UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
ESCOLA DE ENGENHARIA
DEPARTAMENTO DE ENGENHARIA ELÉTRICA
PROGRAMA DE PÓS-GRADUAÇÃO EM ENGENHARIA ELÉTRICA

ELISANDRA PAVONI LAZZARETTI

Avaliação de Desempenho de Implementações em Hardware e Software de Algoritmos para Aplicações de Manutenção Inteligente

Porto Alegre
2012

ELISANDRA PAVONI LAZZARETTI

Avaliação de Desempenho de Implementações em Hardware e Software de Algoritmos para Aplicações de Manutenção Inteligente

Dissertação de mestrado apresentada ao Programa de Pós-Graduação em Engenharia Elétrica, da Universidade Federal do Rio Grande do Sul, como parte dos requisitos para a obtenção do título de Mestre em Engenharia Elétrica.

Área de concentração: Controle e Automação

ORIENTADOR: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre

2012

ELISANDRA PAVONI LAZZARETTI

Avaliação de Desempenho de Implementações em Hardware e Software de Algoritmos para Aplicações de Manutenção Inteligente

Esta dissertação foi julgada adequada para a obtenção do título de Mestre em Engenharia Elétrica e aprovada em sua forma final pelo Orientador e pela Banca Examinadora.

Orientador: _____

Prof. Dr. Carlos Eduardo Pereira, UFRGS

Doutor pela Universidade de Stuttgart – Stuttgart, Alemanha

Banca Examinadora:

Prof. Dr. João César Netto, PPGC-UFRGS

Doutor pela Université Catholique de Louvain – Louvain-La-Neuve, Bélgica

Prof. Dr. Renato Ventura Bayan Henriques, PPGEE-UFRGS

Doutor pela Universidade Federal de Minas Gerais – Belo Horizonte, Brasil

Prof. Dr. Valner João Brusamarello, PPGEE-UFRGS

Doutor pela Universidade Federal de Santa Catarina– Florianópolis, Brasil

Coordenador do PPGEE: _____

Prof. Dr. João Manoel Gomes da Silva Jr.

Porto Alegre, novembro de 2012.

AGRADECIMENTOS

Agradeço primeiramente aos meus pais, Neiva e Antonio, pelo apoio incondicional durante este período, e por compreender os momentos de ausência.

Agradeço ao meu namorado, Thomás, pelo incentivo, carinho e força durante todo o mestrado. Sem este companheirismo eu não teria chegado até aqui.

Agradeço aos colegas do LASCAR pela ajuda, pelas conversas e pelos momentos de descontração. Agradeço em especial ao ex-professor e agora colega Marcos Zuccolotto pela ajuda, pelas sugestões e revisões, pelas conversas e pela amizade.

Agradeço ao pessoal da empresa Novus Produtos Eletrônicos pelo apoio, brincadeiras e compreensão do horário maluco principalmente durante a reta final do trabalho.

Agradeço ao professor Carlos Eduardo Pereira por ter aceitado me orientar neste trabalho, pelas ideias, apoio e cobranças.

RESUMO

No mercado altamente globalizado de hoje, a manutenção dos equipamentos tem se tornado um fator crucial para as empresas dos mais diversos segmentos. Técnicas de manutenção baseadas no nível de degradação dos equipamentos estão sendo preferidas em detrimento das técnicas tradicionais como manutenção corretiva e preventiva, e trazem benefícios como tempos de paradas reduzidos, tarefas de manutenção facilitadas e melhor gerenciamento de ativos. Com o desenvolvimento das técnicas de manutenção inteligente, os sistemas embarcados que comportarão estes algoritmos necessitarão cada vez mais de alta flexibilidade, combinada com alta velocidade de processamento e baixo consumo. Em outras palavras, eles tornam-se cada vez mais complexos, o que tem impacto direto no projeto destes sistemas. Neste contexto, a programação baseada em modelos em conjunto com a capacidade de geração automática de código para uma dada plataforma tem despertado grande interesse. O presente trabalho tem como objetivo realizar a análise dos espaços de projeto e também do desempenho de diferentes implementações para algoritmos de manutenção inteligente quando executados em hardware e software. A partir de implementações disponíveis nos ambientes MATLAB e LabVIEW™ de um sistema de manutenção inteligente chamado Watchdog Agent™, e utilizando ferramentas de geração automática de código, o desempenho dos sistemas de manutenção gerados é comparado usando-se parâmetros como tempo de execução e ocupação de memória ou da área do FPGA. Para os testes são utilizados dados de vibração coletados de uma bancada de testes composta por um atuador eletromecânico para válvulas.

Palavras-chave: Manutenção Inteligente, Watchdog Agent™, FPGA, Sistemas embarcados, Geração automática de código.

ABSTRACT

In today's highly globalized market, equipment maintenance has become a crucial factor for companies from several segments. Maintenance strategies based on equipment's condition level are being preferred in place of traditional techniques such as corrective and preventive maintenance, and incur in benefits such as reduced downtime, facilitated maintenance tasks and better assets management. With the development of intelligent maintenance techniques, the embedded systems that will be used with such algorithms will need increasingly more flexibility, combined with high processing speed and low power consumption. In other words, they became increasingly more complex, what directly impact in their project. Within this context, model based engineering associated with automatic platform-specific code generation capabilities are of great interest. This work has as objective to perform a design space exploration by analyzing the performance of different implementations for intelligent maintenance algorithms when executed in hardware and software. Based on implementations available in MATLAB™ and LabVIEW™ environments of an intelligent maintenance system called Watchdog Agent, and using automatic code generation tools, the performance of the generated systems are compared using parameters such as execution time and memory or FPGA area occupation. For the validation tests, vibration data collected from a test bench composed by an electric mechanical actuator will be used.

Keywords: Intelligent Maintenance, Watchdog Agent™, FPGA, Embedded systems, Automatic code generation.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ASIC: Application Specific Integrated Circuit

CBM: Condition Based Maintenance

cRIO: CompactRIO

CV: Confidence Value

DWT: Discrete Wavelet Transform

FPGA: Field Programmable Gate Array

HDL: Hardware Description Language

IDE: Integrated Development Environment

GPP: General Purpose Processor

IMS Center: Center for Intelligent Maintenance SystemsLR: Logistic Regression

MDE: Model Driven Engineering

MEX: MATLAB™ Executable

MTBF: Mean Time Between Failures

MTTR: Mean Time To Repair

NRE: Non-Recurring Engineering

OSA-CBM: Open Systems Architecture for Condition-Based Maintenance

PC: Personal Computer

PEID: Product Embedded Information Devices

SoC: System-on-a-Chip

VHDL: VHSIC HDL

VHSIC: Very-high-speed integrated circuit

VI: Virtual Instrument

WA: Watchdog Agent™ Toolbox

WPE: Wavelet Packet Energies

1 INTRODUÇÃO

No mercado altamente competitivo e globalizado de hoje, no qual as empresas devem apresentar alta produtividade, alta qualidade e baixo custo em seus produtos e serviços, a manutenção dos equipamentos tem se tornado um fator crucial para atender a estas exigências (LEE ET AL., 2006; LU, DUROCHER, STEMPER, 2009; NIU; YANG, 2010). Equipamentos degradados podem comprometer a qualidade dos produtos finais, e paradas inesperadas em decorrência de falhas geram um enorme custo para empresas e podem resultar em atrasos no cumprimento dos compromissos com os clientes, além de riscos aos funcionários e ao meio ambiente.

Hoje ainda empregam-se principalmente dois tipos de manutenção: a corretiva e a preventiva. A manutenção puramente corretiva consiste no conserto de um equipamento após uma quebra ou falha. Neste caso não há tempo ou recursos despendidos entre as falhas, porém o custo da manutenção torna-se mais elevado e pode haver graves consequências, como as citadas anteriormente. Na manutenção preventiva realizam-se intervenções periódicas assumindo um determinado nível de degradação dos equipamentos. Mesmo com estas intervenções periódicas ainda podem ocorrer paradas inesperadas por falha. Nos últimos anos, um novo paradigma de manutenção está emergindo, transformando as tradicionais técnicas focadas no conserto da falha para uma manutenção mais focada na prevenção e predição das falhas (YAN, LEE, 2005; KOBBACY ET AL., 2008; LEE, GHEFFARI, ELMELIGY, 2011).

Mesmo quando os equipamentos parecem falhar repentinamente, a degradação sofrida por eles é gradual e geralmente pode ser mensurada. Com base nisso, há um aumento no número de sistemas de manutenção que baseados no nível de degradação do sistema buscam prever suas falhas, sistemas estes que recebem denominações como “sistemas inteligentes de manutenção” (LEE ET AL., 2006). O objetivo de um sistema de manutenção inteligente é realizar a migração dos sistemas tradicionais para um sistema proativo, com base no estado de

operação e na degradação dos equipamentos. Assim, a manutenção inteligente envolve a monitoração do desgaste dos equipamentos, geração de diagnósticos, quantificação da perda de desempenho, e atuação sobre o sistema, fornecendo informações importantes e úteis para a equipe de manutenção e alterando o padrão de comportamento dos equipamentos (quando detectada uma provável falha) até a realização da manutenção (ESPÍNDOLA ET AL., 2010).

Paralelamente a estes desenvolvimentos, também existe um crescente interesse em embarcar estes algoritmos diretamente nos produtos, gerando os chamados "*product embedded information devices*" (PEID) (KIRITSIS, 2011). Devido ao desenvolvimento das técnicas de manutenção inteligente, os sistemas embarcados que comportarão estes algoritmos tornam-se cada vez mais complexos, o que tem impacto direto no projeto destes sistemas. Além disso, em função das características do mercado, o tempo de projeto deve ser cada vez menor. Para o desenvolvimento de um algoritmo em um sistema embarcado, o projetista deve ter em mente a plataforma na qual o mesmo será executado (MOON, SEO, KIM, 2007). Neste contexto, a programação baseada em modelos em conjunto com a capacidade de geração automática de código para uma dada plataforma tem despertado grande interesse. No entanto, ainda não há um consenso sobre a melhor forma de implementação: se em software, em hardware, ou explorando arquiteturas mistas.

O presente trabalho visa avaliar o espaço de projeto para sistemas embarcados de manutenção inteligente, analisando o desempenho de algoritmos quando implementados em hardware ou software, em diferentes arquiteturas-alvo. Para tanto, os algoritmos disponíveis na ferramenta *Watchdog Agent*™ desenvolvida pelo *Center for Intelligent Maintenance Systems* (IMS Center) com o qual o Grupo de Controle, Automação e Robótica (GCAR) da Universidade Federal do Rio Grande do Sul (UFRGS) tem cooperação serão utilizados. A partir de implementações deste sistema disponíveis nos ambientes MATLAB™ e LabVIEW™ e utilizando ferramentas de geração automática de código, parâmetros tais como

tempo de execução dos algoritmos e ocupação da área do FPGA serão analisados. Para os testes serão utilizados dados de vibração coletados de uma bancada de testes composta por um atuador eletromecânico para válvulas. Com estas informações, pode-se rapidamente adaptar o sistema de manutenção a diferentes plataformas, ou mesmo alterar os algoritmos disponíveis no sistema, identificar qual plataforma é mais indicada para uma determinada aplicação ou ainda futuramente desenvolver uma plataforma embarcada reconfigurável, combinando processador e FPGA.

Esta dissertação está organizada da seguinte forma: o Capítulo 2 apresenta uma revisão das técnicas de manutenção. São apresentados a classificação adotada para as diferentes estratégias de manutenção e alguns conceitos importantes relacionados ao tema. O Capítulo 3 trata das diferentes arquiteturas de sistemas embarcados, bem como de técnicas de projeto com FPGAs. O Capítulo 4 faz uma breve revisão do estado da arte das técnicas de manutenção preditiva e proativa, bem como de trabalhos relacionados à geração automática de código. No Capítulo 5 trata da ferramenta de manutenção inteligente utilizada neste trabalho, apresentado sua estrutura e os algoritmos utilizados no estudo de caso. No Capítulo 6 é apresentada uma visão geral da proposta da presente dissertação, além da descrição do estudo de caso utilizado neste trabalho. Também é abordado o uso das ferramentas de geração automática de código para o embarque dos algoritmos. Por fim no Capítulo 7 os resultados obtidos são apresentados e discutidos.

2 TÉCNICAS DE MANUTENÇÃO

Em praticamente todos os processos produtivos, a qualidade do produto final está intimamente ligada com o desempenho da máquina ou equipamento que o fabrica (LIMA, MARCORIN, 2003). Com o uso, os equipamentos ficam sujeitos a diversos tipos de degradação, tais como desgastes, poeira, falta de lubrificação, desalinhamento, etc. Esta deterioração gera desvios no processo e diminuição da qualidade do produto final. A ocorrência de falhas graves, que implicam na impossibilidade de operação dos equipamentos e levam à paradas no processo produtivo, ocasiona redução da produtividade e, por consequência, um alto custo para a empresa.

Estas situações podem ser minimizadas com uma manutenção adequada, que garanta o bom funcionamento dos equipamentos. As formas de manutenção mais difundidas são a corretiva e a preventiva, mas com o atual desenvolvimento tecnológico, outras formas vêm ganhando espaço: a manutenção preditiva e a manutenção proativa.

Antes de abordar as características das diferentes técnicas de manutenção, é importante conhecer algumas definições:

- **Função requerida**: É o conjunto de condições de funcionamento para o qual um dispositivo foi projetado, fabricado ou instalado, em outras palavras, condições de funcionamento consideradas necessárias para prover um dado serviço (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 1994).
- **Confiabilidade**: É a habilidade de um sistema ou componente de realizar sua função requerida sob determinadas condições por um período especificado de tempo (IEEE, 1991).
- **Disponibilidade**: É a capacidade de um sistema ou componente estar em condições de executar uma determinada função em um dado instante ou durante um intervalo de tempo determinado, supondo que os recursos externos

requeridos esteja assegurados (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 1994).

- **Falha**: Evento que determina o término da capacidade de um dispositivo de desempenhar sua função requerida (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 1994).
- **Defeito**: É qualquer desvio de uma característica de um dispositivo em relação a seus requisitos. Um defeito pode ou não afetar a capacidade de desempenhar a função requerida (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 1994).
- **Tempo médio entre falhas**: Também conhecido pela sigla MTBF (*Mean Time Between Failures*). É o valor médio dos intervalos de tempo entre falhas consecutivas sob determinadas condições, para um dado período na vida útil de um dispositivo. É utilizado para mesurar a confiabilidade de um equipamento – quanto maior o MTBF, mais confiável é o equipamento (IEEE, 1995).
- **Tempo médio para reparo**: Também conhecido pela sigla MTTR (*Mean Time To Repair*). Para um determinado período da vida útil de um dispositivo, é o tempo médio necessário para realizar uma manutenção corretiva. É utilizado como uma medida da complexidade e da modularidade de um equipamento – quanto maior o MTTR, mais complexo é o equipamento (IEEE, 1995).

Existem diversas classificações das técnicas de manutenção, organizando-as de acordo com com diferentes critérios, tais como pela ocorrência de falhas ou pelo funcionamento dos sistemas. Neste trabalho considera-se a classificação das estratégias mostrada na

Figura 1 (GONÇALVES, 2011):

Figura 1 Classificação das técnicas de manutenção (GONÇALVES, 2011)

2.1 MANUTENÇÃO CORRETIVA

A manutenção corretiva corresponde às medidas tomadas após uma quebra ou falha que impeça o funcionamento adequado do equipamento (IEEE, 1995). Estas medidas podem ser de caráter *paliativo*, quando são provisórias, visando restabelecer o funcionamento mínimo aceitável para depois efetuar um reparo definitivo ou *curativo*, quando são definitivas, restabelecendo completamente a função requerida do equipamento (GONÇALVES, 2011). Neste caso, não há nenhum gasto adicional entre as quebras, nem paradas para monitoramento periódico. No entanto, para uma manutenção mais eficiente é preciso ter as peças para reposição em estoque e em quantidade adequada.

Esta política de manutenção impossibilita um planejamento de custos, visto que as quebras e suas consequências não são previstas, por exemplo, podem ocorrer danos em outros equipamentos de um sistema por causa de uma quebra. Além disso, ela pode implicar em paradas inaceitáveis no processo produtivo. Entretanto, a manutenção corretiva pode ser a mais indicada quando o equipamento em questão não for essencial ao processo, quando a falha pode ser identificada e/ou corrigida rapidamente, e quando as consequências e os custos de uma falha não forem significativos. Em outras palavras, é a política indicada quando os

custos da indisponibilidade do equipamento são inferiores aos custos necessários para evitar uma falha (LIMA, MARCORIN, 2003).

2.2 MANUTENÇÃO PREVENTIVA

A manutenção preventiva corresponde a um conjunto de práticas periódicas, tais como lubrificação, limpeza, ajustes, e troca de peças ou equipamentos, que visam evitar as falhas ou quebras (IEEE, 1995). Ela é baseada em intervenções periódicas, que geralmente são programadas, seguindo uma frequência definida pelo fabricante do equipamento (como por exemplo, baseada no *Mean Time Between Failures* - MTBF) ou pela experiência adquirida da empresa (históricos de manutenção e dados estatísticos), com o objetivo de reduzir a probabilidade de falha do equipamento. Neste caso, da mesma forma que ocorre na manutenção corretiva, é necessário ter um estoque de peças de reposição para que a prática seja eficiente.

Um dos inconvenientes da manutenção preventiva são as constantes intervenções, que por vezes podem ser desnecessárias. Em função das variações que normalmente ocorrem nos materiais utilizados na sua confecção, dificilmente dois equipamentos vão apresentar os mesmos problemas ou o mesmo tempo de funcionamento antes de uma falha, ainda que eles sejam nominalmente idênticos. Além disso, como o desgaste sofrido depende do uso do equipamento (frequência de trabalho e aplicação), é possível que a substituição de uma peça que ainda teria muito tempo de vida útil seja necessária, por exemplo, caso a peça seja utilizada com uma frequência maior do que usualmente previsto. Isto pode reduzir a produtividade, em função das paradas, e eleva o custo total do processo. Também pode ocorrer uma quebra antes do tempo previsto, ou seja, esta técnica não descarta a possibilidade de uma parada para manutenção corretiva.

Esta política de manutenção é realmente eficiente em equipamentos ou peças que

sofram algum tipo de degradação em um ritmo uniforme e conhecido, e para as quais os custos de uma quebra sejam altos quando comparados aos custos da sua manutenção (LIMA, MARCORIN, 2003).

2.3 MANUTENÇÃO PREDITIVA

A manutenção preditiva, também chamada de Manutenção Baseada na Condição (do inglês "*Condition Based Maintenance*"), caracteriza-se pela monitoração constante de sinais provenientes de um equipamento ou da eficiência de um processo, a fim de garantir uma qualidade de serviço desejada e reduzir ao mínimo as ações de manutenção preventiva e diminuir a manutenção corretiva (ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS, 1994). Os sinais de interesse são aqueles que, de alguma forma, trazem informações que possam auxiliar na identificação de uma provável falha. Em outras palavras, buscam-se informações que permitam caracterizar o estado atual do sistema e dar indicações da evolução do seu desempenho.

Estes sinais ou variáveis podem já estar disponíveis no equipamento, ou pode ser necessário instrumentá-lo para a implantação de um sistema de aquisição. Entre os sinais que podem ser utilizados destacam-se: vibrações mecânicas, ruído e temperatura. Sinais de vibração são bastante utilizados para detectar problemas mecânicos em máquinas rotativas. Vazamentos em tubulações produzem um som particular, que pode ser detectado por ultra-som. Já a termografia é aplicada para detectar um aquecimento anormal de algum componente.

O custo de implementação desta política de manutenção é normalmente bastante elevado, uma vez que implica na instalação da infraestrutura necessária à monitoração dos equipamentos. Desta forma, ela torna-se mais vantajosa nos equipamentos mais críticos do processo, para os quais o tempo para manutenção (MTTR) e/ou o custo da sua

indisponibilidade são altos. Normalmente faz-se um estudo detalhado dos pontos críticos do processo a fim de determinar os dispositivos que deverão seguir esta estratégia, analisando o histórico de manutenção e avaliando a frequência e o tempo de parada associado a cada falha (LEE ET AL., 2009; GONÇALVES, 2011).

A maior parte dos sistemas de manutenção baseada na condição envolve a estimação do estado de um equipamento com base no reconhecimento de indicadores de falha, utilizando ferramentas como redes neurais, lógica Fuzzy e modelos estocásticos (LEE ET AL., 2006). Estas abordagens necessitam de uma grande quantidade de informação a priori sobre a máquina ou processo, pois as falhas devem ser conhecidas e descritas para serem identificadas (DJURDJANOVIC, LEE, NI, 2003).

2.4 MANUTENÇÃO PROATIVA

A manutenção proativa, também chamada de manutenção inteligente ou *e-maintenance* (LEE ET AL., 2010), envolve a monitoração do desgaste dos equipamentos ou processos, geração de diagnósticos, quantificação da perda de desempenho, e atuação sobre o sistema, fornecendo informações importantes e úteis para a equipe de manutenção e alterando o padrão de comportamento dos equipamentos (quando detectada uma provável falha) até a realização da manutenção.

Com um sistema proativo pode-se identificar desgaste de peças muito antes da quebra, evitando que outras partes do processo sejam também afetadas. Além disso, saber com antecedência qual peça deve ser consertada ou substituída facilita e acelera o trabalho de manutenção. Um exemplo desta capacidade pode ser vista no trabalho apresentado por (ESPÍNDOLA ET AL., 2010), no qual são abordadas técnicas de realidade mista para orientar o trabalho de manutenção, visando guiar o operador de forma segura durante suas atividades.

Um sistema proativo também permite o gerenciamento de ativos e das operações de

manutenção de uma forma mais eficiente. Entre os exemplos encontrados na literatura pode-se citar o trabalho apresentado por (YANG, DJURDJANOVIC, NI, 2008), onde procedimentos de otimização baseados em algoritmos genéticos são utilizados para buscar o agendamento de maior custo-benefício das atividades de manutenção, considerando ganhos de produção e gastos na manutenção. No trabalho apresentado por (MOORE, STARR, 2006) as tarefas de manutenção são priorizadas utilizando uma estratégia chamada Criticidade Baseada em Custo (CBC - *Cost-Based Criticality*), levando em consideração a probabilidade de falha, informações de custo e fatores de risco, de forma rápida e sem necessidade de basear-se na experiência de um operador. Este tratamento é importante quando, por exemplo, os recursos para a realização das tarefas são limitados.

O conhecimento do real estado dos equipamentos também pode permitir o aumento da vida útil de peças que antes seriam descartadas em manutenções preventivas, reduzindo os desperdícios e preservando os recursos naturais. Essas informações podem ser levadas ao fabricante do equipamento, permitindo que este, baseado nos dados reais de operação, possa melhor avaliar e aperfeiçoar os seus projetos, fechando o laço de gerenciamento de ciclo de vida dos produtos (DJURDJANOVIC, LEE, NI 2003; KIRITSIS, 2011).

A implementação desta estratégia de manutenção possui um espaço de projeto grande, como a escolha de sensores, ferramentas para processamento de sinais, definição da plataforma na qual o sistema será implementado, viabilidade técnica e econômica e treinamento da equipe de manutenção (DJURDJANOVIC, LEE, NI, 2003). Algumas iniciativas vêm sendo tomadas no sentido de se criar uma forma mais genérica para realizar as tarefas de monitoramento de desempenho e estimação do estado do sistema, que possa ser facilmente adaptada para diferentes aplicações ou diferentes modos de uso dos equipamentos. Uma análise extensiva dos trabalhos realizados em âmbito acadêmico e industrial para manutenção baseada em condição é apresentado em (MULLER, MARQUEZ, IUNG, 2008).

Em uma destas inciativas foi criado nos Estados Unidos um centro de parceria entre universidades e empresas, chamado de Center for Intelligent Maintenance Systems (IMS Center), que tem por objetivo auxiliar na migração do paradigma de "falha e conserta" para o paradigma de "predizer e prevenir" (LEE ET AL., 2006; LEE, GHEFFARI, ELMELIGY, 2011). Na abordagem proposta pelo IMS Center, o estado do sistema é mensurado através de um indicador da semelhança entre o comportamento normal e o comportamento recentemente observado, chamado de *Confidence Value*, ou Valor de Confiança (CV). Para tanto, utiliza-se um conjunto de ferramentas de análise e prognóstico de falhas denominado *Watchdog Agent*™ (WA), o qual será abordado no Capítulo 5.

3 ARQUITETURAS DE SISTEMAS EMBARCADOS PARA MANUTENÇÃO INTELIGENTE

Com o desenvolvimento das técnicas de manutenção inteligente, os sistemas embarcados que comportarão os algoritmos necessitarão cada vez mais de alta flexibilidade, combinada com alta velocidade de processamento e baixo consumo. Em outras palavras, eles tornam-se cada vez mais complexos, o que tem impacto direto no projeto destes sistemas.

Diferentemente de um computador de uso geral, o projeto de um sistema embarcado é consideravelmente mais complexo. Ele é uma atividade interdisciplinar, que envolve diferentes áreas de pesquisa. Além de custo de tempo de projeto, outras restrições tais como memória disponível limitada, requisitos de consumo, transformam o projeto em um desafio. Como não há uma plataforma de certa forma padronizada para estes sistemas, como o que o ocorre com os computadores pessoais, por exemplo, tem-se ainda mais graus de liberdade no projeto (GÖTZ, 2007).

Por outro lado, com o rápido desenvolvimento tecnológico dos últimos anos, as plataformas de hardware e software oferecem melhores desempenhos a menores custos (JARITZ, 2012; CARRO, 2001). Acompanhando esta tendência, as plataformas tem se tornado cada vez mais complexas e tempo de projeto deve ser cada vez menor. Para o desenvolvimento de um algoritmo em um sistema embarcado, o projetista deve ter em mente a plataforma na qual o mesmo será executado (MOON ET AL., 2007). Neste contexto, a engenharia baseada em modelos (MDE – *Model driven Engineering*) em conjunto com a capacidade de geração automática de código para uma dada plataforma tem despertado grande interesse.

3.1 PROCESSADORES

Os processadores de propósito geral (GPPs) apresentam baixo custo e facilidade de

implementação, além de oferecerem alta flexibilidade pela sua capacidade de programação. No entanto, como não são especificamente projetados para a tarefa na qual serão utilizados, podem muitas vezes perder em desempenho. Isto se deve ao fato de que as tarefas são executadas de forma sequencial, e o tempo para execução das tarefas é limitada pelo número de ciclos utilizados para carregar e decodificar as instruções e para leitura e escrita de dados na memória (STECHELE ET AL., 2004; ALLGAYER, 2009).

Entre as vantagens operacionais da utilização de GPPs está o fator de escala. Como os microprocessadores são encontrados em milhares de projetos, seu custo dilui-se entre muitos clientes, e por vezes até competidores entre si. Além disso, uma vez que uma plataforma baseada em processador esteja disponível dentro de uma empresa, novas versões de produtos ou mesmo novos produtos, podem ser criadas apenas pela alteração do software desta plataforma. A personalização do sistema dá-se através do software de aplicação, que é a etapa que atualmente toma a maior parte do tempo de projeto. Além destas vantagens competitivas, há ainda o fator treinamento de engenheiros, já que estes geralmente se formam com conhecimentos de programação de microprocessadores (CARRO, WAGNER, 2003).

3.2 ASICs

Circuitos integrados de aplicação específica (ASICs) são utilizados em aplicações onde a necessidade de ter alto desempenho justifica os custos envolvidos na fabricação do circuito. ASICs podem conter centenas de milhares de portas lógicas e podem ser utilizados para realizar funções grandes e complexas (MAXFILED, 2004). Eles são projetados para atender a uma aplicação específica, oferecendo, portanto, algumas vantagens com relação aos GPPs, uma vez que são otimizados para realizar tais tarefas. Isto resulta em um reduzido tempo de execução e baixo consumo, mas não permite nenhum grau de flexibilidade: se for necessário alterar a sua funcionalidade, deve-se construir um novo circuito integrado.

Em muitas aplicações, é adequada a integração do sistema em uma única pastilha, configurando o que se chama de SoC (*System-on-a-Chip*). Em situações onde requisitos de área, potência e desempenho sejam críticos, o projeto do SoC na forma de um ASIC pode ser mandatório. No entanto, o projeto e o processo de fabricação são demorados e caros (MAXFIELD, 2004). Um dos problemas no uso de ASICs diz respeito aos custos de engenharia não-recorrentes (NRE – *Non-Recurring Engineering*), ou seja, aqueles gastos com pesquisa e desenvolvimento que ocorrem uma única vez durante o projeto, uma vez que o reaproveitamento dos circuitos desenvolvidos é baixo (CARRO, 2001).

3.3 FPGAs

FPGAs (*Field Programmable Gate Arrays*) são componentes que contém blocos de lógica programável em conjunto com interligações também programáveis entre estes blocos (MAXFIELD, 2004). Também podem conter barramentos para dados e memória distribuída. Eles podem implementar um ASIC ou uma arquitetura de processador em um dispositivo configurável (STECHELE ET AL., 2004).

Pela sua capacidade de agregar a flexibilidade como os GPPs e a customização de hardware dos ASICs, os FPGAs ocupam uma posição intermediária entre eles (ALLGAYER, 2009; GÖTZ, 2007), como é ilustrado na Figura 2.

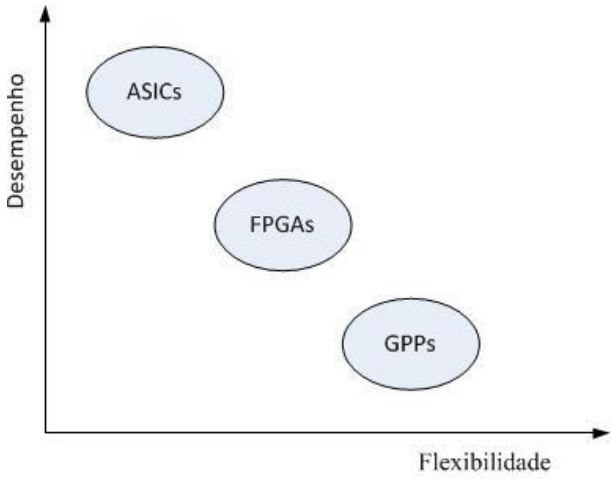

Figura 2 Relação entre desempenho e flexibilidade (ALLGAYER, 2009)

Os FPGAs podem ser configurados diversas vezes após sua fabricação, sendo possível implementar circuitos digitais tais como processadores, interfaces, controladores, etc. A arquitetura interna de um FPGA pode variar de acordo com o fabricante, mas de forma geral é constituída por:

- Blocos lógicos programáveis (CLBs - *Configuration Logical Blocks*): Implementam toda a lógica combinacional e sequencial de um circuito. São constituídos de flip-flops e por lógicas combinacionais.
- Blocos de entrada e saída (IOBs - *Input Output Blocks*): são circuitos responsáveis pela interface entre os CLBs e o exterior do FPGA e o barramento de interconexões. São compostos por buffers bidirecionais.
- Interconexões programáveis: são trilhas ou barramentos para conectar as entradas e saídas dos CLBs e IOBs. O processo de escolha das interconexões e chamado de roteamento. Cada fabricante de FPGA determina o número disponível de interconexões.
- Blocos de memória (BRAM): em alguns modelos de FPGA tem-se a disposição um banco de memória para implementação de funções lógicas mais complexas ou apenas armazenamento de dados.

Atualmente existe no mercado uma grande variedade de modelos de FPGAs de diversos fabricantes. Entre os principais fabricantes pode-se citar Altera, Actel, Xilinx, entre outros. Estes oferecem dispositivos com capacidades de programação distintas, como programação total ou parcial, e características específicas, tais como desempenho, consumo de energia, testabilidade e arranjos de interconexões (BOSA, 2009). Além disso, os FPGAs oferecem funcionalidades como blocos de DSP (*Digital Signal Processor*) e implementações de microcontroladores em lógica programável. Atualmente alguns modelos oferecem

integrados no mesmo encapsulamento o núcleo físico de um processador, configurando um SoC.

Os FPGAs podem ser utilizados para implementar praticamente qualquer projeto de hardware, sendo que um dos seus usos mais comuns é a utilização para a prototipação de ASICs. No entanto, devido ao seu baixo custo de inserção no mercado, ele tem sido cada vez mais utilizado diretamente no produto final (BOSA, 2009).

Arquiteturas reconfiguráveis com processadores, FPGAs e memória embarcados visam combinar a flexibilidade por programação dos microprocessadores com o baixo tempo de execução das tarefas e baixo consumo dos ASICs (STECHELE ET AL., 2004).

3.3.1 Linguagens de descrição de hardware

As linguagens de descrição de hardware, ou HDLs (*Hardware Description Languages*), foram desenvolvidas a partir da necessidade de criação de uma forma simples e completa de se especificar um sistema digital, já se prevendo que o gargalo no desenvolvimento de novos produtos seria o tempo de projeto (CARRO, WAGNER, 2003). Existem diversas linguagens desenvolvidas com capacidade para descrever a especificação e implementação de circuitos digitais, como por exemplo, VHDL, Verilog, ABEL (*Advanced Boolean Equation Language*), HardwareC, XNF (*Xilinx Netlist Format*) e AHDL (*Altera Hardware Description Language*) (ALLGAYER, 2009).

O VHDL (*VHSIC Hardware Description Language*) permite descrições tanto em baixo nível quanto em nível mais abstrato de comportamento. Como originalmente esta linguagem foi desenvolvida para descrição e simulação de sistemas eletrônicos, e não para síntese, ela é muito mais rica do que o subconjunto que normalmente é utilizado pelos fabricantes (CARRO, 2001).

Entres as vantagens do uso de uma linguagem de descrição de hardware pode-se citar

(CARRO, 2001):

- HDLs permitem maior poder de abstração ao projetista, portanto, projetos de maior complexidade são facilitados;
- O fato de se utilizar síntese torna o projeto praticamente independente de tecnologias de fabricação;
- As HDLs servem também para a manutenção de projetos, pois é mais facilmente compreensível do que um esquemático correspondente;
- O uso da síntese melhora a produtividade. O resultado final difere pouco com relação ao obtido por um projetista humano, possibilitando que várias soluções possam ser testadas em um curto período de tempo.

Entre as desvantagens que permanecem ou aparecem ao utilizar este tipo de linguagem pode-se citar (CARRO, 2001):

- O investimento inicial no treinamento dos projetistas;
- Ainda não é possível sintetizar circuitos analógicos e mistos;
- Não é uma solução para todo tipo de projeto.

3.4 TÉCNICAS DE PROJETO DE SISTEMAS EMBARCADOS

O projeto de um sistema embarcado é extremamente complexo, por envolver conceitos pouco analisados pela computação de propósitos gerais (CARRO, WAGNER, 2003). Por exemplo, as questões da portabilidade e do limite de consumo de potência sem perda de desempenho, a baixa disponibilidade de memória, a necessidade de segurança e confiabilidade, a possibilidade de funcionamento em uma rede maior, e o curto tempo de projeto tornam o desenvolvimento de sistemas computacionais embarcados uma área em si.

O projeto de sistemas eletrônicos embarcados enfrenta diversos desafios, uma vez que o espaço de projeto arquitetural a ser explorado é muito vasto. A arquitetura de hardware de

um SoC embarcado pode conter um ou mais processadores, memórias, interfaces para periféricos e blocos dedicados. Além do grande tempo que pode ser gasto com uma exploração sistemática deste espaço de projeto, deve-se considerar ainda o tempo necessário para o projeto e validação individual de todos os componentes dedicados do sistema – processadores, blocos de hardware, rotinas de software, RTOS (Sistema Operacional de Tempo-Real) – assim como o tempo de validação de sua agregação dentro de um mesmo sistema (CARRO, WAGNER, 2003).

3.4.1 Hardware embarcado em FPGA

O fluxo de projeto com FPGAs é basicamente igual ao de qualquer projeto que utilize lógica semi-custom (CARRO, 2001). A diferença fundamental encontra-se na possibilidade de projeto e teste de um circuito em um curto espaço de tempo. Deve-se ressaltar que uma vez obtido o conjunto de portas do circuito a ser implementado, seja por síntese com VHDL ou através de um esquemático feito por um projetista, a configuração do FPGA é imediata, basta ter-se uma placa apta a receber a programação. A Figura 3 mostra o fluxo de projeto com FPGA.

A primeira etapa do fluxo de projeto é a especificação do circuito, o que pode ser feito através de um esquemático realizado manualmente por um projetista, ou utilizando alguma linguagem de descrição de hardware. Em seguida deve-se executar uma simulação do circuito, a fim de verificar a sua correção. Caso encontrem-se erros, deve-se voltar à etapa inicial para corrigi-los. Uma vez que a simulação corresponde ao comportamento esperado do circuito, parte-se para a escolha do FPGA. Em seguida é realizada a síntese do circuito e assinalamento das entradas e saídas do sistema. O último passo é a programação do dispositivo e verificação do funcionamento.

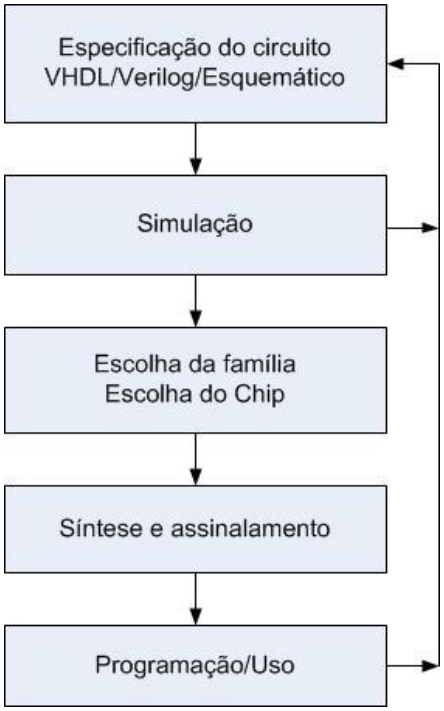

Figura 3 Fluxo de projeto para utilização de FPGA (CARRO, 2001)

Normalmente antes de se partir para o desenvolvimento do hardware, os algoritmos são testados em outra plataforma. Uma possibilidade é desenvolver o algoritmo em ambiente de mais alto nível, tal como o MATLAB, validá-lo e somente então iniciar o procedimento ilustrado anteriormente, como no trabalho realizado por (BOSA, 2009). No trabalho apresentado por (PICCOLI ET AL., 2012), o sistema foi desenvolvido em linguagem C e validado, para futuramente ser implementado em hardware.

A utilização de ferramentas de geração automática de código reduz consideravelmente o tempo de projeto, eliminando a necessidade de realizar a especificação do circuito a; partir do zero depois da etapa de validação dos algoritmos. Com a crescente capacidade dos FPGAs e a necessidade de se colocar os sistemas no mercado rapidamente, uma possível perda de desempenho pode ser aceitável.

4 TRABALHOS RELACIONADOS

Nesta seção será apresentada uma revisão do estado da arte nos tópicos relacionados a este trabalho. Serão abordados trabalhos relacionados com manutenção preditiva e proativa, bem como trabalhos relacionados à geração automática de código encontrados na literatura.

4.1 MANUTENÇÃO PREDITIVA

Como discutido na Seção 2.3, os sistemas de manutenção preditiva necessitam de modelos ou informações a priori sobre os equipamentos fim de serem implantados. Existem diversos exemplos desta estratégia de manutenção na literatura, aplicados nos mais diversos equipamentos. A seguir serão apresentados alguns trabalhos relacionados com a manutenção preditiva:

- Em sistemas de conversão de energia eólica a detecção precoce da degradação é muito importante em função dos geradores estarem em locais de difícil acesso (em uma torre). Para cada tipo de falha são utilizadas técnicas diferentes, mas todas baseadas em modelos do sistema. Entre estas técnicas pode-se citar a análise da densidade espectral de potência que é utilizada para detectar defeitos nas hélices e a análise espectro dos sinais modulantes do rotor para identificar falhas no estator e no rotor (AMIRAT ET AL., 2009).
- Termogramas são utilizados para identificar desgaste em dispositivos de proteção contra surtos utilizados na proteção contra descargas elétricas em sistemas de energia. As imagens são processadas utilizando a transformada Watershed e em seguida uma rede neural faz a classificação entre operação normal, de falha e suspeita (ALMEIDA ET AL., 2009).
- No diagnóstico de falhas relacionadas a curto circuito em motores de indução utiliza-se Análise das Assinaturas da Corrente do Motor (MCSA – *Motor*

Current Signature Analysis) e Lógica Fuzzy. A MCSA possibilita a detecção de diversas falhas comuns em motores, fazendo a decomposição espectral da corrente de regime do estator. As componentes frequenciais que aparecem em caso de curto circuito são conhecidas através de modelos do motor. (PEREIRA ET AL., 2005).

4.2 MANUTENÇÃO PROATIVA

Conforme discutido na Seção 2.4, os sistemas de manutenção de proativa buscam a monitoração do desgaste dos equipamentos, geração de diagnósticos, quantificação da perda de desempenho, e atuação sobre o sistema. Entretanto, observa-se que a maioria dos trabalhos relacionados não foca na implementação prática dos sistemas de manutenção embarcados, restringindo-se ao estudo e validação das técnicas utilizando um computador pessoal, ou então implementando o sistema apenas em software ou apenas em hardware.

- No trabalho apresentado por (GONÇALVES ET AL., 2009) foi desenvolvido um sistema embarcado para detecção e diagnóstico de falhas em um atuador eletromecânico utilizando um processador Microblaze sintetizado em um FPGA para executar o processamento de sinal, e mapas auto-organizáveis foram implementados em hardware no mesmo FPGA para detecção, classificação e predição de falhas. Todo o treinamento dos algoritmos é realizado em um computador pessoal, e o monitoramento do sinal é feito no FPGA.
- O trabalho apresentado por (GONÇALVES, 2011) expande o sistema acima embarcando também filtro adaptativo no FPGA a fim de avaliar qual técnica (mapas auto-organizáveis ou o filtro adaptativo) é mais adequada para embarque na plataforma proposta, em termos de eficiência da identificação de

falha, área do FPGA e tempo de execução.

- Um sistema embarcado para detecção de falhas em atuadores utilizando Transformada Wavelet Discreta e filtros adaptativos LMS (*Least Mean Squares*) foi desenvolvido em linguagem C e validado em software, com o objetivo de futuramente ter um chip (ASIC) embarcado (PICCOLI ET AL., 2012).
- No trabalho apresentado por (LINXIA, LEE, 2010), é abordada uma proposta de sistema de manutenção inteligente com alguma capacidade de reconfiguração. Uma arquitetura baseada em agentes e utilizando um algoritmo baseado em *Quality Function Deployment* (QFD) foi utilizada para reconfigurar as técnicas de avaliação de desempenho de acordo com mudanças do ambiente ou solicitações via uma interface web, no entanto, a aplicação é executada apenas em software, utilizando um computador industrial.

4.3 GERAÇÃO AUTOMÁTICA DE CÓDIGO

Em função da crescente complexidade dos sistemas de manutenção proativa embarcados, e dos recentes desenvolvimentos tecnológicos das plataformas de hardware e software, a programação baseada em modelos, em ambientes de alto nível, aliada a capacidade de geração automática de código tem despertado grande interesse. A seguir alguns trabalhos relacionados com geração automática de código encontrados na literatura serão apresentados:

- Uma proposta para geração automática de código a partir de uma descrição do sistema em UML (*Unified Modeling Language*) foi apresentada por (WEHRMEISTER, FREITAS, PEREIRA, 2009). Nesta proposta uma ferramenta de geração de código chamada GenERTiCA (*Generation of*

Embedded Real-Time Code based on Aspects) é apresentada, a qual utiliza scripts com as regras de geração de código. Estes scripts podem ser escritos para diversas linguagens, tais como C/C++, Java, VHDL e SystemC.

- Trabalhos relacionados com processamento de áudio também têm explorado as possibilidades da programação em linguagens de alto nível aliados com geração automática de código, como mostrado em (JARITZ, 2012). Neste trabalho um mesmo algoritmo é implementado em ambiente MATLAB/Simulink™ utilizando o Simulink Coder™ para gerar código em C, em Python sendo executado em uma máquina virtual, e por fim utilizando Audio Languange e o Audio Processor que visa criar uma camada de abstração entre o algoritmo e o hardware.
- O trabalho apresentado por (MOON, 2007) explora a capacidade de simulação na MDE. Um bloco híbrido para utilização em ambiente MATLAB/Simulink é proposto, que além da capacidade de geração de código C, pode simular o funcionamento do modelo levando em consideração as características de hardware (atrasos temporais), focando em aplicações automotivas utilizando o barramento CAN.
- A análise do tempo de execução do pior caso automatizada foi integrada no ambiente MATLAB/Simulink em (KIRNER ET AL., 2002), de forma a facilitar o desenvolvimento de projetos. Esta análise é feita transformando-se o código e o controle de fluxo de informação, os quais são gerados automaticamente pelo ambiente MATLAB/Simulink, em níveis de representação mais baixos, onde os cálculos do tempo de execução no pior caso são realizados.
- Um sistema de detecção de falhas em motores utilizando sinais de vibração e

extraindo informações temporais como valor RMS (Root Mean Square) e informações frequenciais como a decomposição com FFT (Fast Fourier Transform) é apresentado em (COSTA ET AL., 2010). O sistema foi construído em ambiente MATLAB/Simulink™ através dos blocos disponíveis na ferramenta DSP Build da Altera, a qual é capaz de gerar código VHDL automaticamente para implementação em FPGA.

5 A FERRAMENTA WATCHDOG AGENT™

A metodologia proposta pelo IMS Center preconiza a extração da informação contida nos sinais monitorados, a fim se obter um índice para o estado atual de degradação dos equipamentos. Inicialmente aplica-se uma técnica de processamento de sinais aos dados provenientes dos sensores. Em geral estes dados serão numerosos, portanto é necessária uma etapa de redução de dimensionalidade, resultando no que é chamado de assinatura de desempenho ou características do sinal. Em seguida, aplica-se um algoritmo para classificação das características, obtendo um índice de desempenho. A Figura 4 mostra um diagrama deste processamento de dados, onde pode-se observar os sinais representativos de cada etapa.

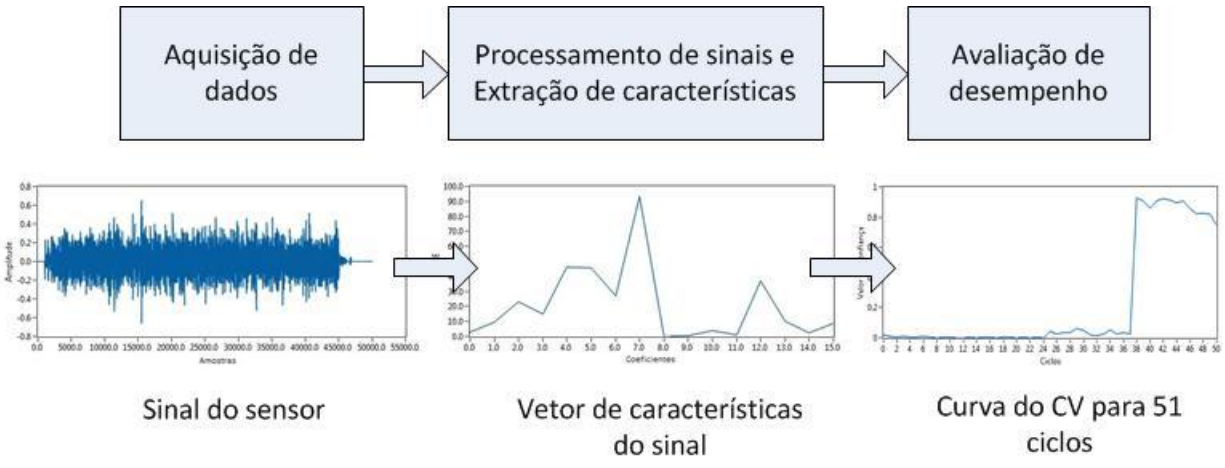

Figura 4 Processamento das informações na metodologia utilizada.

Quando o equipamento entra em um processo de degradação, é esperado que as assinaturas típicas de desempenho atual se distanciem da assinatura típica de funcionamento normal. Um indicador utilizado para a identificação do estado de um equipamento é chamado de Valor de Confiança (ou *Confidence Value* - CV). O CV é definido como uma grandeza que varia entre 1 e 0, sendo que valores próximos de 1 representam funcionamento normal e valores próximos de 0 representam funcionamento em caso de falha do sistema. A Figura 5

apresenta o conceito de Valor de Confiança: no gráfico à esquerda pode-se ver a distância entre o comportamento normal e um comportamento mais recente (de um equipamento degradado) e à direita pode-se observar o decréscimo nos Valores de Confiança indicando redução do desempenho (DJURDJANOVIC, LEE, NI, 2003).

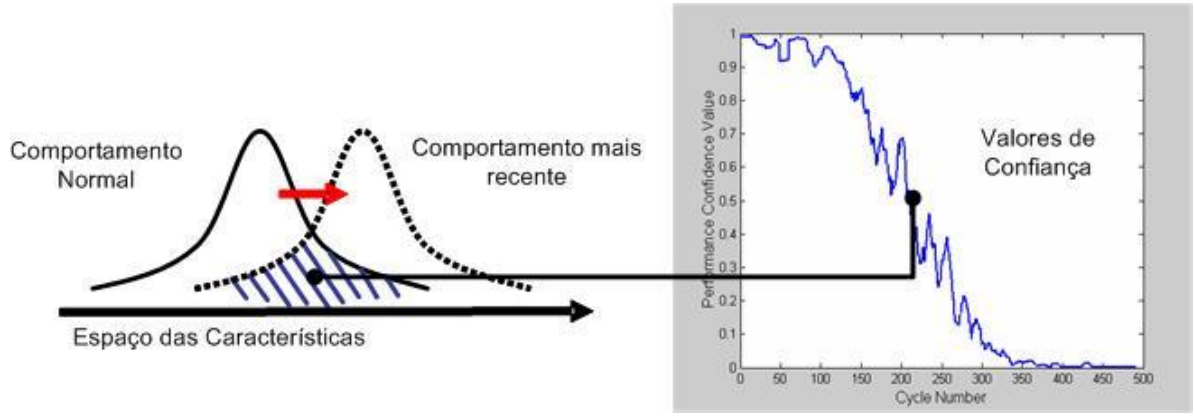

Figura 5 Representação do conceito de Valor de Confiança (DJURDJANOVIC, LEE, NI, 2003).

O *Watchdog Agent*^{TM} *Toolbox* é um conjunto de ferramentas para a análise de desempenho e diagnóstico de falhas. Nele, o índice de desempenho do sistema ou equipamento (CV) é obtido realizando a análise dos sinais oriundos de diversos sensores. Aplicando técnicas de processamento de sinal a estes dados, é possível extrair as características de funcionamento normal do sistema e, com base nestas características, é possível identificar desvios no processo (LEE ET AL., 2006).

O *Watchdog Agent*™ é composto por diversos módulos: Processamento de Sinal e Extração de Características, Avaliação de Desempenho, Diagnóstico de Falhas e Predição da Saúde do Sistema. Ele foi desenvolvido de forma compatível com o modelo OSA-CBM (*Open Systems Architecture for Condition-Based Maintenance*) (THURSTON, 2001), a fim de facilitar tanto a integração de novas ferramentas como também a integração com outros dispositivos compatíveis com este modelo.

No módulo de Processamento de sinais e Extração de Características estão à

disposição os métodos de Energia das bandas de frequência da Transformada de Fourier, de Momentos da Transformada Tempo-Frequência, de Momentos da Transformada Wavelet Packet e de Energia da Transformada Wavelet Packet, além da extração de informações como média, variância e valores de pico dos sinais. No módulo de Avaliação de desempenho tem-se os métodos de Regressão Logística e Reconhecimento Estatístico de Padrões. O módulo de Detecção de Falhas conta com os métodos de Mapas Auto-organizáveis, Redes Neurais e Clusterização Hierárquica. O módulo de Predição da Saúde do sistema é composto pelos métodos de modelo ARMA (*Auto-regressive Moving-Average*) e Match Matrix. Outros métodos podem ser incluídos nestes módulos, desde que mantenham a compatibilidade com o modelo OSA-CBM.

Atualmente este sistema está disponível em duas versões, uma de desenvolvimento implementada em ambiente MATLAB™, ilustrada na Figura 6, e outra comercial implementada em ambiente LabVIEW™, atualmente em versão de demonstração sem todas as funcionalidades á disposição, que pode ser vista na Figura 7. Para este trabalho tem-se acesso a todo o código da versão implementada em MATLAB™.

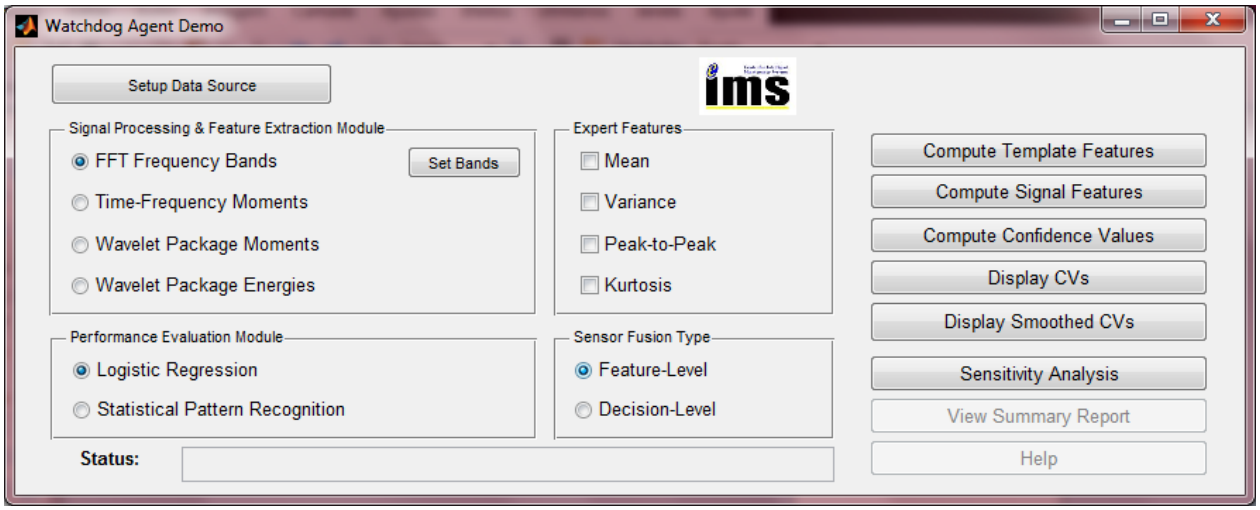

Figura 6 Ferramentas do WA disponíveis na versão para o MATLAB™

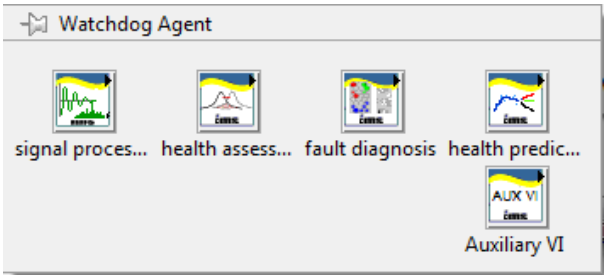

Figura 7 Ferramentas do WA disponíveis na versão para o LabVIEW™

5.1 O MODELO OSA-CBM

O modelo OSA-CBM é uma proposta de padronização para a construção de uma arquitetura aberta para padronizar o fluxo de informações dentro de um sistema de manutenção baseado na condição (THURTON, 2001). Um sistema de manutenção baseado na condição envolve uma grande variedade de componentes, tais como diferentes softwares e hardwares, e este modelo facilita a integração destes componentes estabelecendo uma arquitetura composta por sete blocos funcionais, como pode ser visto Figura 8.

Esta arquitetura se propõe a facilitar a integração e a interoperabilidade de componentes entre diferentes fabricantes de equipamentos. Ela descreve as informações e o fluxo de dados dos módulos especificados, entretanto não especifica a comunicação entre os mesmos.

A seguir é apresentada uma breve descrição das camadas propostas, de acordo com (THURTON, 2001):

- **Aquisição de Dados**: Consiste em um elemento sensor e um elemento de aquisição de dados. É responsável por transformar uma grandeza física em uma grandeza elétrica (já condicionada) e, por fim, em um valor digital. Também pode ser responsável pelo armazenamento em um banco de dados.

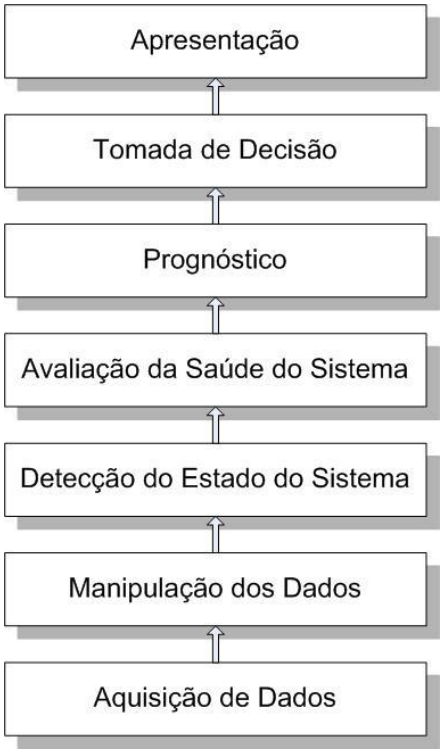

Figura 8 Modelo OSA-CBM

- **Manipulação dos dados**: Realiza uma primeira etapa de cálculos sobre os valores adquiridos no módulo anterior, geralmente utilizando técnicas de processamento de sinais. Os resultados desta etapa podem ser sinais no domínio do tempo, de frequência, ou ainda tempo-frequência. Pode haver armazenamento de informações nesta camada.
- **Detecção do Estado do Sistema**: Calcula continuamente o estado atual ou indicadores de estado atual de cada sistema, subsistema ou componente, com base nos dados já processados. Estes resultados são armazenados para utilização posterior. Pode ainda gerar alertas baseado em condições limite previamente estabelecidas.
- **Avaliação da Saúde do Sistema**: Coloca o resultado da monitoração do estado do sistema no contexto das operações. Utiliza os resultados atuais da monitoração, bem como seus valores anteriores e outros indicadores do estado

para determinar a saúde do sistema ou equipamento monitorado. Além disso, armazena os resultados formando um histórico.

- **Prognóstico**: Utiliza uma metodologia de modelagem para predizer a saúde futura do sistema ou equipamento monitorado (tendência do sistema ou equipamento). Sua função básica é estimar a vida útil restante do equipamento ou probabilidade de falha com relação a um horizonte de predição. Pode haver armazenamento dos resultados.
- **Tomada de Decisão**: Integra as informações necessárias para dar suporte à tomada de decisão com base nos resultados da camada de Prognóstico, em restrições externas, em requisitos de funcionalidade do sistema ou equipamento, em condições financeiras, etc. Fornece um ranking das ações recomendadas de acordo com implicações das decisões.
- **Apresentação**: É a interface homem-máquina, responsável por apresentar a saída desse sistema, o que pode ser feito de forma única por meio de uma tela para um operador ou de forma distribuída pela Internet. Esta camada pode trocar informações com qualquer outra camada do modelo. Também pode incluir técnicas de realidade aumentada (*Augmented Reality*) como proposto em (ESNPÍNDOLA ET AL., 2010).

5.2 ALGORITMOS

O estudo detalhado de todos os algoritmos disponíveis na ferramenta *Watchdog Agent*™ não faz parte do escopo deste trabalho, desta forma, apenas serão apresentados aqueles utilizados no estudo de caso. A justificativa para a escolha de tais algoritmos será apresentada na Seção 6.2.

5.2.1 Energias da Transformada Wavelet Packet

A Transformada Wavelet é uma forma de análise no domínio tempo-frequência, sendo indicada para sinais não estacionários, que apresentam descontinuidades, tendências, etc. Esta análise vem sendo utilizada em diversas aplicações, como na remoção de ruído de sinais ou imagens, na compressão de imagens com baixa degradação da qualidade e na área médica.

Assim como na Transformada de Fourier, o sinal é decomposto em uma soma de funções base, mas no lugar de uma base senoidal, localizada apenas na frequência, são utilizadas funções base localizadas em tempo e frequência, chamadas de Wavelets. As Wavelets são formas de onda oscilantes de duração limitada e com valor médio igual a zero, a Figura 9 mostra alguns exemplos de Wavelets. O fato da função de base ser finita no tempo evidencia o caráter tempo-frequência da análise, sendo que a representação do sinal é obtida decompondo-o em versões dilatadas (ou comprimidas) e transladadas da Wavelet mãe ψ(t) (RIOUL, VETTERLI, 1991).

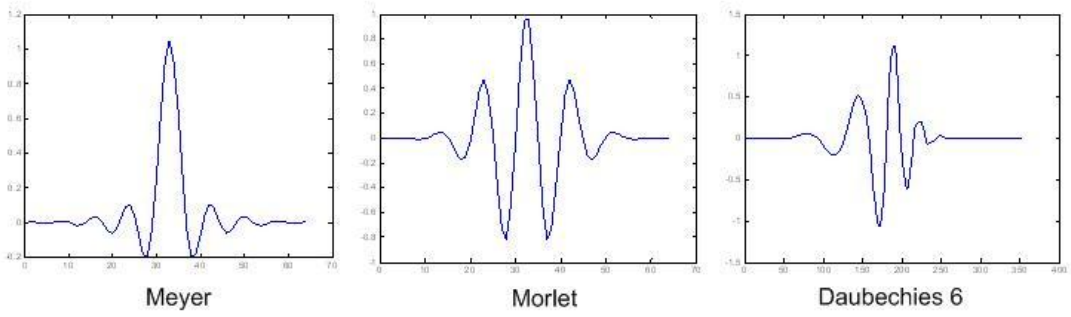

Figura 9 Exemplos de famílias de Wavelets

As Wavelets são definidas por (RIOUL, VETTERLI, 1991; DAUBECHIES, 1992):

$$\psi_{a,b}(t) = \frac{1}{\sqrt{a}} \cdot \psi\left(\frac{t-b}{a}\right)$$

onde *a* é o fator de escala e *b* o fator de translação, sendo que ambos são contínuos.

A Transformada Wavelet Contínua (CWT – *Continuous Wavelet Transform*) de um

sinal x(t) é dada pela seguinte expressão (RIOUL, VETTERLI, 1991; DAUBECHIES, 1992):

$$CWT(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \cdot \psi_{a,b}(t) dt$$

O fator de escala *a* tem influência na largura da Wavelet, e consequentemente na resolução da análise. Quanto menor for *a*, mais comprimida será a Wavelet, e desta forma ela será capaz de detectar eventos de alta frequência. De forma análoga, quanto maior for o fator de escala, mais dilatada será a Wavelet, e os padrões de baixa frequência serão identificados (RIOUL, VETTERLI, 1991). Esta mudança de escala caracteriza uma análise multiresolução do sinal, como pode ser visto na Figura 10:

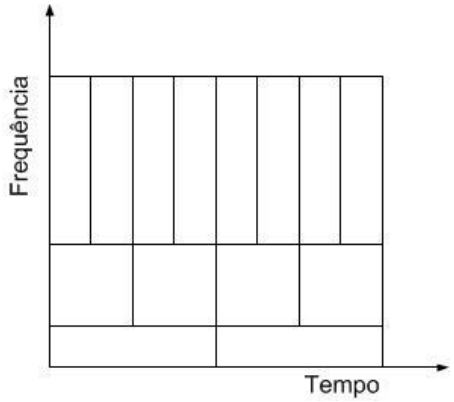

Figura 10 Resolução tempo-frequência da WT

No caso de um sinal discreto, utiliza-se a Transformada Wavelet Discreta (DWT – *Discrete Wavelet Transform*), que pode ser realizada através de um processo de filtragem utilizando um banco de filtros em forma de árvore, onde cada nível possui um filtro passa alta e um filtro passa baixa (filtros em quadratura). Ou seja, em cada nível o sinal é decomposto em uma parte de alta frequência, também chamada de *detalhe*, e outra de baixa frequência, também chamada de *aproximação* (MALLAT, 1989). Este processo pode ser repetido, separando as componentes de alta e baixa frequência da aproximação de cada nível, formando a chamada *árvore de decomposição Wavelets*. Também é possível realizar essa decomposição

dos detalhes, gerando uma árvore mais rica, com mais possibilidades para representar o sinal, formando a chamada *árvore de decomposição Wavelet Packet* (MALLAT, 1989). A Figura 11 mostra a árvore de decomposição Wavelet Packet, onde H_0 é o filtro passa baixa, H_1 é o filtro passa alta, e o bloco $\downarrow 2$ é um decimador.

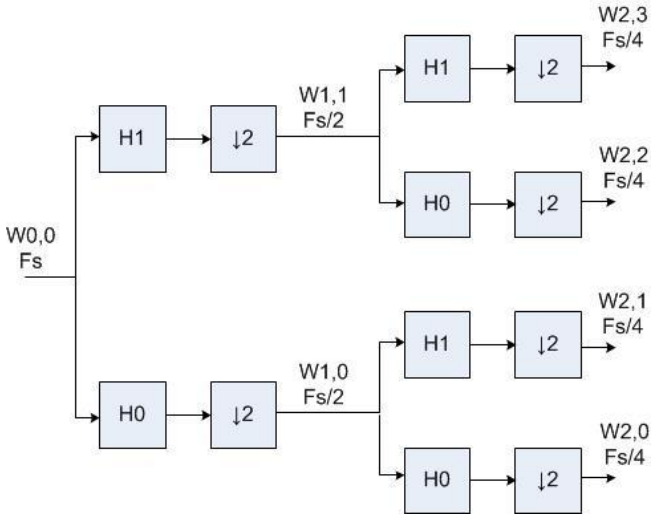

Figura 11 Representação da árvore de decomposição Wavelet Packet com 2 níveis de decomposição

Esta implementação, também chamada de piramidal (MALLAT, 1989; RIOUL, VETTERLI, 1991), possibilita que o cálculo seja feito rapidamente, tornando este método indicado quando os dados disponíveis devem ser processados rapidamente ou quando eles se apresentam em grande quantidade (IMS CENTER, 2007). Como principal desvantagem deste método, pode-se citar a dificuldade em determinar qual família de Wavelets é a mais indicada para realizar a análise, assunto que ainda é tema de estudos (IMS CENTER, 2007).

O vetor de características é finalmente obtido calculando-se a energia dos coeficientes resultantes no nível mais baixo de decomposição da transformada, chegando-se em um vetor de 2^kelementos, onde k é a quantidade de níveis de decomposição (IMS CENTER, 2007), configurando o que se chama de Energia da Transformada Wavelet Packet (WPE – *Wavelet Packet Energies*).

5.2.2 Regressão Logística

A Regressão Logística (LR – *Logistic Regression*) faz parte da categoria de modelos estatísticos chamados de Modelos Generalizados Lineares (AGRESTI, FINLAY, 2007). Este método permite obter uma saída discreta, como uma classificação em um grupo, de um conjunto de dados que pode ser contínuo, discreto, binário, etc (IMS CENTER, 2007). Geralmente a resposta possui dois estados, como presença/ausência ou sucesso/falha, e no estudo de interesse, comportamento normal/comportamento degradado.

A Regressão Logística tenta ajustar um mapeamento do espaço de k dimensões da entrada para um espaço de uma dimensão de saída. Define-se a variável da resposta como sendo y, e denota-se y=1 quando o conjunto de entrada possui a característica de interesse e y=0 quando ele não possui (AGRESTI, FINLAY, 2007). A Figura 12 mostra uma curva logística:

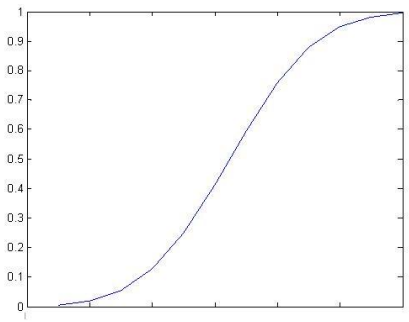

Figura 12 Curva típica de um modelo de regressão logística

Matematicamente, o modelo é expresso da seguinte forma (IMS CENTER, 2007):

$$p(x) = P(y = 1|x) = \frac{1}{1 + e^{-(\alpha + \beta 1 x 1 + \beta 2 x 2 + ... + \beta k x k)}} = \frac{e^{\alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k}}{1 + e^{\alpha + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_k x_k}}$$

onde $x = (x_1, x_2, ..., x_k)$ é a entrada de dimensão k e y é a saída binária.

O modelo acima pode ser reescrito em termos das probabilidades de evento e de não evento, ou p(x) e 1-p(x) respectivamente (AGRESTI, FINLAY, 2007):

$$\frac{p(x)}{1-p(x)} = e^{\alpha+\beta_1 x_1+\beta_2 x_2+\ldots\beta_k x_k}$$

Aplicando o logaritmo natural nos dois lados da equação obtém-se:

$$g(x) = \ln\left(\frac{p(x)}{1-p(x)}\right) = \alpha + \beta_1 x_1 + \beta_2 x_2 + \ldots + \beta_k x_k$$

onde o $\ln\left(\frac{p(x)}{1-p(x)}\right)$ é conhecido como função *logit*, e torna o modelo linear (AGRESTI, FINLAY, 2007).

Na prática, uma amostra da entrada {x} é observada e a probabilidade {p(x)} é estimada da estatística ou especificada subjetivamente de acordo com a necessidade, como por exemplo, estabelecendo uma entrada normal com probabilidade 0,95 e uma entrada anormal com probabilidade 0,1. Em função desta característica, este método é indicado para situações onde os comportamentos aceitáveis e não aceitáveis são explicitamente conhecidos (IMS CENTER, 2007).

A partir das probabilidades {p(x)} e da entrada {x}, os parâmetros α, β_1, ...,β_k são obtidos resolvendo o modelo linear, utilizando mínimos quadrados, máxima verossimilhança, etc., ou seja, o algoritmo precisa ser "treinado" antes da sua utilização. Em função disso, se as características de entrada pertencerem a um espaço com muitas dimensões, o cálculo pode se tornar muito lento e tornar o uso deste método inviável (IMS CENTER, 2007).

O Valor de Confiança é obtido diretamente da primeira equação, uma vez que P(y=1|x) corresponde ao comportamento normal do sistema, e por se tratar de uma função de probabilidade, seu valor naturalmente varia entre 0 e 1.

6 PROPOSTA DA DISSERTAÇÃO

Neste capítulo será apresentada uma visão geral da proposta do presente trabalho, bem como será apresentado o estudo de caso utilizado para teste e validação da mesma. Também serão discutidos e detalhados os procedimentos adotados para a geração automática de código nos ambientes LabVIEW™ e MATLAB™.

6.1 VISÃO GERAL DA PROPOSTA

A proposta da presente dissertação é explorar alguns dos espaços de projeto possíveis para o embarque dos algoritmos previamente desenvolvidos na ferramenta *Watchdog Agent*™ para diferentes plataformas, bem como analisar o desempenho destes algoritmos nestas plataformas. Para tanto, no lugar de se utilizar técnicas tradicionais de projeto, a abordagem adotada é de fazer uso das ferramentas de geração automática de software e hardware disponíveis nos ambientes LabVIEW™ e MATLAB™, para os quais o *Watchdog Agent*™ já possui implementações.

6.1.1 Conceito geral

A partir do sistema *Watchdog Agent*™ disponível nos ambientes LabVIEW™ e MATLAB™, o objetivo principal deste trabalho é analisar o desempenho de diferentes implementações em hardware e software de algoritmos de manutenção inteligente, sendo que os códigos analisados serão, na medida do possível, obtidos de forma automatizada utilizando-se ferramentas disponíveis (vide Figura 13).

Para a geração de código no ambiente MATLAB™ serão utilizadas as ferramentas MATLAB Coder™ e Embedded Coder™. Estas ferramentas permitem geração de software genérico, possibilitando ainda algum nível de personalização do código e otimizações próprias para sistemas embarcados. Para a geração de HDL será utilizada a ferramenta HDL

Coder™, capaz de gerar descrições de hardware portáveis e sintetizáveis em VHDL ou Verilog. Nos dois casos as ferramentas adotadas são capazes de gerar código independentemente da plataforma adotada no sistema embarcado, de forma que não se torna necessário reescrevê-lo para diferentes componentes.

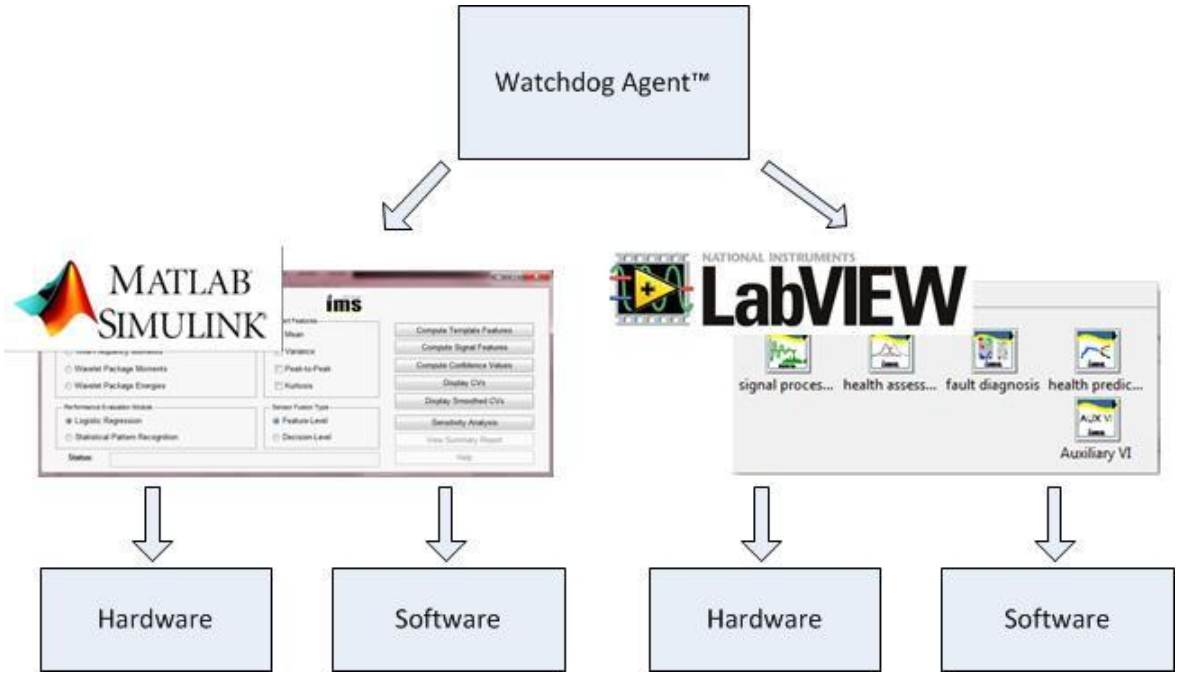

Figura 13 Abordagem de embarque dos algoritmos adotada

No ambiente LABVIEW™ é possível implementar o sistema apenas na plataforma CompactRIO, a qual é composta por uma controladora de tempo real e um chassi que contém um FPGA. A geração de código para esta plataforma se dá de forma integrada e transparente para o usuário. A metodologia adotada no desenvolvimento dos sistemas será abordada no decorrer deste capítulo.

A fim de validar a proposta e avaliar os resultados obtidos, o estudo de caso utilizado neste trabalho é a avaliação de degradação de um atuador eletromecânico para válvulas. Sinais de vibração do motor coletados em situações de funcionamento normal e funcionamento degradado serão utilizados. Os algoritmos escolhidos para extração das características destes sinais e posterior avaliação da degradação são Energias da

Transformada Wavelet Packet e Regressão Logística, respectivamente. Estes algoritmos são indicados neste tipo de aplicação em função da natureza não estacionária dos sinais de vibração e da disponibilidade de sinais representativos de funcionamento normal e degradado à disposição. O desempenho do sistema de manutenção embarcado será analisado em termos das seguintes métricas: percentual de ocupação do processador, ocupação dos recursos (área) do FPGA e por fim, tempo de execução dos algoritmos com relação à frequência de operação do processador..

6.1.2 Planejamento dos experimentos

Nesta seção será discutido o planejamento dos experimentos realizados. O estudo de caso escolhido para validação e testes será apresentado em linhas gerais, e os espaços de projeto a serem comparados serão apresentados.

6.1.2.1 Visão geral do estudo de caso escolhido

Os dados à disposição para este trabalho foram coletados de uma bancada de testes composta por dois atuadores eletromecânicos para válvulas, desenvolvida no âmbito de um projeto de pesquisa cooperativa entre o GCAR/UFRGS com as empresas Transpetro e Coester Automação Ltda. Os atuadores elétricos permitem a abertura e fechamento motorizado de válvulas através da movimentação de sua haste.

As válvulas estão presentes em diversas aplicações, entre as quais estão as plantas de indústrias do setor petroquímico e o setor de saneamento básico. Uma única planta industrial pode ter em suas instalações centenas de válvulas, que podem estar operando sob condições de pressão e temperatura altas, ou mesmo em locais remotos com acesso difícil ou demorado, logo é importante que se tenha informações sobre o estado do equipamento com a antecedência necessária para que a equipe de manutenção chegue antes da falha. Desta forma, uma falha no conjunto atuador-válvula pode trazer riscos e prejuízos inaceitáveis. Existem

diversos tipos de válvulas: gaveta, esfera, ou globo, por exemplo, cada uma indicada para um tipo de aplicação em particular (HENRIQUES ET AL., 2010).

Os atuadores são dispositivos que permitem a automação do acionamento de válvulas, comportas e equipamentos similares, tendo como principal função a movimentação da haste da válvula. Os atuadores elétricos são compostos de uma parte mecânica, que compreende uma série de engrenagens para transferir o movimento do motor para o eixo da válvula, e uma parte elétrica, composta da parte de comando e de um motor para o acionamento. Os atuadores à disposição na bancada de teste são da linha CSR, modelo CSR 06, da empresa Coester Automação Ltda. Este atuador é específico para válvulas multivoltas, com capacidade de torque de 60Nm e com o chamado controle Inteligente, o qual possui uma unidade de processamento local, sensores e sistema de autodiagnose, além de capacidade de comunicação via uma rede Modbus. A Figura 14 mostra um desenho do conjunto atuador-válvula, indicando a nomenclatura das suas partes constituintes.

Figura 14 Conjunto atuador-válvula

6.1.2.2 Implementações a serem comparadas

Conforme discutido no conceito geral da proposta deste trabalho, serão explorados os espaços de projeto para implementações em software e em hardware dos algoritmos de interesse, fazendo uso de ferramentas de geração automática de código presentes nos ambientes MATLAB™ e LabVIEW™. Desta forma é possível implantar o sistema de forma rápida, em diferentes plataformas, sem precisar reescrever os algoritmos para a plataforma em questão. A Figura 15 mostra a proposta de implementação a partir do ambiente MATLAB™, mostrando algumas plataformas que podem ser utilizadas. O procedimento adotado será detalhado na seção 6.3.

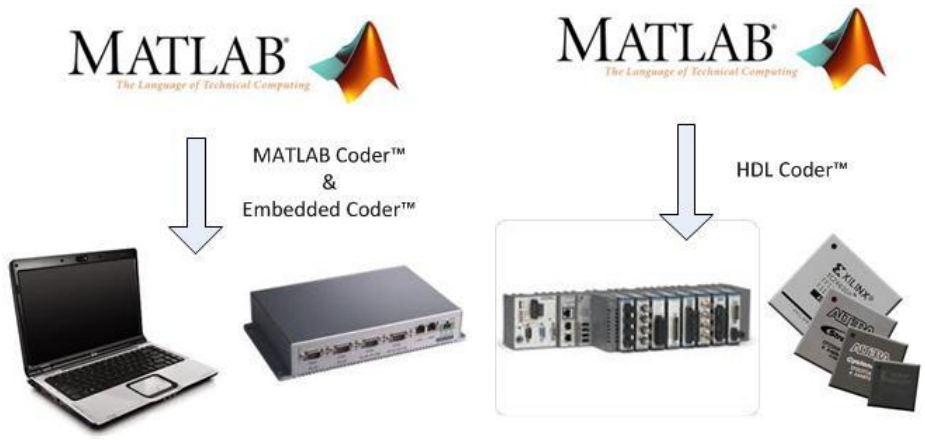

Figura 15 Espaço de projeto de geração de software e hardware a partir do ambiente MATLAB™

A Figura 16 mostra a proposta de implementação a partir do ambiente LabVIEW™. Neste caso, as ferramentas para embarque do código são inerentes ao ambiente de desenvolvimento, de forma que se os módulos utilizados tiverem suporte para execução na plataforma CompactRIO, basta criar um projeto adequado para executar o código em hardware ou software. O procedimento adotado nesta abordagem será detalhado na seção 6.4.

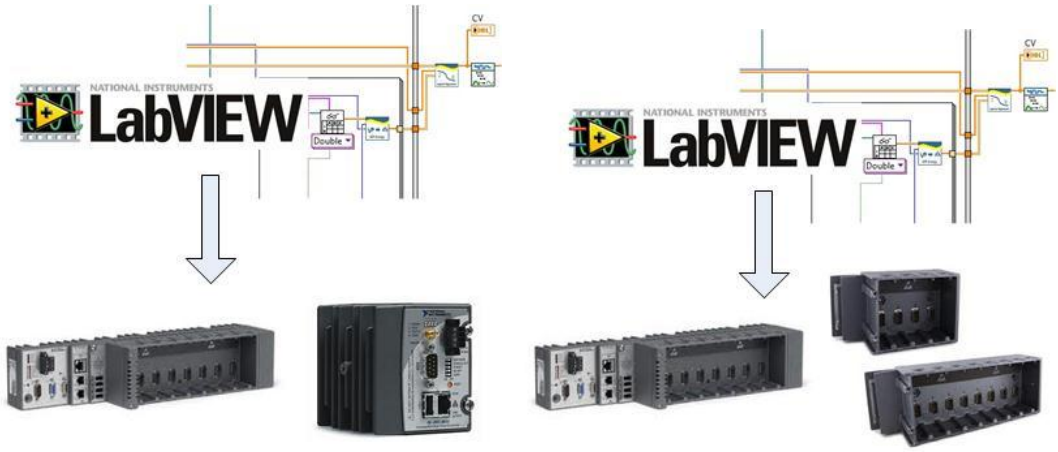

Figura 16 Espaço de projeto de geração de software e hardware a partir do ambiente LabVIEW™

6.2 DESCRIÇÃO DETALHADA DO ESTUDO DE CASO

Os atuadores disponíveis na bancada de testes possuem uma interface de comando local programável, e todos os elementos de controle estão incorporados no próprio equipamento. O acionamento pode ser local ou remoto, sendo que neste último caso os contatos devem ser duplicados até o painel de controle remoto. Os atuadores com comando Inteligente tem a capacidade de detectar alguns problemas, como sobreaquecimento do motor, torque excessivo e falta de fase, e, além disso, possuem a instrumentação necessária para medir os valores de torque exercido na haste por percentual de abertura da válvula. Estes valores são armazenados na memória interna do atuador, a qual tem capacidade para registrar as últimas 500 operações realizadas (abertura ou fechamento). No entanto, estas medidas são disponibilizadas com resolução baixa (1Nm sendo o fundo de escala de 60Nm neste caso) e a cada 5% de abertura. Além disso, a fim de facilmente identificar o início e o fim do movimento do atuador, valores de torque inferiores a 10Nm são registrados como 10Nm e quando não há movimento o torque é registrado como 0Nm.

A bancada de testes foi desenvolvida a fim de que se possam simular condições de

desgaste severo que seriam inaceitáveis no ambiente de operação real do conjunto atuador-válvula. Além disso, tem-se a liberdade de instrumentá-la a fim de verificar diferentes técnicas para identificação de diferentes falhas. Com a bancada à disposição pode-se simular o comportamento do conjunto com peças com diferentes níveis de degradação.

Na bancada de testes, mostrada na Figura 17, um dos atuadores tem um sistema de freio a disco acoplado ao seu eixo para simular os esforços mecânicos exercidos pelo fluxo de fluidos no interior da válvula durante os movimentos de abertura ou fechamento. Este mesmo atuador está instrumentado com três acelerômetros para aquisição de vibração, posicionados no eixo do motor do atuador, na ponta do sem-fim e na pinça do freio. A Figura 18 mostra a localização dos acelerômetros posicionados no sem fim e no eixo do motor, bem como detalha a estrutura interna do atuador.

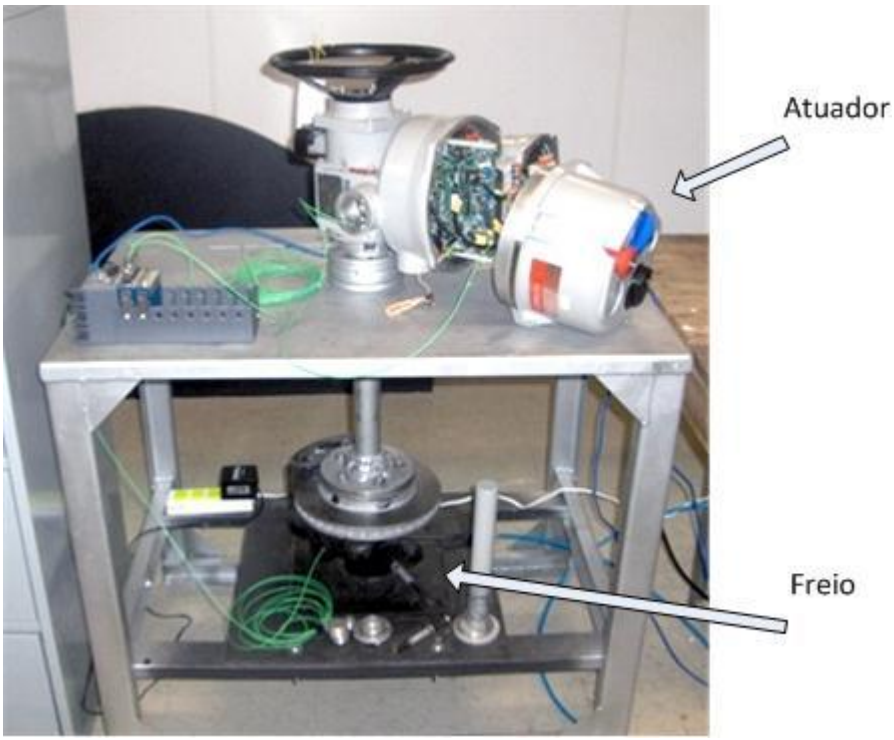

Figura 17 Atuador instrumentado na bancada de testes

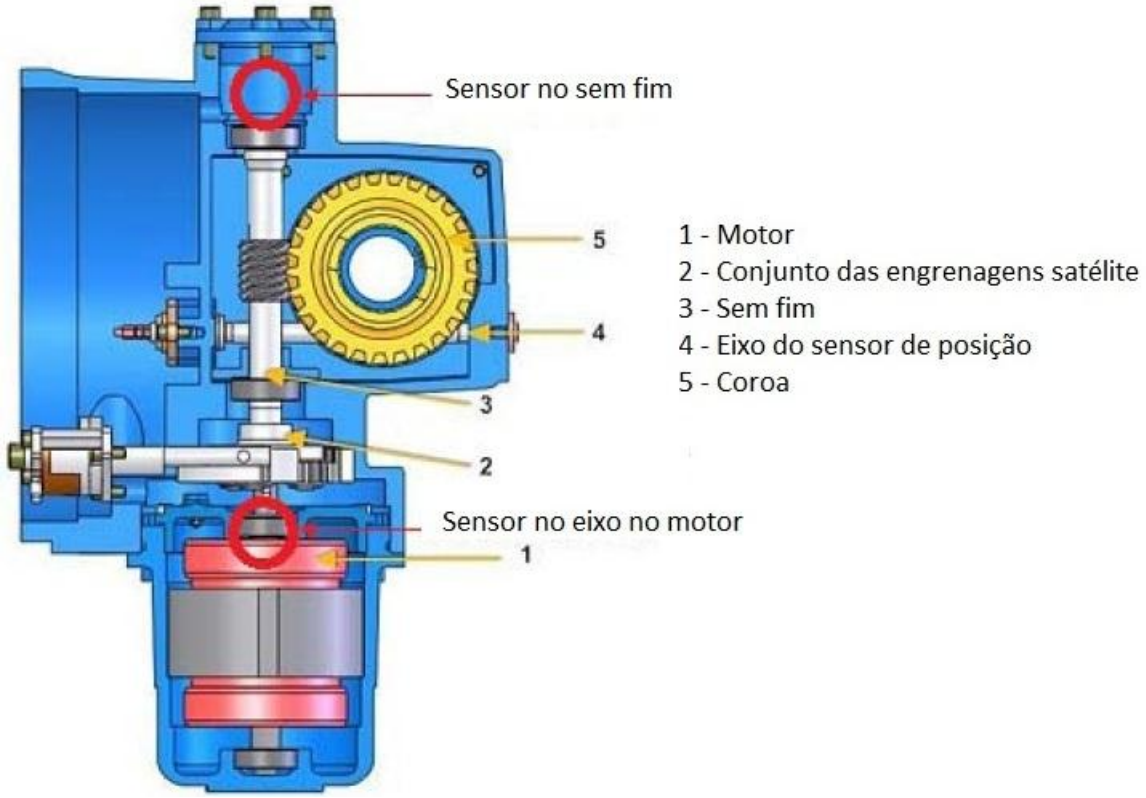

Figura 18 Localização dos sensores no atuador

Também se tem à disposição engrenagens desgastadas e quebradas para serem substituídas no atuador e simular um comportamento de falha, como pode ser visto na Figura 19.

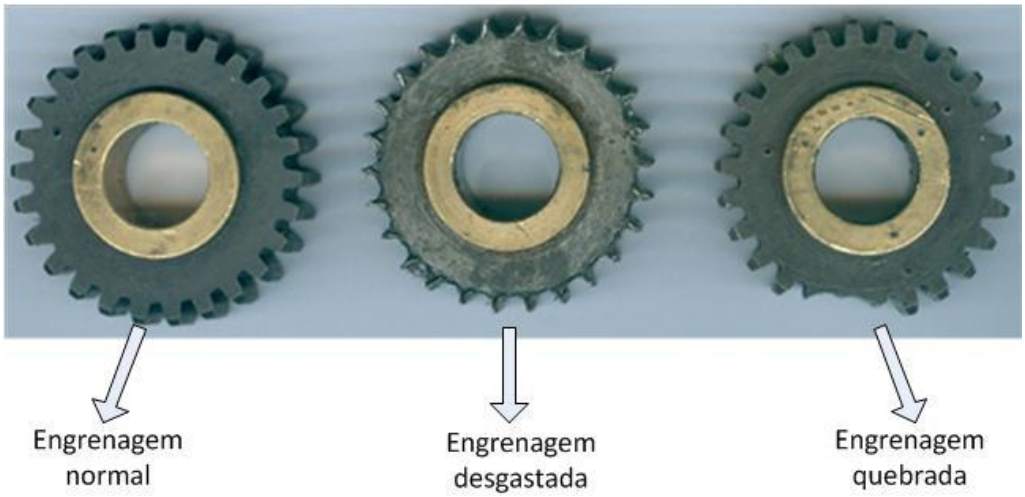

Figura 19 Engrenagens utilizadas nos testes

Estas engrenagens compõe o chamado conjunto de engrenagens satélite do atuador, cuja localização é mostrada na Figura 20 e fazem parte do sistema de transmissão do movimento do motor para a haste da válvula.

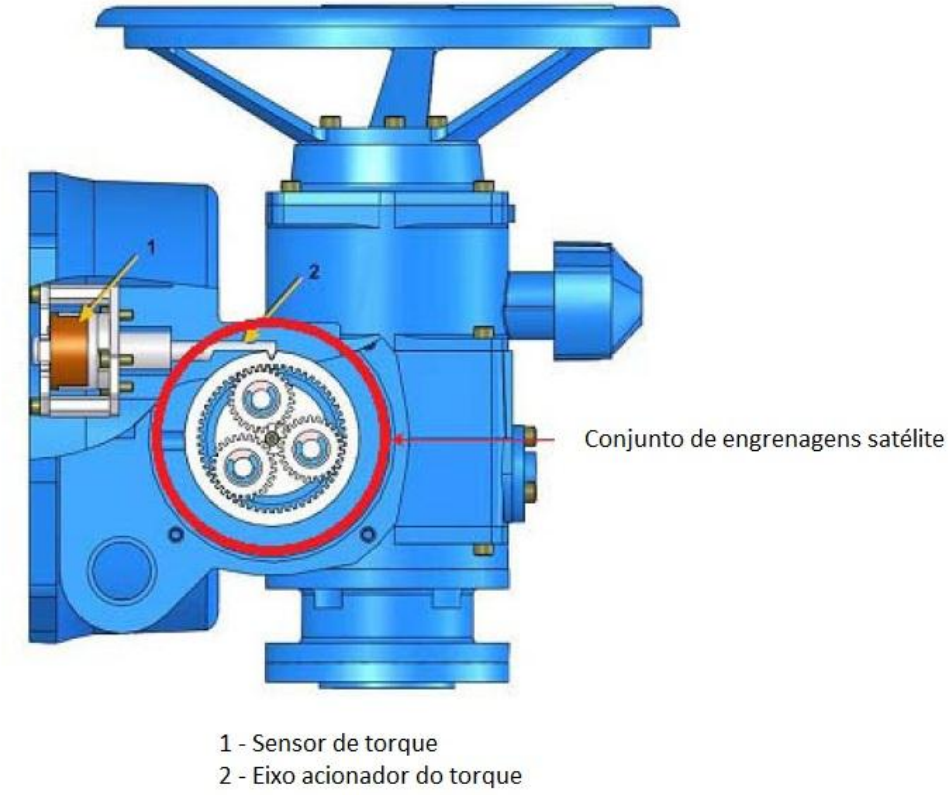

Figura 20 Localização das engrenagens satélite no atuador

Os dados foram obtidos a partir de um sistema de aquisição desenvolvido no LabVIEW™, utilizando o chassis cRIO-9104 e o módulo para aquisição de vibração NI-9233. Os dados foram adquiridos a uma taxa de 2048 amostras por segundo durante o movimento completo de abertura e fechamento, que para o atuador utilizado tem duração de aproximadamente 45 segundos (FACCIN, 2011; BÖESCH, 2011). As informações de cada ciclo de abertura ou fechamento foram salvas como um arquivo de texto.

Para este estudo de caso, portanto, tem-se a disposição 25 curvas de abertura e 25 curvas de fechamento coletadas em cada uma das 6 situações distintas a seguir, totalizando 300 curvas (FACCIN, 2011; BÖESCH, 2011):

- Engrenagens em perfeito estado e sem acionamento do freio;
- Engrenagens em perfeito estado e freio acionado com pressão de 1 bar;
- Engrenagens em perfeito estado e freio acionado com pressão de 3 bar;
- Uma engrenagem desgastada e sem acionamento do freio;
- Três engrenagens desgastadas e sem acionamento do freio;
- Uma engrenagem com dentes quebrados e sem acionamento do freio.

Neste trabalho foram utilizados apenas os dados do acelerômetro acoplado ao eixo do motor do atuador, que de acordo com trabalhos prévios (FACCIN, 2011; BÖESCH, 2011) é o sensor com melhores resultados na detecção de degradação das engrenagens do atuador. Esta escolha foi feita para não ser necessário abordar a fusão dos sensores na análise. Da mesma forma, os dados foram subamostrados para uma taxa de 1024 amostras por segundo, a fim de tornar o processamento menos custoso, o que não mostrou impacto no resultado da análise de acordo com (FACCIN, 2011). Os dados foram adicionalmente selecionados para o movimento de abertura da válvula, uma vez que este é o momento em que há maior força sendo exercida na sede da válvula e consequentemente maior exigência do atuador.

Como o sinal de vibração não é estacionário, técnicas de processamento de sinal de tempo e frequência são as mais indicadas para que as características variantes no tempo sejam captadas. Assim, o método mais indicado para a extração das caraterísticas do sinal é a análise das Energias da Transformada Wavelet Packet (WPE), que por sua capacidade de representação do sinal em bandas de frequência mostra-se eficaz para este tipo de sinal (QIU ET AL., 2006). Além disso, uma vez que se dispõe de peças defeituosas para uso no atuador, tem-se tanto dados de funcionamento normal, como dados de funcionamento de falha, portanto, o método de Regressão Logística (LR), o qual necessita de referência dos dois tipos de comportamento para seu treinamento, pode ser utilizado.

Considerou-se como referência de comportamento normal parte dos dados obtidos com o atuador funcionando a vazio, isto é, sem atuação do freio, e com todas as engrenagens em perfeitas condições. Como referência de comportamento de falha considerou-se parte dos dados obtidos substituindo uma das engrenagens do atuador por uma engrenagem com dentes quebrados e em operação a vazio. Para a validação, utilizou-se o restante dos dados obtidos a vazio e os dados obtidos com a engrenagem quebrada, além de dados obtidos com engrenagens severamente desgastadas.

Neste trabalho optou-se por realizar o treinamento dos algoritmos em um computador pessoal (PC), embarcando apenas a execução dos algoritmos já treinados. O resultado do treinamento é utilizado como um parâmetro de entrada do algoritmo de análise da saúde do sistema, ou seja, para um novo treinamento de uma situação diferente ou com um conjunto de dados diferente, basta passar o novo resultado para o código gerado.

6.3 DESENVOLVIMENTO EM AMBIENTE MATLAB™

A fim de verificar a adequação dos algoritmos escolhidos para a detecção de degradação, inicialmente criou-se um código no ambiente MATLAB™ contemplando as etapas de extração de características e avaliação de desempenho. Utilizou-se a versão R2012a do software MATLAB,

Na extração das caraterísticas utilizando o algoritmo WPE, o código disponível possibilita a escolha de qual Wavelet mãe será utilizada e dos níveis de decomposição da análise, o que impacta diretamente na forma e número de elementos do vetor de características resultante. Optou-se por utilizar a Wavelet mãe Daubechies 6, que já foi utilizada com sucesso em trabalhos anteriores (GONÇALVES, 2011, FACCIN, 2011). Para o nível de decomposição desta análise fizeram-se ensaios com diferentes opções para estabelecer o valor adequado, considerando-se como adequado o menor de nível de

decomposição que ainda gera um vetor de características para o qual o algoritmo de LR consegue distinguir entre as duas classes de comportamento. Desta forma, optou-se por utilizar 4 níveis de decomposição, que geram um vetor de 16 elementos como vetor de características do sinal.

Para o algoritmo LR estipulou-se que o comportamento normal corresponderia a um CV de 0,95 e o comportamento de falha corresponderia a um CV de 0,1. Estes valores foram escolhidos com base nos valores típicos encontrados nos exemplos de uso das ferramentas e na documentação das funções nos dois ambientes. Quanto mais próximos dos limites forem os valores de CV adotados no treinamento, melhor será a separação entre os comportamentos normal e de falha, ficando mais evidente as alterações decorrentes da degradação do sistema.

Para o treinamento dos algoritmos é necessário ter sinais de referência de comportamento normal e de comportamento de falha. Neste trabalho utilizou-se 12 curvas (metade das 25 curvas disponíveis para abertura da válvula) obtidas com todas as engrenagens em perfeito estado e sem pressão do freio sendo exercida como referência de funcionamento normal e 12 curvas obtidas com uma engrenagem quebrada como referência de funcionamento de falha. No teste foram utilizadas as 13 curvas restantes de funcionamento normal, as 13 curvas restantes do funcionamento de falha, além das 25 curvas obtidas com três engrenagens desgastadas, totalizando 51 curvas de teste. Desta forma, das 51 curvas (ou ciclos), 38 correspondem a comportamento de falha. Esta separação foi feita a fim de que nos testes não houvessem dados idênticos aos utilizados no treinamento.

A Figura 21 mostra o gráfico de CV resultante com os arquivos de teste, onde se pode verificar o sucesso na classificação de cada ciclo de operação. Neste caso, um ciclo representa um movimento de abertura da válvula. Os 38 primeiros ciclos são os correspondentes às curvas de falha e os 13 ciclos restantes são de funcionamento normal.

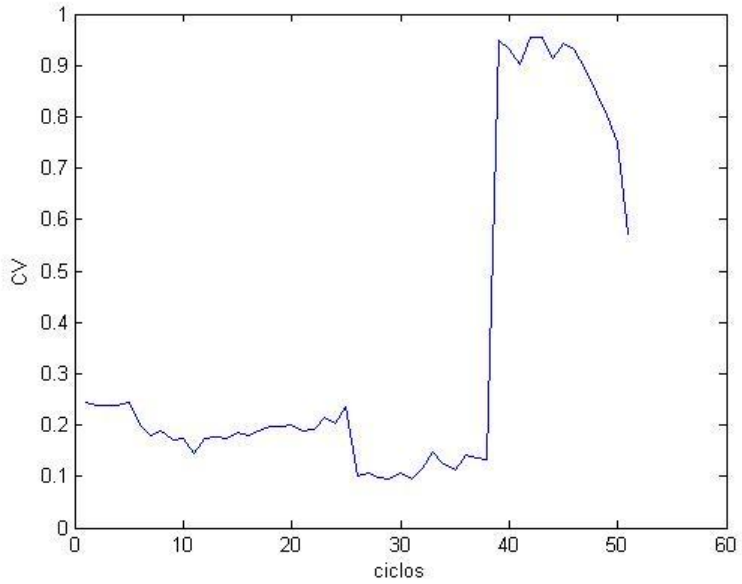

Figura 21 Curva dos CVs obtida no MATLAB™

A Figura 22 e a Figura 23 mostram as curvas de vibração e os vetores de características obtidos com o módulo WPE desenvolvido em MATLAB™ para uma situação de funcionamento normal e funcionamento de falha, respectivamente. Os gráficos de vibração estão plotados em tensão [mV] por amostra, e os gráficos do vetor de características estão plotados como Energia da transformada Wavelet por índice dos vetores de coeficientes da transformada. Observa-se que as curvas de características são similares nos dois casos, mas ainda assim é possível identificar diferenças na amplitude das energias dos coeficientes, mais explicitamente entre os coeficientes 8 e 9. Nota-se que a mesma análise não pode ser feita observado os sinais de vibração diretamente.

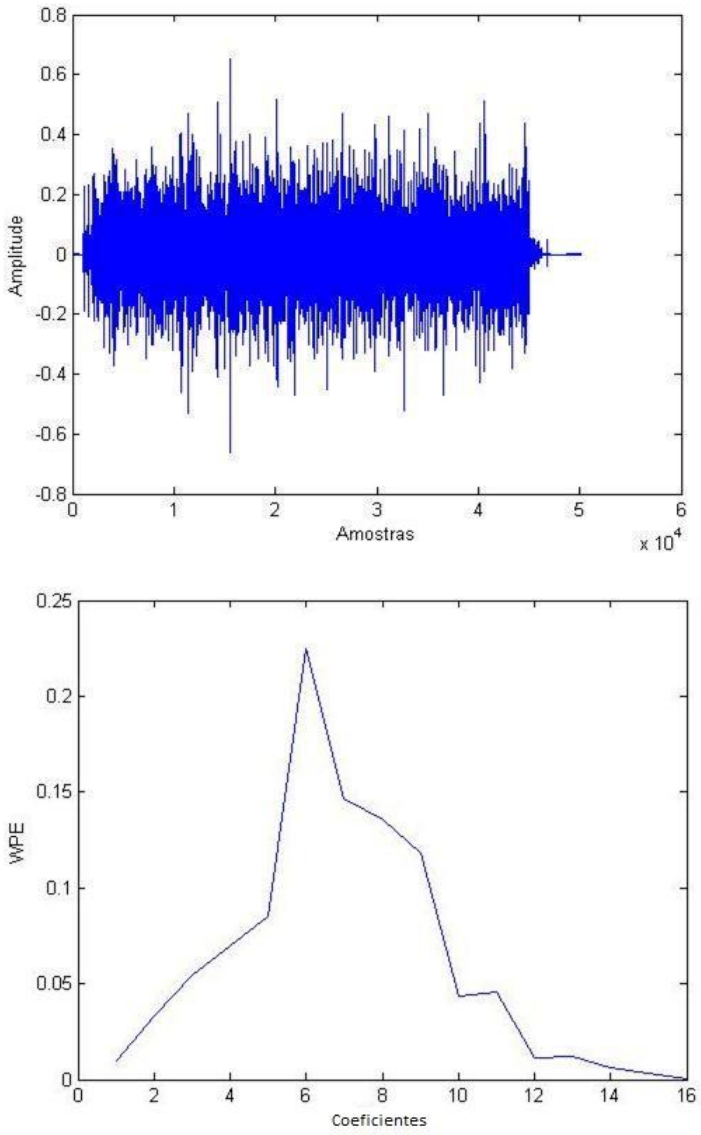

Figura 22 Sinal de vibração e vetor de características de funcionamento normal obtido no MATLAB™

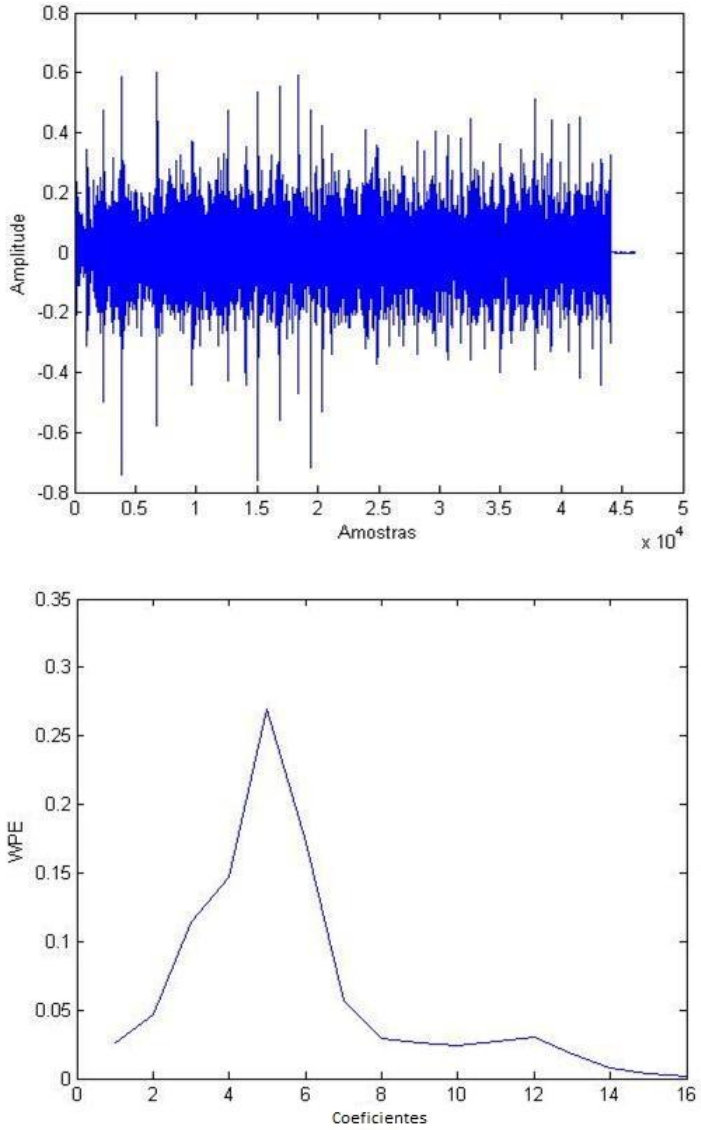

Figura 23 Sinal de vibração e vetor de características de funcionamento de falha obtido no MATLAB™

Para validação e testes da geração automática de software tem-se a disposição um PC (*Personal Computer*) com processador Intel Core™ i5-2410M de segunda geração, com dois núcleos com clock de 2,3GHz cada e sistema operacional Windows 7 e um computador industrial Avantech modelo UNO2170, com processador Intel Celeron de 600MHz e sistema operacional Windows XP SP3. Como será visto nas próximas seções, o código gerado é portável, podendo ser compilado para plataformas diferentes destas à disposição. Na geração automática de HDL, será utilizada a plataforma CompactRIO da National Instruments para a

validação e testes.

6.3.1 Geração de Software Embarcado

O MATLAB™ oferece algumas ferramentas para geração de software, como o MATLAB Compiler™, o MATLAB Coder™ e o Embedded Coder™. O MATLAB Compiler™ permite a geração de arquivos executáveis ou de bibliotecas compartilháveis a partir de código escrito dentro do ambiente MATLAB™ que utilizem os toolboxes compatíveis. Para utilizar a biblioteca ou arquivo executável gerado, é necessário ter uma licença válida do MATLAB™ na plataforma de execução ou então ter o *MATLAB Compiler Runtime* (MCR) instalado. Esta solução é adequada apenas para plataformas Windows, Linux ou Mac.

A ferramenta MATLAB Coder™ oferece geração de código C e C++ e também de arquivos executáveis autônomos ou arquivos MEX (*MATLAB Executable*), sem que seja necessária a instalação de nenhuma outra ferramenta, como ocorre com o MATLAB Compiler™. Já o Embedded Coder™ amplia estas funcionalidades oferecendo características importantes no desenvolvimento de software embarcado, tais como personalização da aparência do código gerado e otimização do código para uma plataforma de hardware específica (MATHWORKS, 2012b).

Para a geração da função MEX basta criar um projeto de geração de código, e indicar qual o arquivo de entrada que contém o código, uma vez que o algoritmo pode estar organizado em mais de um arquivo. Também é necessário explicitar os tipos das variáveis de entrada, bem como suas dimensões caso as entradas sejam vetores ou matrizes. Para a geração de código executável, é preciso ainda fornecer uma função *main* codificada em C ou C++, de acordo com a linguagem na qual o código deve ser gerado.

O fluxo de projeto recomendado ao utilizar o MATLAB Coder™ está ilustrado na

Figura 24 (MATHWORKS, 2012a). Os arquivos MEX mostrados no fluxograma são sub-rotinas escritas em C, C++ ou Fortran que o interpretador do MATLAB™ é capaz de carregar e executar. Estas sub-rotinas funcionam exatamente da mesma forma que scripts ou funções nativas do MATLAB™. Dentro do ambiente MATLAB™ são também usadas para reaproveitar algum trecho de código já disponível em linguagem C, C++ ou Fortran sem a necessidade de reescrevê-lo ou ainda para acelerar alguma porção computacionalmente pesada de um código escrito no MATLAB™. Neste caso são utilizadas para realizar a conferência do código gerado ainda dentro do ambiente MATLAB™.

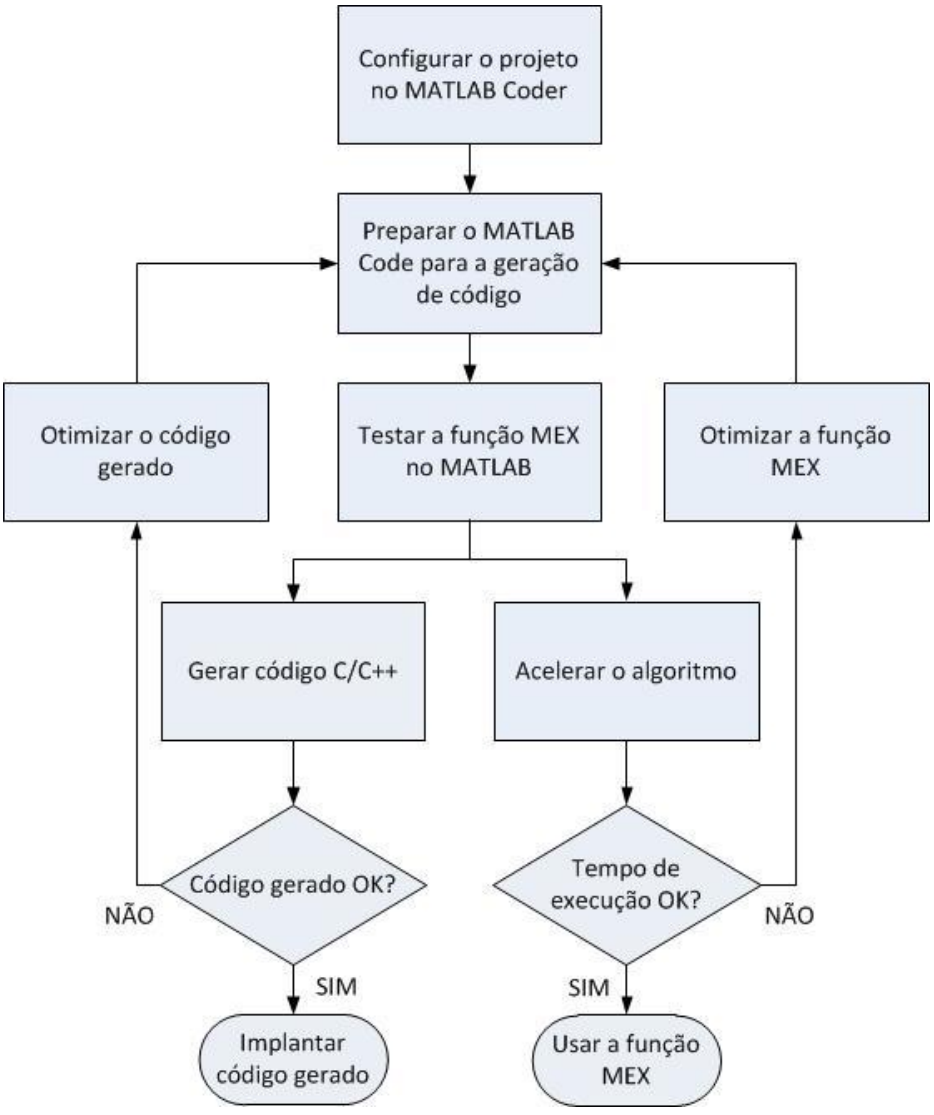

Figura 24 Fluxo de desenvolvimento para geração de software utilizando o MATLAB Coder™ (MATHWORKS, 2012a)

Utilizando o Embedded Coder™ para criação de código C ou C++ é também possível indicar a plataforma de hardware onde o código gerado será testado e onde será posteriormente executado. Neste caso, código adicional é gerado para emular a plataforma de produção na plataforma de teste. É possível selecionar algumas famílias de processadores de fabricantes como Analog Devices, Microchip, Texas Instruments e Freescale. O fluxo de projeto a ser seguido neste caso é mostrado na Figura 25. Observa-se que após a geração do código, ele pode ser integrado a um projeto existente ou compilado em um ambiente IDE (*Integrated Development Environment*) externo ao ambiente MATLAB™.

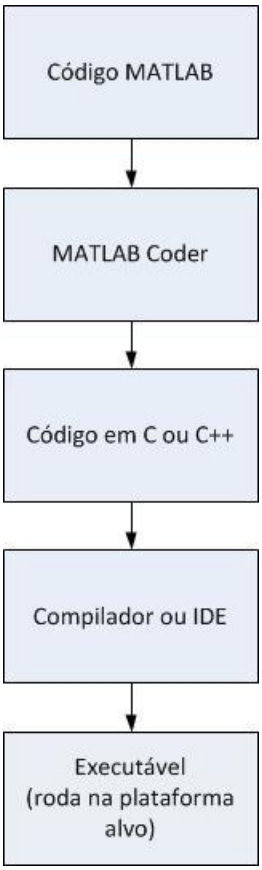

Figura 25 Fluxo de projeto utilizando o Embedded Coder™ (MATHWORKS, 2012b)

É importante ressaltar que o MATLAB Coder™ não é capaz de gerar código (arquivos MEX ou executáveis) para plataformas diferentes, ou seja, caso se deseje criar um executável para uma plataforma de 64 bits será imprescindível gerar este executável em uma versão de

64 bits do MATLAB™ com o compilador correspondente (compatível com o MATLAB™), e o mesmo acontece caso se deseje criar um executável para uma plataforma de 32 bits. Portanto neste trabalho utilizou-se tanto a versão de 32 bits quanto a versão de 64 bits do software. Além disso, utilizou-se o Compilador Microsoft Visual C++ 2008 para geração de código para plataforma de 64 bits, e o compilador Microsoft Visual Studio 10.0 para plataforma de 32 bits.

O procedimento adotado para o desenvolvimento do software a ser embarcado foi de abordar cada algoritmo separadamente. O primeiro passo foi realizar o treinamento dos algoritmos (etapa que não será embarcada nas plataformas de teste e validação) e armazenar o resultado como um arquivo de texto. Em seguida realizou-se a geração de código dos dois algoritmos, utilizando o resultado do treinamento nos testes e validação. Como as plataformas disponíveis para os testes de execução de software (o PC e o computador industrial Uno) possuem instalado o sistema operacional Windows, utilizou-se um arquivo executável gerado automaticamente.

6.3.2 Geração de Hardware

Até a versão R2011a do MATLAB™ a ferramenta para geração automática de HDL disponível, chamada Simulink HDL Coder™, apenas permitia a geração de código a partir de modelos montados no Simulink. Isto limitava seu uso a sistemas descritos em ambiente Simulink, ou forçava a conversão de um código escrito em ambiente MATLAB™ para o Simulink. A partir da versão R2012a do MATLAB™, a ferramenta foi substituída pelo HDL Coder™, que é capaz de gerar descrições portáveis e sintetizáveis em VHDL ou Verilog a partir de funções desenvolvidas em ambiente MATLAB™, modelos do Simulink™ ou gráficos do Stateflow™. Para sua utilização são necessários adicionalmente o MATLAB Coder™ e o Fixed-Point Toolbox™ (MATHWORKS, 2012c).

Além da geração do código em VHDL, através da ferramenta HDL Verifier™ é possível também gerar o *testbench* para simulação e verificação do projeto. O *testbench* é uma função ou script do MATLAB™ codificada com objetivo de testar o algoritmo de interesse. O HDL Verifier™ oferece integração com os ambientes de simulação tais como Cadence Incisive©, Mentor Graphics© e ModelSim©. Desta forma, é possível utilizar o próprio MATLAB™ ou o Simulink para simular o HDL gerado e analisar a resposta obtida (MATHWORKS, 2012d). Também é possível realizar o chamado *FPGA-in-the-Loop* (FIL). O FIL permite testar um projeto em um hardware real utilizando o MATLAB™ ou o Simulink, a partir de qualquer HDL, seja escrito manualmente ou gerado automaticamente de um modelo. Ele fornece um canal de comunicação entre o ambiente MATLAB™ e o FPGA (MATHWORKS, 2012d). No entanto, na versão utilizada do software, esta ferramenta está disponível apenas para algumas placas de desenvolvimento compatíveis, e não pode ser utilizada na plataforma disponível neste trabalho.

A Figura 26 mostra o fluxo de projeto seguido utilizando o HDL Coder™. Esta ferramenta oferece um fluxo de projeto automatizado chamado de *Workflow Advisor* para auxiliar no desenvolvimento. Inicialmente deve-se indicar no projeto a função a ser implementada em hardware e um código de teste a partir do qual será gerado o *testbench*. Um aspecto importante a ser considerado durante o desenvolvimento nesta abordagem é que o código produzido no ambiente MATLAB™ deve estar em ponto fixo para que o HDL Coder™ realize a geração automática de VHDL. Como o HDL Coder™ possui uma etapa de conversão automática para ponto fixo, o projeto pode ser desenvolvido em ponto flutuante, porém deve-se analisar o resultado da etapa de conversão e verificar se este corresponde ao esperado. O HDL Coder™ utiliza as informações coletadas na execução do *testbench* para inferir os tipos dos dados em ponto fixo utilizados na conversão.

Ao final da etapa de conversão para ponto fixo o *Workflow Advisor* exibe o resultado

obtido a fim de que se verifique que este é o mesmo obtido em ponto flutuante. Deve-se notar que nem todas as funções disponíveis para utilização no MATLAB™ são compatíveis com o Fixed-Point Toolbox™, e portanto algumas podem requerer codificação manual. Os tipos de dados propostos podem produzir resultados que apresentam um erro com relação ao resultado obtido em ponto flutuante. Se este erro for maior que o admissível para a aplicação, os tipos das variáveis podem ser ajustados manualmente.

Figura 26 Fluxo de desenvolvimento utilizando o HDL Coder™

Uma vez que a conversão para ponto fixo for concluída com sucesso, pode-se partir para a geração de HDL. Diversas opções de otimização estão disponíveis, tanto com relação à área utilizada quanto ao tempo de execução. O HDL Coder™ suporta um subconjunto das

funções disponíveis para programação com o MATLAB™, logo nesta etapa também pode ser necessário reescrever algumas funções manualmente. Neste caso, deve-se passar pelo passo de conversão para ponto fixo novamente. Depois da geração do código, o mesmo é simulado com uma das ferramentas de simulação compatíveis dentro do próprio ambiente MATLAB™. Além disso, a síntese do HDL pode ser feita diretamente do MATLAB™, utilizando uma das ferramentas de síntese compatíveis (MATHWORKS, 2012c).

Algumas práticas são recomendadas ao se utilizar esta abordagem a fim de gerar um código melhor e de forma mais rápida (MATHWORKS, 2012e):

- Serializar os dados de entrada e saída. Estruturas de processamento de dados paralelo necessitam de mais recursos de hardware e de mais pinos;
- As entradas e saídas da função implementada em hardware não podem ser matrizes ou estruturas;
- Utilizar algoritmos de soma e subtração no lugar de algoritmos que usem funções como seno, divisão e módulo, pois operações de soma e subtração utilizam menos recursos;
- Evitar vetores ou matrizes muito grandes, pois estes demandam mais registradores e espaço em RAM;
- É recomendável fazer uso das opções de otimização à disposição na ferramenta, como as opções *Loop unrolling* ou *Loop streaming.* Na opção *loop unrolling* o HDL Coder™ cria múltiplas instâncias do corpo do laço no código gerado, o que torna a execução do algoritmo mais rápida, mas em contrapartida ocupa uma maior área do FPGA e gera um código menos legível. Na opção *loop streaming* o HDL Coder™ gera uma única instância do corpo do laço, a qual é utilizada a cada iteração. Nesta opção, a área ocupada é menor, porém perde-se em velocidade de execução.

A fim de ilustrar a importância da serialização dos dados na geração do código e das diferenças na ocupação dos recursos do FPGA com as opções de otimização, utilizou-se o código de um filtro FIR (*Finite Impulse Response*) para verificar os resultados. A Tabela 1 mostra a ocupação dos recursos obtida com processamento paralelo e com a serialização dos dados. Neste caso, os dados de entrada utilizados foram um dos sinais de vibração à disposição para os testes deste trabalho.

Tabela 1 Previsão de ocupação do FPGA prevista pelo MATLAB™ com dados de entrada paralelos e seriais

Recursos	Paralelo	Serial
Multiplicadores	301050	6
Somadores/subtratores	1204200	30
Registradores	150535	28

A Tabela 2 mostra as diferenças na ocupação dos recursos no caso de dados de entrada serializados e com diferentes otimizações.

Tabela 2 Previsão de ocupação do FPGA prevista pelo MATLAB™ com diferentes otimizações

Recursos	Unroll loops	Stream loops
Multiplicadores	6	1
Somadores/subtratores	5	5
Registradores	28	95

A abordagem adotada para a geração de HDL foi a mesma adotada para a geração de software, isto é, cada algoritmo foi estudado separadamente. O primeiro algoritmo estudado foi o de LR. Estabeleceu-se como entradas do algoritmo o vetor de características a ser classificado e o resultado do treinamento. Desta forma, a estrutura do algoritmo fica fixa, mas desde que o vetor de características se mantenha com o mesmo tamanho, basta trocar o vetor de resultado do treinamento caso se deseje classificar um novo conjunto de dados de outra aplicação. O código em VHDL gerado pelo HDL Coder™ foi primeiramente simulado em

ambiente ModelSim PE Student Edition versão 10.1a. A Figura 27 mostra o resultado da simulação, exibindo os valores obtidos de valor de confiança como um gráfico. É possível observar que a curva obtida para os CVs corresponde à curva esperada obtida em ambiente MATLAB™, mostrada na Figura 21, confirmando a correção da lógica do código gerado.

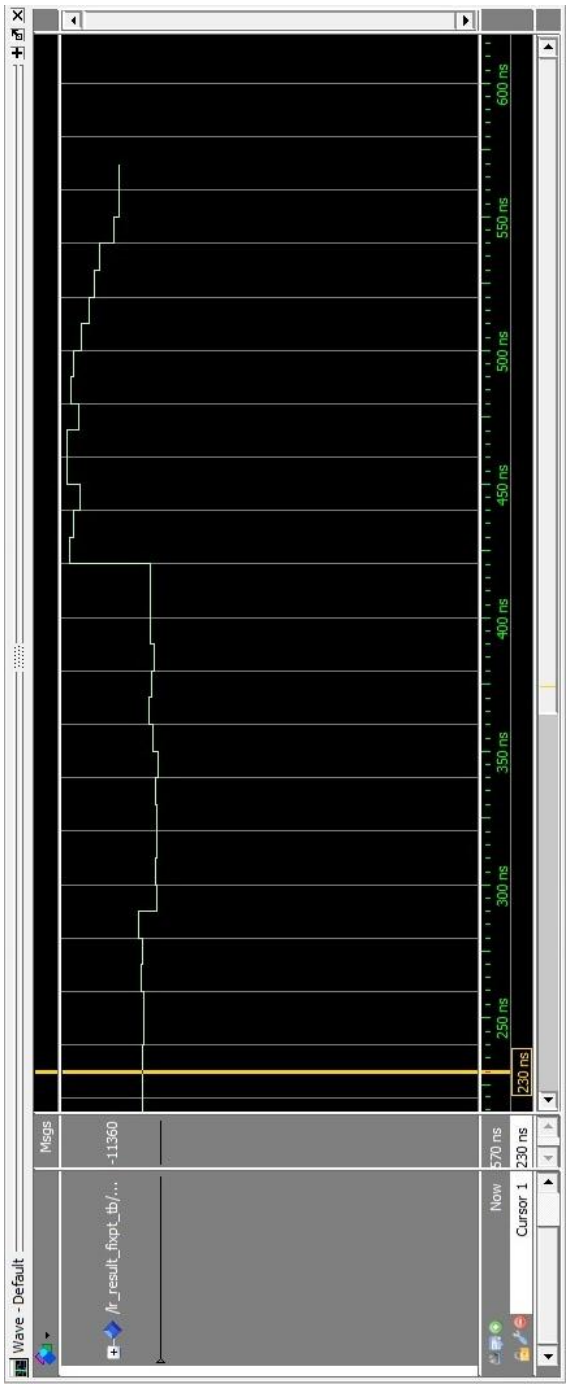

Figura 27 Resultado da simulação do código gerado pelo MATLAB™

Em seguida o algoritmo WPE foi estudado. Fixou-se no código a Wavelet mãe a ser utilizada e os níveis da decomposição, e como parâmetros de entrada deixou-se apenas o sinal. Em função da forma de implementação do algoritmo no MATLAB™, não foi possível realizar a conversão para HDL. O HDL Coder™ não é capaz de gerar código para vetores de dimensão variável, e como este algoritmo foi implementado como um laço para os níveis de decomposição, e a cada nível os sinais a serem filtrados tem dimensão de diferente em função da etapa de decimação, não é possível gerar o HDL sem realizar alterações drásticas na forma de implementação, o que foge do escopo deste trabalho. Em ambos os casos funções como exponencial, potenciação e filtro FIR tiveram de ser manualmente codificados a fim de compatibilizá-las com o Fixed-Point Toolbox™ e o HDL Coder™.

6.4 DESENVOLVIMENTO EM AMBIENTE LABVIEW™

Repetindo o procedimento adotado no desenvolvimento em ambiente MATLAB™, inicialmente criou-se um VI (*Virtual Instrument*) contemplando o sistema de avaliação de desempenho completo para verificar a adequação dos resultados obtidos. A Figura 28 mostra o diagrama de blocos do sistema completo, incluindo o treinamento e execução dos algoritmos de LR e WPE.

Os programas criados neste ambiente são chamados de VIs pois sua aparência e operação emulam um instrumento físico, como um osciloscópio ou um multímetro (NATIONAL INSTRUMENTS, 2010). Cada VI é composto por um painel frontal, onde ficam os controles e indicadores, onde os resultados são exibidos, e um diagrama de blocos, onde o programa é codificado de forma gráfica utilizando blocos funcionais. Os parâmetros dos dois algoritmos foram os mesmos utilizados no ambiente MATLAB™, e utilizou-se o mesmo conjunto de dados para treinamento e testes. A Figura 29 mostra a curva de CV obtida, onde se pode observar que a detecção da falha ocorre corretamente.

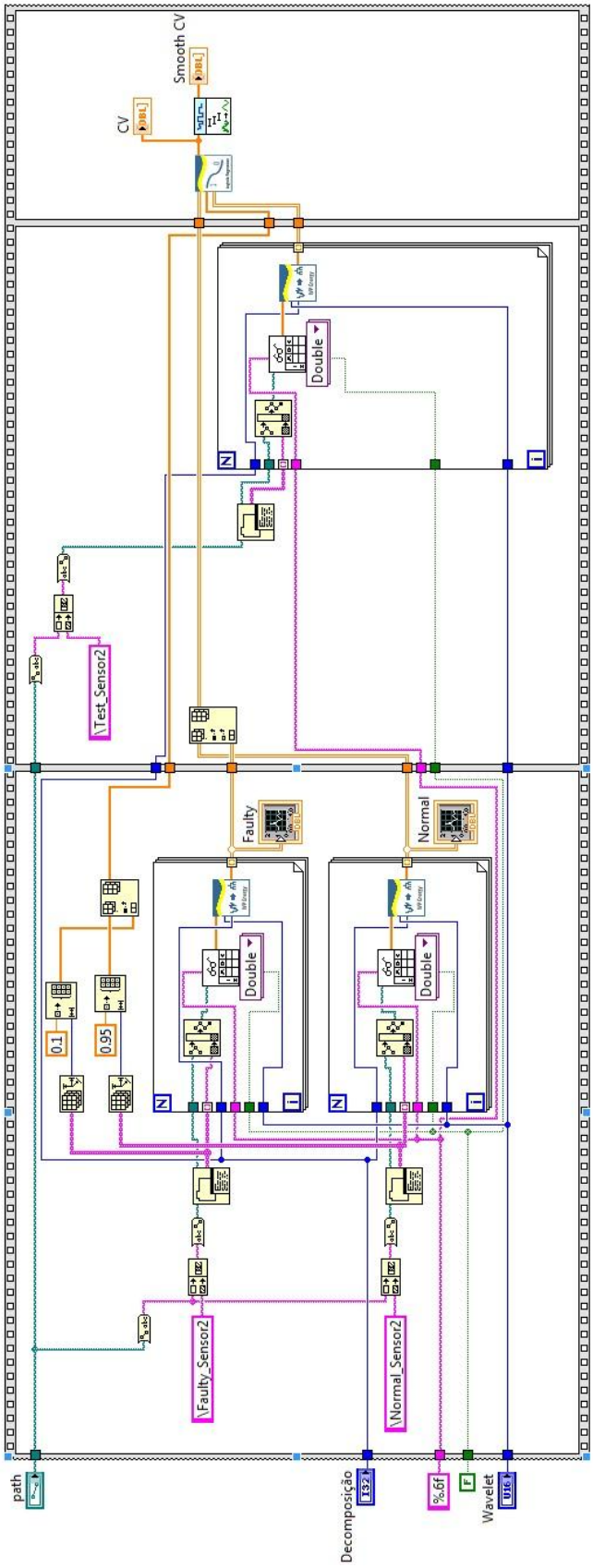

Figura 28 Sistema completo implementado no LabVIEW™

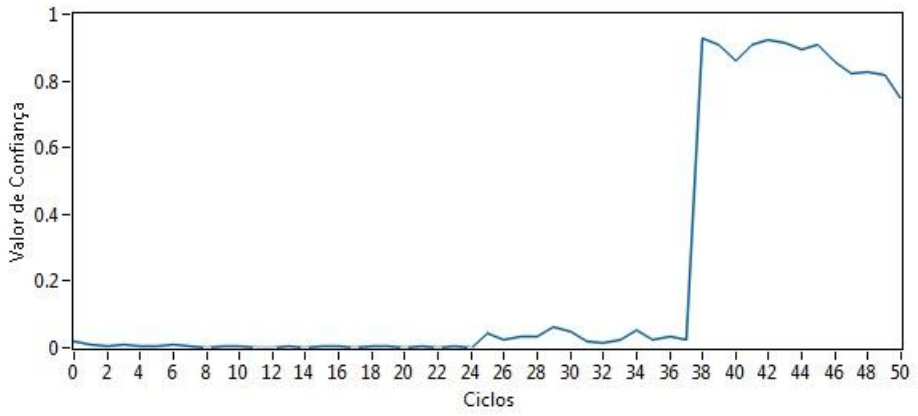

Figura 29 Gráfico de CV obtido no LabVIEW™

Utilizou-se a versão 2011 do software LabVIEW™ e o Watchdog Agent™ Prognostics Toolkit. Para o uso do Watchdog Agent™ Toolkit são necessários adicionalmente os módulos Advanced Signal Processing Toolkit, Sound and Vibration Measurement Suite 2011 e Mathscript RT Module.

A Figura 30 e a Figura 31 mostram os vetores de características correspondentes de uma situação de funcionamento normal e de funcionamento degradado, respectivamente. Foram utilizados os mesmos parâmetros e sinais da avaliação do código em MATLAB™, apresentados nas figuras 22 e 23.

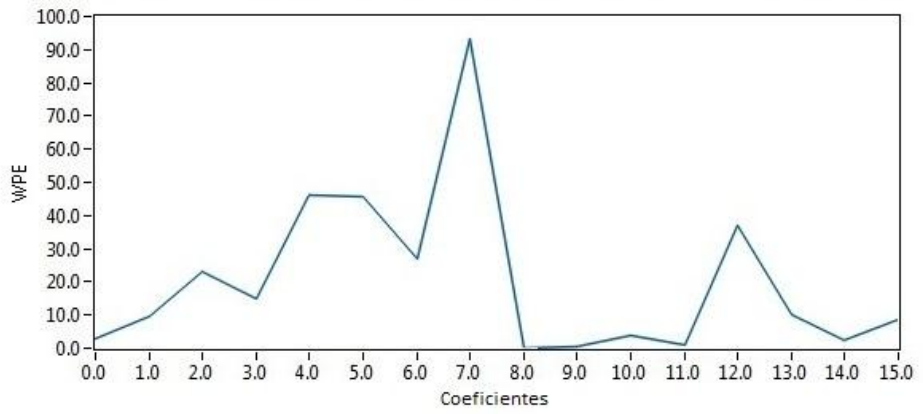

Figura 30 Vetor de características de funcionamento normal obtido no LabVIEW™

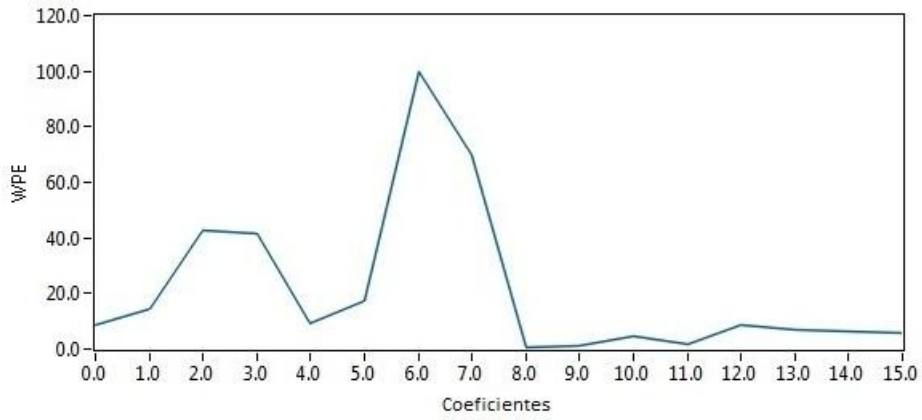

Figura 31 Vetor de características de funcionamento degradado obtido no LabVIEW™

Observa-se que os resultados numéricos desta etapa diferem dos obtidos anteriormente em ambiente MATLAB™. Como não se tem acesso à forma como o código foi implementado em LabVIEW™, não é possível saber as diferenças na implementação que levam a estes resultados. No entanto, observa-se que com duas implementações do algoritmo é possível reconhecer as situações de funcionamento normal e funcionamento de falha corretamente.

Como plataforma para os testes se tem à disposição dois modelos do sistema reconfigurável embarcado de controle e aquisição, o cRIO (CompactRIO). A sua arquitetura de hardware é composta por uma controladora tempo-real embarcada e um chassi, ao qual são conectados os módulos de entrada/saída. O chassi possui um FPGA, que pode ser programado para estabelecer a conexão dos módulos à controladora ou mesmo realizar o processamento dos dados de entrada/saída de forma autônoma, criando desta forma um canal de entrada ou laço de controle rápido (em comparação com os laços de controle estabelecidos na controladora). A programação do sistema é feita diretamente dentro do ambiente gráfico do LabVIEW™ (NATIONAL INSTRUMENTS, 2009b). Dos modelos à disposição, um é composto pelo chassi cRIO-9104 e pela controladora cRIO-9004, e o outro é composto pelo conjunto de chassi e controladora cRIO-9082. A controladora cRIO-9004 possui um

processador com clock de 195MHz, e o conjunto cRIO-9082 possui um processador Intel Core™ i7-660UE com dois núcleos com clock de 1,33GHz cada, e com um FPGA Spartan-6 LX150.

As aplicações de tempo-real, assim como a troca de informações entre o processador da controladora e o FPGA podem ser realizadas utilizando as funcionalidades inerentes ao LabVIEW™. Para a parte de entrada e saída, pode-se escolher entre os diversos módulos compatíveis, tanto analógicos como digitais (NATIONAL INSTRUMENTS, 2009b). A grande vantagem desta abordagem é a integração existente entre o software de desenvolvimento e a plataforma de hardware, bem como a integração dos módulos de entrada e saída na plataforma de hardware, permitindo a criação de sistemas complexos em curto espaço de tempo, se comparado às abordagens tradicionais.

Dos algoritmos desenvolvidos para a ferramenta *Watchdog Agent*™, apenas um subconjunto está disponível na versão para o LabVIEW™. Atualmente, dos módulos existentes na versão em questão, apenas alguns podem ser embarcados na controladora, tais como os módulos de processamento de sinais e avaliação de desempenho, e nenhum deles é compatível com a execução em FPGA. Como não se tem acesso à construção destes módulos, não é possível passar nenhuma etapa do processamento para o FPGA. Para tanto seria necessário reescrever todo o modelo dos algoritmos de interesse, o que não é o foco deste trabalho. Desta forma, para a implementação do WA™ disponível no ambiente LabVIEW™, apenas a geração de software será abordada.

6.4.1 Geração de Software Embarcado

O procedimento adotado foi de realizar a análise de cada etapa da avaliação de degradação (extração de características e avaliação da saúde do sistema) separadamente. Para tanto foram criados três VIs distintos: um para o realizar o treinamento e salvar o resultado

em um arquivo de texto, um para o cálculo do vetor de características e por fim um VI para obtenção dos valores de CV. Conforme se definiu anteriormente, o treinamento dos algoritmos não será embarcado na plataforma, apenas os VIs desenvolvidos para os algoritmos LR e WPE serão individualmente implementados e analisados.

Neste caso, como todos os blocos utilizados são compatíveis com a execução na controladora, o embarque do código se dá de forma extremamente simples. Para tanto, basta criar um projeto de Tempo-Real e adicionar o VI desejado. De forma automática todos os VIs e módulos necessários para a execução do programa, bem como as configurações necessárias, são carregados na memória da controladora. Neste modo, o código é executado na controladora, e o painel frontal do VI pode ser acessado e visualizado em um computador pessoal. Os arquivos de texto com os dados de vibração, vetores de características, e dados resultantes do treinamento são enviados também para a memória da controladora. A Figura 32 mostra o projeto para execução do código na controladora do cRIO, onde pode-se observar o VI desenvolvido para análise do algoritmo LR localizado na controladora.

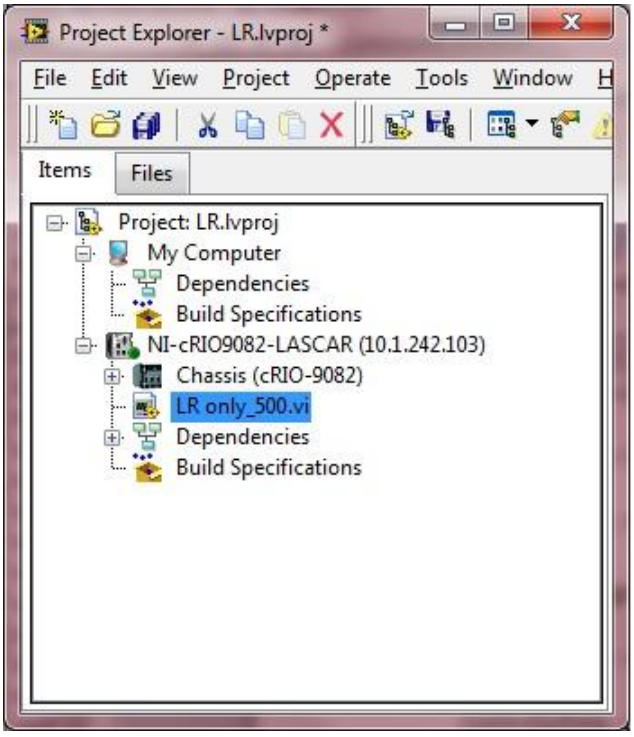

Figura 32 Projeto do código para execução na controladora do cRIO

6.4.2 Utilização do FPGA do cRIO

A fim de explorar a utilização da FPGA disponível no chassi do cRIO, realizou-se o embarque do código em VHDL gerado automaticamente pelo MATLAB™. Para tanto, utilizou-se o bloco *IP Integration Node* (*Intellectual Property Integration Node*), o que permite integrar à programação LabVIEW™ um código desenvolvido em VHDL. Para o seu uso deve-se ter as ferramentas de compilação da Xilinx instaladas no computador de desenvolvimento, e, além disso, ele deve estar contido dentro do bloco *single-cycle Timed Loop*, que irá executá-lo no período especificado correspondente ao clock principal ou derivado utilizado no FPGA.

Para que se possa executar um código no FPGA do cRIO são necessários dois VIs. Um deles, chamado de Host VI, pode ser executado tanto na controladora do cRIO quando em um computador pessoal, e o outro VI que é executado diretamente no FPGA (NATIONAL INSTRUMENTS, 2009a). A Figura 33 mostra o projeto para execução no FPGA, e pode-se ver o Host VI localizado na controladora e o VI localizado no chassi, onde encontra-se o FPGA. No caso desta aplicação, o Host VI envia para o FPGA os vetores dos sinais a serem processados, exibe o resultado do processamento e o salva em disco. Com esta abordagem, diferentes conjuntos de dados podem ser processados pelo FPGA. Caso se optasse por embarcar os vetores de dados no projeto do FPGA, a cada novo vetor de teste seria necessária uma nova compilação e, com isso, poderia-se obter um desempenho ligeiramente diferente, em função do reposicionamento das estruturas dentro do FPGA.

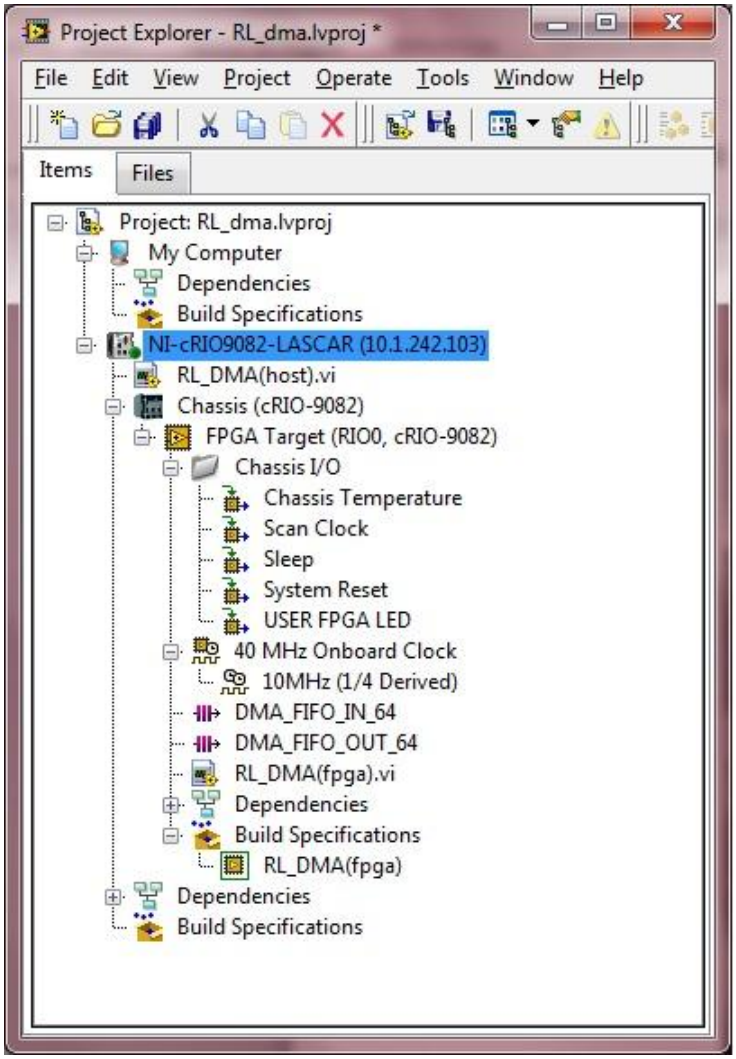

Figura 33 Projeto do código para execução no FPGA do cRIO

O Host VI, neste caso chamado de RL_DMA(host), abre o arquivo especificado, converte os dados para formato de ponto fixo e envia para o processamento no FPGA, através de uma memória FIFO (*First In - First Out*) implementada no próprio FPGA e acessada via DMA (*Direct Memory Access*). Esta estrutura garante a transferência de dados sem perdas entre dois os domínios de clock diferentes, ou seja, o host e o FPGA. A seguir, através de uma segunda FIFO, os dados resultantes do processamento são lidos, exibidos no painel de controle e salvos em arquivo.

O tempo de execução do processo de escrita, processamento e leitura é mensurado através de dois blocos *Tick Counter*, configurados para operar na base de tempo de µs. Para

incluir na medida dos tempos de execução as variações temporais inerentes ao sistema, projetou-se um sistema que possa realizar N medidas e posteriormente calcular o tempo médio de execução da tarefa.

O código executado na controladora, neste caso chamado de VI RL_DMA(fpga), por sua vez, recebe os dados enviados pelo Host VI, executa em hardware o bloco de processamento da rotina de Regressão Logística e devolve ao Host VI o resultado, bem como a mensuração do tempo necessário para execução do *IP Integration Node*.

O uso do *IP Integration Node*, representado na Figura 34, apresenta algumas restrições:

- É executado através de um *single-cycle TimedLoop*, que sincroniza a execução do *IP Integration Node* com o clock principal ou derivado da FPGA.
- Suporta como entrada/saída somente sinais do tipo *std_logic* e *std_logic_vector*, o que limita a interface de sinais a tipos de dados tais como bit, conjunto de bits, ponto fixo (FXP) e inteiros, com e sem sinal. Não é possível conectar ao *IP Integration Node* uma entrada tipo conjunto de inteiros, por exemplo.
- Não é possível até a versão 2011 do LabVIEW™ o uso de representação em ponto flutuante no FPGA.

Estas restrições impactam na codificação do RL_DMA(fpga), como pode ser visto na Figura 34. Optou-se por utilizar ponto fixo com 1 bit para representação da parte inteira. As conversões de tipo devem ser feitas explicitamente, para evitar perda de informação. O conjunto de dados de entrada deve ser desmontado manualmente para ser lido pelo *IP Integration Node*.

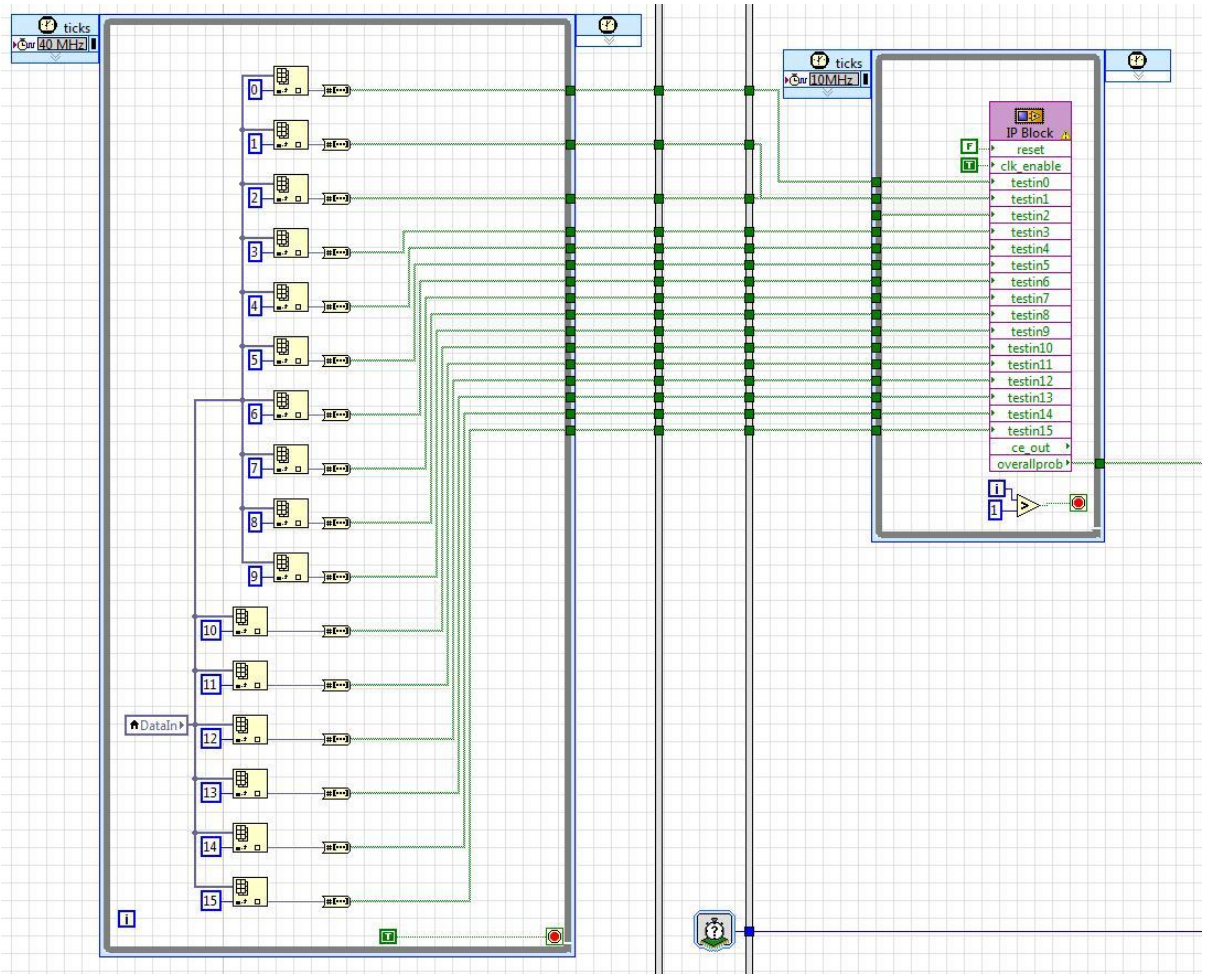

Figura 34 Implementação do *IP Integration Node* com o algoritmo LR

Para simplificar a integração dos arquivos VHDL produzidos pelo HDL Coder™ com o *IP Integration Node*, codificou-se o arquivo *lr_wrapper.vhd*, que estabelece as conexões entre as estruturas de dados exigidas pelo *IP Integration Node* com os arrays de vetores utilizados pelo MATLAB™ na geração de código. Desta forma, se um novo HDL for gerado para este algoritmo, não é necessário alterar o VI desenvolvido no LabVIEW™. Por fim, foi necessária a geração de um clock derivado, com valor de 10MHz, para atender as requisições temporais da compilação do *IP Integration Node.*

7 RESULTADOS

Neste capítulo serão apresentados os resultados práticos obtidos com as diferentes implementações geradas para os algoritmos de interesse quando executadas nas diferentes plataformas a disposição para testes, conforme discutido no capítulo anterior. Além disso, os resultados são analisados e discutidos.

7.1 RESULTADOS DO CÓDIGO GERADO NO MATLAB™

A fim de se obter as medidas de tempo de execução de forma não intrusiva no código, estas foram realizadas utilizando o *cmdlet* (Command-let) *Measure-Command* nativo do Windows PowerShell, que é uma ferramenta de interface por linha de comando (shell). Além disso, os dados de teste foram inicializados dentro do código para que o tempo de leitura dos arquivos não influencie no resultado final. Para o algoritmo LR, os tempos foram medidos para o cálculo dos CVs dos 51 arquivos de teste. Para o algoritmo WPE o tempo corresponde a extração das caraterísticas de uma curva de vibração. A Tabela 3 mostra os resultados de tempo de execução com relação à frequência de operação do processador para o código gerado a partir do MATLAB™. Deve-se observar que os valores de tempo de execução podem variar de acordo com as demais tarefas sendo executadas pelo sistema operacional.

Tabela 3 Tempos de execução em software em relação à frequência de operação para os algoritmos desenvolvidos no MATLAB™

Plataforma	WPE	LR
PC	0,038 ms/MHz	0,023 us/MHz
Advantech Uno	0,653 ms/MHz	0,113 us/MHz

A Tabela 4 mostra a taxa de ocupação do processador durante a execução do código gerado a partir da implementação no MATLAB™. Os resultados foram obtidos utilizando o software *Process Explorer*.

Tabela 4 Taxa de ocupação do processador para os algoritmos desenvolvidos no MATLAB™

Plataforma	WPE	LR
PC	0,3%	0,15%
Advantech Uno	1,98%	0,99%

A Figura 35 mostra a curva obtida no *Host VI* com a execução do código na FPGA do cRIO, para o código gerado pelo HDL Coder™ para o algoritmo LR. Observa-se que o resultado está de acordo com o esperado, repetindo a curva obtida na análise dos algoritmos.

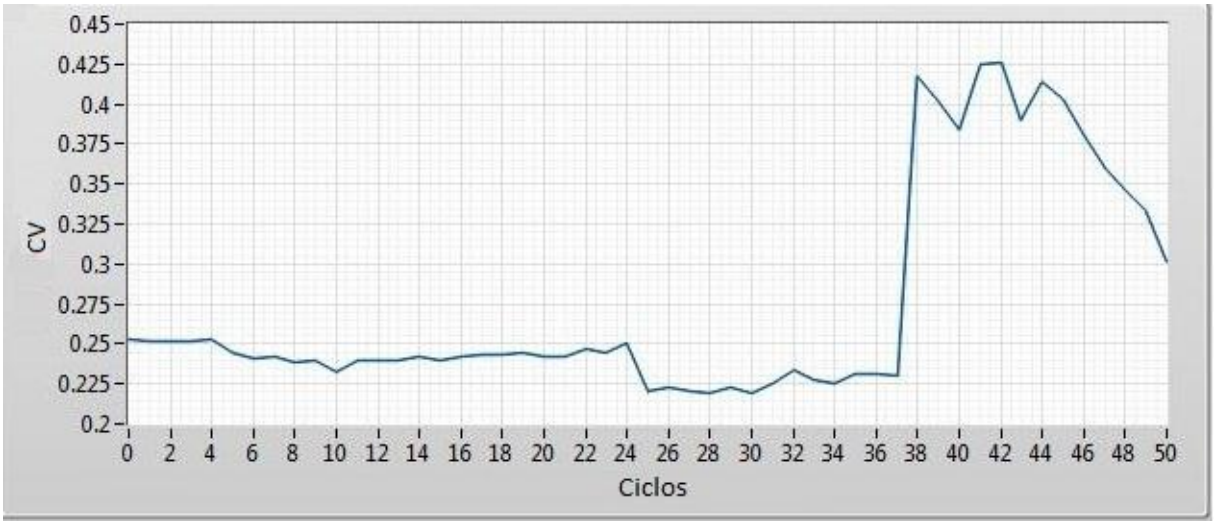

Figura 35 Resultado do código gerado pelo MATLAB™ sendo executado na FPGA do cRIO

A Tabela 5 mostra o tempo para a execução do algoritmo LR no FPGA, sendo que o cálculo foi realizado com 51 ciclos como nas medidas de desempenho em software.

Tabela 5 Tempo de execução em hardware do algoritmo LR desenvolvido no MATLAB™

Plataforma	LR
cRIO	1 us

Durante o processo de geração de HDL, o HDL Coder™ fornece uma previsão do número de recursos de hardware do FPGA utilizados no código gerado. O objetivo principal deste relatório é fornecer uma métrica para avaliação das opções de otimização que podem ser adotadas. No caso desta implementação, optou-se por não utilizar nenhuma otimização. O

relatório de ocupação prevista gerado pelo MATLAB™ pode ser visto na Tabela 6.

Tabela 6 Previsão de ocupação do FPGA prevista pelo MATLAB™ para o algoritmo LR

Recursos	Quantidade utilizada
Multiplicadores	56
Somadores/subtratores	178
Registradores	4
RAM	0
Multiplexadores	40

Na Tabela 7 é apresentada a ocupação dos recursos do FPGA do cRIO, fornecida pelo LabVIEW™ ao final do processo de compilação e síntese.

Tabela 7 Ocupação dos recursos do FPGA na implementação no cRIO do algoritmo LR

Recursos	Ocupação	Percentual ocupado
Total de slices	1253	5,4%
Registradores	3453	1,9%
LUTs	2412	2,6%
DSP48s	16	8,9%
Blocos de RAM	3	1,1%

7.2 RESULTADOS DO CÓDIGO GERADO NO LABVIEW™

Foram utilizados os blocos *Tick Count* para a medição dos tempos de execução, o qual retorna o valor de um contador na unidade especificada de acordo com o tempo médio de execução de cada algoritmo. Este bloco é disponibilizado junto com o Real-Time Toolkit. Também se utilizou a ferramenta Profiler, disponível no ambiente LabVIEW™, que gera um relatório detalhado de tempo de execução e uso de memória de cada VI e sub-VI do sistema. Novamente os VIs foram codificados de forma a executar os algoritmos N vezes a fim de se obter um tempo médio de execução. A Tabela 8 mostra os resultados de tempo de execução obtidos na execução do código em software. Para o algoritmo LR novamente os dados foram

adquiridos para o cálculo de 51 ciclos do atuador.

Tabela 8 Tempos de execução em software em relação à frequência de operação para os algoritmos desenvolvidos no LabVIEW™

Plataforma	WPE	LR
PC	0,015 ms/MHz	0,032 us/MHz
cRIO 9004	6,19 ms/MHz	5,046 us/MHz
cRIO 9082	0,021 ms/MHz	0,0248 us/MHz

Foram utilizados os blocos *RT Get CPU Loads* e *RT Get Memory Usage* para obter dados como a taxa de ocupação da CPU, os quais estão disponíveis apenas para uso com a controladora do cRIO. Para os dados da execução do código no PC utilizou-se novamente o *Process Explorer*. Os resultados podem ser vistos na Tabela 9.

Tabela 9 Percentual de ocupação do processador para os algoritmos desenvolvidos no LabVIEW™

Plataforma	WPE	LR
PC	24%	15%
cRIO 9004	100%	100%
cRIO 9082	0,83%	0,43%

7.3 CONSIDERAÇÕES FINAIS

A Tabela 10 mostra um comparativo qualitativo das diferentes implementações propostas neste trabalho.

Tabela 10 Comparativo qualitativo das implementações propostas

	Plataforma de embarque	Algoritmo	Software	Hardware
MATLAB™	Genérica	WPE	Sim	Não
		LR	Sim	Sim
LabVIEW™	cRIO	WPE	Sim	Não
		LR	Sim	Não

A abordagem de implementação do sistema embarcado utilizando as plataformas de desenvolvimento e de hardware da National Instruments tem como grande vantagem a integração destes sistemas. Existem diversos módulos à disposição para realizar a etapa de aquisição e condicionamento dos sinais de interesse, sem que seja necessário realizar o projeto e validação desta parte do sistema. Ferramentas para se obter métricas de desempenho do sistema também são integradas diretamente no ambiente ou então disponibilizadas como módulos que podem ser facilmente implementadas em um VI.

No entanto, em função das limitações atuais do *Watchdog Agent*™ para o LabVIEW™, não é possível tirar proveito desta integração para implementá-lo no FPGA existente no cRIO. Além disso, não é possível utilizar os recursos do cRIO sem passar pelo ambiente LabVIEW™. Em função disso, a fim de se utilizar um circuito descrito em VHDL desenvolvido em outro ambiente é necessário realizar algumas adaptações para compatibiliza-lo com os requisitos dos módulos disponíveis no LabVIEW™.

As ferramentas disponíveis no MATLAB™ para geração de software e hardware possibilitam o embarque dos algoritmos rapidamente após a validação das técnicas e sem necessidade de se desenvolver um novo projeto do começo ou de se aprender outra linguagem. No entanto, também há uma curva de aprendizado das ferramentas, as quais oferecem uma série de opções de otimização e diretivas de codificação dentro do MATLAB™ para geração de código mais eficiente. Para o desenvolvimento do sistema de manutenção inteligente embarcado, esta abordagem tem a desvantagem de não oferecer as ferramentas para aquisição dos dados como ocorre na abordagem do LabVIEW™.

A Tabela 11 apresenta um resumo dos tempos de execução obtidos para os algoritmos WPE e LR nas diferentes implementações propostas neste trabalho. Outros parâmetros não são diretamente comparáveis, tais como a taxa de ocupação da memória do processador e ocupação dos recursos do FPGA.

Tabela 11 Resumo dos tempos de execução dos algoritmos nas diferentes plataformas

Espaço de projeto	Plataforma	WPE	LR
MATLAB – Software	PC	88 ms	52 us
MATLAB – Software	Advantech Uno	392 ms	68 us
LabVIEW – Software	PC	35 ms	75 us
LabVIEW – Software	cRIO 9004	1207 ms	984 us
LabVIEW – Software	cRIO 9082	28 ms	33 us
MATLAB – Hardware	cRIO 9082	-	1 us

Quanto ao desempenho dos softwares gerados, observa-se um melhor desempenho dos algoritmos quando executados na plataforma cRIO-9082. Este resultado justifica-se pela maior capacidade do processador e também por este estar dedicado a executar o software em questão, ao contrário do que ocorre quando os algoritmos são executado em um computador pessoal no qual outras aplicações estão sendo executadas ao mesmo tempo. No caso da execução do software no PC, observa resultados próximos para as duas implementações (LabVIEW™ e MATLAB™). Quando a plataforma utilizada foi o cRIO-9004, os tempos de execução foram consideravelmente superiores, e observa-se que neste caso, a taxa de ocupação do processador chegou a 100%, o que pode ser um problema se o sistema deve realizar alguma outra tarefa. Já o desempenho dos algoritmos executados no computador industrial foi um pouco inferior ao obtido no PC e no cRIO-9082, no entanto ele ainda fica na ordem de poucas centenas de milissegundos.

O desempenho em hardware foi superior àquele obtido em software, uma vez que esta implementação possibilita que haja paralelismo no processamento. Nota-se que o tempo para execução do algoritmo foi dezenas de vezes inferior ao melhor tempo obtido em software, sendo que o clock utilizado no FPGA também foi dezenas de vezes inferior aquele do processador, comprovando o melhor desempenho esperado da execução em hardware. No entanto, pelas limitações encontradas e mencionadas anteriormente, não foi possível comparar

as implementações realizadas a partir do MATLAB™ e do LabVIEW™. Deve-se ressaltar que o tempo e recursos necessários para o fluxo de dados no sistema de manutenção embarcado não foi abordado neste trabalho, e tendo em vista que é necessário acrescentar diversos blocos para troca de dados com o FPGA do cRIO, este fator deve ser futuramente analisado.

8 CONCLUSÃO

Como já mostrado em trabalhos prévios, o monitoramento dos dados de vibração em conjunto com as técnicas de processamento de sinais utilizadas no sistema de manutenção inteligente Watchdog Agent™ mostrou-se capaz de detectar corretamente as falhas induzidas no atuador. A mudança do valor de confiança obtida é brusca pois os dados de teste foram simulados através da injeção de falhas grosseiras no atuador, numa situação real de degradação espera-se que a curva seja mais suave.

Além disso, da revisão do estado da arte, observa-se que existem poucos trabalhos que implementam um sistema embarcado para manutenção proativa, e estes não exploram a sua implementação em diferentes plataformas. As diferentes opções de plataformas para o sistema embarcado foram discutidas, e também abordou-se o crescente interesse em técnicas de projeto baseados em modelos, ou seja, representações de alto nível do sistema, combinando capacidades de geração automática de código.

A utilização de ferramentas de geração automática de código torna o desenvolvimento do sistema mais rápido, podendo-se partir rapidamente da especificação do sistema e do estudo e validação das técnicas para a implantação em um sistema embarcado para utilização em campo. Como as ferramentas utilizadas geram código portável, genérico, também possibilitam a implementação em diferentes arquiteturas com diferentes processadores ou FPGAs, sem que seja necessário codificar os algoritmos novamente tendo em mente as peculiaridades da plataforma a ser utilizada.

O MATLAB™ já é amplamente utilizado na etapa de criação e validação dos algoritmos, portanto não é necessário passar por toda uma curva de aprendizagem de uma nova linguagem ou ambiente de desenvolvimento ao se utilizar seus recursos de geração de código. Claramente há uma curva de aprendizado para o funcionamento dos geradores de código, uma vez que estes tem diversos parâmetros que podem ser ajustados a fim de se obter

diferentes otimizações no código final e também há uma série de recomendações para o estilo de código que auxiliam a se obter uma conversão mais eficiente e otimizada.

Algumas limitações foram encontradas no uso das ferramentas para geração de VHDL, não sendo possível utilizar os algoritmos sem algumas alterações na implementação. No entanto, se as recomendações fornecidas para o uso destas ferramentas forem seguidas desde o início do desenvolvimento dos algoritmos, o código é gerado de forma rápida, sem necessidade de se realizar um novo projeto do início ao final da etapa de validação.

O código gerado partir das duas implementações do WA™ pode ser prontamente embarcado em um sistema de manutenção. Apenas deve-se considerar que o sistema desenvolvido em ambiente LabVIEW™ pode somente ser executado na plataforma cRIO. Já o código gerado no ambiente MATLAB™, tanto software quanto hardware, é portável para diferentes plataformas. Desta forma, a abordagem proposta neste trabalho mostra-se viável e válida, propiciando um tempo mais curto entre o desenvolvimento dos algoritmos e o seu embarque para utilização em campo.

As principais contribuições deste trabalho foram o estudo dos espaços de projeto para embarque dos algoritmos de manutenção inteligente utilizando os recursos de geração automática de código presente nos ambientes para os quais a ferramenta Watchdog Agent™ está disponível e a análise do desempenho do código gerado. A utilização das ferramentas já disponíveis nestes ambientes, os quais são bastante utilizados para validação dos algoritmos, faz com que não seja necessário o aprendizado de outras ferramentas ou linguagens para o desenvolvimento dos sistemas de manutenção embarcados, como mencionado anteriormente.

O estudo do desempenho dos algoritmos quando implementados em hardware e software também é de extrema importância para o desenvolvimento futuro de uma plataforma reconfigurável de manutenção inteligente. Com este conhecimento pode-se determinar a necessidade de executar os algoritmos em software ou hardware, possibilitando inclusive que

o sistema migre a execução entre software e hardware dinamicamente.

Como trabalhos futuros pode-se citar a geração de código para os demais algoritmos disponíveis na ferramenta, bem como realizar as modificações necessárias nos algoritmos para que os mesmos sejam compatíveis com as ferramentas de geração de código e sigam as diretrizes para uma geração mais eficiente. Também pode-se realizar a codificação manual, tanto para software quanto para hardware dos algoritmos para comparar seu desempenho com aquele obtido com a geração automática, ou, por exemplo, utilizar as ferramentas para geração de código dos fabricantes de FPGA disponíveis para o MATLAB™. Outro fator importante que seve ser aprofundado diz respeito à aquisição e fluxo de dados dentro do sistema de manutenção embarcado, que não foi abordado neste trabalho. Além disso, pode-se imediatamente proceder à implantação do sistema de manutenção completo, consistindo da extração de características e avaliação da degradação, em uma plataforma para embarque e validação em campo.

REFERÊNCIAS

AGRESTI, A.; FINLAY, B. Logistic Regression: modeling categorical responses. In: **Statistical Methods for the Social Sciences**. Lebanon: Prentice Hall, 2007. p. 687-737. ISBN: 0130272957.

ALLGAYER, R. S. **FEMTONODE**: arquitetura de nó-sensor reconfigurável e customizável para rede de sensores sem fio. 2009. 119 p. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2009.

ALMEIDA, C. A. L. et al. Intelligent Thermographic Diagnostic Applied to Surge Arresters: a new approach. **IEEE Transactions on Power Delivery,** New York, v. 24, n. 2, p. 751-757, 2009.

AMIRAT, Y. et al. A brief status on condition monitoring and fault diagnosis in wind energy conversion systems. **Renewable and Sustainable Energy Reviews**, Oxford, v. 13, p. 2629 – 2636, 2009.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR-5462**: confiabilidade e mantenabilidade. Rio de Janeiro, 1994. 37p.

BÖESCH, K. **Detecção de falhas por fusão de sensores em atuadores elétricos**. 2011. 93 p. Projeto de Diplomação (Graduação em Engenharia Elétrica) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2011.

BOSA, J. L. **Sistema Embarcado para a Manutenção Inteligente de Atuadores Elétricos**. 2009. 169 p. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2009.

CARRO, L. **Projeto e Prototipação de Sistemas Digitais**. Porto Alegre: Editora da UFRGS, 2001. v. 1, 234 p. ISBN: 8570255896.

CARRO, L.; WAGNER, F. Sistemas Computacionais Embarcados. In: JORNADAS DE ATUALIZAÇÃO EM INFORMÁTICA, 22, 2003, Campinas. **Anais...** Campinas: Sociedade Brasileira de Computação. 2003. v. 1, p. 45-94.

COSTA, C. da et al. A new approach for real time fault diagnosis in induction motors based on vibration measurement. In: IEEE International Instrumentation and Measurement Technology Conference, 2010, Austin. **Proceedings…** Piscataway: IEEE. 2010. p. 1164-1168.

DAUBECHIES, I. **Ten Lectures on Wavelets**. Philadelphia: Society for Industrial and Applied Mathematics, 1992. 357 p.

DJURDJANOVIC. D.; LEE, J.; NI, J. Watchdog Agent^{TM} - an infotronics-based prognostics approach for product performance degradation assessment and prediction**. Advanced Engineering Informatics**, Oxford, v. 17, p. 109-125, 2003.

ESPÍNDOLA, D. B. et al. A Visualização Mista nos Sistemas de Automação de Processos de Manutenção Preditiva de Máquinas da Indústria. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 18, 2010, Bonito, **Anais...** Bonito: Sociedade Brasileira de Automática, 2010. p. 3021-3027.

FACCIN, F. C. **Manutenção Inteligente**: fusão de sensores aplicada na detecção de falhas em atuadores elétricos. 2011. 89 p. Projeto de Diplomação (Graduação em Engenharia Elétrica) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2011.

GÖTZ, M. **Run-time Reconfigurable RTOS for Reconfigurable Systems-on-Chip**. 2009. 154p. Tese (Doutorado em Engenharia) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2007.

GONÇALVES, L. F. **Desenvolvimento de um sistema de manutenção inteligente embarcado**. 2011. 233 p. Tese (Doutorado em Engenharia) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2011.

GONÇALVES, L. F. et al. Design of an embedded system for the proactive Maintenance of electrical valves, In: ANNUAL SYMPOSIUM ON INTEGRATED CIRCUITS AND SYSTEMS DESIGN, 22, 2009, Natal. **Proceedings...**, New York: Association for Computing Machinery, 2009.

HENRIQUES, R. V. B. et al. Detecção de falhas em atuadores nas linhas de transporte de petróleo e/ou derivados, In: CONGRESSO INTERNACIONAL E EXPOSIÇÃO SUL-AMERICANA DE AUTOMAÇÃO, SISTEMAS E INSTRUMENTAÇÃO, 14, 2010, São Paulo. **Anais...**. São Paulo: International Society of Automation, 2010.

IEEE. **IEEE Standard Computer Dictionary**: a compilation of IEEE standard computer glossaries. Washington: Institute of Electrical and Electronics Engineers, 1991. 218 p.

IEEE. **IEEE Standard Glossary of Computer Hardware Termonilogy**. Washington: Institute of Electrical and Electronics Engineers, 1995. 109 p.

IMS CENTER. **Documentation of Watchdog Agent^{TM} Toolbox Algorithms**. [S. l]: IMS Center, 2007. 61 p.

JARITZ, S. Embedding audio signal processing on different hardware and software platforms by using frameworks. In: EMBEDDED WORLD 2012 EXHIBITION & CONFERENCE, 10, 2012, Nuremberg. **Proceedings…** [S. l.: s. n., 2012]. 11 p. Disponível em: <http://www.et.fh-jena.de/Jaritz/embeddedWorldConference2012StefanJaritzNoPageNumbers.pdf>. Acesso em: 20 ago. 2012.

KIRITSIS, D. Closed-loop PLM for intelligent products in the era of the internet of the things, **Computer-Aided Design**, Guildford, v. 43, issue 5, p. 479-501, 2011.

KIRNER, R. et al. Fully Automatic Worst-Case execution Time Analysis for MATLAB/Simulink Models. In: EUROMICRO CONFERENCE ON REAL-TIME

SYSTEMS, 14, 2002, Vienna. **Proceedings…** [S. l]: IEEE Computer Society, 2002. p. 31-40.

KOBBACY, K. A. et al. New technologies for maintenance. In: **Complex System Maintenance Handbook**. London: Springer, 2008. p. 49 – 78.

LEE, J. et al. Intelligent prognostics tools and e-maintenance. **Computers in Industry**, Amsterdam, v. 57, n. 6, p. 476-489, 2006.

LEE, J. et al. Informatics platform for designing and deploying e-manufacturing systems. In: **Collaborative Design and Planning for Digital Manufacturing**. London: Springer, 2009. p. 1-35.

LEE, J.; GHEFFARI, M.; ELMELIGY, S. Self-maintenance and engineering immune systems: towards smarter machines and manufacturing systems. **Annual Reviews in Control**, Oxford, v. 35, issue 1, p. 11-122, 2011.

LIMA, C. R. C. de; MARCORIN, W. R. Análise de custos de manutenção e de não manutenção em equipamentos produtivos. **Revista de Ciência e Tecnologia**, Piracicaba, v. 11, n. 22, p. 35-42, 2003.

LINXIA, L.; LEE, J. Desing of a reconfigurable prognostics platform for machine tools. **Expert Systems with Applications**, New York, v. 37, p. 240-252, 2010.

LU, B.; DUROCHER, D.; STEMPER, P. Predictive maintenance techniques, **IEEE Industry Applications Magazine,** New York, v. 15, n. 6, p. 52-60, Nov.-Dec. 2009.

MALLAT, A. G. A Theory for Multiresolution Signal Decomposition: the wavelet representation. **IEEE Transactions on Pattern Analysis and Machine Intelligece**, New York, v. 11, n. 7, p. 674-693, 1989.

MATHWORKS. **MATLAB Coder™**: getting started guide. [S. l], 2012a. 125 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/coder/coder_gs.pdf >. Acesso em: 10 abr. 2012.

MATHWORKS. **Embedded Coder™**: getting started guide. [S. l]. 2012b. 123 p. Disponível em: < http://www.mathworks.com/help/pdf_doc/ecoder/ecoder_gs.pdf>. Acesso em: 10 abr. 2012.

MATHWORKS, **HDL Coder™**: getting started guide. [S. l], 2012c. 66 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_gs.pdf >. Acesso em: 10 abr. 2012.

MATHWORKS, **HDL Verifier™**: getting started guide. [S. l], 2012d. 15 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/hdlverifier/hdlv_gs_book.pdf>. Acesso em: 10 abr. 2012.

MATHWORKS, **HDL Coder™**: user's guide. [S. l], 2012e. 1128 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_ug.pdf >. Acesso em: 10 abr. 2012.

MAXFIELD, C. **The Design Warrior's Guide to FPGAs**. Oxford: Newnes, 2004. 542 p. ISBN: 0750676043.

MOON, T. Y.; SEO, S. H.; KIM, J. H. Simulation with Consideration of Hardware Caracteristics and Auto-generated code Using Matlab/Simulink. In: INTERNATIONAL CONFERENCE ON CONTROL, AUTOMATION AND SYSTEMS, 2007, Seoul. **Proceedings…** New York: IEEE, 2007. p. 1494-1498.

MOORE, W.; STARR, A. An intelligent maintenance system for continuous cost-based prioritisation of maintenance activities. **Computers in Industry**, Amsterdam, v. 57, n. 6, p. 595-606, 2006.

MULLER, A.; MARQUEZ, A. C.; IUNG, B. On the concept of E-maintenance: review and current research. **Reliability Engineering and System Safety**, Barking, v. 93, p. 1165-1187, 2008.

NATIONAL INSTRUMENTS. **Getting Started with CompactRIO™ and LabVIEW™**. [S. l], 2009a. 31 p. Disponível em: <http://www.ni.com/pdf/manuals/372596b.pdf>. Acesso em: 20 fev. 2012.

NATIONAL INSTRUMENTS. **CompactRIO Developers Guide**: Recommended LabVIEW Architectures and Development Practices for Machine Control Applications. [S. l], 2009b. 226 p. Disponível em: <http://www.mit.bme.hu/system/files/oktatas/targyak/7258/criodevgudfull.pdf>. Acesso em: 20 fev. 2012.

NATIONAL INSTRUMENTS. **LabVIEW™**: getting started with LabVIEW. [S. l.], 2010. 97 p. Disponível em: < http://www.ni.com/pdf/manuals/373427g.pdf >. Acesso em: 20 fev. 2012.

NIU, G.; YANG, B.-S. Intelligent condition monitoring and prognostic system based on data-fusion strategy. **Expert Systems with Applications**, New York, v. 37, n.12, p. 8831-8840, Dec. 2010.

PEREIRA, L. A.; GAZZANA, D. S.; PEREIRA, L. F. A. Motor Current Signature Analysis and Fuzzy Logic Applied to the Diagnosis of Short-Circuit Faults in Induction Motors. In: ANNUAL CONFERENCE OF INDUSTRIAL ELECTRONICS SOCIETY, 31, 2005, Raleigh. **Proceedings…** Raleigh: IEEE Industrial Electronics Society, 2005. p. 6-10.

PICCOLI, L. et al. Uma solução embarcada para a predição e detecção de falhas em válvulas de transmissão de petróleo e derivados. In: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 19, 2012, Campina Grande, **Anais...** Campina Grande: Sociedade Brasileira de Automática, 2012. p. 4930-4937.

QIU, H. et al. Wavelet filter - based weak signature detection method and its application on rolling element bearing prognostics. **Journal of Sound and Vibration**, London, v. 286, issues 4-5, p. 1066-1090, 2006.

RIOUL, O.; VETTERLI, M. Wavelets and signal processing. **IEEE Signal Processing Magazine**, New York, v. 8, n. 4, p. 14-38, 1991.

STECHELE, W.; HERRMANN, S.; HERKERSDORF, A. Towards a Dynamically Reconfigurable System-in-Chip Platform for Video Signal Processing, In: ARCS ORGANIC

AND PERVASIVE COMPUTING WORKSHOPS, 2004, Augsburg. **Preceedings…** Bonn: Gesellschaft für Informatik E.V., 2004. v. 42, p. 225-234.

THURSTON, M. G. An open standard for Web-based condition-based maintenance systems. In: IEEE SYSTEMS READINESS TECHNOLOGY CONFERENCE, 2001, Valey Forge. **Proceedings…** New York: IEEE, 2001. p. 401-415.

WEHRMEISTER, M. A.; FREITAS, E. P.; PEREIRA, C. E. An Infrastructure for UML-Based Code Generation Tools. In: **IFIP International Federation for Information Processing 2009**. Heidelberg: Springer, 2009, v. 310, p. 32-43, 2009. ISBN: 9783642042836.

YANG, Z.; DJURDJANOVIC, D.; NI, J. Maintenance scheduling in manufacturing systems based on predicted machine degradation. **Journal of Intelligent Manufacturing**, Secaucus, v. 19, issue 1, p. 87-98, 2008.

YAN. J; LEE, J. Degradation Assessment and Fault Modes Classification using Logistic Regression. **Journal of Manufacturing Science and Engineering**, New York, v. 127, n. 4, p. 912-914, 2005.