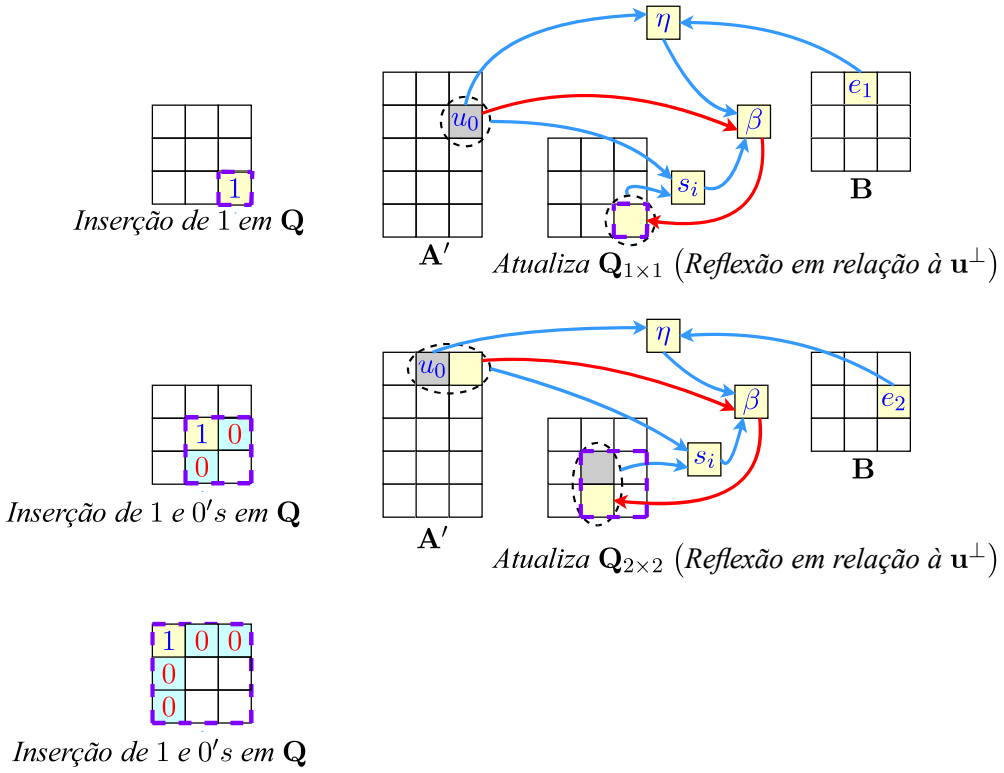


Figura 4.20 – Acumulações de **Q**

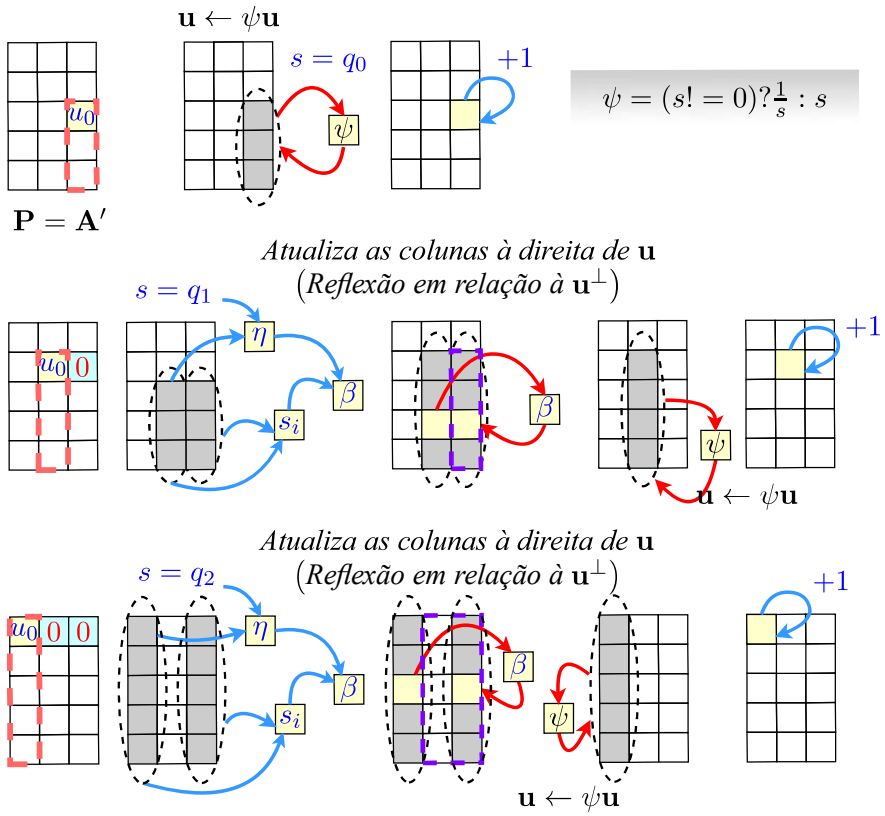


Figura 4.21 – Acumulações de **P**

Os kernels adicionais que completam o cálculo de $\mathbf{Q}$ e $\mathbf{P}$ compreendem a inserção de 1, $0's$ e a multiplicação de uma coluna pelo fator ψ. Elas são funções em que não há dependência de dados e podem ser facilmente divididas em *work-groups*. Dessa maneira, são implementadas como *kernels* NDRange conforme a Figura 4.22.

```
1  // Kernel utilizado para operacoes adicionais que realizam as
       acumulacoes de Q e P
2  __kernel void update(__global float* restrict U,
3                       __global float* restrict diagonal,
4                       int sel, int offsetA,
5                       int offsetB, float fator,
6                       int iplus, int ncols)
7  {
8      int j = get_global_id(0); //posicao global
9
10     float svar, mul;
11
12     switch (sel){
13         case 1: //Insere zeros em Q
14             U[offsetA + j] = 0.0;
15             U[offsetB + j*ncols] = 0.0;
16         break;
17
18         case 2: //Soma um em P
19             U[offsetA] += 1.0;
20         break;
21
22         case 3://Insere zeros em P
23             U[offsetA + j] = fator;
24         break;
25
26         case 4: //Multiplica por phi os vetores-coluna de P
27             svar = diagonal[iplus];
28             mul = (svar != 0) ? 1/svar : svar;
29             U[offsetA + j*ncols] *= mul;
30         break;
31
32         default: break;
33     }
34 }
```

Figura 4.22 – Kernels adicionais para atualização de $\mathbf{Q}$ e $\mathbf{P}$

Após o cálculo das matrizes $\mathbf{P}$, $\mathbf{B}$ e $\mathbf{Q}$ no FPGA, é necessário transferi-las para o *host* utilizando a função de leitura fornecida pela API do SDK OpenCL. De posse dessas matrizes, a próxima etapa consiste em diagonalizar $\mathbf{B}$ iterativamente por meio das transformações ortogonais $\mathbf{G}$ e $\mathbf{H}$, as quais concomitantemente multiplicam $\mathbf{P}$ e $\mathbf{Q}$, respectivamente. Esse processo é realizado no *host*, com o auxílio da biblioteca Eigen para operações vetoriais, sendo obtidas as matrizes $\mathbf{U}$, $\mathbf{S}$ e $\mathbf{V}$ da decomposição SVD de $\mathbf{A}$.

4.4.5 Kernels de paralelização do método Hestenes-Jacobi

Para a implementação deste método, três *kernels* principais foram utilizados, baseados no Algoritmo 4 do capítulo 2. Na linha 7 do algoritmo, as colunas p e q da matriz de entrada, nas quais serão aplicadas as transformações de Jacobi, são obtidas a partir do vetor $\mathbf{W}$. Dessa forma, o vetor $\mathbf{W}$ é atualizado (linha 13) para que se realize todas as combinações da varredura. Serão aplicadas α transformações em paralelo a cada iteração. Como a função de atualização de $\mathbf{W}$ não possui dependência de dados, o *kernel* correspondente pode ser programado como NDRange conforme a Figura 4.23. A regra para mudança no conteúdo do vetor $\mathbf{W}$ é realizada de acordo com a Figura 2.5.

```
1  //  Kernel utilizado para atualizar os elementos do vetor, cujos
       valores de cada posicao corresponde a um vetor-coluna da matriz de
        entrada. Uma varredura consiste no numero de execucoes deste
       kernel para realizar todas as combinacoes possiveis entre duas
       colunas
2
3  __kernel void RefreshW(__global int* restrict W, int ncols)
4  {
5    int j = get_global_id(0); //posicao global
6
7    // Mudanca nos valores de W
8    if(W[j] == 2 || W[j] == ncols-1)
9      W[j] -= 1;
10   else if(W[j]%2 != 0)
11     W[j] += 2;
12   else if(W[j] > 0)
13     W[j] -= 2;
14 }
```

Figura 4.23 – Atualização do vetor que seleciona as colunas

O identificador global `j` determina as colunas de $\mathbf{H}[1]_{p,q}$, que serão selecionadas a cada iteração e esse atributo possui tamanho global $\alpha = p$. O segundo *kernel* é representado na Figura 4.24. Ele se refere as linhas 8 a 11 do algoritmo 4. Este *kernel* realiza a aplicação das rotações na matriz de entrada mediante o cálculo dos parâmetros c e s da matriz de Jacobi.

As normas n_p, n_q e a perpendicularidade entre os vetores cov são representadas, respectivamente, por `alpha`, `beta` e `gamma`. Essas variáveis são utilizadas no cálculo de c e s, os quais são aplicados nas rotações de Jacobi à direita das matrizes $\mathbf{V}$ e $\mathbf{B}$. Em seguida, a variável `gamma` é armazenada na matriz `E` para ser possível verificar (em outro *kernel*) o erro máximo obtido em uma varredura. As rotações de Jacobi possuem independência de dados e o *kernel* pode ser escrito como sendo NDRange.

```
1  //  Kernel utilizado para aplicar as rotacoes de Jacobi
2  __kernel void Jacobi(__global float* restrict R, __global float*
       restrict V, __global float* restrict E, __global int* restrict D,
       int passo, int nrows, int ncols){
3    int j = get_global_id(0);
4    int p, q;
5    float alpha, beta, gamma, c, s, e, t, x, y, w, z;
6    p = D[2*j]; q = D[2*j+1]; // seleciona as colunas
7    alpha = 0; beta = 0; gamma = 0;
8    for (int b = 0; b < ncols; b++){
9      alpha += R[b*ncols + p] * R[b*ncols + p];
10     beta +=  R[b*ncols + q] * R[b*ncols + q];
11     gamma += R[b*ncols + p] * R[b*ncols + q];}
12   e = (beta - alpha)/ (2*gamma);
13   t = sign(e)/(fabs(e) + sqrt(1 + e*e));
14   c = 1/(sqrt(1 + t*t));
15   s = t*c;
16   for (int k= 0; k < ncols; k++){// Aplica as rotacoes em V
17     x = V[k*nrows + p]; y = V[k*nrows + q];
18     V[p*nrows + k] =  c * x - s * y;
19     V[q*nrows + k] =  s * x + c * y;}
20   for (int i= 0; i < nrows; i++){// Aplica as rotacoes em R
21     w = R[i*ncols + p]; z = R[i*ncols + q];
22     R[i*ncols + p] =  c * w - s * z;
23     R[i*ncols + q] =  s * w + c * z;}
24   E[(passo-1)*(nrows/2) + j] = gamma;//Armazena os erros na matriz E
25 }
```

Figura 4.24 – Aplicação das rotações de Jacobi

Por fim, o terceiro *kernel*, apresentado na Figura 4.25, é referente as linhas 17 a 20 do Algoritmo 4. Ele calcula os valores singulares Σ e a matriz $\mathbf{U}$. No cálculo de Σ (linhas 16 a 22 do Algoritmo 4) não é possível realizar a vetorização manual, pois os valor de `ncols` é desconhecido pelo compilador. Dessa forma, não é possível executar as operações em barramentos SIMD diferentes. No cálculo de $\mathbf{U}$ há independência entre os dados, assim, o kernel também é escrito como NDRange.

```
1  // Kernel que obtem a base ortogona U e os valores singulares da SVD
2  __attribute__((reqd_work_group_size(64,64,1)))
3  __kernel void get_Matrices(__global float* restrict C, __global float
       * restrict A, __global float* restrict B, int sel, int nrows, int
       ncols){
4      int j = get_global_id(0); int i = get_global_id(1);
5      float value; int r;
6      switch (sel){
7          case 1:  // Valores singulares
8              value = 0;
9              for (r = 0; r < nrows; r++){value += A[r*ncols + j];}
10             C[j] = sqrt(value); break;
11         case 2:  // Base ortogonal U
12             C[j*ncols + i] = A[j*ncols + i]/B[j]; break;
13         default: break;}
14 }
```

Figura 4.25 – Cálculo de Σ e $\mathbf{U}$

4.5 Programação do Raspberry Pi

O Raspberry Pi comanda a aplicação do sinal de excitação nas entradas de perturbação e controle da estrutura, bem como a aquisição dos seus sinais de vibração. A Figura 4.26 apresenta as configurações iniciais do programa embarcado, que consiste em definir os pinos de entrada e saída do dispositivo e configurar os parâmetros do *socket* UDP. Além disso, são mostradas as funções de leitura do sinal de vibração digital e sua conversão D/A, `ReadChannel()` e `ConvertVolts()`, e as funções de comunicação UDP com o servidor DE10-Nano, `envio()` e `requisicao()`.

```
HOST = '192.168.0.102' # Endereco do servidor
PORT = 5000            # Porta que de acesso ao servidor
udp = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
dest = (HOST, PORT)
# Ativa o barramento SPI
spi = spidev.SpiDev()
spi.open(0,0)
spi.max_speed_hz=1000000
# Define o canal do sensor
PZT_channel = 0
# Define os pinos do Motor Inferior
pwmInf = 38
dirInf = 12
# Define os pinos do Motor Superior
pwmSup = 40
dirSup = 11
# Configuracao dos pinos dos motores:
GPIO.setmode(GPIO.BOARD)
GPIO.setup(dirInf, GPIO.OUT) # Direcao configurada como saida
GPIO.setup(pwmInf, GPIO.OUT) # PWM configurada como saida
pwm_inf = GPIO.PWM(pwmInf, 500)  # Configura PWM em 500Hz
GPIO.setup(dirSup, GPIO.OUT) # Direcao configurada como saida
GPIO.setup(pwmSup, GPIO.OUT) # PWM configurada como saida
pwm_sup = GPIO.PWM(pwmSup, 500)  # Configura PWM em 500Hz
# Adquire o sinal de vibracao
def ReadChannel(channel):
  adc = spi.xfer2([1,(8+channel)<<4,0])
  data = ((adc[1]&3) << 8) + adc[2]
  return data
# Converte o sinal
def ConvertVolts(data,places):
  Volts = (data * 3.3) / float(1023)
  volts = round(volts,places)
  return volts
# Envia dados ao servidor
def envio(sens):
  udp.sendto(str.encode(str(sens)), dest)
  return()
# Recebe dados do servidor
def requisicao():
  msg, serv = udp.recvfrom(1024)
  control=float(msg)
  return (control)
```

Figura 4.26 – Configurações iniciais do Raspberry Pi

Com o intuito de obter os parâmetros de Markov, é preciso medir a vibração estrutural em resposta aos sinais de identificação aplicados nas entradas de perturbação $\mathbf{w}$ e controle $\mathbf{u}$. A função `excsup`, ilustrada na Figura 4.27, aciona a entrada de controle $\mathbf{u}$, medindo a vibração estrutural $\mathbf{y}_u$, após a aplicação um filtro de média móvel para eliminar o ruído do sistema eletrônico. De maneira similar a função `excsup()`, existe também no programa do Raspberry Pi a função `excinfe()`, que adquire a vibração estrutural $\mathbf{y}_w$ em resposta ao sinal de identificação plicado na entrada $\mathbf{w}$. Dessa forma, é possível calcular os parâmetros de Markov e enviá-los ao servidor DE10-Nano, conforme mostrado na Figura 4.28.

```
1
2  def excsup(): # Aciona motor superior a adquire sinal de vibracao
3    i=0
4    while (i < npt):
5      if(uN[i] > 15.0): # Limita o sinal aplicado em |15|V
6        pwm_sup.ChangeDutyCycle(100)
7      elif(uN[i] < -15.0):
8        pwm_sup.ChangeDutyCycle(100)
9      else:
10       pwm_sup.ChangeDutyCycle( (abs(uN[i])/15)*100 )
11     if(dir_uN[i]== 1):
12       GPIO.output(dirSup, GPIO.HIGH)
13     else:
14       GPIO.output(dirSup, GPIO.LOW)
15     pzt_level = ReadChannel(PZT_channel)
16     pzt_volts = ConvertVolts(pzt_level,4)
17     y = pzt_volts-1.83
18     yu[i]=y
19     if(i> TAM-1):  # Media movel do sinal
20       yu_movel[i] = (np.sum(yu[i-TAM:i]))/(TAM)
21     else:
22       yu_movel[i] = (np.sum(yu[0:i]))/(i+1)
23     time.sleep(dt)
24     i+=1
```

Figura 4.27 – Excitação para identificação estrutural

```
1  def sendMarkov(): # Funcao para enviar os parametros de Markov
2    media_yu = np.mean(yw_movel)
3    yu_movel= yu_movel - media_yu     # Retira a media do sinal
4    media_yw = np.mean(yw_movel)
5    yw_movel = yw_movel - media_yw
6    Hyu, fyu = get_FRF(uN, yu_movel,t)# Calcula as FRF
7    Hyw, fyw = get_FRF(wN, yw_movel,t)
8    Mu = fun.impulse_resp(Hyu)        # Calcula os parametros de Markov
9    Mw = fun.impulse_resp(Hyw)
10   for i in range(0,n):              # Envia os parametros via UDP
11     envio(round(float(Mu[i]),8))
12   for i in range(0,n):
13     envio(round(float(Mw[i]),8))
14   udp.close()
```

Figura 4.28 – Envio Markov via UDP

Os parâmetros da resposta ao impulso são calculados através da aplicação da transformada inversa de Fourier sobre as FRFs obtidas experimentalmente, de acordo com as linhas 8 e 9 da Figura 4.28. O código das funções `get_FRF` e `impulse_resp` é apresentado na Figura 4.29. As FRFs experimentais são obtidas pelo estimador H_1 (Pintelon; Schoukens, 2012).

```
1  import numpy as np
2  # Funcao que calcula as FRFs experimentais
3  def get_FRF(entrada, saida, tempo):
4    dt = tempo[2]-tempo[1]
5    n = len(tempo)       # Numero de pontos de um periodo
6    npt = len(entrada)   # Numero total de pontos
7    N = int(npt/n)       # Numero de periodos
8    df = 1/(n*dt)        # Resolucao da frequencia
9    f = np.linspace(0, n*df, n) # vetor de frequancia
10   f = f[0:int(n/2)]           # frequencia de Nyquist
11
12   # cria a estrutura complexa do sinal de entrada e saida
13   in_novo = np.zeros(npt, dtype=complex)
14   out_novo = np.zeros(npt, dtype=complex)
15   in_novo.real = entrada
16   out_novo.real = saida
17
18   # Separa estrutura para cada periodo
19   X = np.zeros((N,n), dtype=complex)
20   Y = np.zeros((N,n), dtype=complex)
21   for i in range(0,N):
22     X[i] = np.fft.fft(in_novo[ i*n : (i+1)*n])
23     Y[i] = np.fft.fft(out_novo[i*n : (i+1)*n])
24
25   Xhalf = np.zeros((N,int(n/2)), dtype=complex)
26   Yhalf = np.zeros((N,int(n/2)), dtype=complex)
27   for i in range(0,N):
28     Xhalf[i] = X[0:int(n/2)]
29     Yhalf[i] = Y[0:int(n/2)]
30
31   # Calcula as correlacoes
32   Gyx = np.zeros(int(n/2), dtype=complex)
33   Gxx = np.zeros(int(n/2), dtype=complex)
34   H = np.zeros(int(n/2), dtype=complex)
35   for i in range(0,N):
36     Gyx = Gyx + Yhalf[i]*np.conj(Xhalf[i])
37     Gxx = Gxx + Xhalf[i]*np.conj(Xhalf[i])
38
39   # Adota estimador H_um
40   H = Gyx / Gxx
41   return (H, f)
42 #Funcao que calcula os parametros de Markov
43 def impulse_resp(H):
44   Hflip = np.flipud(H)
45   Hflipconj = np.conj(Hflip)
46   Hnovo = np.concatenate((H, Hflipconj), axis=0)
47   ynovo = np.fft.ifft(Hnovo)
48   yres = np.round(np.real(ynovo),14)
49   return (yres)
```

Figura 4.29 – Cálculo dos parâmetros de Markov

A função que avalia o desempenho do sistema de controle em reduzir a vibração estrutural é apresentada na Figura 4.30. A variável `N` é definida com valor igual a 4. Dessa forma, esta função aplica quatro períodos de um sinal específico na entrada de perturbação $\mathbf{w}$, sendo que o sinal de controle é aplicado no entrada $\mathbf{u}$, somente nos 2 últimos períodos. Com isso, é possível comparar a vibração estrutural do sistema em malha aberta e com a aplicação do sinal de controle, calculado na linha 32, em função dos estados estimados na linha anterior. Vale ressaltar que a aplicação de todos os sinais pelo Raspberry Pi foi limitado em $\pm 15V$.

```
1  # Funcao que avalia o desempenho do sistema de controle aplicando
2  # quatro periodos de disturbio na entrada de perturbacao e fechando
3  # a malha de controle nos dois ultimos periodos
4  def excMotores():
5    x = np.zeros((2,1))
6    con = 0
7    i=0
8    while (i < n*N):
9      if(wN[i] > 15.0):  # Sinal de perturbacao limitado em |15|V
10       pwm_inf.ChangeDutyCycle(100)
11     elif(wN[i] < -15.0):
12       pwm_inf.ChangeDutyCycle(100)
13     else:
14       pwm_inf.ChangeDutyCycle((abs(wN[i])/15)*100)
15     if(dir_wN[i]== 1):
16       GPIO.output(dirInf, GPIO.HIGH)
17     else:
18       GPIO.output(dirInf, GPIO.LOW)
19     pzt_level = ReadChannel(PZT_channel)  # leitura tensao PZT
20     pzt_volts = ConvertVolts(pzt_level,4)
21     y = pzt_volts-1.83
22     ymed[i]=y
23     # Calcula media movel do sinal de leitura
24     if(i> TAM-1):
25       ymovel[i] = (np.sum(ymed[i-TAM:i]))/(TAM)
26     else:
27       ymovel[i] = (np.sum(ymed[0:i]))/(i+1)
28     if(i > 2*n-1):  # Malha fechada
29       # Calcula o sinal de controle
30       yest[i] = MC*x;
31       x = MA*x + MB[:,0]*con + L*(ymovel[i]-yest[i]);
32       con = -K*x
33       contr_d[i] = con
34       # Aplica o sinal de controle limitado em |15|V
35       if(con > 15.0):
36         pwm_sup.ChangeDutyCycle(100)
37       elif(con < -15.0):
38         pwm_sup.ChangeDutyCycle(100)
39       else:
40         pwm_sup.ChangeDutyCycle((abs(con)/15)*100)
41           if(con > 0):
42         GPIO.output(dirSup, GPIO.HIGH)
43       else:
44         GPIO.output(dirSup, GPIO.LOW)
45     time.sleep(dt)
46     i+=1
```

Figura 4.30 – Controle da vibração estrutural

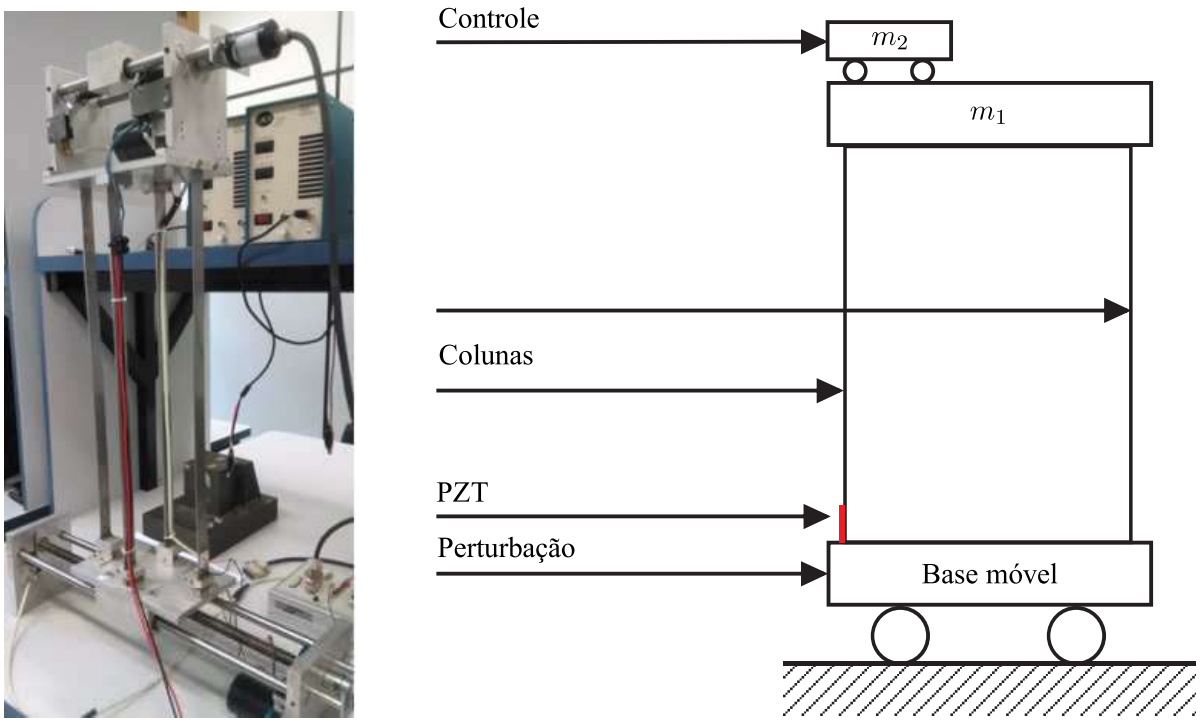
5 RESULTADOS EXPERIMENTAIS

Este capítulo descreve inicialmente a montagem física do protótipo utilizado, bem como a instrumentação desenvolvida para a realização dos ensaios experimentais. Em seguida, são apresentados os sinais utilizados para a identificação da estrutura e validação do sistema de controle para reduzir a vibração estrutural. Na sequência, expõe os resultados experimentais obtidos na identificação da estrutura na forma saudável e, posteriormente, apresenta os resultados experimentais do sistema de controle sob os parâmetros do modelo identificado. Em seguida, são induzidos dois tipos de danos na estrutura, o primeiro consiste na remoção de uma coluna de quatro e, o segundo, na remoção de outra coluna, totalizando duas colunas removidas. Na sequência do capítulo, é apresentado o procedimento experimental para detectar esses dois diferentes danos e analisar a severidade de cada um. A estratégia compreende em identificar periodicamente a estrutura com sinal determinístico, a fim de atualizar o modelo da estrutura, e os ganhos do controlador e observador, de acordo com a severidade do dano. Posteriormente, são apresentados os resultados experimentais da identificação e controle da estrutura para o dano 2, o mais severo. Por último, é descrito uma análise do tempo de execução da identificação estrutural, utilizando a biblioteca Eigen em comparação com a implementação em OpenCL no FPGA.

5.1 Estrutura flexível inteligente

A estrutura utilizada para validação da metologia é mostrada na Figura 5.1a. Ela possui massa total de $16kg$ e altura de $824mm$, podendo ser dividida em três partes principais: base, sistema flexível que simula um prédio alto e uma massa ativa AMD (do inglês, *Active Mass Driver*). A base é acoplada à estrutura por um fuso. Assim, os movimentos rotacionais de um motor CC de $35W$ e $5500rpm$ são convertidos em movimentos lineares. A estrutura compreende a massa m_1 suportada por quatro vigas de alumínio anexadas à base e há um elemento piezoelétrico colado na extremidade inferior de uma coluna para medir a vibração lateral. A massa ativa m_2 é movida linearmente por outro motor DC, usando um outro fuso de avanço semelhante. Além disso, os motores CC são ativados por unidades de potência, baseadas em sinais PWM.

O sistema de controle é projetado para minimizar vibrações causadas pelo movimento da base. Para esse fim, um sinal de controle é usado para criar um movimento específico da AMD que compense as perturbações. Assim, as forças dinâmicas produzidas pelo deslocamento de m_2 sobre a estrutura é responsável pela atenuação da vibração. A Figura 5.1b ilustra o diagrama icônico que descreve a estrutura com os principais sinais de entrada e saída.



(a) Montagem física (b) Diagrama icônico

Figura 5.1 – Estrutura vertical

O acionamento dos motores via PWM e a aquisição do valor de tensão do sensor piezoelétrico é realizado pelo Raspberry Pi. Inicialmente, conforme a Figura 1.1, caso não exista perturbação externa, o Raspberry Pi adquire os sinais de vibração e calcula a resposta ao impulso de cada entrada (parâmetros de Markov), para enviar ao sistema embarcado DE10-Nano. O kit SoC DE10-Nano é responsável por fazer a identificação do sistema e reprojetar o controlador e o observador, caso seja detectado dano na estrutura. Em seguida, o DE10-Nano envia ao Raspberry Pi o modelo atualizado do sistema de controle ou a confirmação de integridade estrutural. Assim, o Raspberry Pi é habilitado para controlar a vibração da estrutura quando ela está sujeita a perturbações externas. A Figura 5.2 ilustra o procedimento de instrumentação do sistema.

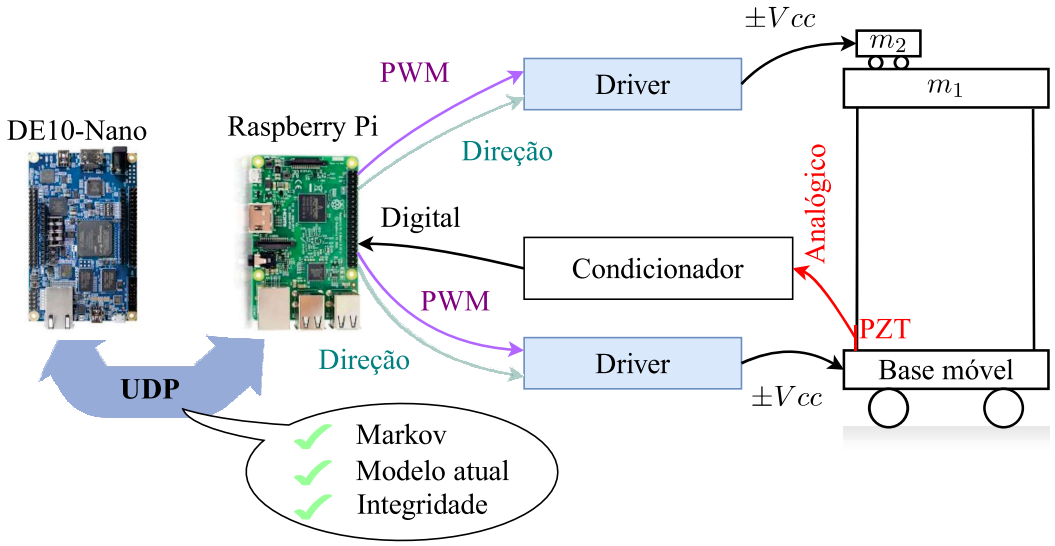


Figura 5.2 – Instrumentação do sistema

O Raspberry Pi adquire apenas sinais digitais e, por isso, é necessário utilizar um conversor analógico-digital para adquirir o valor de tensão do PZT. Como a entrada analógica do conversor $MCP3008$ utilizado pode receber apenas valores positivos entre $0V$ e $5V$, é necessário condicionar o valor de tensão fornecida pelo sensor para que ele fique dentro dos limites de entrada deste circuito integrado, o que consiste basicamente em reduzir a amplitude pico-a-pico e aplicar um *offset* ao sinal de tensão do PZT. Para este fim, utilizam-se os dois amplificadores operacionais do chip $CA3240$. O primeiro amplificador atua como *buffer* e, assim, fornece alta impedância de entrada e o segundo é usado como amplificador inversor. O circuito integrado $AD620$ é um amplificador de instrumentação que fornece o *offset* desejado ao sinal. A Figura 5.3 apresenta o circuito de condicionamento projetado.

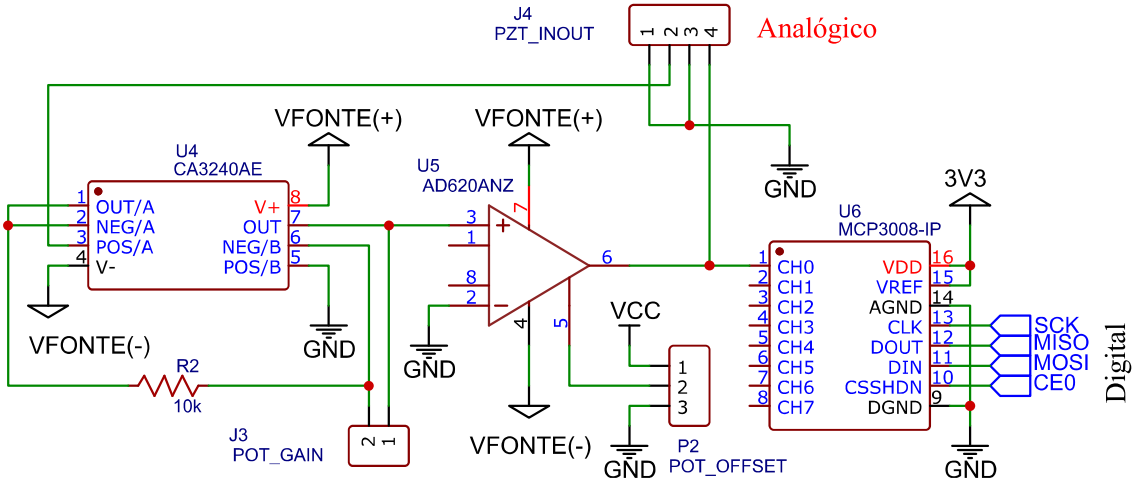


Figura 5.3 – Circuito eletrônico para condicionamento do sinal

Utilizou-se também um filtro de média móvel para melhorar a relação de sinal-ruído, implementado no Raspberry Pi. Em contrapartida, há um pequeno atraso do sinal filtrado em relação ao sinal condicionado, como ilustrado na Figura 5.4. O valor da frequência natural identificada pelo modelo é ligeiramente deslocado do valor real. Entretanto, essa mudança é embutida no modelo, na identificação experimental da planta.

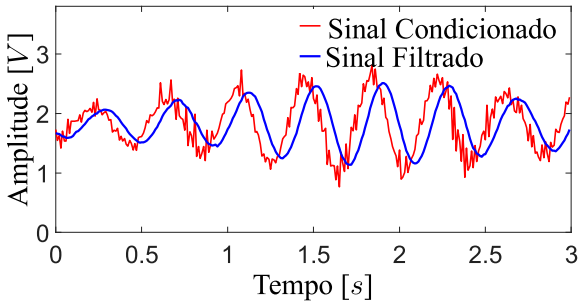
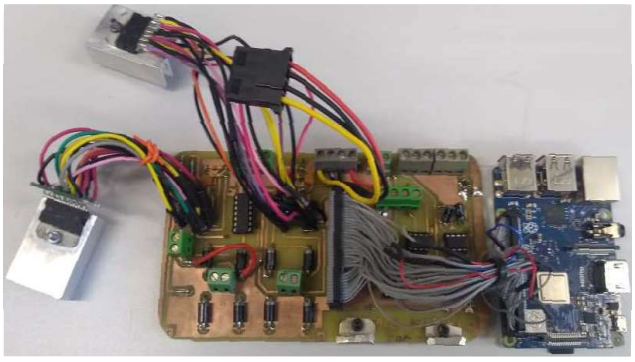


Figura 5.4 – Filtragem do sinal condicionado

O acionamento de cada motor DC é feito por um circuito em ponte H, denominado *driver*, o qual utiliza o chip $L298N$. O *driver* recebe o sinal de PWM e a direção de rotação do Raspiberry Pi, determinando a amplitude da tensão e o sentido da corrente sobre o motor. Vale a pena salientar que a tensão média máxima aplicada a cada motor, consiste no valor de tensão da fonte de alimentação ajustado em 15V, ou seja, com o PWM igual a 100%. A Figura 5.6 ilustra o circuito de acionamento dos motores implementado. Os circuitos das Figuras 5.3 e 5.6 foram confeccionados em uma placa de circuito impresso, conforme ilustrado na Figura 5.5a. A placa confeccionada foi embutida, em conjunto com o Raspberry Pi, em um protótipo mostrado na Figura 5.5b.



(a) Placa de interface entre o PZT e os motores com Raspberry Pi

(b) Montagem física do dispositivo de interface

Figura 5.5 – Integração dos componentes

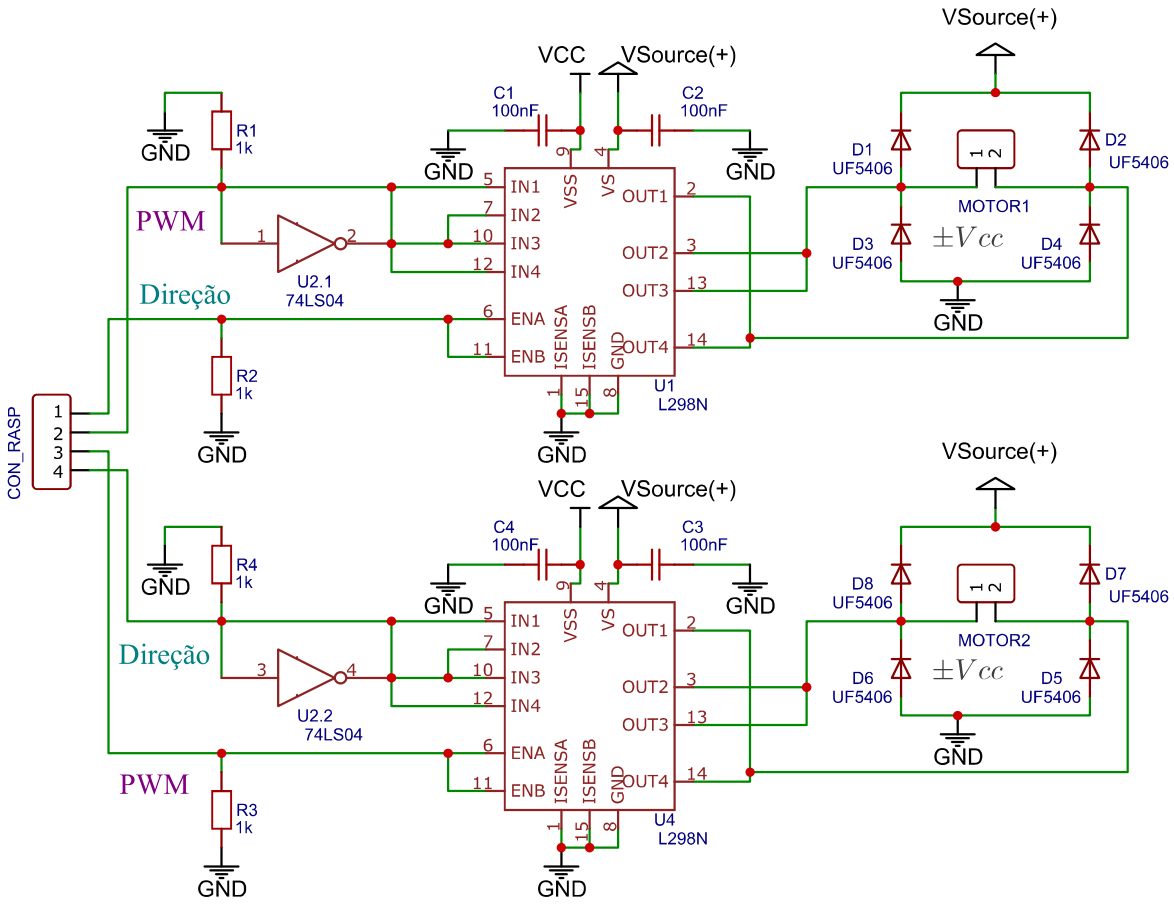


Figura 5.6 – Circuito de acionamento dos motores

5.2 Sinais para identificação e controle

Esta seção descreve a forma do sinal determinístico utilizado para identificar a estrutura flexível e apresenta os três tipos de sinais aplicados como perturbação, com o intuito de validar o controlador. A identificação e o controle da estrutura ocorrem em instantes diferentes. Em ambos procedimentos, os sinais foram empregados para a estrutura saudável e com o segundo dano.

5.2.1 Identificação com sinal determinístico

A identificação é realizada para modelar a estrutura no intervalo de frequências entre 0 e 6Hz, tendo em vista que a maior parte da energia real dos eventos sísmicos está localizada nesse intervalo, considerando como referência os terremotos de San Jose N59E, Kern County N90E e El Centro (Spencer *et al.*, 1985; Abreu; Lopes Jr., 2009; Abreu; Lopes Jr., 2010; Genari *et al.*, 2015). Nesse contexto, a entrada determinística utilizado na identificação

é o sinal de Schroeder (Hoagg; Lacy, 2006), sendo mostrado na Figura 5.7. Esse sinal possui duração de 16 segundos, banda de interesse limitada em 0 a 6Hz, sendo amostrado em 100Hz. O sinal de Schroeder tem um espectro plano, permitindo excitar a estrutura com igual energia na banda de frequência de interesse.

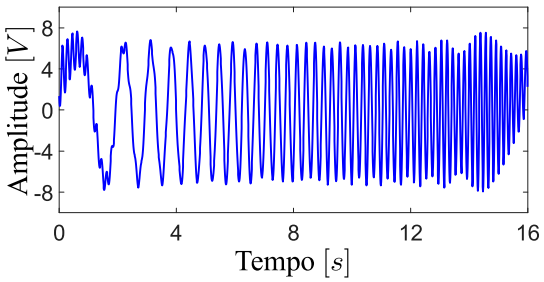
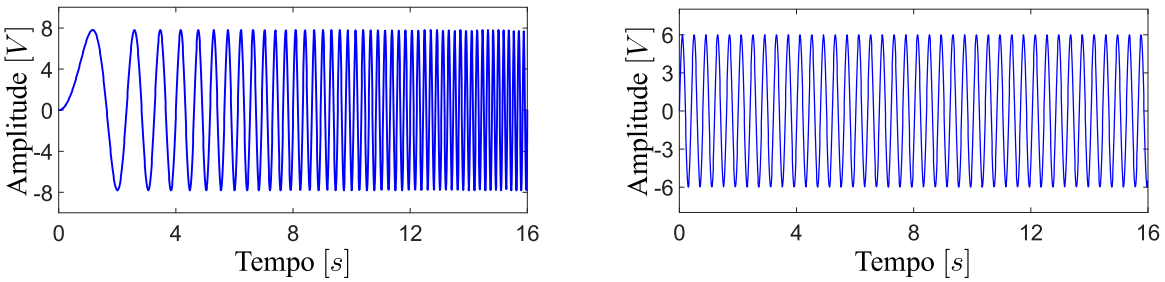


Figura 5.7 – Sinal de Schroeder

5.2.2 Sinal de distúrbio para a validação do controlador

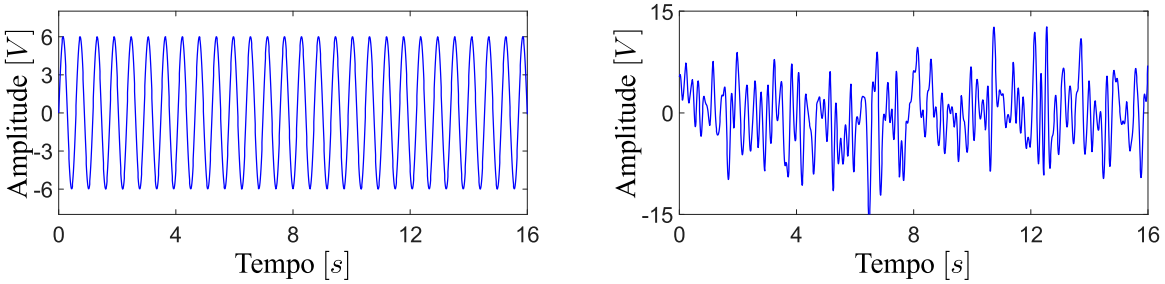
Neste trabalho são utilizados três sinais diferentes de distúrbio. O primeiro é o chirp presente na Figura 5.8a, com banda de frequência entre 0 e $6Hz$, amplitude de $7V$, duração de $16s$, discretizado na taxa de $100Hz$, aplicado para avaliar o controle da estrutura saudável e com dano 2. A segunda forma de perturbação é uma senoide aplicada na estrutura saudável e outra na estrutura com dano 2, cujas frequências são ajustas de acordo com a frequência natural de cada configuração. Para a estrutura saudável, a senoide foi configurada em $2,49Hz$ (Figura 5.8b) e para a estrutura com dano 2, a senoide foi configurada com $1,72Hz$, conforme ilustrado na Figura 5.8c. Em ambas as situações, os sinais possuem amplitude de $7V$, duração de $16s$ e são discretizadas na taxa de $100Hz$. O terceiro sinal é um ruído colorido, criado a partir de um ruído gaussiano branco, com semente igual a oito e duração de $16s$, discretizado na taxa de $100Hz$, com média zero, desvio padrão 15 e limitado por uma frequência de $6Hz$ por um filtro Butterworth Passa-Baixa de quarta ordem (Figura 5.8d). Este sinal foi aplicado para validar o sistema de controle da estrutura saudável e com o dano 2.

O sistema de controle deve ser projetado com o intuito de minimizar as vibrações da estrutura vertical levando em conta o requisito de instrumentação, ou seja, o sinal de controle é limitado pela tensão de alimentação dos *drivers*, de $\pm 15V$.



(a) Chirp

(b) Seno na ressonância da estrutura saudável



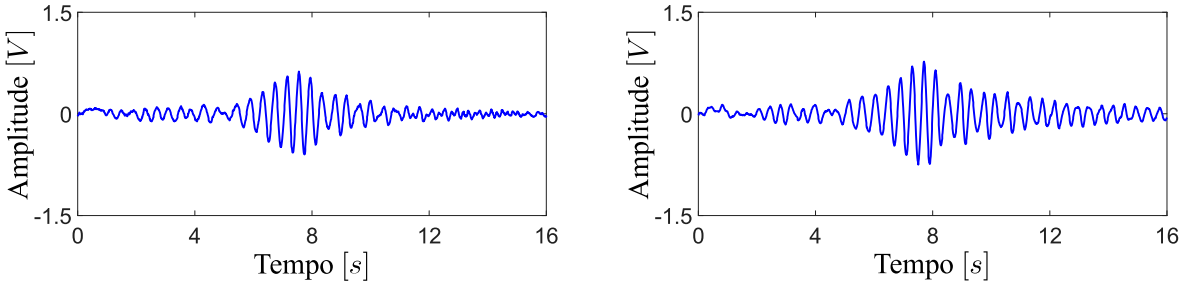
(c) Seno na ressonância da estrutura com dano 2

(d) Ruído colorido limitado em $6Hz$

Figura 5.8 – Sinais aplicados na entrada de perturbação para a validação do controlador

5.3 Identificação da estrutura saudável

A relação entre o sinal da entrada superior de controle e a vibração estrutural é obtida através da aplicação do sinal de Schroeder nesta entrada e definindo que a inferior seja nula, relação mostrada na Figura 5.9a. De maneira análoga, para obter a relação entre evento sísmico da entrada de perturbação e a vibração estrutural, o sinal de controle é definido como zero e o sinal determinístico é aplicado na entrada de distúrbio, conforme ilustrado na Figura 5.9b.



(a) Schroeder aplicado na entrada de controle

(b) Schroeder aplicado na entrada de perturbação

Figura 5.9 – Vibração da estrutura saudável em resposta ao sinal de Schroeder

O ponto de partida para a identificação de um sistema consiste em limitar a ordem máxima do modelo que o represente. No contexto da estrutura vertical desse trabalho, considera-se que o valor da ordem máxima suficiente seja igual a 10. Esse fato é evidenciado através da inspeção dos valores singulares de $\mathbf{H}[1]_{p,q}$, conforme descrito pela Equação 2.30. Na Figura 5.10, os dois primeiros valores singulares possuem as maiores magnitudes e, portanto, é possível realizar o truncamento da SVD, considerando que o sistema modelado seja de segunda ordem.

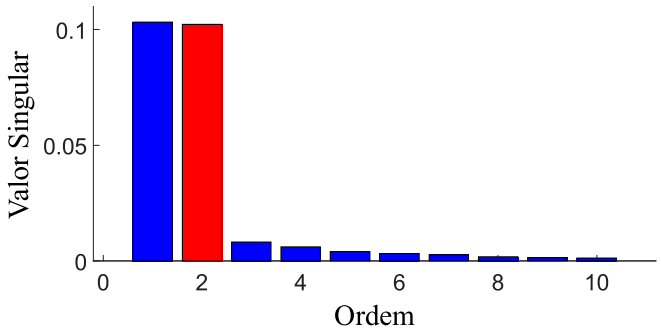
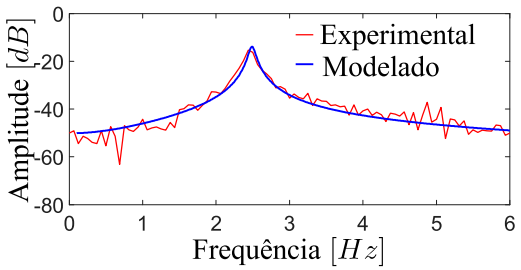
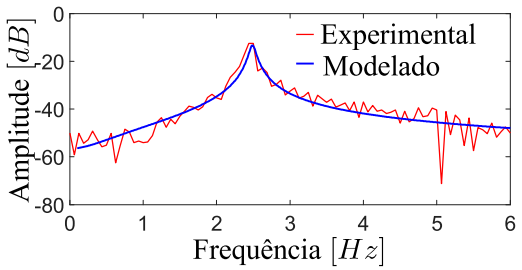


Figura 5.10 – Comparação dos valores singulares com a ordem do modelo

De acordo com a Figura 3.1, a função de resposta em frequência (FRF) experimental entre o sinal do sensor PZT e o sinal da entrada de controle é definida como P_{yu}, enquanto a FRF entre o sinal do sensor PZT e o sinal da entrada de perturbação é definida como P_{yw}. A Figura 5.11a compara a FRF experimental P_{yu} com a função de transferência obtida no modelo identificado e a Figura 5.11b compara a FRF experimental P_{yw} com a função de transferência obtida no modelo identificado. É possível notar que existe uma forte correspondência entre as FRFs experimentais e o modelo, demonstrando que ele representa adequadamente a estrutura inteligente saudável na banda de interesse.



(a) Comparação da FRF experimental P_{yu} com o modelo identificado



(b) Comparação da FRF experimental P_{yw} com o modelo identificado

Figura 5.11 – Identificação da estrutura saudável

O modelo identificado é representado pelas seguintes matrizes:

$$\mathbf{A}_d = \begin{bmatrix} 0.9825 & 0.1564 \\ -0.1542 & 0.9873 \end{bmatrix}, \quad \mathbf{B}_{1d} = \begin{bmatrix} 0.0228 \\ 0.104 \end{bmatrix}, \quad \mathbf{B}_{2d} = \begin{bmatrix} -0.0234 \\ 0.0042 \end{bmatrix}, \tag{5.1}$$

$$\mathbf{C}_d = \begin{bmatrix} -0.0496 & -0.0082 \end{bmatrix}, \quad \mathbf{D}_{1d} = \begin{bmatrix} 0 \end{bmatrix} \quad \text{e} \quad \mathbf{D}_{2d} = \begin{bmatrix} 0 \end{bmatrix}. \tag{5.2}$$

5.4 Controle da estrutura saudável

Na implementação do controlador para a estrutura saudável, foi ajustado um fator de amortecimento $\xi_1 = 0.12$. Este valor é fixo e permite que com a alocação de polos, o sinal de controle permaneça dentro dos limites requeridos pela instrumentação, considerando uma classe de distúrbios esperada. Dessa forma, os atuadores não saturam e permanecem dentro da faixa linear de operação. Caso a vibração estrutural esteja fora do previsto e demande um sinal de controle fora dos limites de operação da instrumentação, é necessário reconfigurar os parâmetros do controlador, reduzindo também o desempenho do sistema. Assim, os polos de malha fechada, respectivos ao primeiro modo, são alocados no plano s como

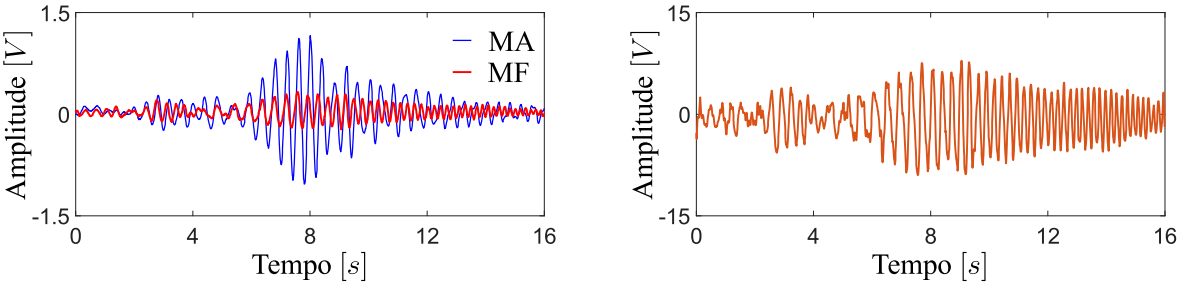
$$s = -\xi_1 \omega_{n_1} \pm j\left(\omega_{n_1}\sqrt{1-\xi_1^2}\right). \tag{5.3}$$

Estes polos são mapeados para o plano discreto z através da expressão $z = e^{Ts}$ (Ogata, 1994), em que T é o período de amostragem. Dessa forma, os polos da função de transferência em malha fechada no plano z foram alocados em $0.9696 \pm j0.1518$ e os polos do observador foram alocados para serem quatro vezes mais rápidos que os polos de malha fechada. Assim, os ganhos do controlador $\mathbf{K}_g$ e do observador $\mathbf{L}_g$ são calculados pelo método de Ackerman, em função das matrizes do sistema no espaço de estado em tempo discreto e obtidos na identificação usando a técnica ERA, sendo dados por

$$\mathbf{K}_g = \begin{bmatrix} -1.2321 & 0.4381 \end{bmatrix} \quad \text{e} \quad \mathbf{L}_g = \begin{bmatrix} -15.7106 \\ -39.0706 \end{bmatrix}. \tag{5.4}$$

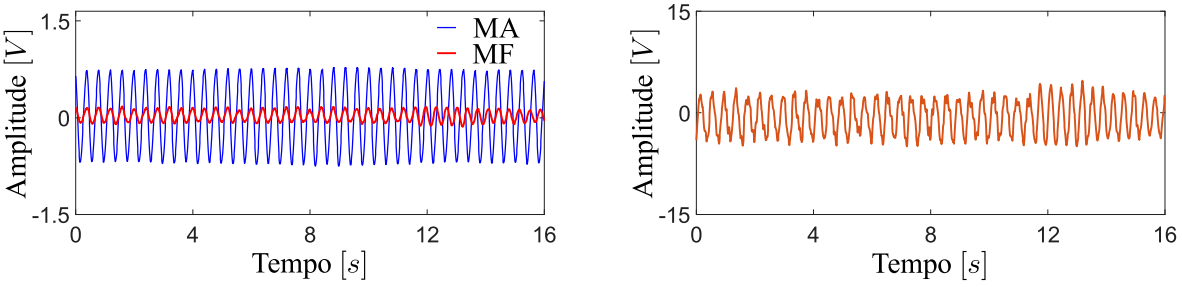
As medidas de vibração do sistema de controle da estrutura são atenuadas pela ação do controlador em malha fechada (MF) em relação ao sistema em malha aberta (MA), sem a ação do controlador. As Figuras 5.12, 5.13 e 5.14 ilustram as vibrações estruturais experimentais para a estrutura em malha aberta e em malha fechada, quando as entradas são o chirp, seno na frequência natural da estrutura ($2,49$Hz) e ruído colorido, respectivamente. Também é possível visualizar o sinais de controle do sistema em malha fechada para cada configuração.

Quando é aplicado o sinal de controle, ocorre redução da vibração em 74% e 78%, para as entradas de distúrbio chirp e seno, respectivamente. A energia do sinal de vibração experimental em malha fechada é reduzida em 80%, quando o ruído colorido é aplicado como entrada de distúrbio. Além disso, os sinais de controle permanecem dentro dos limites requeridos pela instrumentação em todos os casos.



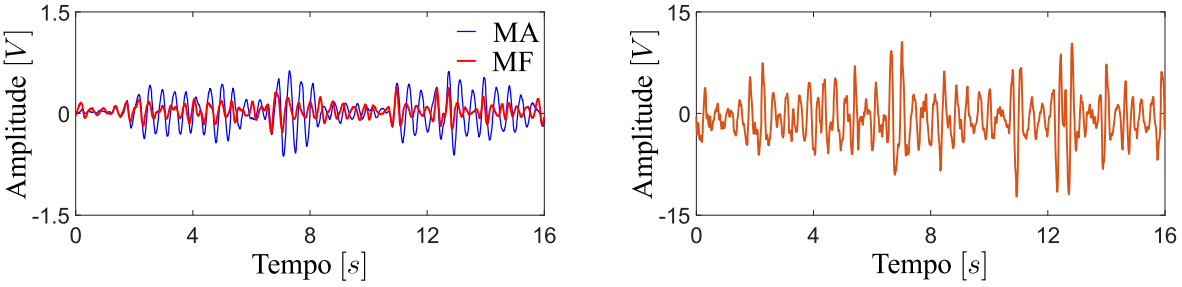
(a) Vibração (b) Sinal de controle

Figura 5.12 – Controle de vibração da estrutura saudável para o distúrbio chirp



(a) Vibração (b) Sinal de controle

Figura 5.13 – Controle de vibração da estrutura saudável para o distúrbio seno

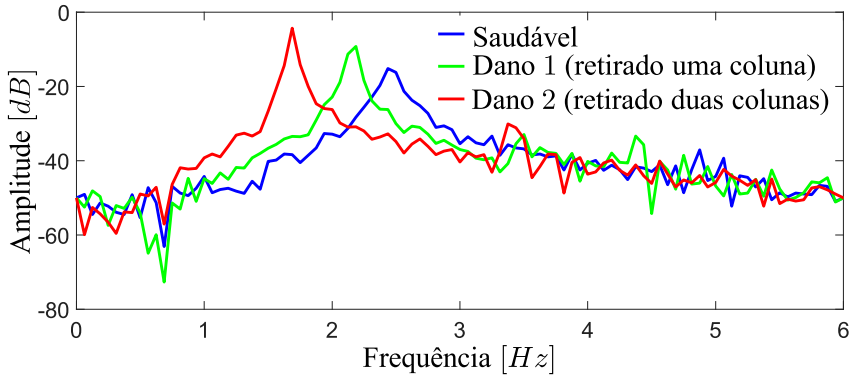


(a) Vibração (b) Sinal de controle

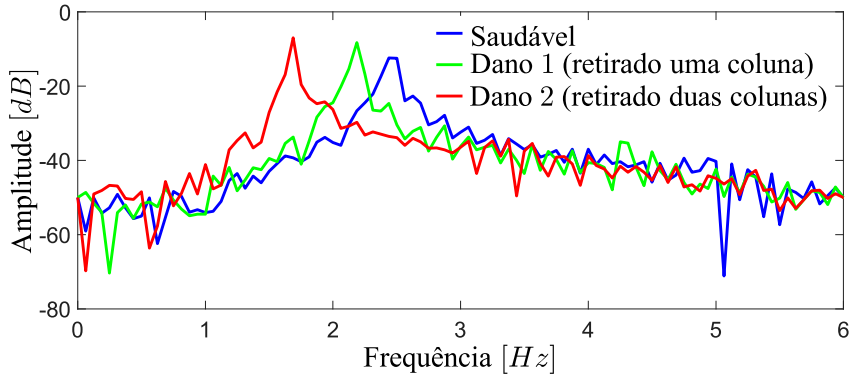
Figura 5.14 – Controle de vibração da estrutura saudável para o distúrbio ruído

5.5 Efeito do dano na dinâmica estrutural

Os testes experimentais da seção anterior foram aplicados no protótipo definido como saudável. Entretanto, mesmo com o controle ativo das vibrações, caso a estrutura seja danificada de tal forma que ocorra uma mudança significativa em sua dinâmica, o sistema de controle projetado pode perder desempenho e até mesmo a estabilidade. Dessa forma, o método para detecção de danos deve ser capaz de capturar essa modificação, sinalizando a necessidade da reconfiguração do controlador e do observador para assegurar um desempenho adequado e a estabilidade. A Figura 5.15 ilustra as FRFs experimentais da estrutura saudável e com a inclusão dos dois tipos de danos. O dano 1 consiste na remoção de uma coluna das quatro da estrutura e o dano 2, na remoção de mais outra coluna, totalizando duas colunas removidas. Verifica-se que a remoção gradativa das colunas provoca o deslocamento na frequência natural. Além disso, é fundamental perceber a alteração na magnitude dos picos das FRFs em cada configuração, principalmente no caso de P_{yw}, pois o objetivo do sistema de controle consiste em reduzir esse valor.



(a) Comparação de P_{yu}



(b) Comparação de P_{yw}

Figura 5.15 – Comparação das FRFs experimentais para o dano gradativo

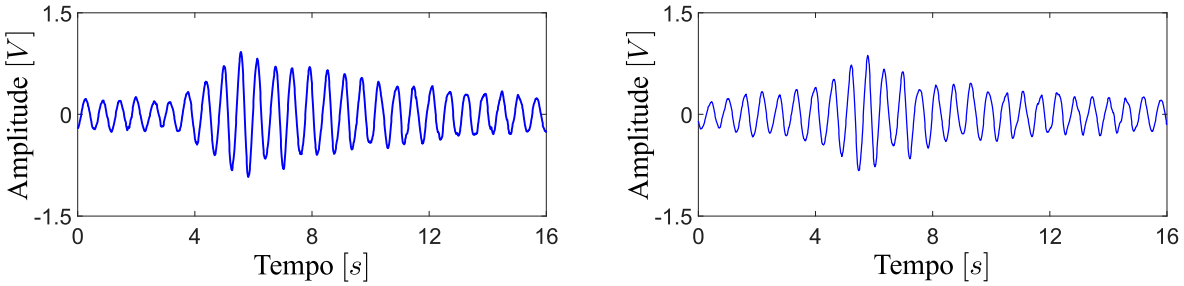
As distâncias entre o modelo de referência considerado saudável e os obtidos mediante a aplicação de danos na estrutura cresceu de forma gradativa com o aumento da severidade do dano, como mostrado na Tabela 5.1. É possível notar que a técnica para detecção de danos é capaz de detectar o dano e, também, quantizar sua severidade.

Comparação dos modelos	**Distância**
Saudável e sem uma coluna (dano 1)	1.5334
Sem uma coluna (dano1) e sem duas colunas (dano 2)	1.8715
Saudável e sem duas colunas (dano 2)	2.0113

Tabela 5.1 – Detecção de danos na estrutura flexível

5.6 Identificação da estrutura com dano

O procedimento de identificação para a estrutura danificada é similar ao da estrutura saudável. A Figura 5.16a apresenta a resposta da estrutura para a estrada Schroeder como sinal de controle, enquanto, a Figura 5.16b mostra a saída da estrutura quando o Schroeder é aplicado como perturbação. Além disso, a estrutura também é definida como um sistema de segunda ordem, de acordo com as magnitudes dos valores singulares apresentados na Figura 5.17.



(a) Schroeder aplicado na entrada de controle

(b) Schroeder aplicado na entrada de perturbação

Figura 5.16 – Vibração da estrutura com o dano 2 em resposta ao sinal de Schroeder

A Figura 5.18a compara a FRF experimental P_{yu} com a função de transferência obtida no modelo identificado e a Figura 5.18b compara a FRF experimental P_{yw} com a função de transferência obtida no modelo identificado, considerando a estrutura com o dano 2. É possível notar que ocorre também uma forte correspondência entre as FRFs experimentais e o modelo, demonstrando que o modelo representa adequadamente a estrutura com o dano 2. As matrizes identificadas do modelo identificado são:

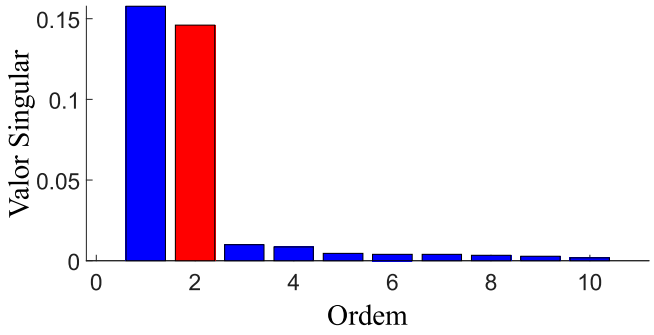
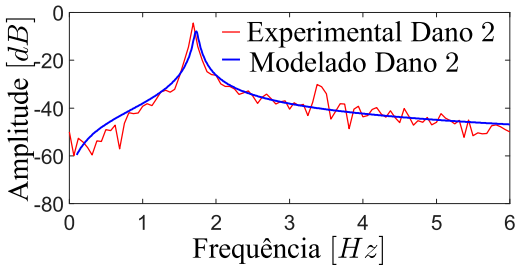
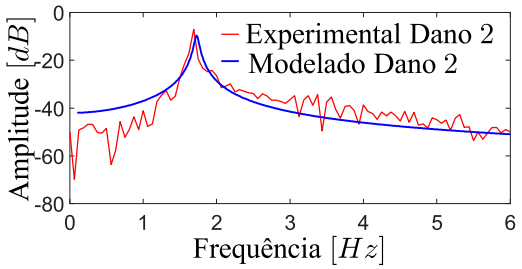


Figura 5.17 – Comparação dos valores singulares com a ordem do modelo



(a) Comparação da FRF experimental P_{yu} com o modelo identificado



(b) Comparação da FRF experimental P_{yw} com o modelo identificado

Figura 5.18 – Identificação da estrutura com o dano 2

$$\mathbf{A}_d = \begin{bmatrix} 0.9928 & 0.1137 \\ -0.1028 & 0.9916 \end{bmatrix}, \quad \mathbf{B}_{1d} = \begin{bmatrix} 0.0007 \\ -0.0211 \end{bmatrix}, \quad \mathbf{B}_{2d} = \begin{bmatrix} -0.0195 \\ 0.0178 \end{bmatrix}, \tag{5.5}$$

$$\mathbf{C}_d = \begin{bmatrix} -0.0427 & 0.0408 \end{bmatrix}. \tag{5.6}$$

5.7 Controle da estrutura com dano

O sistema de controle é reconfigurado para que, em malha fechada, possua fator de amortecimento igual ao da estrutura saudável controlada, que na banda de frequência analisada é único e de valor $\xi_1 = 0.12$, com o intuito de atender aos critérios de operação linear dos atuadores. As frequências naturais dos modos de malha fechada acompanham as frequências da dinâmica estrutural, pois o critério de desempenho neste trabalho é reduzir a vibração estrutural, desconsiderando qualquer performance em relação a frequência natural de malha fechada. Vale a pena ressaltar que isso pode ser configurado no sistema de controle, caso as frequências naturais sejam um requisito de projeto. Assim, com a inserção do dano 2, os polos de malha

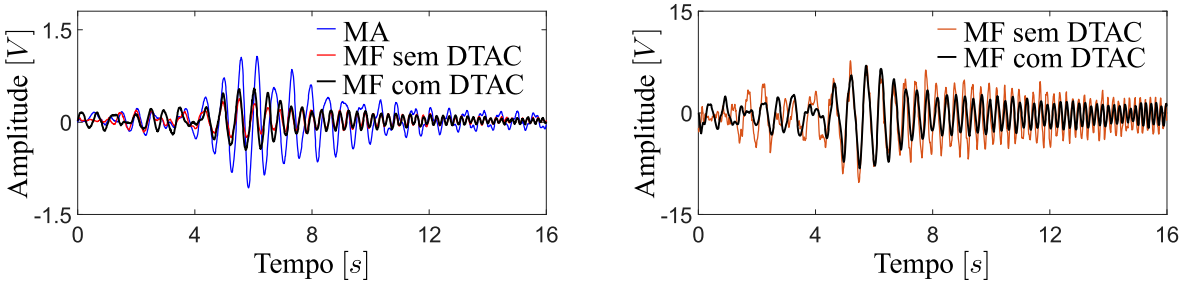
fechada foram automaticamente alocados no plano z em $0.9813 \pm 0.1062j$ e os polos do observador foram também automaticamente alocados para serem quatro vezes mais rápidos que os polos de malha fechada. Dessa forma, os ganhos do controlador e de observador para estrutura danificada são dados por

$$\mathbf{K}_g = \begin{bmatrix} -0.6509 & 0.5091 \end{bmatrix} \quad \text{e} \quad \mathbf{L}_g = \begin{bmatrix} -20.7393 \\ -15.3594 \end{bmatrix}. \tag{5.7}$$

As Figuras 5.19, 5.20 e 5.21 ilustram o controle de vibrações para a estrutura com dano 2 utilizando o controlador e o observador projetados para a estrutura saudável (MF sem DTAC) e empregando o novo sistema de controle (MF com DTAC), o qual reprojeta os ganhos do controlador e observador, considerando o segundo dano. Esses sinais foram avaliados pela aplicação das entradas de perturbação chirp, seno na frequência natural da estrutura com dano 2 $(1,72\text{Hz})$ e ruído colorido, respectivamente. Também é possível visualizar o sinais de controle do sistema em malha fechada para cada configuração. Quando o monitoramento detecta a existência de danos, o modelo da estrutura é atualizado e os ganhos do controlador e observador são reprojetados. É possível notar que o desempenho do sistema de controle reprojetado, considerando o segundo dano, é análogo ao desempenho do controlador projetado anteriormente para a estrutura saudável, testado na estrutura danificada. Entretanto, apenas o sistema reprojetado DTAC garante estabilidade do sistema de controle, gerando também um sinal de controle menor. Com a aplicação do sinal de controle em malha fechada pelo novo controlador reprojetado, ocorre redução da vibração em 53% e 69%, para as entradas de distúrbio chirp e seno, respectivamente. A energia do sinal de vibração experimental em malha fechada é reduzida em 89% para o ruído colorido como entrada de distúrbio. Além disso, os sinais de controle permanecem também dentro dos limites requeridos pela instrumentação em todos os casos. O sistema de controle mantem um ótimo desempenho, considerando os três sinais utilizados como distúrbio, mostrando a eficácia e a segurança de operação da técnica de controle proposta para controlar estruturas na condição de saudável e quando sofrem danos.

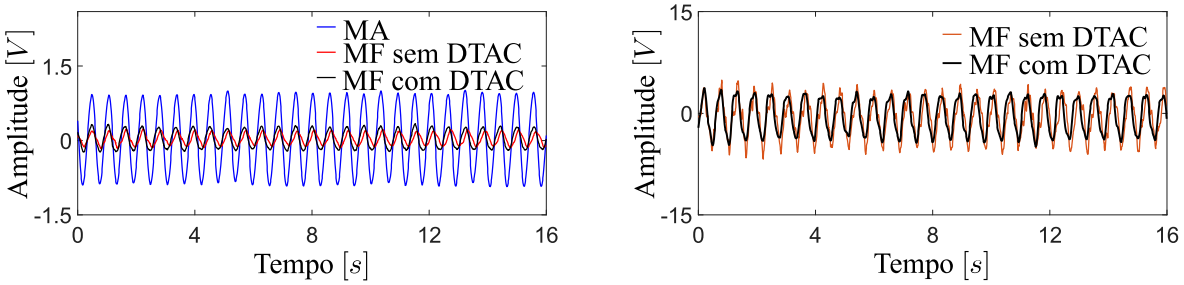
5.8 Avaliação da implementação no FPGA

A identificação da estrutura pela a técnica ERA, consiste em construir as matrizes de Hankel $\mathbf{H}[1]_{p,q}$ e $\mathbf{H}[2]_{p,q}$, conforme as Equações 3.35 e 3.36, decompor a matriz $\mathbf{H}[1]_{p,q}$ em valores singulares e, após a inspeção dos valores singulares, efetuar as multiplicações matri-



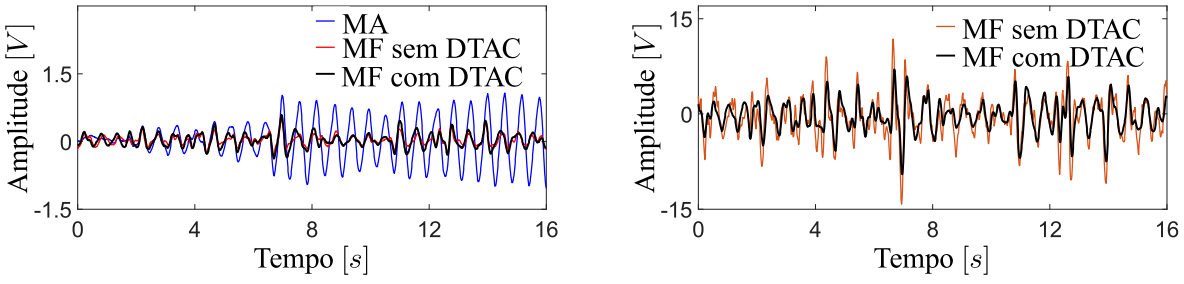
(a) Vibração (b) Sinal de controle

Figura 5.19 – Controle de vibração da estrutura com o dano 2 para o distúrbio chirp



(a) Vibração (b) Sinal de controle

Figura 5.20 – Controle de vibração da estrutura com o dano 2 para o distúrbio seno



(a) Vibração (b) Sinal de controle

Figura 5.21 – Controle de vibração da estrutura com o dano 2 para o distúrbio ruído

ciais para obter as matrizes que representam o modelo dinâmico estrutural, de acordo com as Equações 3.45 e 3.46. Conforme descrito na Seção 3.1, considerando que a estrutura flexível possua s entradas, r saídas e que o sistema modelado possua ordem m, as matrizes obtidas nas Equações 3.45 e 3.46, podem ser reescritas como

$$\mathbf{A}_d = \boldsymbol{\Sigma}_m^{-1/2} \underbrace{\mathbf{U}_m^T \mathbf{H}[1]_{p,q} \mathbf{V}_m}_{\mathbf{MA}_m} \boldsymbol{\Sigma}_m^{-1/2}, \tag{5.8}$$

$$\mathbf{B}_d = \mathbf{\Sigma}^{1/2} \underbrace{\mathbf{V}^T(1:m, 1:s)}_{\mathbf{MB}_m} \quad \text{e} \quad \mathbf{C}_d = \underbrace{(1:m, 1:r)\mathbf{U}}_{\mathbf{MC}_m} \mathbf{\Sigma}^{1/2}. \tag{5.9}$$

A implementação paralela em OpenCL no FPGA consiste em construir as matrizes de Hankel, conforme descrito na Seção 4.2 e, em seguida, calcular a SVD de $\mathbf{H}[1]_{p,q}$ por meio de um dos métodos descritos na Seção 2.6. Para o cálculo da SVD via iterações QR, a paralelização consiste em calcular a matriz bidiagonal $\mathbf{B}$ e as bases ortogonais $\mathbf{P}$ e $\mathbf{Q}$, conforme apresentado na Seção 4.3, transferindo as matrizes resultantes para o processador ARM aplicar as iterações QR e obter $\mathbf{U}_m$, $\mathbf{\Gamma}_m$ e $\mathbf{V}_m$ e, assim, calcular as matrizes das Equações 5.8 e 5.9, com o auxílio da biblioteca Eigen. No método Hestenes-Jacobi, as matrizes que representam a SVD de $\mathbf{H}[1]_{p,q}$ são calculadas iterativamente no FPGA. Ademais, são realizadas as multiplicações matriciais de $\mathbf{MA}_m$, $\mathbf{MB}_m$ e $\mathbf{MC}_m$, com o intuito de transferir para o processador ARM somente matrizes com dimensão da ordem do modelo e, assim, calcular as matrizes $\mathbf{A}_d$, $\mathbf{B}_d$ e $\mathbf{C}_d$, através da utilização da biblioteca Eigen.

Além disso, a biblioteca Eigen é utilizada para calcular os ganhos do sistema de controle $\mathbf{K}_g$ e $\mathbf{L}_g$. Esta biblioteca é frequentemente utilizada no auxílio de cálculos numéricos (Bates; Eddelbuettel, 2013; Ikeno, 2018; Eray; Temizel, 2020), sendo compatível com outras bibliotecas tais como Lapack e BLAS (Eigen, 2021), amplamente utilizadas nos cálculos de álgebra linear. A biblioteca Eigen fornece atualmente duas funções que calculam a SVD de uma matriz. A primeira consiste em utilizar o método denominado *jacobiSvd()*, que implementa iterações de Jacobi em ambos os lados da matriz de entrada, sendo numericamente preciso e rápido para matrizes de ordem reduzida. A outra maneira se dá pelo uso do método *bdcSvd()*, a qual implementa uma estratégia recursiva, denominada divisão e conquista, a partir da matriz de entrada reduzida para a forma bidiagonal superior. Além disso, este procedimento é ligeiramente mais rápido para matrizes de ordem elevada. Entretanto, a recursão não é suportada na linguagem OpenCL (Munshi, 2012). Dessa forma, intuito é analisar a viabilidade de uso do FPGA como acelerador OpenCL no cálculo da SVD em relação aos métodos disponíveis pela biblioteca Eigen. Na identificação da estrutura realizada neste trabalho, foram construídas as matrizes de Hankel em função dos parâmetros $q = 101$ e $p = 800$. Com o intuito de verificar o tempo de identificação para matrizes maiores, na hipótese do sistema vir a ser constituído de mais sensores e atuadores, foi realizado uma comparação do tempo de identificação ERA pelo DE10-Nano, no qual a matriz de Hankel em cada identificação possui como parâmetro $q = 128$ e p variando de 1024 à 32768, conforme ilustrado na Figura 5.22.

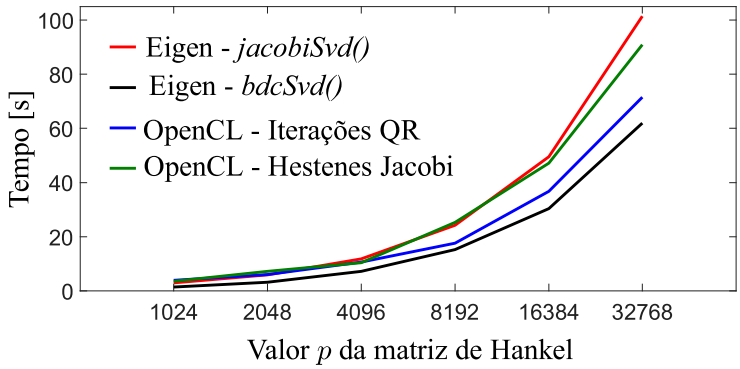


Figura 5.22 – Tempo de identificação ERA para diferentes tamanhos da matriz de Hankel

O cálculo da decomposição em valores singulares é muito efetivo quando processado de forma serial, mas não é prontamente receptivo em implementações paralelas devido a dependência de dados para sua execução. A maior contribuição do tempo total gasto pelo método de interação QR, proposto por Golub e implementado em OpenCL, consiste na aplicação das transformações de HouseHolder sobre $\mathbf{A}$. Os *kernels* responsáveis pelas operações do método Hestenes-Jacobi também possuem fatores internos ao *loop* desconhecidos pelo compilador e a vetorização é inviabilizada. Além disso, o uso de múltiplas unidades de computação que poderiam acelerar o cálculo, ultrapassa os recursos do dispositivo DE10-Nano. Desta forma, a proposta deste trabalho foi implementar uma estratégia de paralelização mais conservadora devido as deficiências do dispositivo DE10-Nano (pouca memória, falta de recursos em ampliar as unidades de computação). Mesmo assim, a implementação nesse dispositivo chegou próximo ao tempo de execução da divisão e conquista *bdcSvd()*. Entretanto, é preciso realizar testes experimentais com sistemas mais complexos com múltiplas entradas e saídas, para validar a comparação com matrizes maiores. Ainda é possível melhorar o desempenho do cálculo da SVD através do método Hestenes-Jacobi. Por exemplo, Wang *et al.*, 2020 utilizaram um SoC composto com FPGA, denominado Xilinx Zynq-7000®, para calcular, em paralelo no FPGA, a SVD de uma matriz de forma eficiente comparada aos métodos disponíveis na biblioteca Eigen. A estratégia proposta neste trabalho foi baseada no método Hestenes-Jacobi, através do compartilhamento máximo de dados durante a execução. Como resultado, o cálculo da SVD foi acelerado em $300\times$ comparado as duas implementações disponíveis na biblioteca Eigen.

6 CONCLUSÕES E PERSPECTIVAS

Neste trabalho foi desenvolvido um sistema automatizado que monitora a integridade e atenua as vibrações de estruturas inteligentes flexíveis sujeitas a danos. O monitoramento periódico é implementado em uma arquitetura de processamento paralelo e consiste em identificar a estrutura, detectar a existência de danos, quantificar a sua severidade e reprojetar o sistema de controle de forma automática. A seguir são apresentadas as conclusões detalhadas e as perspectivas para a continuação do trabalho.

6.1 Conclusões

Inicialmente, o segundo capítulo apresentou os conceitos fundamentais de álgebra linear que serviram de base para o cálculo prático da decomposição em valores singulares utilizados pela técnica de identificação ERA, implementada em OpenCL no FPGA, e apresentou conceitos necessários para o cálculo da distância entre os subespaços, utilizados na detecção de danos. O terceiro capítulo descreveu a representação matemática de uma estrutura flexível no espaço de estados, a discretização de sistemas contínuos, o métodos de identificação ERA, a estratégia de projeto do controlador, do observador e o método de detecção de danos utilizando subespaços. O quarto capítulo apresentou a programação paralela em OpenCL, sua implementação em sistemas embarcados com FPGA e demonstrou a ideia para acelerar o cálculo nas etapas de construção da matriz de Hankel e da decomposição em valores singulares, ambas utilizadas na identificação do sistema pelo método ERA. No quinto capítulo, foi feita a validação experimental da identificação e controle de vibração de uma estrutura inteligente, monitorando-a periodicamente. Como resultado, o sistema de controle foi reconfigurado automaticamente quando o dano foi detectado. Além disso, tanto na forma saudável como na presença de dano, o controlador projetado foi capaz de reduzir significantemente as vibrações.

Os resultados experimentais apresentaram uma forte correspondência, demonstrando que o método de identificação e detecção de danos proposto foi capaz de modelar adequadamente a dinâmica da estrutura flexível saudável e danificada. Dessa forma, como resultado prático, esta metodologia investigada tem potencial de ser embarcada e de monitorar a saúde de estruturas flexíveis inteligentes reais e atenuar as respectivas vibrações, providenciando mais segurança na operação e possibilitando construir estruturas mais leves e maiores. Vale a pena

ressaltar que a realização deste trabalho envolveu uma revisão bibliográfica abrangente dos assuntos relacionados ao tema de controle ativo de vibrações, controle tolerante a falhas, controle tolerante a danos, monitoramento da integridade de estruturas, identificação por subespaços de sistemas determinísticos e estocásticos, técnicas de detecção de danos baseadas em séries temporais, identificação clássica de sistemas determinísticos e estocásticos, aquisição e processamento de sinais. materiais inteligentes, sistema embarcados, processamento paralelo em OpenCL com FPGA, programação e aplicação de dispositivos heterogêneos, e valorosos conceitos em álgebra linear e geometria analítica.

6.2 Perspectivas

As estruturas atuais são projetadas para terem a máxima de produtividade. Por isso, é importante realizar o monitoramento da estrutura de forma preventiva, objetivando reduzir o tempo de interrupção da operação. Nesse contexto, é possível tornar a estratégia proposta na dissertação mais eficiente, implementando o sistema embarcado com a metodologia de identificação ERA de forma *online*, assim, permitindo que as estruturas sejam testadas em modo de operação.

No sistema embarcado proposto neste trabalho, foram utilizados apenas um sensor piezoelétrico (PZT) para medição dos sinais de vibração e, como atuador, um amortecedor de massa ativa (AMD), para atenuar a vibração estrutural. Entretanto, as estruturas inteligentes complexas como aeronaves, são constituídas de muitos sensores e atuadores para realizar o monitoramento da sua integridade e o controle de sua vibração. A quantidade de sensores e atuadores fazem com que a dimensão necessária das matrizes, para identificar o modelo estrutural pela técnica ERA, aumente significativamente. Dessa forma, é fundamental implementar metodologias para processar sinais de múltiplos sensores e atuadores, em um período tempo satisfatório, visando não afetar as atividades regulares de operação estrutural. A técnica proposta neste trabalho cumpre este objetivo, entretanto, ainda é necessário testá-la em estruturas mais complexas, com dinâmica mais sofisticada e com múltiplos atuadores e sensores, para analisar melhor o seu desempenho frente a abordagens puramente seriais.

Evidentemente, o cálculo da SVD em computadores com melhor poder de processamento, como *clusters*, pode ser realizado em um tempo menor. No entanto, o foco deste trabalho foi implementar um sistema embarcado, compatível com aplicações práticas, que demanda baixo consumo de energia, utilizando o kit educacional DE10-Nano como servidor, com o qual

ainda é possível acelerar a identificação. Ainda assim, é possível otimizar o kernel implementado mediante melhores práticas de programação disponíveis em Altera, 2020. Além disso, a utilização de dispositivos OpenCL com mais recursos, sendo executados por processadores mais robustos, permitem maior aceleração do cálculo. A estratégia proposta por Wang *et al.*, 2020, apresenta uma aceleração de até $300\times$ em comparação as duas implementações disponíveis na biblioteca Eigen.

REFERÊNCIAS

Abbaspour, A.; Mokhtari, S.; Sargolzaei, A.; Yen, K. K. A Survey on Active Fault-Tolerant Control Systems. **Electronics**, 2020.

Abdulameer, H. A.; Wasmi, H. R. Vibration Control Analysis of Aircraft Wing by Using Smart Material. **Innovative Systems Design and Engineering**, v. 6, n. 8, 2015.

Abreu, G. L. C. M.; Lopes Jr., V. Mixed H_2/H_∞ control of a two-floors building model using the linear matrix inequality approach. In: **20th International Congress of Mechanical Engineering – COBEM2009**. Gramado, Brazil: [s.n.], 2009.

Abreu, G. L. C. M.; Lopes Jr., V. H_2 Optimal Control for Earthquake Excited Structures. In: **VI Congresso Nacional de Engenharia Mecânica - CONEM2010**. Campina Grande, Brazil: [s.n.], 2010.

Adhikari, S. **Damping Models for Structural Vibration**. Tese (Degree of Doctor of Philosophy) — Cambridge University, 2000.

Ali, A. A.; Mahmood, H. Y.; Saeed, M. Active Vibration Control of Aircraft Wing Using Piezoelectric Transducers. **International Journal of Scientific and Engineering Research**, 2016.

Altera. **Implementing FPGA design with the OpenCL standard**. 2013. "<https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01173-opencl.pdf>. Acesso em: 13 dec. 2020.

Altera. **Intel FPGA SDK for OpenCL - Pro Edition Best Practices Guide**. 2020. "<https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/opencl-sdk/aocl-best-practices-guide.pdf>. Acesso em: 13 dec. 2020.

Alwi, H.; Edwards, C.; Tan, C. **Fault Detection and Fault-Tolerant Control Using Sliding Modes**. [S.l.]: Springer, 2011.

Ambrosio, P.; Cazzulani, G.; Resta, F.; Ripamonti, F. An optimal vibration control logic for minimising fatigue damage in flexible structures. **Journal of Sound and Vibration**, v. 333, n. 5, p. 1269–1280, 2014.

Anton, H.; Rorres, C. **Algebra Linear com aplicacoes**. 10th edition. ed. [S.l.]: Bookman, 2017.

Balageas, D.; Fritzen, C.-P.; Güemes, A. **Structural Health Monitoring**. [S.l.]: John Wiley and Sons, 2010.

Barreto, G. **Modelagem Computacional Distribuída e Paralela de Sistemas e de Séries Temporais Multivariáveis no Espaço de Estados**. Tese (Degree of Doctor in Electrical Engineering) — University of Campinas, Brazil, 2002.

Bates, D.; Eddelbuettel, D. Fast and elegant numerical linear algebra using the RcppEigen package. **Journal of Statistical Software, Articles**, v. 52, n. 5, p. 1–24, 2013.

Brent, R. P.; Luk, F. T. The solution of singular-value and symmetric eigenvalue problems on multiprocessor arrays. **Society for Industrial and Applied Mathematics**, 1985.

Caicedo, J.; Dyke, S.; Johnson, E. A. Natural excitation technique and eigensystem realization algorithm for phase I of the IASC-ASCE benchmark problem: Simulated data. **Journal of Engineering Mechanics**, v. 130, n. 1, p. 49–60, 2004.

Chen, C. T. **Linear System Theory and Design**. [S.l.: s.n.], 1995.

Chen, H.-P.; Ni, Y.-Q. **Structural Health Monitoring of Large Civil Engineering Structures**. [S.l.: s.n.], 2018.

Chen, J.; Qiu, Q.; Han, Y.; Lau, D. Piezoelectric materials for sustainable building structures: Fundamentals and applications. **Renewable and Sustainable Energy Reviews**, v. 101, p. 14 – 25, 2019.

Chiang, D.-Y.; Lin, C.-S.; Su, F.-H. Identification of modal parameters from ambient vibration data by modified eigensystem realization algorithm. **Journal of Aeronautics, Astronautics and Aviation**, v. 42, n. 2, p. 79–86, 2010.

De Cock, K. **Principal Angles in System Theory, Information Theory and Signal Processing**. 2002.

Daraji, A. H.; Hale, J. M.; Ye, J. New methodology for optimal placement of piezoelectric sensor/actuator pairs for active vibration control of flexible structures. **Journal of Vibration and Acoustics**, v. 140, 2018.

Datta, B. N. **Numerical Linear Algebra and Applications**. 2nd edition. ed. [S.l.]: Society for Industrial and Applied Mathematics (SIAM), 2009.

Demmel, J. W. **Applied Numerical Linear Algebra**. 1st. ed. [S.l.]: Society for Industrial and Applied Mathematics (SIAM), 1997.

Du, H.; Y., S. K.; J., L. Semi-Active H_∞ Control of vehicle suspension with Magneto-Rheological Dampers. **Journal of Sound and Vibration**, n. 283, p. 981–996, 2005.

Eigen. **Using BLAS/LAPACK from Eigen**. 2021. "<https://eigen.tuxfamily.org/dox-devel/TopicUsingBlasLapack.html>. Acesso em: 19 apr. 2021.

Eldén, L. **Matrix Methods in Data Mining and Pattern Recognition**. [S.l.]: SIAM, 2007.

Eray, H.; Temizel, A. **Performance Analysis of Noise Subspace-based Narrowband Direction-of-Arrival (DOA) Estimation Algorithms on CPU and GPU**. 2020.

Fernandes, L. G. L. **Programação Paralela Distribuída**. 2020. "<https://www.inf.pucrs.br/~gustavo/disciplinas/ppd/material/aula-sockets.pdf>. Acesso em: 15 jan. 2021.

Fisco, N.; Adeli, H. Smart structures: Part I — Active and semi-active control. **Scientia Iranica**, v. 18, n. 3, p. 275 – 284, 2011.

Gawronski, W. K. **Advanced Strutural Dynamics and Active Control of Structures**. 1st edition. ed. [S.l.]: Springer, 2004.

Genari, H.; Mechbal, N.; Coffignal, G.; Nóbrega, E. Damage-Tolerant Active Control using a Modal H_∞-Norm-based methodology. **Control Engineering Practice**, v. 60, p. 76–86, 2017.

Genari, H.; Mechbal, N.; Coffignal, G.; Nóbrega, E. A Modal H_∞-Norm-Based performance requirement for Damage-Tolerant Active Controller Design. **Journal of Sound and Vibration**, v. 394, p. 15–30, 2017.

Genari, H.; Mechbal, N.; Coffignal, G.; Nóbrega, E. A reconfigurable Damage-Tolerant Controller based on a Modal Double-Loop framework. **Mechanical Systems and Signal Processing, Elsevier**, n. 88, p. 334–353, 2017.

Genari, H.; Oliveira Neto, O.; Nóbrega, E.; Mechbal, N.; Coffignal, G. Robust vibration control of a vertical flexible structure subject to seismic events. In: **XVII International Symposium on Dynamic Problems of Mechanics-DINAME2015**. Natal, Brazil: [s.n.], 2015.

Genari, H. F. G. **Métrica baseada em projeção de modelos para detecção de danos em estruturas**. 91 p. Dissertação (Mestrado em Engenharia Mecânica) — Universidade Estadual de Campinas, Campinas, 2012.

Genari, H. F. G. **Métodos de Controle Modal Tolerante a Danos para Estruturas Flexíveis**. Tese (Degree of Doctor in Mechanical Engineering) — University of Campinas, Brazil, 2016.

Genari, H. F. G.; Nóbrega, E.; Mechbal, N. Structural damage diagnosis method based on subspace identification metric. In: **22nd International Congress of Mechanical Engineering (COBEM 2013)**. Ribeirão Preto, SP, Brazil: [s.n.], 2013.

Golub, G. H.; Kahan, W. Calculating the singular values and pseudo-inverse of a matrix. **Journal of the Society for Industrial and Applied Mathematics**, v. 2, n. 2, p. 205–224, 1965.

Golub, G. H.; Loan, C. F. V. **Matrix Computations**. 4th. ed. [S.l.]: The Johns Hopkins University Press, 2013.

He, T.; Zhu, G. G.; Swei, S. S.; Su, W. Optimal Sensor Placement for Vibration Control of a Flexible Aircraft Wing. In: **IEEE Conference on Control Technology and Applications (CCTA)**. Hong Kong, China: [s.n.], 2019.

Hoagg, J.; Lacy, S.; Babuska, V.; Bernstein, D. Sequential multisine excitation signals for system identification of large space structures. In: . [S.l.: s.n.], 2006. v. 2006, p. 6 pp.

Hoagg, J. B.; Lacy, S. Sequential multisine excitation signals for system identification of large space structures. **American Control Conference**, 2006.

Ikeno, H. MXPFIT: A library for finding optimal multi-exponential approximations. **Computer Physics Communications**, v. 230, p. 135 – 144, 2018.

Juang, J. N.; Pappa, R. S. An eigensystem realization algorithm for modal parameter identification and model reduction. **Journal of Guidance, Control and Dynamics**, v. 8, n. 5, p. 620–627, 1985.

MalekzadehFard K Gholami M, R. F. l. M. Free vibration and buckling analyses of cylindrical sandwich panel with Magneto- Rheological fluid layer. **Journal of Sandwich Structures and Materials**, v. 19, n. 4, p. 397–423, 2017.

Khronosgroup. **The open standard for parallel programming of heterogeneous systems**. 2010.

Kim, Y. M. K.; You, K.; You, J.; Paek, S.; Nam, B. LQR Control of Along-Wind Responses of a Tall Building Using Active Tuned Mass Damper. In: **2016 World Congress on Advances in Civil, Environmental, and Materials Research (ACEM16)**. Jeju-do, Korea: [s.n.], 2016.

Klimkhieo, S. **On-line Estimation Approaches to Fault-Tolerant Control of Uncertain Systems**. Tese (Degree of Doctor in Electronic Engineering) — University of Hull, England, 2009.

Liu, L.; Liu Yan-Jun andChen, A.; Tong, S.; Chen, C. L. P. Integral barrier lyapunov function-based adaptive control for switched nonlinear systems. **Science China Information Sciences**, 2020.

Ma, Z.; Ahuja, S.; Rowley, C. W. Reduced-order models for control of fluids using the eigensystem realization algorithm. **Theoretical and Computational Fluid Dynamics**, p. 233–247, 2010.

Martin, R. A metric for arma processes. **Signal Processing, IEEE Transactions on**, v. 48, p. 1164 – 1170, 05 2000.

Martowicz, A.; Sendecki, A.; Salamon, M.; Rosiek, M.; Uhl, T. Application of electromechanical impedance-based SHM for damage detection in bolted pipeline connection. **Nondestructive Testing and Evaluation**, v. 31, n. 1, p. 17–44, 2016.

Mechbal, N.; Nóbrega, E. G. O. Damage Tolerant Active Control: Concept and state of the art. In: **8th IFAC Symposium on Fault Detection, Supervision and Safety of Technical Processes**. Mexico City, Mexico: [s.n.], 2012.

Mechbal, N.; Nóbrega, E. G. O. Adaptive strategy to Damage Tolerant Active Control. In: **9th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes**. Paris, France: [s.n.], 2015.

Mechbal, N.; Nóbrega, E. G. O. Spatial H_∞ approach to damage-tolerant active control. **Structural Control and Health Monitoring.**, v. 22, n. 9, p. 1148–1172, 2015.

Moor, T. A discussion of Fault-Tolerant supervisory control in terms of formal languages. **Annual Reviews in Control**, n. 41, p. 159–169, 2016.

Munshi, A. **The OpenCL Specification**. 2012. Disponível em: <https://www.khronos.org/registry/OpenCL/specs/opencl-1.2.pdf>. Acesso em: 13 dec. 2020.

Na, W. S.; Baek, J. A review of the piezoelectric electromechanical impedance based structural health monitoring technique for engineering structures. **Sensors**, v. 18, n. 5, 2018.

Ogata, K. **Discrete-time control systems**. 2nd edition. ed. [S.l.]: Prentics-Hall, Inc, 1994.

Ogata, K. **Modern Control Engineering**. 5th edition. ed. [S.l.]: Pearson Education, 2009.

Olver, P.; Shakiban, C. **Applied Linear Algebra**. 2nd. ed. [S.l.]: Springer, 2018.

Ortolano, F.; Genari, H.; Nóbrega, E. G. O. Controle Ativo de Vibrações por realimentação de estados de uma estrutura vertical flexível sujeita a eventos sísmicos. In: **X Congresso Nacional de Engenharia Mecânica**. Salvador, Brazil: [s.n.], 2018.

Pintelon, R.; Schoukens, J. **System Identification: A Frequency Domain Approach**. 2nd. ed. [S.l.]: John Wiley and Sons, 2012.

Press, H.; Teukolsky, A.; Vetterling, T.; Flannery, B. **Numerical Recipes: The Art of Scientific Computing**. 3rd edition. ed. [S.l.]: Cambridge University Press, 2007.

Press, W.; Teukolsky, S. A.; T., V. W.; B.P., F. **Numerical Recipes Web note No.2, Rev. 1 - SVD Implementation**. 2007. Disponível em: <http://numerical.recipes/webnotes/nr3web2.pdf>. Acesso em: 13 dec. 2020.

Preumont, A. **Vibration Control of Active Structures: An Introduction**. 3rd edition. ed. [S.l.]: Springer, 2011.

Souza, P. R. d.; Nóbrega, E. G. d. O. A Fault location method using lamb waves and discrete wavelet transform. **Journal of the Brazilian Society of Mechanical Sciences and Engineering.**, v. 346, n. 4, p. 515–524, 2012.

Spencer, B.; Suhardjo, J.; Sain, M. Frequency domain optimal control strategies for a seismic protection. **Journal of Engineering Mechanics**, v. 120, p. 135–159, 1985.

Stepinski, T.; Manka, M.; Martowicz, A. Interdigital lamb wave transducers for applications in structural health monitoring. **NDT&E International**, v. 86, p. 199 – 210, 2017.

Sun, S.; Yang, J.; Du, H.; Zhang, S.; Yan, T.; Nakano, M.; Li, W. Development of magnetorheological elastomers–based tuned mass damper for building protection from seismic events. **Journal of Intelligent Material Systems and Structures**, v. 29, n. 8, p. 1777–1789, 2018.

Sun, W.; Pan, H.; Gao, H. Filter-based adaptive vibration control for active vehicle suspensions with electrohydraulic actuators. **IEEE Transactions on Vehicular Technology**, v. 65, n. 6, p. 4619–4626, 2016.

Tabbache, B.; Rizoug, N.; Benbouzid, M.; Kheloui, A. A control reconfiguration strategy for post-sensor FTC in induction motor-based evs. **Annual Reviews in Control**, n. 62, p. 965–971, 2012.

Tang, D.; Chen, J.; Wu, W.; Jin, L.; Yue, Q.; Xie, B.; Wang, S.; Feng, J. Research on sampling rate selection of sensors in offshore platform SHM based on vibration. **Applied Ocean Research**, v. 101, p. 102192, 2020.

Trefethen, L. N.; Bau, D. **Numerical Linear Algebra**. [S.l.]: Society for Industrial and Applied Mathematics (SIAM), 1997.

Tsogka, C.; Daskalakis, E.; Comanducci, G.; Ubertini, F. The stretching method for vibration-based structural health monitoring of civil structures. **Computer-Aided Civil and Infrastructure Engineering**, v. 32, n. 4, p. 288–303, 2017.

Waidyasooriya, H. M.; Hariyama, M.; Uchiyama, K. **Design of FPGA-Based Computing Systems with OpenCL**. [S.l.]: Springer, 2018.

Wang, P.; Kozim, F.; Amini, F. Vibration control of tall buildings. **Engineering Structures**, p. 282–288, 1983.

Wang, Y.; Lee, J.; Ding, Y.; Li, P. A Scalable FPGA Engine for Parallel Acceleration of Singular Value Decomposition. In: **2020 21st International Symposium on Quality Electronic Design (ISQED)**. [S.l.: s.n.], 2020. p. 370–376.

Wu, S.-T.; Shao, Y.-J. Adaptive vibration control using a virtual-vibration-absorber controller. **Journal of Sound and Vibration**, v. 305, n. 4, p. 891–903, 2017.

Yu, W.; Thenozhi, S.; Li, X. Stable Active Vibration Control System for Building Structures using PD/PID Control. In: **19th IFAC World Congres**. Cape Town, South Africa: [s.n.], 2014.

Yu, X.; Jiang, J. A survey of Fault-Tolerant Controllers based on Safety-Related issues. **Annual Reviews in Control**, n. 39, p. 46–57, 2015.

Zhang, G.; Zhang, H.; Huang, X.; Wang, J.; YU, H.; Graaf, R. Active Fault-Tolerant Control for Electric Vehicles With Independently Driven Rear In-Wheel Motors Against Certain Actuator Faults. **IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY**, v. 24, n. 5, p. 1557–1572, 2016.

Zhang, Y.; Jiang, J. Bibliographical review on reconfigurable fault-tolerant control systems. **Annual Reviews in Control**, v. 32, n. 2, p. 229 – 252, 2008.

Zheng, H.; Mita, A. Damage indicator defined as the distance between ARMA models for structural health monitoring. **Structural Control and Health Monitoring**, v. 15, n. 7, p. 992–1005, 2008.