Universidade do Estado do Rio de Janeiro
Centro de Tecnologia e Ciências
Faculdade de Engenharia

Yuri Marchetti Tavares

Sistema Integrado de Hardware/Software para Rastreamento de Alvos

Rio de Janeiro
2016

Yuri Marchetti Tavares

Sistema Integrado de Hardware/Software para Rastreamento de Alvos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Orientadora: Prof.ª Dr.ª Nadia Nedjah
Coorientadora: Prof.ª Dr.ª Luiza de Macedo Mourelle

Rio de Janeiro
2016

CATALOGAÇÃO NA FONTE
UERJ / REDE SIRIUS / BIBLIOTECA CTC/B

T231 Tavares, Yuri Marchetti.
Sistema integrado de hardware/software para rastreamento de alvos / Yuri Marchetti Tavares. – 2016.
126f.

Orientador: Nadia Nedjah.
Coorientador: Luiza de Macedo Mourelle.
Dissertação (Mestrado) – Universidade do Estado do Rio de Janeiro, Faculdade de Engenharia.

1. Engenharia Eletrônica. 2. Rastreamento automático - Dissertações. 3. Sistemas embarcados - Dissertações. 4. Otimização - Enxame de partículas - Dissertações. I. Nedjah, Nadia. II. Mourelle, Luiza de Macedo. III. Universidade do Estado do Rio de Janeiro. IV. Título.

CDU 004.031.6

_____ _____

Assinatura Data

Yuri Marchetti Tavares

Sistema Integrado de Hardware/Software para Rastreamento de Alvos

Dissertação apresentada, como requisito parcial para obtenção do título de Mestre, ao Programa de Pós-Graduação em Engenharia Eletrônica, da Universidade do Estado do Rio de Janeiro. Área de concentração: Sistemas Inteligentes e Automação.

Aprovado em: 31 de agosto de 2016.

Banca Examinadora:

Prof.ª Dr.ª Nadia Nedjah (Orientadora)
Faculdade de Engenharia - UERJ

Prof.ª Dr.ª Luiza de Macedo Mourelle (Orientadora)
Faculdade de Engenharia - UERJ

Prof. Dr. Thomas Walter Rauber
Departamento de Informática - UFES

Prof. Dr. Paulo Victor Rodrigues de Carvalho
Instituto de Engenharia Nuclear - IEN

Rio de Janeiro
2016

DEDICATÓRIA

Dedico este trabalho aos meus pais pela orientação, pela ajuda e pela minha formação técnica e moral. Vocês são a minha referência! À minha esposa Andressa pelo amor, cuidado e compreensão, principalmente no momentos em que precisei me dedicar aos estudos. Você é um presente de Deus para mim! Aos meus irmãos Caio e Lucas, que sempre torcem pelas minhas conquistas.

AGRADECIMENTOS

Agradeço primeiramente a Deus pela vida, saúde e por esta oportunidade de aperfeiçoamento profissional que Ele me concedeu. Sem Ele nada posso fazer.

Agradeço às minhas orientadoras Nadia Nedjah e Luiza de Macedo Mourelle pela orientação, apoio, atenção e disponibilidade que dispensaram ao longo do presente trabalho.

Agradeço aos meus chefes Vice-Almirante Alipio Jorge, Contra-Almirante Álvaro e Capitão de Mar e Guerra (EN) Marzullo, que me autorizaram a realizar esse curso, em caráter excepcional.

Agradeço aos meus chefes Capitão de Mar e Guerra (EN) Alexandre de Vasconcelos Siciliano, Capitão de Fragata (EN) Fernando Antonio Almeida Coelho e Capitão de Fragata (EN) Rogério Araujo de Barros pela compreensão, apoio e orientação nos momentos cruciais deste trabalho.

Agradeço aos colegas de mestrado Alan Sá, Alejandra, Alexandre Cardoso, Luneque e Nicolás pela troca de conhecimentos e companheirismo.

Agradeço aos professores José Franco Machado do Amaral, Jorge Luís Machado do Amaral e Maria Luiza Fernandes Velloso pelos ensinamentos que recebi durante o curso.

"... Há mais pessoas que desistem do que pessoas que fracassam ..."

Henry Ford

RESUMO

TAVARES, Yuri Marchetti. *Sistema Integrado de Hardware/Software para Rastreamento de Alvos*. 2016. 126f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

O *template matching* é uma técnica importante para rastreamento de padrões em imagens. O objetivo consiste em encontrar um objeto, a partir de um modelo pré-estabelecido, em uma sequência de *frames*. Para avaliar o grau de similaridade entre duas imagens, o Coeficiente de Correlação de Pearson é amplamente utilizado. Esse coeficiente é calculado para cada pixel, o que é computacionalmente muito custoso. O presente trabalho implementa a tarefa de *template matching* em um sistema embarcado, conferindo-lhe grande versatilidade, desempenho e possibilitando seu uso em equipamentos portáteis. Para obter melhor tempo de processamento, o cálculo do coeficiente de correlação foi implementado através de coprocessador dedicado, aproveitando o paralelismo inerente do processo. Além disso, o processo de busca do ponto de correlação foi auxiliado por uma técnica de otimização baseada em enxame de partículas, executado em software por um processador de propósito geral, em uma abordagem integrada de hardware/software. Os testes realizados demonstraram resultados compatíveis com aplicações em tempo real.

Palavras-chave: Sistemas embarcados. Co-design. Otimização por enxame de partículas. Template matching. Correlação. Rastreamento.

ABSTRACT

Sistema Integrado de Hardware/Software para Rastreamento de Alvos
TAVARES, Yuri Marchetti. *Integrated Hardware/Software System for Target Tracking.* 2016. 126f. Dissertação (Mestrado em Engenharia Eletrônica) – Faculdade de Engenharia, Universidade do Estado do Rio de Janeiro, Rio de Janeiro, 2016.

The template matching is an important technique used for pattern tracking. The goal consists of finding a given pattern, given as a prescribed model, in a sequence of frames. In order to evaluate the similarity of two images, Pearson's Correlation Coefficient is widely used. This coefficient is computed for each one of the image pixels, which entails a computationally very expensive operation. This work implements the template matching as an embedded system. This approach allows for a great versatility to use the final design in portable equipment. In order to achieve lower processing time, a dedicated co-processor to perform the correlation computation is designed and implemented. The Particle Swarm Optimization is used to improve the search for the maximum point of correlation. It is implemented in software and run by general purpose processor. This is usually called the co-design approach. The tests show promissing results that are compatible with real time requirement applications.

Keywords: Embedded systems. Co-design. Particle swarm optimization. Template matching. Correlation. Tracking.

LISTA DE FIGURAS

1 Míssil típico (SIOURIS, 2004) 16
2 Erro: distância do alvo até centro da imagem (COLLINS; ZHOU; TEH, 2005) 17

3 Exemplo de rastreamento para imagem "cameraman"(a) por pontos, (b) por *kernel*, (c) por silhueta. 23
4 Matrizes que representam uma imagem em preto e branco (esquerda) e um *template* (direita). 26
5 Ilustração do processo de *template matching* 27

6 Imagem aeronave (esquerda) com correlação calculada em cada pixel (direita) 39
7 Componentes do vetor velocidade da partícula. 42
8 Fluxograma do PSO 42
9 Duas topologias típicas do PSO 43
10 Fluxograma do GA 45
11 Imagem pickup (esquerda) com correlação calculada em cada pixel (direita) 49
12 Imagem carros (esquerda) com correlação calculada em cada pixel (direita) 50
13 Comportamento das partículas como enxame. 51

14 Diagrama em blocos da placa SVDK (SENSORTOIMAGE, 2015) 55
15 Blocos funcionais da família Zynq-7000 (XILINX, 2015) 56
16 Tempo gasto com cada tarefa (*ms*) em uma iteração (escala logarítmica) . . 59
17 Macro-arquitetura do sistema proposto 60
18 Implementação do sistema no Vivado. 60
19 Macro-arquiteura do coprocessador 61
20 Arquitetura do Bloco 1 62
21 Arquitetura do Bloco 2 63
22 Arquitetura do Bloco 3 63

23 Componentes *media* e *soma A2* 67
24 Componente *count until* 67
25 Componente *subt A2* 68
26 Multiplicação em binário 70
27 Circuito para multiplicar dois números inteiros positivos 70
28 Componente *multCLK*. 71
29 Diagrama em Blocos de uma Máquina de Estados Finitos 72
30 Diagrama de Estados da máquina FSM-3B 73
31 Ilustração da divisão inteira de 157 por 4. 75
32 Ilustração da divisão fracionária de 157 por 4. 76
33 Ilustração da divisão inteira de 157 por 4 em base binária 77
34 Diagrama do componente *DIV_inteiro* 78
35 Diagrama do componente *DIV_frac* 79

36 SQRT 81
37 Diagrama de Estados da máquina FSM-SR 81
38 Componente Sincro 82
39 Diagrama de Estados da máquina FSM-SY 83

40 Variação de w para rastreamento na imagem *hollywood* 87
41 Variação de c_1 para rastreamento na imagem *hollywood* 87
42 Variação de c_2 para rastreamento na imagem *hollywood* 88
43 Variação do número de partículas e IT_{MAX} para imagem *hollywood* 89
44 Pedaços retirados da imagem *redcar* 90
45 Imagem *pickup* e seu comportamento para correlação. 93
46 Gráficos de barras para imagem *pickup* 94
47 Imagem *truck* e seu comportamento para correlação 95
48 Gráficos de barras para imagem *truck* 96
49 Imagem *redcar* e seu comportamento para correlação. 97
50 Gráficos de barras para imagem *redcar* 98
51 Imagem *hollywood* e seu comportamento para correlação 99
52 Gráficos de barras para imagem *hollywood* 100
53 Imagem *carros* e seu comportamento para correlação 101
54 Gráficos de barras para imagem *carros* 102
55 Imagem *sedan* e seu comportamento para correlação 104
56 Gráficos de barras para imagem *sedan* 105
57 Imagem *IRcar1* e seu comportamento para correlação 106
58 Gráficos de barras para imagem *IRcar1* 107
59 Imagem *IRcar2* e seu comportamento para correlação 108
60 Gráficos de barras para imagem *IRcar2* 109
61 Imagem *IRcar3* e seu comportamento para correlação 110
62 Gráficos de barras para imagem *IRcar3* 111
63 Tempo de processamento para as imagens de referência (BE). 112
64 Tempo de processamento para as imagens de referência (PSO). 112
65 Confiabilidade para as imagens de referência 113
66 Dimensões e centro dos pixels nos *templates* 115

LISTA DE TABELAS

1 Alguns resultados para BE, GA e PSO 50
2 Simulações Monte Carlo para GA 52
3 Simulações Monte Carlo para PSO-G. 52
4 Simulações Monte Carlo para PSO-L 52

5 Alguns resultados para *aeronave* 57
6 Alguns resultados para *carros* 57
7 Média dos resultados para 100 repetições 57
8 Tempos de processamento para uma iteração da imagem *aeronave*. 58

9 Saídas associadas aos estados de FSM-3B 73
10 Saídas associadas aos estados de FSM-SR 81
11 Saídas associadas aos estados de FSM-SY 82

12 Confiabilidade média para 1000 rastreamentos com parâmetros iniciais . . . 86
13 Taxa de confiabilidade média para imagem *hollywood* (1000 rastreamentos) 86
14 Confiabilidade média para 1000 rastreamentos com parâmetros otimizados 86
15 Variação do número de partículas e número máximo de iterações. 88
16 Valor de PCC para 5 imagens da Figura 44 com relação ao *template* de *redcar* 90
17 Valor do PCC calculado para 50 imagens de *redcar* 90
18 Resultados de BE para imagem *pickup* 94
19 Resultados de PSO médios para imagem *pickup* (1000 rastreamentos) 94
20 Resultados de BE para imagem *truck* 96
21 Resultados de PSO médios para imagem *truck* (1000 rastreamentos) 96
22 Resultados de BE para imagem *redcar* 98
23 Resultados de PSO médios para imagem *redcar* (1000 rastreamentos) 98
24 Resultados de BE para imagem *hollywood* 100
25 Resultados de PSO médios para imagem *hollywood* (1000 rastreamentos). . 100
26 Resultados de BE para imagem *carros* 102
27 Resultados de PSO médios para imagem *carros* (1000 rastreamentos) 102
28 Resultados de BE para imagem *sedan* 103
29 Resultados de PSO médios para imagem *sedan* (1000 rastreamentos). 104
30 Resultados de BE para imagem *IRcar1* 105
31 Resultados de PSO médios para imagem *IRcar1* (1000 rastreamentos). . . . 106
32 Resultados de BE para imagem *IRcar2* 108
33 Resultados de PSO médios para imagem *IRcar2* (1000 rastreamentos). . . . 108
34 Resultados de BE para imagem *IRcar3* 110
35 Resultados de PSO médios para imagem *IRcar3* (1000 rastreamentos). . . . 110
36 *Speedup* de PSO_{HP} com relação aos outros cenários. 112

LISTA DE ALGORITMOS

1 Divisão $Q = A/B$ com 24 bits de precisão fracional 64
2 Divisão inteira $Q = A/B$ 80

LISTA DE SIGLAS

BE	Busca Exaustiva
BRAM	Block RAM
FFT	Fast Fourier Transform
FPGA	Field Programmable Gate Array
FSM	Finite State Machine
GA	Genetic Algorithms
HP	Hardware Pipeline
HS	Hardware Sequencial
LUT	Look-up Table
MHT	Multiple Hypothesis Tracking
PCC	Pearson's Correlation Coeficient
PL	Programmable Logic
PS	Processing System
PSO	Particle Swarm Optimization
PSO-G	PSO *global best*
PSO-L	PSO *local best*
SAD	Sum of Absolute Differences
SIFT	Scale-invariant Feature Transform
SO	Software
SSD	Sum of Squared Differences
SURF	Speeded Up Robust Features
SVDK	Smart Vision Development Kit
VHDL	Very high speed integrated circuits Hardware Description Language

SUMÁRIO

INTRODUÇÃO **15**

1 RASTREAMENTO DE PADRÕES **19**
1.1 Rastreamento de Padrões/Objetos 19
1.1.1 Detecção do Objeto 20
1.1.2 Classificação do Objeto 21
1.1.3 Técnicas para Rastreamento 22
1.1.3.1 Rastreamento por Pontos 23
1.1.3.2 Rastreamento por Núcleo 24
1.1.3.3 Rastreamento por Silhueta 25
1.2 *Template Matching* 25
1.3 Correlação Cruzada Normalizada 28
1.4 Dificuldades em Rastreamento de Padrões 29
1.5 Considerações Finais do Capítulo 30

2 TRABALHOS RELACIONADOS **31**
2.1 *Template Matching* 31
2.2 Implementação em Hardware 33
2.3 Correção do *Drift* 35
2.4 Considerações Finais do Capítulo 36

3 ESCOLHA DA TÉCNICA **38**
3.1 Problema de Otimização 38
3.2 Técnicas de Inteligência Computacional 40
3.2.1 Otimização por Enxame de Partículas 41
3.2.1.1 PSO *Global Best* 43
3.2.1.2 PSO *Local Best* 43
3.2.2 Algoritmos Genéticos 44
3.2.3 Critérios de Parada e Medidores de Desempenho 46
3.3 Comparação das Técnicas 47
3.4 Considerações Finais do Capítulo 53

4 IMPLEMENTAÇÃO DA ARQUITETURA PROPOSTA **54**
4.1 Plataforma de Hardware 54
4.2 Motivação para uso do Coprocessador Dedicado 56
4.3 Descrição da Macro-arquitetura 59
4.3.1 Coprocessador Dedicado 60
4.3.1.1 Macro-arquitetura do Bloco 1 61
4.3.1.2 Macro-arquitetura do Bloco 2 62

4.3.1.3 Macro-arquitetura do Bloco 3 62
4.3.2 Controladores das Memórias 64
4.4 Considerações Finais do Capítulo 65

5 MICRO-ARQUITETURA **66**
5.1 Micro-arquitetura do Bloco 1 66
5.2 Micro-arquitetura do Bloco 2 68
5.2.1 Método de Multiplicação 69
5.2.2 Multiplicação em um Pulso de Clock 69
5.3 Micro-arquitetura do Bloco 3 71
5.3.1 Divisão em Hardware 73
5.3.2 Raiz Quadrada 79
5.4 Micro-arquitetura do Componente Sincro 82
5.5 Considerações Finais do Capítulo 83

6 ANÁLISE DOS RESULTADOS **84**
6.1 Métricas Utilizadas 84
6.2 Escolha dos Parâmetros do PSO 85
6.3 Avaliação do Coprocessador 89
6.4 Avaliação do Desempenho 91
6.4.1 Imagem *pickup* 93
6.4.2 Imagem *truck* 95
6.4.3 Imagem *redcar* 97
6.4.4 Imagem *hollywood* 99
6.4.5 Imagem *carros* 101
6.4.6 Imagem *sedan* 103
6.4.7 Imagem *IRcar1* 105
6.4.8 Imagem *IRcar2* 107
6.4.9 Imagem *IRcar3* 109
6.5 Comparação dos Resultados 111
6.6 Limitações do Hardware 114
6.7 Considerações Finais do Capítulo 115

7 CONCLUSÕES E TRABALHOS FUTUROS **117**
7.1 Conclusões 117
7.2 Trabalhos Futuros 118

REFERÊNCIAS **120**

INTRODUÇÃO

OBTENÇÃO de informações por meio de imagens e vídeos tornou-se uma importante área de pesquisa sobretudo com o desenvolvimento e aprimoramento de sensores e equipamentos inteligentes capazes de capturar, armazenar, editar e transmitir imagens. O aprimoramento destes equipamentos possibilita a obtenção em tempo real destas informações.(NETO, 2014)

Uma das técnicas amplamente utilizadas para encontrar e rastrear padrões em imagens é chamada de *template matching* (AHUJA; TULI, 2013) (MAHALAKSHMI et al., 2012). O objetivo consiste em encontrar um padrão, a partir de um modelo pré-estabelecido, em uma sequência de frames. Para avaliar o grau de similaridade entre duas imagens, o Coeficiente de Correlação de Pearson (Pearson's Correlation Coeficient - PCC) é amplamente utilizado. Essa tarefa é demorada e muito custosa computacionalmente, sobretudo quando são considerados grandes padrões e extensos conjuntos de imagens (SHARMA; KAUR, 2013).

Na área de segurança e defesa, esse tipo de pesquisa é de grande importância e pode proporcionar soluções para o desenvolvimento de sistemas de vigilância (NARAYANA, 2007), monitoramento, controle de fogo (ALI; KAUSAR; KHAN, 2009), guiamento (CHOI; KIM, 2014), navegação (FORLENZA et al., 2012), biometria remota (BENFOLD; REID, 2011), armas guiadas (OLSON; SANFORD, 1999), entre outros.

Em sistemas de armas guiadas, o desenvolvimento de sistemas de guiamento de mísseis é uma das aplicações do rastreamento de alvos. Mísseis podem ser guiados por radar, laser, por imageamento infravermelho ou eletro-óptico. A Figura 1 ilustra os diversos sistemas de um míssil típico.

O sistema de guiamento de um míssil tem a função de identificar as correções necessárias para mantê-lo em direção ao alvo, enviando sinais apropriados ao autopiloto (BLAKELOCK, 1991). O *seeker* é o principal componente do sistema de guiamento e permite realizar, basicamente, as seguintes funções (SIOURIS, 2004):

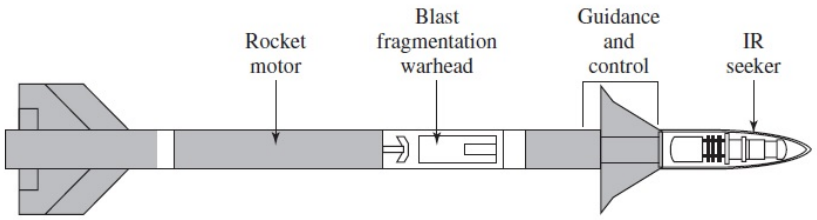


Figura 1: Míssil típico (SIOURIS, 2004)

1. fornecer medidas de movimento do alvo a serem utilizadas pela lei de guiamento;
2. manter a antena (ou sensor) sempre apontado na direção do alvo;
3. acompanhar o alvo continuamente após sua aquisição;
4. medir a taxa de variação angular da linha de visada; e
5. estabilizar a medida da variação angular da linha de visada contra grandes variações na atitude do míssil.

Quando o sistema de guiamento é baseado em imagens (eletro-óptico ou infravermelho), o *seeker* é composto basicamente dos seguintes componentes (KORETSKY; NICOLL; TAYLOR, 2013):

1. um sensor que recebe a informação do alvo e a converte em um sinal elétrico condizente;
2. um sistema eletrônico que processa a saída do sensor, produzindo um sinal de erro;
3. um sistema de apontamento que permite ao sensor acompanhar o movimento do alvo; e
4. um sistema de estabilização.

Em um míssil típico, guiado por imageamento, um operador humano identifica o alvo previamente na tela do console do armamento e, após o alvo ser adquirido, um laço de controle ajusta continuamente o sistema a fim de sempre manter o alvo no centro do campo de visão do sensor. Essa correção é proporcional a um sinal de erro, que corresponde a distância do alvo para o centro da imagem, conforme ilustrado na Figura 2. O maior desafio do processamento de imagens no *seeker* é conseguir realizar a tarefa em tempo real.

Figura 2: Erro: distância do alvo até centro da imagem (COLLINS; ZHOU; TEH, 2005)

O objetivo desta dissertação é implementar a tarefa de *template matching* em um sistema embarcado, que deverá ser capaz de localizar uma imagem pré-selecionada (alvo) em um *frame*, conferindo-lhe grande versatilidade, desempenho e possibilitando seu uso em equipamentos portáteis utilizados, por exemplo, na área de segurança e defesa e nos *seekers* de mísseis modernos. A posição do alvo no *frame* pode ser usada pelo sistema de guiamento para produzir o sinal de erro. Para obter melhor tempo de processamento, o cálculo do PCC foi implementado em coprocessador dedicado, aproveitando o paralelismo inerente do processo. Além disso, o processo de busca do maior ponto de correlação foi auxiliado por técnica de otimização baseada em enxame de partículas (PSO), executado em software pelo processador de propósito geral.

O hardware foi modelado utilizando a linguagem de descrição de hardware VHDL e foi validado na placa *Smart Vision Development Kit* (SENSORTOIMAGE, 2015) com síntese em chip da família Zynq-7000. Esse tipo de circuito integrado possui arquitetura baseada em processador dual-core ARM Cortex-A9 e lógica programável do tipo FPGA, em um único chip (XILINX, 2015).

O restante desta dissertação está organizado em seis capítulos, cujos conteúdos são brevemente descritos a seguir.

Inicialmente, o Capítulo 1 apresenta uma introdução teórica do assunto. Também, são apresentados conceitos relativos a rastreamento de objetos, *template matching* e correlação de imagens.

O Capítulo 2 apresenta, de maneira sucinta, alguns trabalhos relacionados à rastreamento de padrões em imagens e sua implementação em hardware. Ressalta-se a

importância desta dissertação, alinhada com diversos trabalhos, na busca por soluções em tempo real.

O Capítulo 3 tem por objetivo apresentar técnicas de inteligência computacional que possam ser empregadas para otimizar o rastreamento de objetos. As técnicas de otimização por enxame de partículas (PSO) e algoritmos genéticos foram comparadas, sendo que o PSO *global best* obteve melhores resultados.

No Capítulo 4, o projeto é proposto em uma plataforma portátil via um projeto integrado de software/hardware, utilizando a técnica PSO *global best*, implementada em software, e cálculo da coeficiente de correlação, implementado em hardware via um co-processador dedicado. A arquitetura é descrita de forma macro, com apresentação dos principais blocos funcionais do sistema.

O Capítulo 5 descreve o projeto detalhadamente, em sua micro-arquitetura. São apresentados os circuitos, diagramas e algoritmos associados a cada componente. Em especial, destaca-se a implementação da multiplicação em um pulso de clock, da divisão e da raiz quadrada no hardware.

No Capítulo 6 são apresentados e analisados os resultados obtidos com a implementação do projeto na placa SVDK.

Finalmente, o Capítulo 7 completa esta dissertação, apresentando as principais conclusões obtidas do desenvolvimento do presente trabalho. São apresentadas também algumas perspectivas e possibilidades para trabalhos futuros.

Capítulo 1

RASTREAMENTO DE PADRÕES

RECONHECIMENTO de padrões é um dos mais importantes e ativos ramos da inteligência artificial. É a ciência que tenta fazer máquinas tão inteligentes quanto os seres humanos em reconhecer padrões e classificá-los, dentre as categorias desejadas, de forma simples e confiável (SHARMA; KAUR, 2013). É também definido como o estudo de como as máquinas podem observar o ambiente, distinguir os vários padrões de interesse e tomar decisões racionais. Reconhecimento de padrões proporciona soluções para problemas nas mais diversas áreas tais como análise de imagens, automação industrial, visão computacional, identificação biométrica, sensoriamento remoto, reconhecimento de voz, reconhecimento de faces, vigilância e defesa.

Reconhecer padrões em imagens e acompanhar suas posições em vídeos tem sido alvo de diversas pesquisas e tem se destacado por ser uma área exigente de processamento de imagens e visão computacional (PRAJAPATI; GALIYAWALA, 2015).

Este Capítulo tem por objetivo realizar uma revisão e apresentar os conceitos básicos associados a rastreamento de padrões em imagens. Para tal, na Seção 1.1 são apresentados os conceitos de detecção, classificação e rastreamento de objetos. Na Seção 1.2, a abordagem de template matching é discutida. Na Seção 1.3, o conceito e a fórmula para cálculo da correlação cruzada normalizada são apresentados. Na Seção 1.4 são apresentadas as dificuldades e complexidades de pesquisa da área. Finalmente, na Seção 1.5, são realizadas considerações sobre o assunto voltado para aplicação nesta dissertação.

1.1 Rastreamento de Padrões/Objetos

As pesquisas atuais na área de rastreamento de padrões tem atraído muitos pesquisadores. Identificar objetos em vídeos e acompanhar seus movimentos a fim de identificar

suas propriedades tem emergido como áreas de pesquisa em visão computacional (DEORI; THOUNAOJAM, 2014).

Padrão é um arranjo, ou coleção de objetos que são similares entre si, e é caracterizado pela arrumação de seus elementos, contrariamente a sua disposição natural (GONZALEZ; WOODS, 2008). Também pode ser definido como o oposto do caos, uma entidade, vagamente definida, no qual se pode dar um nome (WATANABE, 1985). Na área de rastreamento, muitos pesquisadores chamam padrões de objetos (GONZALEZ; WOODS; EDDINS, 2004). Objetos podem ser definidos como algo que desperte interesse para futura análise. Por exemplo, barcos no mar, peixes em um aquário, veículos na estrada, aeronaves no ar ou pessoas andando na rua são conjuntos de objetos no qual o acompanhamento possa ser de interesse para algum determinado fim (YILMAZ; JAVED; SHAH, 2006).

O assunto é muito amplo e possui diversas interpretações e abordagens. Basicamente, podemos dividir a tarefa de rastreamento de objetos em imagens em três etapas: detecção do objeto, classificação do objeto e rastreamento propriamente dito.

1.1.1 Detecção do Objeto

Esta etapa do rastreamento tem por objetivo identificar o objeto de interesse na sequência de vídeo e isolar os pixels deste objeto (SINDHUJA; RENUKA, 2015)(PRAJAPATI; GALIYAWALA, 2015). Todo método de rastreamento necessita de um mecanismo que possa identificar o objeto na primeira vez que ele aparece no vídeo e também em cada frame. As abordagens mais comuns utilizadas para este fim são: segmentação, modelagem do plano de fundo, detecção de pontos e aprendizado supervisionado.

A segmentação tem por objetivo particionar a imagem em regiões similares de forma a obter o objeto de interesse. Os algoritmos de segmentação tem que equilibrar o critério para um bom particionamento e que também atinja um particionamento eficiente. Alguns exemplos de algoritmos usados para segmentação: *mean-shift* (COMANICIU; MEER, 1999), *graph-cut* (SHI; MALIK, 2000), *active contours* (CASELLES; KIMMEL; SAPIRO, 1995).

A modelagem do plano de fundo visa construir uma representação da cena e realizar a detecção do objeto a partir dos desvios observados em cada frame (YILMAZ; JAVED; SHAH, 2006). Os objetos da cena são classificados através da formação de fronteira entre o plano de fundo (*background*) e o primeiro plano (*foreground*). O primeiro plano contém todos os objetos de interesse. Alguns exemplos de algoritmos utilizados para modelagem

do plano de fundo: *background subtraction* (*frame differencing*), *mixture of gaussians* (STAUFFER; GRIMSON, 2000), *eigenbackground* (OLIVER; ROSARIO; PENTLAND, 2000) e *optical flow*.

Detectores de ponto são utilizados para encontrar pontos de interesse em imagens. Esses pontos são chamados *features* e são destacados por suas características diferenciadas em termos de, por exemplo, cor, textura, geometria ou variação de gradiente de intensidade. A detecção do objeto é realizada pela comparação dos pontos. Uma característica interessante desta abordagem é sua invariância a mudanças na iluminação e no ponto de vista da câmera (YILMAZ; JAVED; SHAH, 2006). Alguns exemplos de algoritmos baseados em *features*: *Scale Invariant Feature Transform* (SIFT) (LOWE, 2004), *affine invariant point detector* (MIKOLAJCZYK; SCHMID, 2002) e *Speeded-Up Robust Features* (SURF) (BAY; TUYTELAARS; GOOL, 2006).

No aprendizado supervisionado, a detecção é realizada pelo aprendizado dos diferentes pontos de vista do objeto, a partir de conjunto de exemplos e um mecanismo de aprendizado supervisionado. Este método normalmente requer uma grande coleção de amostras de cada classe de objetos. Além disso, as amostras devem ser manualmente rotuladas, tarefa demorada e tediosa (YILMAZ; JAVED; SHAH, 2006). A seleção das características dos objetos de forma a diferenciar as classes também é uma tarefa de extrema importância para a eficácia do método. Após a aprendizagem, as classes são separadas por hiper superfícies no hiperespaço de características. Alguns exemplos de uso desta abordagem: redes neurais (ROWLEY; BALUJA; KANADE, 1998), *adaptive boosting* (VIOLA; JONES; SNOW, 2003), árvore de decisão (GREWE; KAK, 1995) e *support vector machines* (PAPAGEORGIOU; OREN; POGGIO, 1998).

A detecção e o rastreamento de objetos são processos muito próximos e relacionados porque o rastreamento começa, normalmente, com a detecção de objetos enquanto que a repetida detecção do objeto, em *frames* subsequentes, é necessária para ajudar e realizar o rastreamento (PRAJAPATI; GALIYAWALA, 2015).

1.1.2 Classificação do Objeto

A fim de rastrear objetos e analisar seus comportamentos, é essencial classificá-los corretamente. A classificação está diretamente ligada às características de cada objeto e como eles são representados. As abordagens para classificação são frequentemente relacionadas

a forma, movimento, cor e textura dos objetos. Na representação por forma, os objetos são representados e classificados por suas características de forma tais como pontos, centróides, formas geométricas, contornos e *blobs*. Os dados são processados em cada frame e armazenados, normalmente, em histogramas. Exemplos de populares detectores de bordas são *canny edge detector* (CANNY, 1986) e sua evolução (BOWYER; KRANENBURG; DOUGHERTY, 2001).

A representação por movimento utilizada para classificação de objetos, normalmente articulados e não rígidos, que possuem algum tipo de movimento periódico. O método *optical flow*, por exemplo, é utilizado para diferenciar objetos rígidos de não rígidos. Baseado nisso, movimentos tipicamente humanos podem ser diferenciados de movimentos simples de veículos, pedestres e animais (SINDHUJA; RENUKA, 2015).

Na representação por cores, os objetos são classificados baseados em suas diferenças de cores. Embora não seja sempre utilizada isoladamente para classificar objetos, ela é uma característica desejável a ser explorada pois apresenta pouca variação em termos de rotação, translação e escala e requer baixo custo de processamento (PRAJAPATI; GALIYAWALA, 2015). Histogramas para descrever distribuição de cores, em uma região, também são bastante utilizados nesta representação.

Na representação por textura, os objetos são classificados baseados na variação de sua superfície, em termos de intensidade, que descreve características de textura, suavidade e regularidade do objeto. Similarmente com representações de forma, características de textura são menos sensíveis a variações de iluminação se comparadas com representação por cores (YILMAZ; JAVED; SHAH, 2006). Exemplos de descritores de textura: *gray-level cooccurrence matrices* (HARALICK; SHANMUGAM; DINSTEIN, 1973), *Law's texture measures* (LAWS, 1980), *wavelets* (MALLAT, 1989) e *steerable pyramids* (GREENSPAN et al., 1994).

1.1.3 Técnicas para Rastreamento

Rastreamento pode ser definido como um problema de aproximar a trajetória de um objeto em uma determinada cena. O propósito principal é encontrar a trajetória desse objeto por achar sua posição em cada *frame* do vídeo (SINDHUJA; RENUKA, 2015). Basicamente, pode ser dividido nas seguintes categorias: rastreamento baseado em pontos, rastreamento baseado em núcleos (*kernel*) e rastreamento baseado em silhuetas. A Figura 3 ilustra as

três categorias para rastreamento da câmera na famosa imagem ”cameraman”. As tarefas de detectar o objeto e estabelecer a correspondência com aqueles dos *frames* anteriores e subsequentes podem ser executadas juntas ou separadas (YILMAZ; JAVED; SHAH, 2006).

Figura 3: Exemplo de rastreamento para imagem ”cameraman”(a) por pontos, (b) por *kernel*, (c) por silhueta

1.1.3.1 Rastreamento por Pontos

Os objetos detectados são representados por pontos e a posição dos pontos na sequência de frames permite o rastreamento. Esta abordagem requer um mecanismo para detectar os objetos em cada *frame*.

O filtro de Kalman é um algoritmo recursivo que proporciona um meio computacional eficiente para estimar o estado de um sistema. Em aplicações de rastreamento, é um filtro utilizado para estimar a posição de objetos, baseado na dinâmica do movimento ao longo do vídeo, que apresentam uma trajetória suave e bem comportada, descrita por um modelo de espaço de estado linear com ruído gaussiano. Essa informação proporciona o acompanhamento do alvo, mesmo em situações de oclusão, e resultados mais robustos. Possui aplicação nas áreas de controle, navegação, visão computacional e economia.

Uma limitação do filtro de Kalman é a hipótese de que as variáveis são normalmente distribuídas. Assim, quando as variáveis de estado não seguirem uma distribuição gaussiana, a estimativa não produzirá bons resultados (YILMAZ; JAVED; SHAH, 2006). Esta limitação pode ser superada com o Filtro de Partículas que utiliza um modelo de espaço de estados mais flexível. As funções utilizadas nas equações de estado e de observação são em geral não-lineares. A probabilidade a *posteriori* é representada por uma distribuição

discreta de probabilidade definida através de um conjunto amostral (partículas) e seus respectivos pesos.

O Multiple Hypothesis Tracking (MHT) é um método geralmente utilizado para resolver problemas de rastreamento de múltiplos alvos. É um algoritmo iterativo baseado em hipóteses para as trajetórias dos objetos. Cada hipótese é um conjunto de trajetórias disconexas. Para cada hipótese, obtém-se a estimativa do alvo no próximo *frame*. Essa estimativa é então comparada com a medida atual por meio de uma medição de distância. Este algoritmo pode lidar com oclusões e tem a capacidade de criar novas trajetórias, para objetos que entrem na cena, e finalizar aquelas relacionadas a objetos que desapareçam da cena.

1.1.3.2 Rastreamento por Núcleo

Kernel é um termo em inglês que se refere a uma região notável do objeto relacionada com sua forma e aparência. Um *kernel* pode ser uma área retangular ou uma forma elíptica, por exemplo. Os objetos são rastreados pela localização de seus movimentos, da região embrionária representada pelo *kernel*, de um *frame* para o próximo. Esses movimentos são usualmente representados por transformações paramétricas como translação, rotação e *affine*. Uma das dificuldades desta abordagem é a possibilidade do *kernel* não abranger todo o objeto de interesse ou possuir conteúdo do plano de fundo.

O *template matching*, ou casamento por modelo, é um método de força bruta que busca regiões da imagem que sejam similares a uma imagem de referência que representa o objeto, chamada de modelo (*template*). A posição do *template* na imagem é computada a partir de medidas de similaridade tais como soma das diferenças absolutas (SAD), soma do quadrado das diferenças (SSD), correlação cruzada, correlação cruzada normalizada, dentre outras. Este método é capaz de lidar com rastreamento de imagens isoladas e oclusão parcial. Uma limitação do *template matching* é o alto custo computacional associado a força bruta. Muitos pesquisadores, a fim de reduzir este custo, limitam a área de busca à vizinhança do objeto no *frame* anterior (YILMAZ; JAVED; SHAH, 2006)

Mean Shift é uma abordagem eficiente para rastreamento de objetos cuja aparência é definida por histogramas. O algoritmo é iterativo e busca encontrar no *frame* o local que mais se assemelha com o histograma da imagem de referência. A similaridade é avaliada por meio do coeficiente de Bhattacharya e o procedimento de gradiente ascendente é utilizado para mover o rastreador para o local que maximiza a similaridade.

Support Vector Machine é um método de rastreamento baseado em aprendizado supervisionado. Durante o treinamento, são apresentados conjuntos com exemplos positivos (imagens do objeto a ser rastreado) e exemplos negativos (regiões do plano de fundo que podem confundir o sistema). A abordagem consegue lidar com oclusões parciais mas possui necessidade de treinamento.

Layering based tracking é um método baseado na modelagem da imagem como um conjunto de camadas. Uma camada é associada ao plano de fundo e as outras associadas a cada objeto. A probabilidade de cada pixel pertencer a uma camada (objeto) é considera características de forma e movimentos anteriores do objeto. Este método é geralmente utilizado para rastrear múltiplos objetos.

1.1.3.3 Rastreamento por Silhueta

Os objetos podem ter formas complexas que, muitas das vezes, podem não ser bem descritas com formas geométricas simples (YILMAZ; JAVED; SHAH, 2006). Métodos de rastreamento por silhueta tem por objetivo identificar as formas precisas dos objetos em cada *frame*. Ver Figura 3(c). Esta abordagem pode ser dividida em duas categorias, dependendo da forma como objeto é rastreado: por contornos ou por formas.

Métodos de rastreamento por contornos (*contour matching*) evoluem o contorno inicial do objeto para sua nova posição. É necessário que parte do objeto no *frame* anterior se sobreponha sobre o objeto no *frame* seguinte. Um exemplo de algoritmo é aquele proposto em (KASS; WITKIN; TERZOPOULOS, 1988), chamado *snakes*, baseado na deformação do contorno inicial em pontos determinados. A deformação é direcionada para as bordas do objeto pela minimização da energia do *snake*, empurrando-a para linhas e bordas.

Métodos de rastreamento por forma da silhueta (shape matching) são métodos muito parecidos com *template matching*. A principal diferença é que o modelo representa a forma exata do objeto. (HUTTENLOCHER; NOH; RUCKLIDGE, 1993) é um exemplo deste tipo de método que utiliza a distância Hausdorff para encontrar a localização do objeto.

1.2 *Template Matching*

O *template matching*, ou casamento por modelos, é amplamente utilizado em processamento de imagens para determinar a similaridade entre duas entidades (pixels, curvas

ou formas) de mesmo tipo. O padrão a ser reconhecido é comparado com um modelo previamente armazenado, levando-se em conta todas as posições possíveis.

Na área de rastreamento, o *template matching* é um importante método de rastreamento de objetos em imagens a partir de núcleos (*kernel*). A tarefa basicamente se resume em achar ocorrências de uma imagem pequena, considerada como *template*, em uma sequência de imagens maiores (*frames*). A Figura 4 mostra duas matrizes que representam duas imagens em preto e branco. A imagem da direita seria o *template* a ser encontrado na imagem da esquerda. Em representações do tipo byte inteiro para imagens em preto e branco, quanto maior o valor de um pixel, mais próximo do branco ele estará, e quanto menor o valor do pixel, mais próximo do preto ele estará.

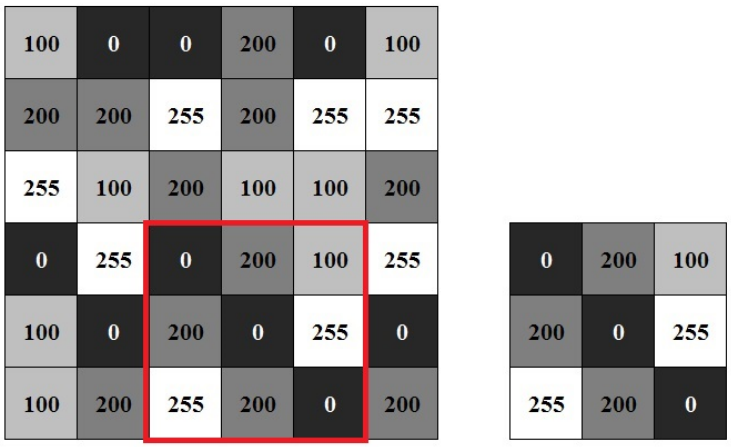


Figura 4: Matrizes que representam uma imagem em preto e branco (esquerda) e um *template* (direita)

A busca no *frame* ocorre comparando o *template*, em cada pixel, com pedaços de imagem de mesmo tamanho que ele. A Figura 5 ilustra esse processo para a Figura 4. O *template* desliza pixel a pixel na imagem principal até que todas as posições sejam atingidas. Em cada posição, é utiliza uma medida de similaridade para comparar as imagens. Após o cálculo da similaridade para todas as possibilidades, aquela que apresentar o maior valor, acima de um limiar pré-estabelecido, é considerada como sendo o local do objeto buscado dentro da imagem (NIXON; AGUADO, 2002). Esta operação é muito custosa quando são considerados grandes modelos e extensos conjuntos de imagens (SHARMA; KAUR, 2013). A vantagem do *template matching* é que o *template* armazena em si diversas características particulares do objeto (cor, textura, forma, bordas, centróide, etc), que o diferenciam dos outros, permitindo maior exatidão e rastreamento de um objeto

específico dentro de um grupo. Além disso, a detecção do objeto não fica comprometida com a escolha da forma como classificá-lo ou representá-lo. A desvantagem é o alto custo computacional necessário para o cálculo em todas as posições.

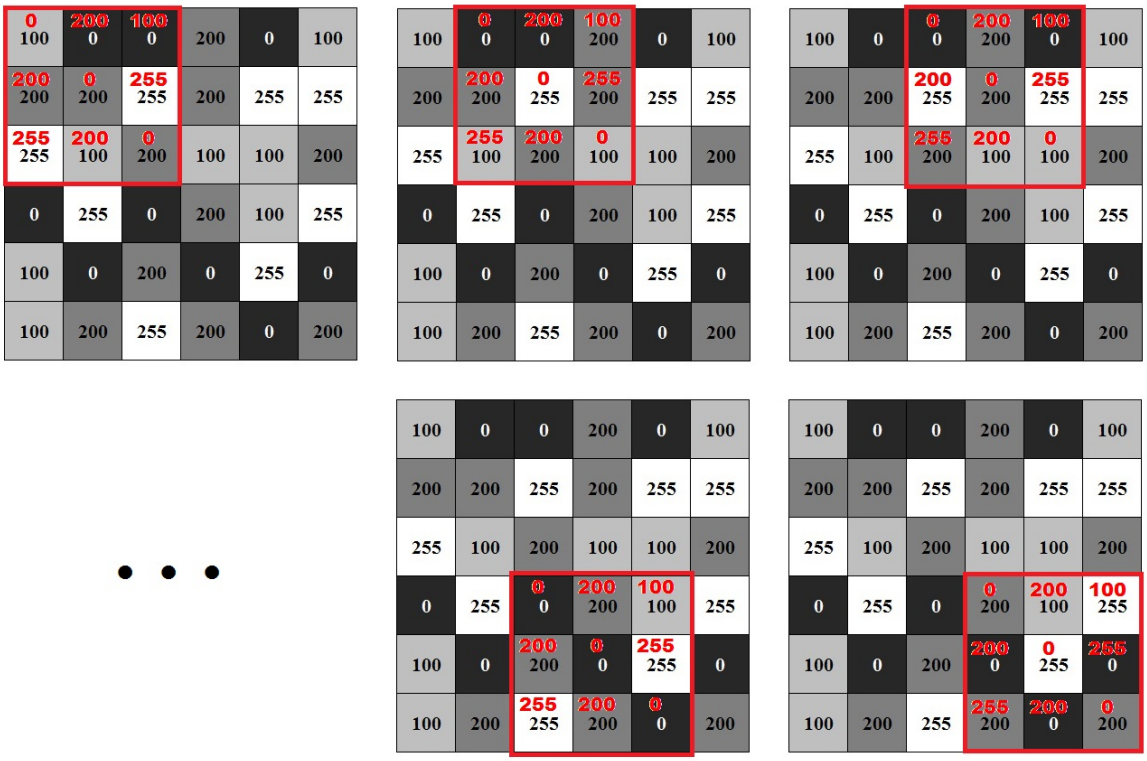


Figura 5: Ilustração do processo de *template matching*

Para avaliar o grau de similaridade do template ao longo do frame, algumas técnicas são utilizadas, dentre elas: soma das diferenças absolutas (*Sum of Absolute Differences* - SAD), soma dos quadrados das diferenças (*Sum of Squared Differences* - SSD) e correlação cruzada.

SAD é uma das mais simples medidas de similaridade entre duas imagens. É baseada na subtração da intensidade absoluta de cada um dos pixels. As diferenças para todos os pixels são somadas criando uma métrica para comparação. Quanto mais perto de zero, mais próximas são as imagens. A Equação 1 apresenta sua fórmula para cálculo:

$$SAD = \sum_{i=1}^{N} |(p_i - a_i)|, \tag{1}$$

onde p_i é a intensidade do pixel i na imagem padrão (*template*) e a_i é a intensidade do pixel i no pedaço de imagem original A.

Em SSD, a subtração da intensidade absoluta de cada um dos pixels é elevada ao quadrado antes da somá-las. Da mesma forma que SAD, quanto mais perto de zero, mais

próximas são as imagens. A Equação 2 apresenta a fórmula para cálculo da soma dos quadrados das diferenças:

$$SSD = \sum_{i=1}^{N} (p_i - a_i)^2. \tag{2}$$

A correlação cruzada é outra medida de similaridade para imagens. Ao invés de serem subtraídos, os pixels são multiplicados, conforme Equação 3:

$$corr = \sum_{i=1}^{N} (p_i)(a_i). \tag{3}$$

Esta técnica é sensível a mudanças de amplitude das imagens (GONZALEZ; WOODS, 2000). Para superar tal desvantagem, utiliza-se a correlação cruzada normalizada, descrita em detalhes na Seção 1.3.

1.3 Correlação Cruzada Normalizada

O termo correlação é amplamente utilizada no senso comum significando algum tipo de relação entre duas coisas ou fatos. Na área de processamento de sinais, a correlação cruzada é obtida pela convolução de um sinal pelo conjugado do outro. Nesta dissertação, o termo correlação tem um sentido mais restrito e se refere a medida de similaridade associada a correlação cruzada normalizada entre duas imagens. Esta técnica é uma versão aperfeiçoada da correlação cruzada. Ela apresenta um termo normalizador no denominador que lhe confere invariância a mudanças globais de brilho e resultados sempre dentro de um intervalo definido $[-1, 1]$.

A expressão para o cálculo encontra-se na Equação 4, também conhecida como Coeficiente de Correlação de Pearson (PCC) para imagens (MIRANDA, 2014):

$$PCC = \frac{\sum_{i=1}^{N} (p_i - \overline{p})(a_i - \overline{a})}{\sqrt{\sum_{i=1}^{N} (p_i - \overline{p})^2} \sqrt{\sum_{i=1}^{N} (a_i - \overline{a})^2}}, \tag{4}$$

onde p_i é a intensidade do pixel i na imagem padrão (*template*); $\overline{p}$ é a média das intensidades dos pixels da imagem padrão; a_i é a intensidade do pixel i no pedaço da imagem A; $\overline{a}$ é a média da intensidade dos pixels no pedaço de imagem A. O *template* e o pedaço da imagem A devem ter o mesmo tamanho.

O coeficiente pode ser entendido como um índice adimensional com valores situados entre -1 e 1 inclusive, que reflete a intensidade do grau de relacionamento entre duas

variáveis. Coeficiente igual a 1 significa uma correlação perfeita positiva entre as duas variáveis. Coeficiente igual a -1 significa uma correlação negativa perfeita entre as duas variáveis. Se uma variável aumenta, a outra sempre diminui. Coeficiente igual a zero significa que as duas variáveis não dependem linearmente uma da outra.

A utilização ideal da correlação cruzada normalizada, dada pela Equação 4, considera que a aparência do alvo permanece a mesma durante todo o vídeo (MATTHEWS; ISHIKAWA; BAKER, 2004). Qualquer alteração de escala ou rotação do alvo podem influenciar a resposta do sistema. A própria mudança de iluminação, contraste ou ruído (*clutter*) que sejam inseridos no ambiente podem ocasionar erros. Uma possível solução para esse problema é atualizar o *template* a cada *frame* (correlação adaptativa).

1.4 Dificuldades em Rastreamento de Padrões

Após a apresentação das diversas técnicas nas seções anteriores, é importante registrar as dificuldades e complexidades enfrentadas ao se tentar rastrear objetos em imagens.

As formas complexas e as mudanças rápidas de aparência e movimentos complexos dos objetos, conferem dificuldades no rastreamento, sobretudo para objetos não-rígidos. A rotação e mudanças de escala do objeto também são fatores de grande importância.

A dinâmica da cena também influencia no rastreamento. O ruído nas imagens, a oclusão completa ou parcial do objeto e as mudanças de iluminação e contraste também podem complicar o processo. Outro fator importante é a interação entre múltiplos objetos na cena, que podem se sobrepor e atrapalhar a identificação de alguns deles.

Uma dificuldade associada com rastreamento por *template matching* é o *drift*. Quando ocorre a atualização automática, muitas das vezes o *template* atualizado não corresponde exatamente ao objeto e pequenos erros vão sendo acumulados, podendo ocasionar desvio no rastreamento do objeto original. Este problema é geralmente identificado como *drift*.

Em virtude do exposto, é de vital importância a seleção do algoritmo correto, dependendo dos requisitos da aplicação e das características do objeto. O processamento em tempo real ainda é um grande desafio no rastreamento de objetos por imagens.

1.5 Considerações Finais do Capítulo

Neste Capítulo foram apresentados os conceitos básicos associados a detecção, classificação e rastreamento de objetos em imagens. Diversas técnicas e as maiores dificuldades sobre o assunto foram apresentadas. Foi dada ênfase ao método de rastreamento por *template matching* e correlação cruzada normalizada, técnicas escolhidas para uso nesta dissertação.

O *template matching* apresenta fácil implementação em um sistema de arma. Os objetos identificados como alvos não necessitam ser previamente modelados, permitindo que diversos tipos de alvos sejam acompanhados (carros, barcos, aeronaves, construções). Basta somente que o alvo seja identificado no *frame* inicial e suas características ficarão armazenadas no *template*. O processo de detecção do objeto é realizado ao mesmo tempo que seu rastreamento. Também não necessita de treinamento.

Um outro fator importante é a fácil implementação, em hardware, da correlação cruzada normalizada, que não apresenta funções matemáticas complexas nem transformadas. A utilização de técnicas de sistemas inteligentes podem ajudar a superar limitações de tempo de processamento associadas a força bruta do *template matching*. O maior desafio é ter que lidar com oclusão, mudanças de escala, mudanças de comportamento do objeto e com *drift*.

Capítulo 2

TRABALHOS RELACIONADOS

ESTE Capítulo, organizado em três seções, apresenta alguns trabalhos relacionados à rastreamento de objetos em imagens e sua implementação em hardware. Como o assunto é muito vasto, foi concebida uma noção geral e chamada atenção para o estado da arte do tema. Na Seção 2.1 são apresentadas algumas técnicas de *template matching* com trabalhos relevantes na área. Na Seção 2.2 são apresentadas implementações de *template matching* em hardware na tentativa de aumentar o desempenho do sistema. Na Seção 2.3 são expostos trabalhos voltados para eliminação do problema do *drift* presente em sistemas de rastreamento por *template matching*. Por último, como considerações finais, realiza-se a comparação do presente trabalho com aqueles apresentados e ressalta-se a relevância da pesquisa.

2.1 *Template Matching*

Nesta Seção são apresentados trabalhos voltados para rastreamento de objetos em imagens, com foco na utilização da metodologia de *template matching*. Esta abordagem tem sido amplamente utilizada para localização e reconhecimento de objetos em imagens. A tarefa basicamente se resume em achar ocorrências de uma imagem pequena, considerada como modelo (*template*), dentro de uma imagem maior. As técnicas de *template matching* podem ser divididas em dois tipos de grupos: baseadas em área e baseadas em *features*. Dentre elas, podem ser citadas: correlação cruzada normalizada (calculada pelo coeficiente de correlação de Pearson), soma das diferenças absolutas (*Sum of Absolute Differences - SAD*), soma das diferenças quadráticas (*Sum of Square Differences - SSD*), *Grayscale Based Matching*, *Edge Based Matching*, *Scale-invariant Feature Transform - SIFT* e *Speeded Up Robust Features - SURF*, dentre outras (AHUJA; TULI, 2013) (PERVEEN; KUMAR;

BHARDWAJ, 2013). Este último artigo ainda afirma que a correlação cruzada normalizada é invariante a variações lineares de brilho e contraste e que sua simples implementação em hardware a faz útil para aplicações em tempo real.

Em (AHUJA; TULI, 2013) foi realizada uma comparação entre duas técnicas de *template matching*: correlação cruzada normalizada e o correlação de fase. Primeiramente, foram comparados os tempos de processamento dos dois algoritmos para cinco imagens diferentes. Posteriormente, para avaliar o comportamento dos algoritmos, as imagens foram submetidas a rotação e novamente o tempo de processamento foi avaliado. Os autores concluíram que a correlação de fase é mais rápida até determinado tamanho de imagem, a partir da qual, a correlação leva vantagem. Também foi exposto que a correlação é mais veloz para imagens com rotação.

Em (NETO, 2014), o coeficiente de correlação de Pearson foi utilizado para encontrar o grau de similaridade de frames em aplicações de robótica autônoma. Pequenas distorções ou ruído, leves alterações na posição dos objetos e outros fatores podem produzir alterações neste coeficiente. Considerando ainda que frames próximos possuem regiões homogêneas, o trabalho propõe fusão de regiões das imagens para melhorar resultados e trazer os valores da correlação mais próximos da realidade. O método se resume em identificar o plano de fundo do frame atual e copiá-lo no frame de referência. Para identificação do plano de fundo foram utilizados o *Otsu Thresholding Method* e o *Canny Edge Detector*. Para o caso estudado no artigo, o valor da correlação aumentou de 0,790 para 0,800.

O algoritmo *mean-shift* é um procedimento iterativo não paramétrico de estimação do gradiente de densidade para busca local utilizado, inicialmente, para clusterização. Comaniciu (COMANICIU; RAMESH; MEER, 2003) foi o primeiro a aplicá-lo em rastreamento de alvos. O modelo do alvo é obtido do histograma de cores do objeto em movimento. A medida de similaridade, entre o possível alvo e o modelo do alvo, é computada usando o coeficiente de Bhattacharya. Um dos problemas desse método é que ele, normalmente, converge muito lentamente.

O trabalho (SUN, 2012) propõe um rápido algoritmo para rastreamento de alvos baseado em *mean-shift*. Segundo o autor, esta tarefa se resume a um problema de maximização de bordas (*bound maximization*). Um dos passos é encontrar o ponto máximo do limite inferior da função de custo. No algoritmo *mean-shift* tradicional, são mantidos pa-

râmetros de otimização conservadores para garantir a convergência. No algoritmo rápido proposto, foi utilizada uma otimização adaptativa, onde os parâmetros foram ajustados avaliando-se a função de custo.

A obstrução do alvo, chamado de oclusão, é um desafio no campo de rastreamento por imagem. No caso do artigo (SUN, 2012), o autor propõe lidar com esse problema por meio de um método chamado de *template matching* local. O coeficiente de Bhattacharyya é utilizado para determinar se o alvo encontra-se obstruído. Caso isso aconteça, a localização final é definida pelo *template matching* local, dentro da região de interesse definida pelo algoritmo *mean-shift*. Para melhorar o tempo de processamento, o trabalho também conta com um algoritmo para predição da posição do alvo chamado de predição linear (*linear prediction algorithm*).

O sistema foi implementado em uma plataforma de processamento de imagens TDS642EVM, da TY company, com DSP TMXDM642 e FPGA embutidos. Foram realizados quatro experimentos para avaliar o sistema proposto: o primeiro utilizando o algoritmo *mean-shift* tradicional, o segundo utilizando o algoritmo proposto e rastreamento com variação de posição, o terceiro utilizando o algoritmo proposto na presença de oclusão parcial e o último utilizando o algoritmo proposto em cena complexa com variação de posição. Os resultados demonstraram que o algoritmo rápido possui maior robustez que o algoritmo *mean-shift* tradicional com relação a oclusão e variações de posição do alvo. Ele também levou vantagem ao conseguir realizar o processamento três vezes mais rápido.

2.2 Implementação em Hardware

Esta Seção apresenta alguns trabalhos em que soluções de hardware são propostas a fim de aumentar o desempenho de sistemas de vídeo e rastreamento de objetos.

Em (NGO et al., 2013), é utilizada a metodologia *co-design* para implementar, em uma plataforma programável do tipo FPGA, um sistema de vigilância de vídeo automático. Como toda aplicação de processamento de imagens, o sistema deve possuir alto grau de desempenho para suportar aplicações em tempo real.

Co-design é uma metodologia para o desenvolvimento integrado de sistemas implementados utilizando componentes de hardware e software, dado um conjunto de requisitos de desempenho e uma tecnologia de implementação. Uma arquitetura desse tipo

consiste, geralmente, de componentes de software, executados em processador reprogramável, assistidos por componentes de hardware desenvolvidos especificamente para a aplicação (NEDJAH; MOURELLE, 2007). No caso do artigo, as operações computacionalmente mais intensivas foram implementadas em FPGA, maximizando o processamento paralelo e aproveitando a arquitetura *pipeline*. O controle das aplicações e interface com usuário foram desenvolvidos em software e executados por um soft-processador Altera Nios-II. Para detecção dos objetos foram utilizados filtros, morfologia matemática e segmentação das imagens. Para a aplicação de vigilância e acompanhamento dos objetos foi utilizada uma técnica chamada *background subtraction*. Nesta técnica, os objetos são identificados pelos desvios, a cada frame, do plano de fundo. Qualquer mudança nos pixels de uma região é traduzido como sendo um objeto em movimento. Apesar de eficiente, esta abordagem requer que o plano de fundo seja estático, não sendo aplicável para situações em que tanto o alvo quanto o resto da imagem sofrem modificações ao longo dos frames.

Para avaliar o sistema proposto, o autor realizou, inicialmente, uma comparação do desempenho das tarefas computacionalmente mais intensivas (dilatação, erosão e filtro, por exemplo) executadas somente pelo processador reprogramável e com auxílio dos componentes personalizados, chamados de aceleradores de hardware. A abordagem com os aceleradores diminuiu significativamente o tempo de processamento. Depois, foram realizadas comparações do sistema proposto com o algoritmo implementado em computadores pessoais (MATLAB e OpenCV) e com outras implementações no estado da arte. A arquitetura proposta superou todas as outras implementações com ampla margem, demonstrando ser factível sua aplicação em tempo real, o ganho no tempo de execução com os aceleradores de hardware e a flexibilidade em programar as tarefas rotineiras no processador reprogramável.

Em (MAHALAKSHMI et al., 2012), várias técnicas de *template matching* são discutidas e uma arquitetura baseada em correlação espectral, implementada em FPGA, é proposta. A correlação é calculada utilizando a transformada rápida de Fourier (*Fast Fourier Transform - FFT*) em duas dimensões e multiplicação complexa. Foram utilizados blocos de hardware para o cálculo da FFT, e sua inversa, e blocos para a multiplicação complexa simples. O autor conclui que o *template matching* utilizando correlação cruzada é melhor que outras técnicas baseadas em características (*feature based*), sobretudo quando não existem características fortes o suficiente nem detalhes aparentes (cor/forma/tamanho).

Outrossim, métodos baseados em correlação são mais utilizados para desenvolvimento de hardware por sua simplicidade de implementação.

Em (VINOD; ANITTA, 2013) uma técnica, implementada em FPGA, é proposta para geração de imagem panorâmica a partir da combinação de imagens. O conjunto é formado de duas ou mais imagens digitais tiradas de uma cena por diferentes sensores ou diferentes pontos de vista. O método foi dividido em três fases. A primeira, chamada de registro, realiza o casamento das imagens utilizando a correlação de fase. A informação no domínio da frequência é utilizada para encontrar os parâmetros de rotação e translação entre as imagens. A segunda fase é a integração da imagem. Nesta fase as imagens são sobrepostas baseadas nas informações da fase anterior. Para se chegar a montagem final foi utilizado o método de mapeamento de pixels. A terceira e última fase é a harmonização das imagens. Nesta etapa, as variações de luminosidade e marcas de 'costura' são eliminadas com o método da combinação média ponderada (*weighted average blending*). Todas as fases foram implementadas em FPGA Spartan 6, da Xilinx, com eficiente utilização dos recursos de hardware e resultados dentro do esperado. Não foram apresentados resultados em termos de tempo de processamento.

2.3 Correção do *Drift*

Esta Seção apresenta alguns trabalhos voltados para a correção do efeito do *drift*, quando o modelo (*template*) é atualizado. A metodologia de *template matching* para rastreamento de padrões pressupõe que o objeto mantém a mesma aparência, o que razoável para pequenos períodos de tempo. Uma possível e simples solução é atualizar o modelo a cada *frame*. Essa solução acaba causando um outro problema: o *template* acaba desviando do alvo original (*drift*). A cada atualização, pequenos erros são introduzidos na localização do alvo.

O artigo (MATTHEWS; ISHIKAWA; BAKER, 2004) propõe um algoritmo para atualizar o modelo padrão sem apresentar o problema de desvio. Para rastreamento do objeto, o método utiliza o algoritmo proposto por Lucas-Kanade (LUCAS; KANADE et al., 1981), também conhecido como *Optical Flow*. Este método, *Optical Flow*, apresenta dificuldades para implementação em tempo real e é sensível a mudanças de iluminação e ruído (SUN, 2012). A estratégia do autor é manter o *template* do primeiro frame para corrigir os demais. O *template* é utilizado para estimar a posição do alvo e depois é alinhado

com primeiro *template* para eliminar o *drift*. Para avaliação do algoritmo, o autor realiza comparações qualitativas e quantitativas. Os resultados demonstraram que o método proposto aumenta a robustez do rastreamento. Por outro lado, a constante atualização do aumenta o custo computacional tornando o processo mais lento. Além disso, o algoritmo não prevê mudanças na estrutura, como aparecimento de partes novas do objeto.

Já o trabalho (SCHREIBER, 2007), apresenta uma solução para o efeito do *drift*, prevendo mudanças no objeto e oclusão. Um algoritmo robusto para *template matching* é proposto utilizando processo, baseado em mínimos quadrados ponderados, em que os pesos são atualizados adaptativamente. O processo é utilizado de tal forma que pixels do plano de fundo, regiões com oclusão e que tenham havido mudanças de brilho são considerados pontos fora da curva e são suprimidos.

O autor realizou comparações qualitativas e quantitativas entre o método anterior de correção de drift (MATTHEWS; ISHIKAWA; BAKER, 2004) e o método proposto. Os resultados demonstraram que o algoritmo anterior, sob condições de oclusão parcial e dificuldades de iluminação, apresenta falha ao tentar alinhar o objeto com o primeiro *template* e, consequentemente, sofre com *drift*. Apesar de mais robusta, essa versão é mais lenta que a versão anterior.

Em (FAN et al., 2011) um algoritmo robusto com correção de *drift* ativo e ponderado é proposto. Para minimizar o *drift*, uma função de energia total é minimizada. Esta função possui um termo para o rastreamento e outro para a correção. A soma do erro quadrático dos pixels do *template* é utilizado como medida de similaridade para comparação com os frames. O *template* é atualizado com partes do *template* anterior, com menor erro de similaridade, e com o primeiro *template*. O rastreamento, também baseado em *Optical Flow*, apresentou tempo de processamento, número de iterações e erro médio quadrático melhores que o algoritmo com correção de drift passivo (MATTHEWS; ISHIKAWA; BAKER, 2004).

2.4 Considerações Finais do Capítulo

Neste Capítulo foram apresentados, resumidamente, diversos trabalhos sobre rastreamento de objetos em imagens e sua implementação em hardware. Ressalta-se a importância desta dissertação, alinhada com diversos trabalhos, na busca por soluções em tempo real. Apesar

de ser um tema relevante, a busca por solução para problemas de *drift*, oclusão, mudanças de posição do objeto, iluminação e ruído não fazem parte do escopo desta dissertação.

Todos os métodos avaliados apresentam vantagens e desvantagens. *Meanshift* e *Optical Flow* são métodos iterativos que precisam convergir e que, normalmente, são mais lentos. Este último é sensível a mudanças de iluminação e ruído. Métodos baseados em *features* são extremamente versáteis e robustos mas exigem que as imagens possuam características e detalhes marcantes, não sendo recomendados para aplicações com pequenas imagens, baixa resolução e presença de ruído. Apesar de simples e eficiente, *background subtraction* necessita que o plano de fundo seja estático, sendo recomendado para aplicações de vigilância, por exemplo. A simplicidade de implementação da correlação cruzada normalizada e sua invariância a brilho e contraste tornam atrativa sua implementação em hardware, apesar de ser sensível a mudanças na posição do objeto. A abordagem de *co-design* é perfeita para implementação desta dissertação aliando a velocidade e paralelismo do hardware com a flexibilidade em colocar tarefas rotineiras e de interface com usuário em software.

Capítulo 3

ESCOLHA DA TÉCNICA

O *TEMPLATE matching* foi escolhido nesta dissertação, conforme detalhado no Capítulo 1, como método de rastreamento de objetos em imagens. É um método de busca por força bruta em que o *template* desliza pixel a pixel na imagem até que seja comparado em todas as posições. Também foi definido que, em cada posição, será utilizada correlação cruzada normalizada como medida de similaridade. Esta operação é muito custosa quando são considerados grandes modelos e extensos conjuntos de imagens (SHARMA; KAUR, 2013).(AVNET, 2015)

Este Capítulo tem por objetivo apresentar técnicas de inteligência computacional que possam ser empregadas para otimizar o rastreamento do objeto, evitando a busca exaustiva, demorada e custosa computacionalmente. Ademais, é realizada uma comparação de desempenhos de duas técnicas na resolução do problema de interesse. Para tal, na Seção 3.1 o problema de rastreamento é comparado com o problema de otimização. Na Seção 3.2, são apresentadas técnicas de inteligência computacional, com ênfase naquelas utilizadas nesta dissertação. Nesta Seção também são apresentados critérios de parada e medidores de desempenho amplamente utilizados. Na Seção 3.3, o desempenho dos algoritmos PSO global best, PSO local best e GA são comparados. Por fim, na Seção 3.4, são realizadas considerações finais, com a apresentação do algoritmo mais indicado.

3.1 Problema de Otimização

Problemas de otimização são aqueles que recaem na busca por valores ótimos em funções objetivo, isto é, o maior ou o menor valor da função em um dado intervalo. Geralmente possuem os seguintes elementos (ENGELBRECHT, 2006):

- uma função objetivo, que representa a quantidade a ser otimizada, que deve ser maximizada ou minimizada;
- um conjunto de variáveis, que afetam o valor da função objetivo; e
- um conjunto de restrições, que limitam os valores que podem ser atribuídos às variáveis.

A Figura 6 mostra um exemplo de imagem com a correlação calculada em cada pixel. O *template* está destacado pelo quadrado verde. Observa-se que o gráfico da correlação possui um pico máximo bem visível, no local aonde se encontra o centro do objeto. Esta figura nos fornece a percepção que o rastreamento do objeto na imagem se resume em um problema de otimização, onde o máximo global deve ser encontrado. A função objetivo seria o próprio valor do coeficiente de correlação, as variáveis seriam as linhas e colunas e os limites seriam as dimensões da imagem (espaço de busca). Algoritmos de otimização são métodos de busca onde o objetivo é encontrar uma solução ótima, por meio de atribuição de valores às variáveis, para um dado problema de otimização. Para tal, o algoritmo examina o espaço de soluções na busca de candidatas, que atendam às restrições do problema (ENGELBRECHT 2006). Diversas técnicas associadas a inteligência computacional atendem a esse propósito. Eles serão brevemente discutidos na Seção 3.2.

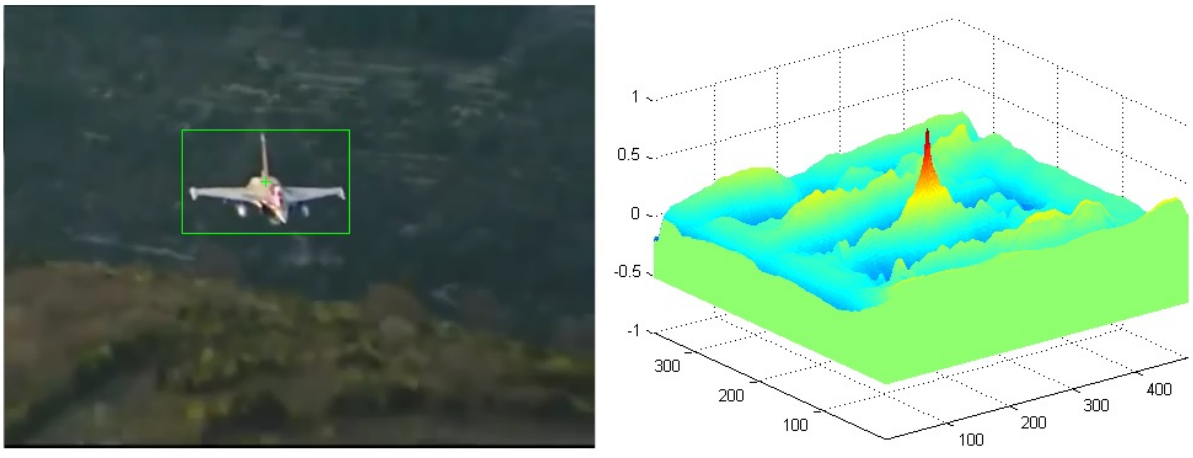


Figura 6: Imagem aeronave (esquerda) com correlação calculada em cada pixel (direita)

3.2 Técnicas de Inteligência Computacional

As técnicas de inteligência computacional fornecem soluções para problemas complicados por meio da modelagem da inteligência biológica e natural (ENGELBRECHT, 2007). Muitas delas atendem ao propósito de otimização, podendo ser citadas a computação evolucionária e a inteligência de enxame.

A computação evolucionária tem por objetivo imitar o processo natural de evolução, onde o indivíduo mais apto tem maior chance de sobrevivência. Os algoritmos associados a este tipo de abordagem utilizam uma população de indivíduos que corresponde às possíveis soluções do problema. Cada indivíduo possui cromossomos que definem suas características. Em cada geração, os indivíduos competem para reprodução. A chance de sobrevivência do indivíduo está associada a função objetivo, que define o problema. Dentre as técnicas utilizadas, Algoritmos Genéticos é aquele baseado diretamente na evolução genética das espécies, sendo um método amplamente utilizado para resolução de problemas científicos e de engenharia complexos (NP-hard), incluindo aqueles que recaiam em tarefas de otimização e aprendizado de máquina (MITCHELL, 1995).

As técnicas de inteligência de enxame tiveram origem no estudo do comportamento social de colônias ou enxames de organismos sociais como, por exemplo, abelhas, formigas, peixes e aves. São compostos por indivíduos simples, dotados de pouca inteligência e recursos sensoriais limitados, que interagem com o ambiente e cooperam entre si para solucionar problemas complexos. Também é conhecido como inteligência coletiva (ENGELBRECHT, 2006).

O algoritmo de otimização por enxame de partículas surgiu da observação do comportamento social de bandos de aves e cardumes de peixes (KENNEDY; EBERHART, 1995). As partículas, consideradas como possíveis soluções, se comportam como os pássaros. Se um dos elementos do grupo descobre um caminho onde há facilidade para encontrar o alimento, os outros componentes do grupo tendem a seguir também este caminho. Este método possui alta convergência e grande simplicidade de implementação e estruturação.

O algoritmo de otimização por colônia de formigas (DORIGO, 1992), se inspirou no comportamento de formigas em busca de alimento. A troca de informações entre os indivíduos é realizado por meio de depósito de feromônio no meio e permite que o menor caminho seja encontrado.

O comportamento coletivo das abelhas também inspirou o desenvolvimento do algoritmo de colônias de abelhas artificiais (KARABOGA, 2005). Na busca por néctar, as abelhas que encontram uma boa área para extração retornam a colmeia e compartilham essa informação com outras por meio de uma dança chamada *waggle dance*.

Esta dissertação aborda somente dois tipos de algoritmos de inteligência computacional para serem aplicados no processo de rastreamento por *template matching*: otimização por enxame de partículas e algoritmos genéticos, detalhados nas seções 3.2.1 e 3.2.2, respectivamente.

3.2.1 Otimização por Enxame de Partículas

O algoritmo de otimização por enxame de partículas, ou *Particle Swarm Optimization* (PSO), surgiu da observação do comportamento social de bandos de aves e cardumes de peixes (KENNEDY; EBERHART, 1995). As partículas, consideradas como possíveis soluções, se comportam como os pássaros à procura de alimento, utilizando o aprendizado próprio (componente cognitiva) e o aprendizado do bando (componente social). O problema é expresso por meio de uma função objetivo. Cada partícula possui coordenadas (que definem a sua posição no espaço de busca) e uma aptidão (que corresponde ao valor da função objetivo em sua posição atual. A aptidão define a qualidade da solução representada pela partícula. Por curiosidade, o termo partícula é utilizado, em analogia a física, por possuir posição e vetor velocidade bem definidos mas não possuir massa nem volume. Já o termo enxame, representa um conjunto de possíveis soluções. Para navegar no espaço de busca, a cada ciclo iterativo, a posição de cada partícula é atualizada segundo Equação 5:

$$x_i(t+1) = x_i(t) + v_i(t+1), \tag{5}$$

onde $x_i(t)$ é a posição da partícula i no tempo t e $x_i(t+1)$ e $v_i(t+1)$ são a posição e a velocidade da partícula i no tempo $t+1$, respectivamente.

O vetor velocidade também é atualizado a cada iteração e corresponde a soma de três componentes. As três componentes são inércia, memória e cooperação, ilustradas na Figura 7. A inércia tende a manter a partícula em uma direção idêntica à da iteração anterior, de acordo com um peso chamado coeficiente de inércia, introduzido por (SHI; EBERHART, 1998). A memória, também chamada de componente cognitiva, conduz a partícula na direção da melhor posição encontrada pela própria partícula em sua trajetória, adicionando uma tendência de retorno à essa posição. A cooperação, também chamada

de componente social, decorre da troca das experiências adquiridas entre as partículas do enxame. Ela conduz a partícula na direção da melhor posição encontrada pelo enxame ou por sua vizinhança, dependendo da topologia considerada. A estrutura social do PSO é determinada pela forma como as partículas trocam informações e exercem influência umas sobre as outras (ENGELBRECHT, 2006). Inicialmente, duas versões, chamadas global best PSO e local best PSO, foram propostas. Elas foram descritas mais detalhadamente nas seções 3.2.1.1 e 3.2.1.2, respectivamente.

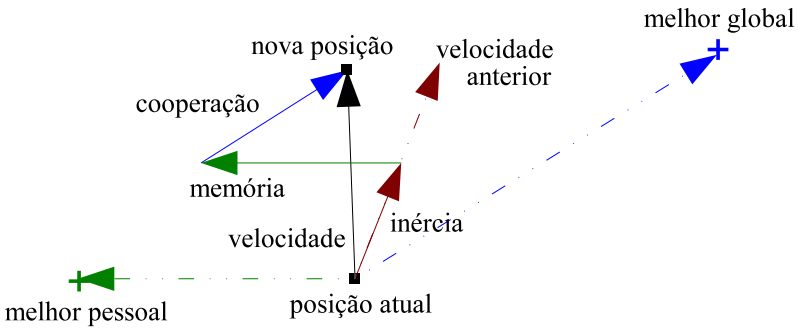


Figura 7: Componentes do vetor velocidade da partícula

Nas primeiras aplicações do PSO, verificou-se que a velocidade crescia rapidamente, especialmente para partículas distantes das melhores posições global e individual (ENGELBRECHT, 2006). Consequentemente, a cada iteração, a partícula percorria grandes distâncias, e acabava saindo do espaço de busca ou divergindo. Para controlar esse aspecto, estabeleceu-se um limite de velocidade para cada dimensão (ENGELBRECHT, 2006).

A Figura 8 apresenta um fluxograma com passos básicos para implementação do PSO. Os passos se repetem até que o critério de parada seja atingido. Os critérios de parada mais utilizados estão descritos na Seção 3.2.3.

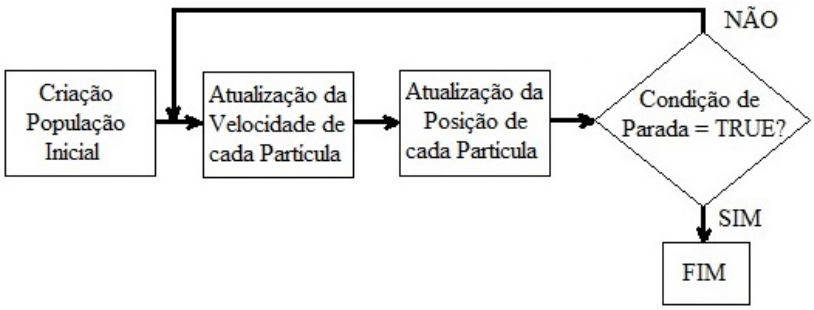


Figura 8: Fluxograma do PSO

3.2.1.1 PSO *Global Best*

Para o PSO *global best* (PSO-G), ou gbest PSO, a vizinhança da partícula consiste de todas as partículas do enxame. Este comportamento reflete a topologia em estrela da Figura 9. A informação obtida por qualquer das partículas está disponível para todas as outras. Neste caso, o vetor velocidade é atualizado conforme Equação 6:

$$v_i(t+1) = wv_i(t) + c_1r_1(pbest_i - x_i(t)) + c_2r_2(gbest - x_i(t)), \tag{6}$$

onde w é uma constante que representa a inércia da partícula, c_1 e c_2 são constantes que atribuem pesos das componentes cognitiva e social, respectivamente, r_1 e r_2 são números aleatórios entre 0 e 1, $pbest_i$ é a melhor posição encontrada pela partícula i e $gbest$ é a melhor posição encontrada pelo enxame. As posições das partículas se confinam no espaço de busca e a velocidade máxima é estabelecida para cada dimensão.

Segundo (ENGELBRECHT, 2006), esta topologia converge mais rápido que outras estruturas, embora seja mais suscetível a ficar presa em mínimos/máximos locais.

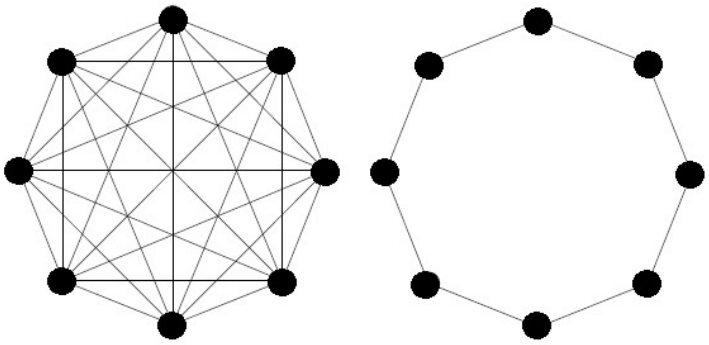


Figura 9: Duas topologias típicas do PSO

3.2.1.2 PSO *Local Best*

Para o PSO *local best* (PSO-L), ou lbest PSO, a vizinhança de cada partícula está limitada à duas partículas vizinhas. Este comportamento reflete a topologia em anel da Figura 9.

A informação obtida pela partícula é compartilhada, primeiramente, no âmbito de sua vizinhança. Como a informação flui mais lentamente entre as partículas, esta topologia é mais lenta que outras estruturas mas consegue explorar maiores partes do espaço de busca (ENGELBRECHT, 2006) e, normalmente, consegue obter melhores desempenhos em termos de qualidade da solução.

Para o PSO *local best*, o vetor velocidade da próxima iteração ($v_i(t+1)$) é atualizado conforme Equação 7:

$$v_i(t+1) = wv_i(t) + c_1r_1(pbest_i - x_i(t)) + c_2r_2(lbest - x_i(t)), \quad (7)$$

onde w é uma constante que representa a inércia da partícula, c_1 e c_2 são constantes que atribuem pesos das componentes cognitiva e social, respectivamente, r_1 e r_2 são números aleatórios entre 0 e 1, $pbest_i$ é a melhor posição encontrada pela partícula i e $lbest$ é a melhor posição encontrada pela vizinhança da partícula. Da mesma forma que no gbest PSO, as posições das partículas se confinam no espaço de busca e a velocidade máxima é estabelecida para cada dimensão.

3.2.2 Algoritmos Genéticos

Algoritmos Genéticos (*Genetic Algorithms* - GA), proposto por John Holland (HOLLAND, 1975), é uma das técnicas mais consolidadas da inteligência computacional. É inspirado na teoria da evolução genética das espécies (WHITLEY, 1994). A melhor solução para um determinado problema é encontrada a partir da combinação de possíveis soluções que são aprimoradas a cada geração. A cada geração, ou iteração, uma nova população de possíveis soluções, ou indivíduos, é criada a partir das informações genéticas dos melhores indivíduos da população da geração anterior.

O algoritmo representa uma possível solução utilizando uma estrutura simples, chamada cromossomo (MITCHELL, 1995). O cromossomo armazena as características do indivíduo por meio de genes, que representam as variáveis do problema a ser otimizada (ENGELBRECHT, 2006). No cromossomo, são aplicados operadores, geneticamente inspirados, de seleção, cruzamento e mutação, simulando o processo de evolução da solução.

Nesta dissertação, foi implementado o algoritmo genético tradicional, seguindo o fluxograma da Figura 10. No começo, a população inicial de possíveis soluções é gerada aleatoriamente a fim de assegurar uma uniforme representação do espaço de busca. Cada indivíduo é formado por um cromossomo, que são representados por cadeias binárias. Esta população é avaliada por uma função de avaliação de aptidão (*fitness function*), permitindo atribuir um valor de aptidão proporcional à qualidade da solução em resolver o problema.

Para criação da próxima geração, os indivíduos passam por um processo de seleção, que permite escolher aqueles indivíduos da população que são mais aptos a gerar os

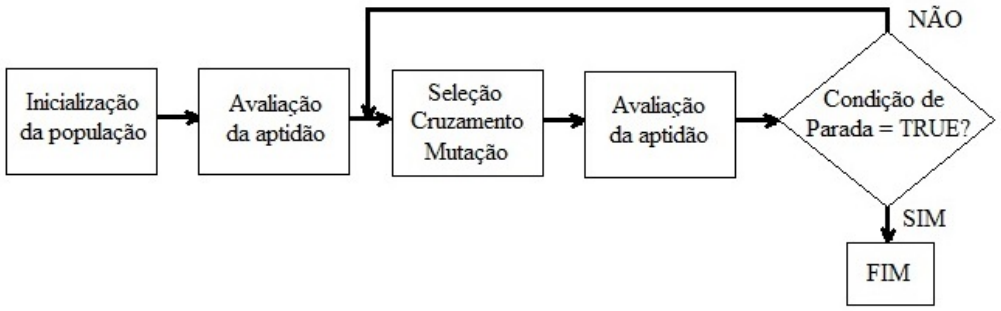


Figura 10: Fluxograma do GA

descendentes da geração seguinte. Quanto maior a aptidão do indivíduo, maior a chance dele se reproduzir e passar sua informação para a próxima geração (ENGELBRECHT, 2006). No algoritmo proposto em (HOLLAND, 1975), o processo de seleção é do tipo roleta viciada, onde cada indivíduo é representado por uma fatia do espaço proporcional à sua aptidão. O processo de seleção é repetido, com reposição, até que se atinja o número desejado de indivíduos necessários para formar a nova população. Torneio e rank também são outros meios de se realizar a seleção.

Para que não haja risco de se perder a melhor solução alcançada até o momento, algumas implementações de algoritmos genéticos utilizam o princípio de elitismo. Este operador consiste na repetição incondicional dos indivíduos melhores adaptados, sem mutação, de uma geração na geração seguinte (ENGELBRECHT, 2006).

Após o processo de seleção, realiza-se o cruzamento dos indivíduos selecionados. Cruzamento (*crossover*) é o processo de criar novos indivíduos através do material genético dos genitores (ENGELBRECHT, 2006). Os genitores são separados dois a dois e, com probabilidade P_c, realiza-se o cruzamento. Se o cruzamento não ocorrer, os dois descendentes serão cópias exatas de seus genitores (MITCHELL, 1995). Os algoritmos genéticos tradicionais geralmente usam o cruzamento simples ou de um ponto, onde os cromossomos de cada um dos genitores é cortado em um ponto aleatório e seus segmentos, a partir do ponto de corte, são permutados.

Após algumas gerações, a população terá indivíduos semelhantes, localizados em uma mesma região do espaço de busca, que pode não ser a região onde encontra-se o ótimo global, indicando uma convergência prematura. Para superar este problema, utiliza-se o operador mutação que fornece um comportamento exploratório, induzindo o algoritmo a amostrar novos pontos do espaço de busca. No operador proposto em (HOLLAND, 1975),

a mutação é caracterizada pela inversão, com uma certa probabilidade, de alguns bits do cromossomo. O processo é repetido, com a avaliação e criação de novas populações, até que um critério de parada seja estabelecido. Os critérios de parada mais utilizados estão descritos na Seção 3.2.3.

3.2.3 Critérios de Parada e Medidores de Desempenho

As técnicas de inteligência computacional buscam sempre boas soluções, que atendam satisfatoriamente a resolução de um problema. A solução obtida nem sempre é a melhor possível. Sendo assim, é necessário estabelecer critérios que indiquem o momento em que o algoritmo deve parar a busca. (ENGELBRECHT, 2006) cita alguns desses critérios, dos quais foram considerados como os mais relevantes:

- **Número máximo de iterações**: este critério é importante quando se deseja avaliar a melhor solução que o algoritmo é capaz de encontrar, dentro de um determinado período de tempo. Ele também é utilizado, em conjunto com outro critério, para terminar o algoritmo caso não haja convergência. A escolha do número de iterações deve ser criteriosa pois pode levar ao término prematuro do processo sem que uma boa solução seja encontrada, caso o número de iterações seja pequeno, ou elevar, sem necessidade, o custo computacional, caso o número de iterações seja grande.

- **Obtenção de uma solução aceitável**: o processo de busca termina quando a diferença entre a solução ótima e a melhor solução encontrada for menor que um limiar de erro pré-estabelecido. Da mesma forma, esse limiar deve ser cuidadosamente escolhido pois pode levar a soluções não ótimas, se for muito grande, ou dificultar o término da busca, se for muito pequeno. Este critério só pode ser usado se houver conhecimento prévio do valor ótimo esperado.

- **Nenhuma melhoria é observada durante um determinado número de iterações**: este critério verifica a estagnação do processo e pode ser utilizado para situações em que não se conhece o valor ótimo. A ausência de melhoria significativa na aptidão da melhor solução, durante um determinado número de iterações, pode indicar convergência dos indivíduos. No caso do PSO, uma variação pequena da velocidade média do enxame também pode indicar convergência.

Além do critério de parada, o desempenho dos algoritmos e a qualidade das soluções obtidas precisam ser monitorados. (ENGELBRECHT, 2006) cita alguns desses medidores, dos quais foram considerados os mais importantes:

- **acurácia (A)**: se refere diretamente a qualidade da solução obtida. Quando se conhece a solução ótima, corresponde a diferença do valor da função objetivo nesta posição, $f(otimo)$, com valor da função objetivo na posição da melhor solução obtida, $f(gbest)$, conforme Equação 8:

$$\text{Acurácia (A)} = |f(gbest) - f(otimo)|. \tag{8}$$

  Quanto mais próximo de zero, melhor a solução.

- **confiabilidade (C)**: se refere a porcentagem das simulações que convergiram para determinado valor da função objetivo ($N_{convergiram}$), com relação ao número total de simulações (N_{total}). Quanto maior, mais confiável é o algoritmo. A fórmula para cálculo é dada na Equação 9:

$$\text{Confiabilidade (C)} = \frac{N_{convergiram}}{N_{total}}. \tag{9}$$

- **eficiência (E)**: se refere ao tempo necessário para obter a solução com a precisão desejada. Vale ressaltar que quanto maior for a acurácia exigida, maior será esse tempo.

3.3 Comparação das Técnicas

Para comparação das técnicas, foram realizadas simulações computacionais por meio do software MATLAB 7.10.0 (R2010a), instalado em um computador ASUS, intel Core i7-2630 QM 2GHz, memória de 6Gb e com sistema Operacional Windows 7 Home Premium 64 bits. As técnicas gbest PSO, lbest PSO e GA foram comparadas com a busca exaustiva. Para implementação do algoritmo genético, foi utilizado um pacote de ferramentas (*toolbox*) externo ao MATLAB.

Para a busca exaustiva (BE), foi desenvolvido um *script* em MATLAB com os seguintes passos:

1. transformação da imagem principal e do *template* para uma escala de cinza. Os cálculos são efetuados em termos da informação de intensidade dos pixels;

2. extração de um pedaço (recorte), de mesmo tamanho do *template*;

3. cálculo do coeficiente de correlação de Pearson;

4. repetir passos 2 e 3 para cada um dos pixels da imagem principal. As bordas da imagem principal foram completadas com 0s (preto); e

5. comparar os resultados, um a um, a fim de encontrar aquele no qual o valor da correlação seja o mais elevado. Este pixel será designado como centro do objeto na imagem.

Para algoritmos genéticos foi implementado um outro *script* de MATLAB, observando os seguintes passos:

1. transformação da imagem principal e do *template* para uma escala de cinza. Os cálculos são efetuados em termos da informação de intensidade dos pixels;

2. geração da população inicial, com cromossomos correspondendo a posições aleatórias na imagem principal;

3. para cada indivíduo, extração de recorte, de mesmo tamanho do *template* e cálculo do coeficiente de correlação de Pearson. O valor da correlação de cada indivíduo equivale a sua aptidão. As bordas da imagem principal foram completadas com 0s, representando a cor preta;

4. geração de nova população, utilizando a seleção por roleta e elitismo, o cruzamento a um ponto e a mutação. Como a correlação possui valores negativos, antes da seleção pela roleta os valores de aptidão foram normalizados; e

5. repetição dos passos 3 e 4 até que um valor aceitável da correlação seja atingido ou um número máximo de iterações seja ultrapassado.

Para ambos algoritmos do PSO, o *script* para MATLAB obedeceu os seguintes passos:

1. transformação da imagem principal e do *template* para uma escala de cinza. Os cálculos são efetuados em termos da informação de intensidade dos pixels;

2. geração da população inicial das partículas, com posições e velocidades aleatórias;

3. para cada partícula, extração de um pedaço (recorte), de mesmo tamanho do *template* e cálculo do coeficiente de correlação de Pearson na posição da partícula. As bordas da imagem principal foram completadas com 0s (preto);

4. armazenar melhor valor encontrado pela partícula e melhor valor encontrado pela vizinhança;

5. atualização da posição e velocidade de cada partícula; e

6. repetição dos passos 3, 4 e 5 até que um valor aceitável do coeficiente de correlação seja atingido ou que um número máximo de iterações seja ultrapassado.

As imagens utilizadas para teste, ao lado do gráfico da correlação calculado para cada *template*, estão exibidas nas Figuras 6, 11 e 12. A Figura 6, denominada aeronave, foi retirada de um vídeo de aviões da internet (YOUTUBE, 2016). A Figura 11, denominada pickup, corresponde ao *frame* n° 715 do *benchmark* EgTest05, baixado de (COLLINS; ZHOU; TEH, 2005). E a Figura 12, denominada carros, corresponde ao primeiro *frame* do *benchmark* EgTest02, baixado de (COLLINS; ZHOU; TEH, 2005).

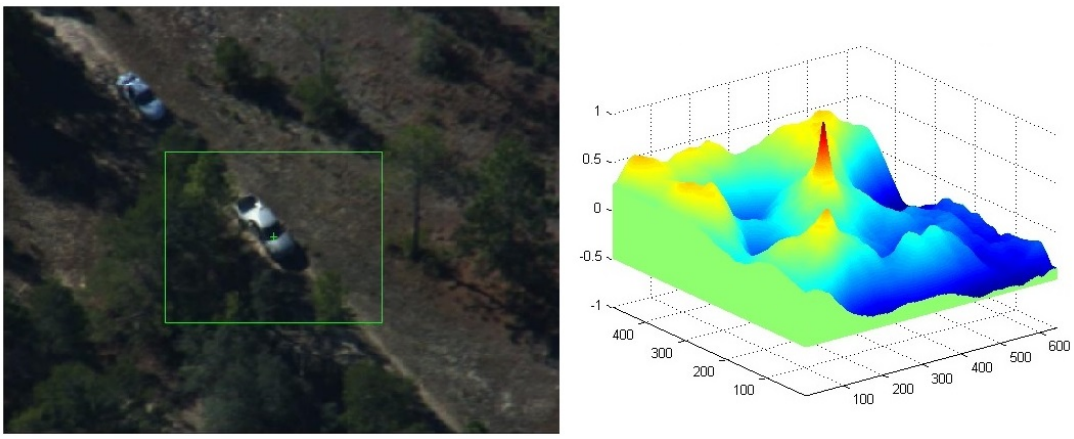


Figura 11: Imagem pickup (esquerda) com correlação calculada em cada pixel (direita)

Todos os algoritmos foram utilizados para encontrar a localização do pixel referente ao centro do *template* na imagem original. O tempo de processamento e o valor máximo da correlação para as imagens aeronave e pickup estão exibidos nas colunas 3 e 4 da Tabela 1, respectivamente. Neste caso, os parâmetros do GA, PSO-G e PSO-L foram configurados, empiricamente, da seguinte forma, denominada $config_1$:

- $GA_{config1}$: população de 30 indivíduos; seleção por roleta e elitismo; taxa de cruzamento de 80%; taxa de mutação de 12%.

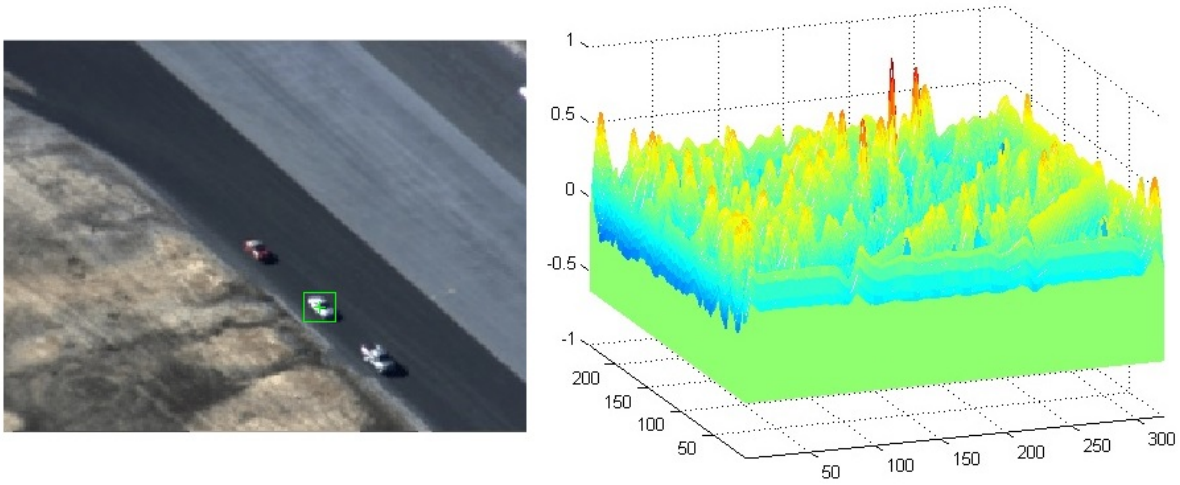


Figura 12: Imagem carros (esquerda) com correlação calculada em cada pixel (direita)

- $PSO_{config1}$: enxame de 100 partículas, velocidade máxima permitida V_{max} de 10, $w = 1$, $c_1 = 1,5$ e $c_2 = 2$.

Como critério de parada, foram considerados o valor de 0,95 para o coeficiente de correlação ou um número máximo de iterações. Para o algoritmo genético foi estipulado um número máximo de 300 gerações e para o PSO-G e PSO-L, 50 iterações.

Tabela 1: Alguns resultados para BE, GA e PSO

		aeronave $config_1$	pickup $config_1$	carros $config_1$	carros $config_2$
Posição (linha,coluna)	BE	(145,214)	(259,310)	(164,194)	(164,194)
	GA	(144,214)	(260,310)	(164,194)	(164,194)
	PSO-G	(144,215)	(260,312)	(164,194)	(164,194)
	PSO-L	(146,214)	(258,310)	(164,194)	(164,194)
PCC	BE	0.9997	0.9994	0.9987	0.9987
	GA	0.9596	0.9850	0.9987	0.9987
	PSO-G	0.9699	0.9556	0.9987	0.9987
	PSO-L	0.9597	0.9849	0.9987	0.9987
Tempo (seg)	BE	34.5	188.53	4.67	4.67
	GA	0.7318	0.6814	0.6882	2.1382
	PSO-G	0.3075	0.5209	0.2472	0.4705
	PSO-L	0.6683	1.1117	0.2852	0.9458

Durantes as simulações, observa-se que todos algoritmos conseguem localizar o objeto desejado em um tempo menor do que a busca exaustiva. Para a busca exaustiva, o tempo de processamento está diretamente ligado ao tamanho da imagem e do *template*. Já para os algoritmos avaliados, esta afirmação não é sempre verdadeira pois o tempo de processamento também é influenciado pela forma como os parâmetros são escolhidos.

No caso do PSO, observou-se claramente o comportamento em enxame das partículas, conforme animação ilustrada na Figura 13, onde as cruzes indicam as posições das partículas e o número indicado no canto superior esquerdo informa o número da iteração considerada. A animação referente a movimentação das partículas pode ser encontrado no site http://youtu.be/CFEmWdjYQag.

Figura 13: Comportamento das partículas como enxame

Para estender um pouco mais a tarefa e dar mais amplitude ao estudo, foi avaliada a imagem carros, Figura 12. Observa-se um fator complicativo por meio do gráfico da correlação que apresenta muitos máximos locais. Desta vez, os algoritmos precisaram ser configurados de forma a evitar esses máximos locais. Os resultados desta configuração, chamada $config_2$, estão exibidos na última coluna da Tabela 1.

- $GA_{config2}$: população de 35 indivíduos; seleção por roleta e elitismo; taxa de cruzamento mantida em 80%; taxa de mutação de 15%; número máximo de gerações de 400.

- $PSO_{config2}$: enxame de 200 partículas; w =1,4; c_1 = 1,5; c_2 = 1,5; velocidade máxima mantida em V_{max} = 10. O número máximo de iterações para o PSO-G foi configurado para 50 enquanto que para o PSO-L foi configurado em 150.

Tendo em vista o caráter estocástico do GA e PSO, é preciso avaliar o seu grau de confiabilidade, considerando as configurações escolhidas. Para isso, realizou-se simulações do tipo Monte Carlo, no qual a busca foi repetida 100 vezes a fim de avaliar a confiabilidade dos algoritmos (probabilidade deles convergirem). Os resultados estão exibidos nas Tabelas 2, 3 e 4 para o GA, PSO-G e PSO-L, respectivamente. Observa-se que o PSO-G é

mais rápido (eficiente) e tem maior confiabilidade que ambos PSO-L e GA. PSO-L é mais eficiente que GA. Para a imagem carros, o *speed-up* obtido com GA é aproximadamente 2, com confiabilidade de 71%. Além disso, os resultados apontam que os algoritmos são sensíveis à escolha dos parâmetros. Em um problema real, existe a possibilidade que não haja convergência e que seja indicado um valor errôneo para a posição do alvo. Para mitigar os impactos, sugere-se repetir a busca algumas vezes até que o resultado desejado seja alcançado.

Um artigo sobre a comparação destas duas técnicas foi submetido e aceito para apresentação oral no 12º Congresso Brasileiro de Inteligência Computacional (TAVARES; NEDJAH; MOURELLE, 2015). Na ocasião, os autores foram convidados a submeter uma versão estendida do artigo para uma *special issue* do *International Journal of Swarm Intelligence Research* e uma do *International Journal of Bio-Inspired Computation*.

Tabela 2: Simulações Monte Carlo para GA

Imagem	Tempo (seg)		Gerações		Confiabilidade
	média	desvio	média	desvio	(%)
aeronave	0.7740	0.5715	47.02	34.85	100
pickup	4.2493	3.2171	140.78	106.45	85
carros ($config1$)	2.2791	1.2127	204.26	108.68	52
carros ($config2$)	2.7883	1.9209	221.14	152.25	71

Tabela 3: Simulações Monte Carlo para PSO-G

Imagem	Tempo (seg)		Gerações		Confiabilidade
	média	desvio	média	desvio	(%)
aeronave	0.2559	0.1103	11.86	5.12	100
pickup	0.7984	0.4032	12.56	6.43	99
carros ($config1$)	0.1948	0.0992	33.37	17.05	56
carros ($config2$)	0.3163	0.1750	27.02	14.98	81

Tabela 4: Simulações Monte Carlo para PSO-L

Imagem	Tempo (seg)		Gerações		Confiabilidade
	média	desvio	média	desvio	(%)
aeronave	0.5448	0.2585	24.69	11.78	98
pickup	1.2108	0.5366	19.29	8.56	100
carros ($config1$)	0.2725	0.0652	43.92	10.55	31
carros ($config2$)	1.0257	0.6466	82.25	51.79	72

3.4 Considerações Finais do Capítulo

Neste Capítulo a tarefa de *template matching* foi apresentada como problema de otimização e algumas técnicas de inteligência computacional foram sugeridas para sua resolução. Duas técnicas foram efetivamente comparadas: algoritmos genéticos e otimização por enxame de partículas (algoritmos *global best* e *local best*).

Os testes possibilitaram confirmar que todos os algoritmos conseguiram rastrear o *template* em menor tempo que a busca exaustiva. Os dois algoritmos de PSO obtiveram melhores resultados que o algoritmo de GA. PSO *global best* foi melhor que *PSO local best*, em termos de eficiência (tempo de processamento) e confiabilidade. Conclui-se que o PSO global best é a ferramenta mais indicada para realizar a otimização desejada, trazendo grande agilidade na tarefa. Conseguiu-se acelerar o tempo de processamento das imagens aeronave, pickup e carros em 134x, 236x e 14x, respectivamente, comparando as simulações Monte Carlo e a busca exaustiva.

O maior problema diz respeito a sensibilidade dos algoritmos à escolha dos parâmetros. Em um problema real, existe a possibilidade que não haja convergência. Para corrigir tal problema, é possível repetir o algoritmo PSO algumas vezes, até que o objeto seja encontrado. Contudo, este procedimento pode diminuir o desempenho do algoritmo.

Capítulo 4

IMPLEMENTAÇÃO DA ARQUITETURA PROPOSTA

NO Capítulo anterior, duas técnicas de inteligência computacional foram comparadas na resolução do problema de rastreamento de objetos por *template matching* e correlação: algoritmos genéticos e otimização por enxame de partículas As simulações em MATLAB indicaram que o PSO *global best* obteve melhores resultados, em termos de eficiência (tempo de processamento) e confiabilidade, sendo a ferramenta mais indicada. Esta abordagem proporciona boa solução para o problema, contudo, em diversas aplicações de segurança e defesa, como, por exemplo, em mísseis, drones e aeronaves, o sistema precisa estar embarcado em equipamento portátil.

Neste Capítulo, a solução elaborada é proposta em uma plataforma portátil via um projeto integrado de software/hardware, utilizando a técnica PSO *global best*, implementada em software, e cálculo da coeficiente de correlação, implementado em hardware via um coprocessador dedicado. Para tal, a Seção 4.1 descreve a plataforma de hardware sobre o qual o sistema foi implementado. A Seção 4.2 apresenta as motivações para utilização da abordagem integrada de software/hardware, chamada *co-design*. Na Seção 4.3, a arquitetura proposta é descrita no nível macro. E por último, na Seção 4.4, são realizadas considerações finais.

4.1 Plataforma de Hardware

O projeto proposto nesta dissertação utiliza a placa *Smart Vision Development Kit* rev 1.2 (SVDK) da *Sensor to Image* (SENSORTOIMAGE, 2015). A Figura 14 apresenta o diagrama em blocos da placa, que possui um módulo da Xilinx chamado *PicoZed 7Z015 System-On Module* rev.C que proporciona um ambiente de hardware voltado para o desenvolvimento

e avaliação de aplicações de visão de máquina. A placa SVDK também possui diversos recursos associados a sistemas de processamento embarcado, incluindo interface UART, encoder de vídeo HDMI, Ethernet PHYs tri-mode, I/O de propósito geral, e também componentes padrão para visão de máquina como as interfaces USB3, GigE Vision e CoaXPress. Sensores também podem ser conectados através do barramento PCI Express.

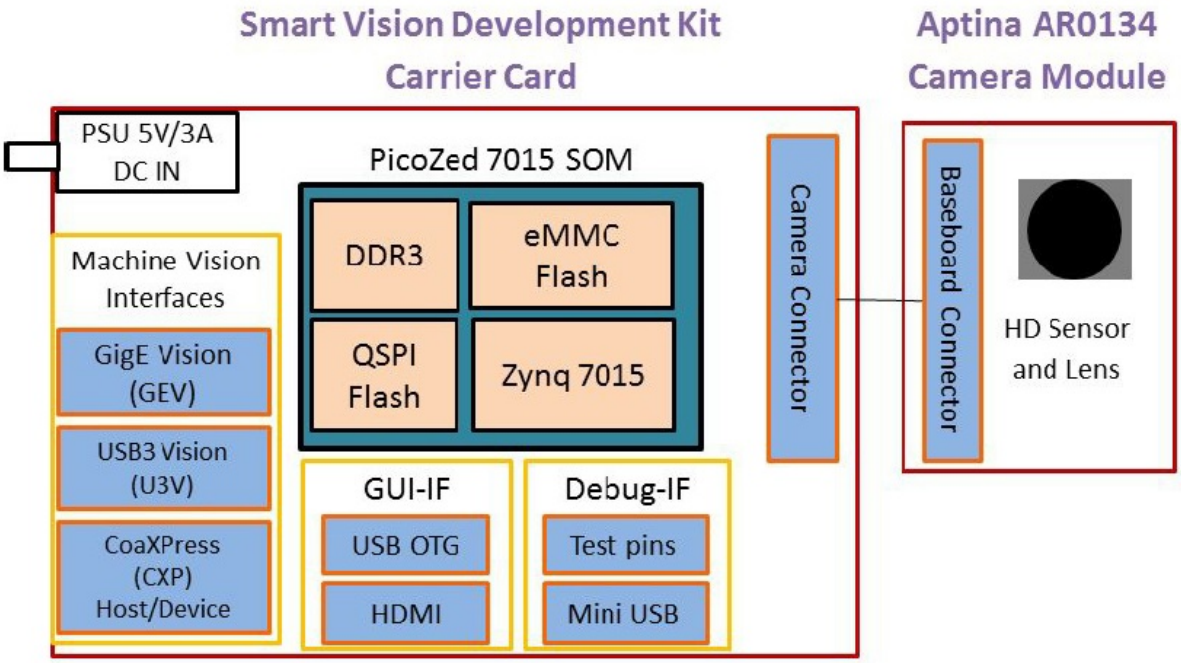


Figura 14: Diagrama em blocos da placa SVDK (SENSORTOIMAGE, 2015)

O módulo *PicoZed 7Z015 SOM* é uma placa de desenvolvimento de baixo custo voltada para aplicações diversas. Dentre seus recursos, possui 1 GB de memória DDR3 (x32), oscilador de 33,333 MHz, além do chip XC7Z015, da Xilinx (AVNET, 2015).

O chip XC7Z015 é um produto da família Xilinx Zynq-7000, com arquitetura *All Programmable System on Chip*. A Figura 15 ilustra os blocos funcionais deste componente que possui sistema de processamento (PS), baseado em processador dual-core ARM Cortex-A9, e lógica programável (PL) em um único chip (XILINX, 2015). Esse componente fornece a escalabilidade e flexibilidade de um FPGA com o desempenho, capacidade e facilidade de uso associados com ASIC e é perfeito para a implementações do tipo co-design.

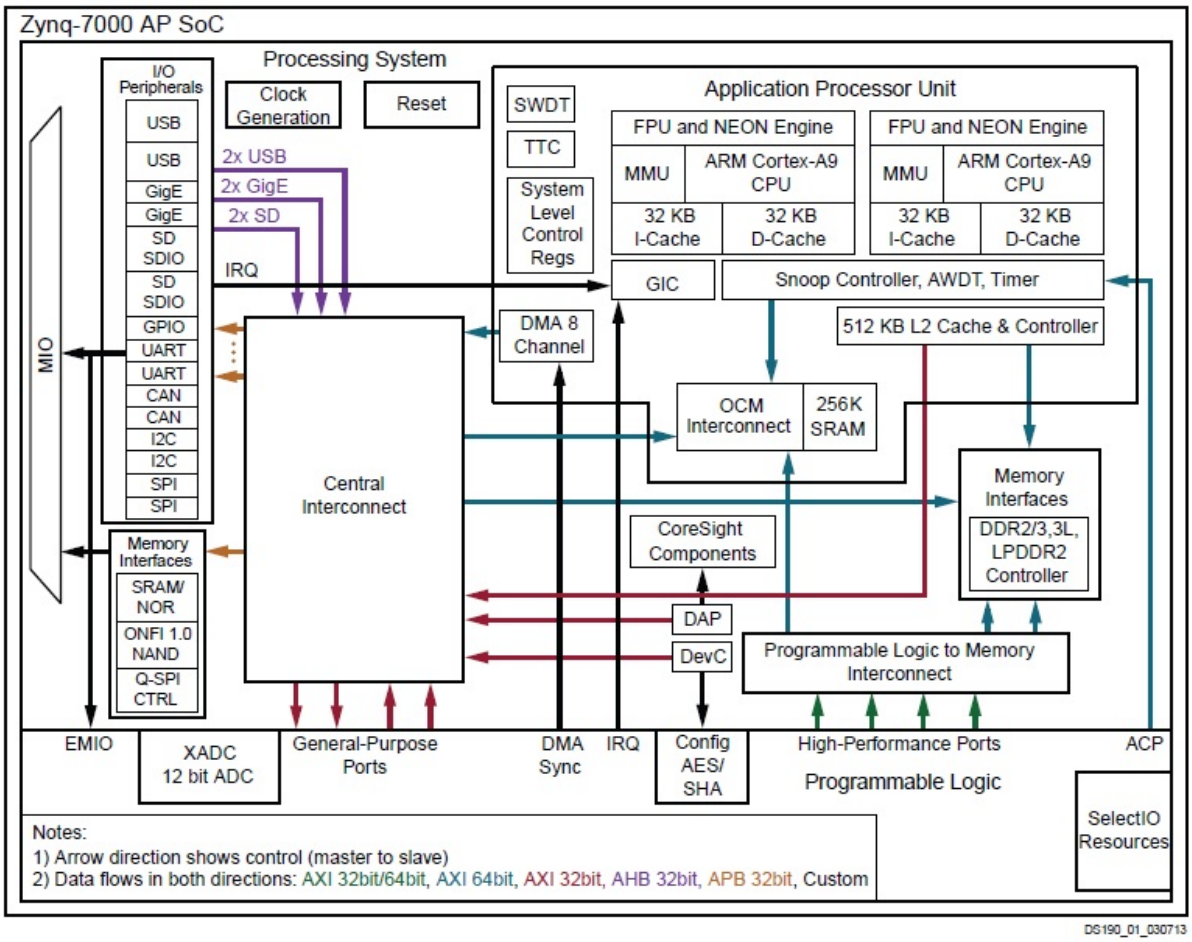


Figura 15: Blocos funcionais da família Zynq-7000 (XILINX, 2015)

4.2 Motivação para uso do Coprocessador Dedicado

Especificações de sistemas embarcados normalmente impõe restrições bastante limitadas com respeito ao custo, tempo de resposta e consumo de energia. Eles normalmente incluem software, hardware e subsistemas de interface. Além disso, o desenvolvimento deste tipo de sistema usualmente requer otimização concomitante de vários objetivos de projeto, que são conflitantes, em muitos dos casos. Ademais, esses projetos visam resolver problemas do tipo NP-hard (NEDJAH; MOURELLE, 2014). Com objetivo de realizar uma avaliação inicial do projeto em um sistema embarcado, a técnica PSO *global best*, indicada no Capítulo 3 para otimização do problema de rastreamento, foi implementada na placa SVDK. Seguindo os mesmos passos para elaboração do script para MATLAB (Seção 3.3), foi desenvolvido um programa, na linguagem C, a ser executado pelo processador do chip Zynq XC7Z015 a fim de rastrear os *templates* nas Figuras 6 e 12. Nestes testes iniciais, somente o sistema de processamento do chip foi empregado. Foram utilizados os mesmos

parâmetros e critérios de parada. Todos os *templates* tiveram seus tamanhos padronizados em 64x64 pixels. Alguns dos resultados (tempo de processamento, número de iterações e valor máximo de correlação) estão apresentados nas Tabelas 5 e 6.

Tabela 5: Alguns resultados para *aeronave*

	BE	**PSO**	**PSO**	**PSO**
Posição (*linha,coluna*)	(154,215)	(154,215)	(54,321)	(154,216)
PCC	0,999	0,999	0,554	0,964
Iterações	-	7	30	22
Tempo (*ms*)	392064	1568	5854	4771

Tabela 6: Alguns resultados para *carros*

	BE	**PSO**	**PSO**	**PSO**
Posição (*linha, coluna*)	(323,389)	(252,310)	(324,390)	(252,310)
PCC	0,999	0,758	0,962	0,758
Iterações	-	30	20	30
Tempo (*ms*)	768001	6113	4411	6124

O rastreamento foi repetido 100 vezes em cada imagem, para avaliar o tempo de processamento médio. Os resultados encontram-se na Tabela 7.

Tabela 7: Média dos resultados para 100 repetições

	aeronave	***carros***
Iterações	16,71	22,53
Tempo (*ms*)	3600	4842

Os resultados confirmam que o PSO *global best* é uma excelente ferramenta para diminuir o tempo de processamento no problema abordado. Comparando-se com a busca exaustiva, conseguiu-se um *speed-up* médio de 108x e 158x para as imagens *aeronave* e *carros*, respectivamente. Mesmo assim, em aplicações de tempo real os resultados não são suficientes.

O movimento de uma cena de um vídeo é traduzido pela exibição de vários *frames* consecutivos, transmitindo ao nosso cérebro a sensação de animação. Um vídeo tradicional apresenta a taxa de 30 *frames* por segundo. Nesse caso, o objeto precisaria ser rastreado em um período de tempo menor que $\frac{1}{30} = 33,333$ *ms*. Com este propósito em mente, alguns testes foram realizados para descobrir qual parte do código era mais custosa computacionalmente.

Estimou-se o tempo total de uma iteração como uma soma de fatores, conforme Equação 10:

$$T_T = T_{PCC} + T_R + T_C + T_A, \tag{10}$$

onde T_T é o tempo total de uma iteração; T_{PCC} é o tempo gasto pelo processador para realizar o cálculo de PCC; T_R corresponde ao tempo gasto para extração do recorte da imagem principal; T_C corresponde ao tempo gasto para comparação e verificação das melhores partículas e, finalmente, T_A é o tempo gasto para atualização das posições e velocidade das partículas.

Com objetivo de quantificar os tempos estimados na Equação 10, foram realizados cinco experimentos com a imagem *aeronave*, relacionados a uma única iteração, cujos resultados encontram-se na Tabela 8. O tempo T_1 corresponde a T_T; T_2 corresponde a T_T menos T_{PCC} ($T_2 = T_R + T_C + T_A$); T_3 corresponde a T_T menos T_{PCC} e T_R ($T_3 = T_C + T_A$); T_4 corresponde a T_A e, por último, T_5 corresponde a T_C.

Tabela 8: Tempos de processamento para uma iteração da imagem *aeronave*

	$\boldsymbol{T}_1$	$\boldsymbol{T}_2$	$\boldsymbol{T}_3$	$\boldsymbol{T}_4$	$\boldsymbol{T}_5$
Tempo (μs)	226200	24570	95.9	77.8	36.9

É possível identificar que o cálculo de PCC é a parte mais custosa do algoritmo, seguido pelo acesso a memória para retirar os recortes da imagem principal. Esta dissertação propõe, como forma de melhorar o tempo de processamento, a implementação do cálculo do PCC em um coprocessador dedicado, explorando sua capacidade de paralelizar o processo. Além disso, propõe a utilização de uma memória local a esse processador, de forma que o acesso a cada dado seja no nível de registradores. Enquanto que um *frame* está sendo tratado pelo coprocessador, o próximo já pode ser carregado nesta memória. A arquitetura foi proposta desta forma, com PSO sendo executado pelo processador de propósito geral do chip (PS) e com coprocessador e memórias dedicados implementados na lógica programável do chip (PL). Esta abordagem, chamada *co-design*, é uma metodologia para desenvolver um sistema integrado, utilizando componentes de hardware e software, para atender requisitos de desempenho e limitações de custo. A arquitetura final tem, normalmente, componentes de software executados pelo processador que é auxiliado por componentes de hardware dedicados desenvolvidos especialmente para a aplicação (NEDJAH; MOURELLE, 2007).

Um artigo sobre a implementação do *Template Matching* em um sistema embarcado, contendo também essa análise de tempos, foi submetido e aceito para a *16th International Conference on Computational Science and Its Applications* (TAVARES; NEDJAH; MOURELLE, 2016a).

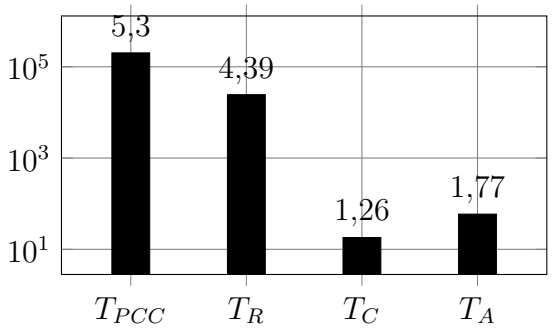


Figura 16: Tempo gasto com cada tarefa (ms) em uma iteração (escala logarítmica)

4.3 Descrição da Macro-arquitetura

O projeto proposto, utilizando a placa de desenvolvimento, descrita na Seção 4.1, e a abordagem co-design, descrita na Seção 4.2, é apresentado em detalhes nesta Seção. A macro-arquitetura desse sistema integrado é apresentada na Figura 17 e contempla um processador de propósito geral para execução do passo do PSO, um coprocessador para o cálculo do PCC, os blocos de memória dedicados (`BRAM IMG` e `BRAM TMP`) para armazenar a imagem principal e o *template*, respectivamente, e os blocos do controle de acesso a essas memórias (`GET IMG` e `GET TMP`). Os componentes foram descritos por meio da linguagem de descrição de hardware VHDL (*Very high speed integrated circuits Hardware Description Language*) e sintetizados por meio da ferramenta de software Vivado, da própria Xilinx. O sistema integrado desenvolvido no Vivado pode ser visto na Figura 18. Uma vez desenvolvido, o sistema de hardware é exportado para o Software Development Kit (SDK), também da Xilinx. O SDK proporciona um ambiente de desenvolvimento de projetos de aplicação de software e permite a configuração da lógica programável do chip XC7Z015, através da placa SVDK, e a programação de seu processador.

Dois artigos sobre a implementação do projeto integrado de software/hardware utilizando PSO e o coprocessador foram submetidos e aceitos para o 21^o Congresso Brasileiro de Automática (TAVARES; NEDJAH; MOURELLE, 2016c) e para a *Latin American Conference on Computational Intelligence* (TAVARES; NEDJAH; MOURELLE, 2016b).

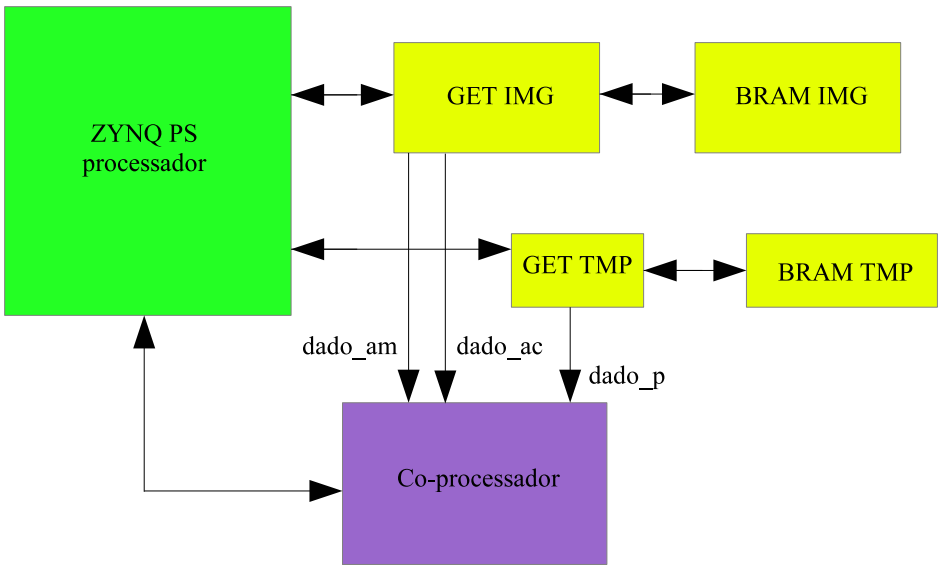


Figura 17: Macro-arquitetura do sistema proposto

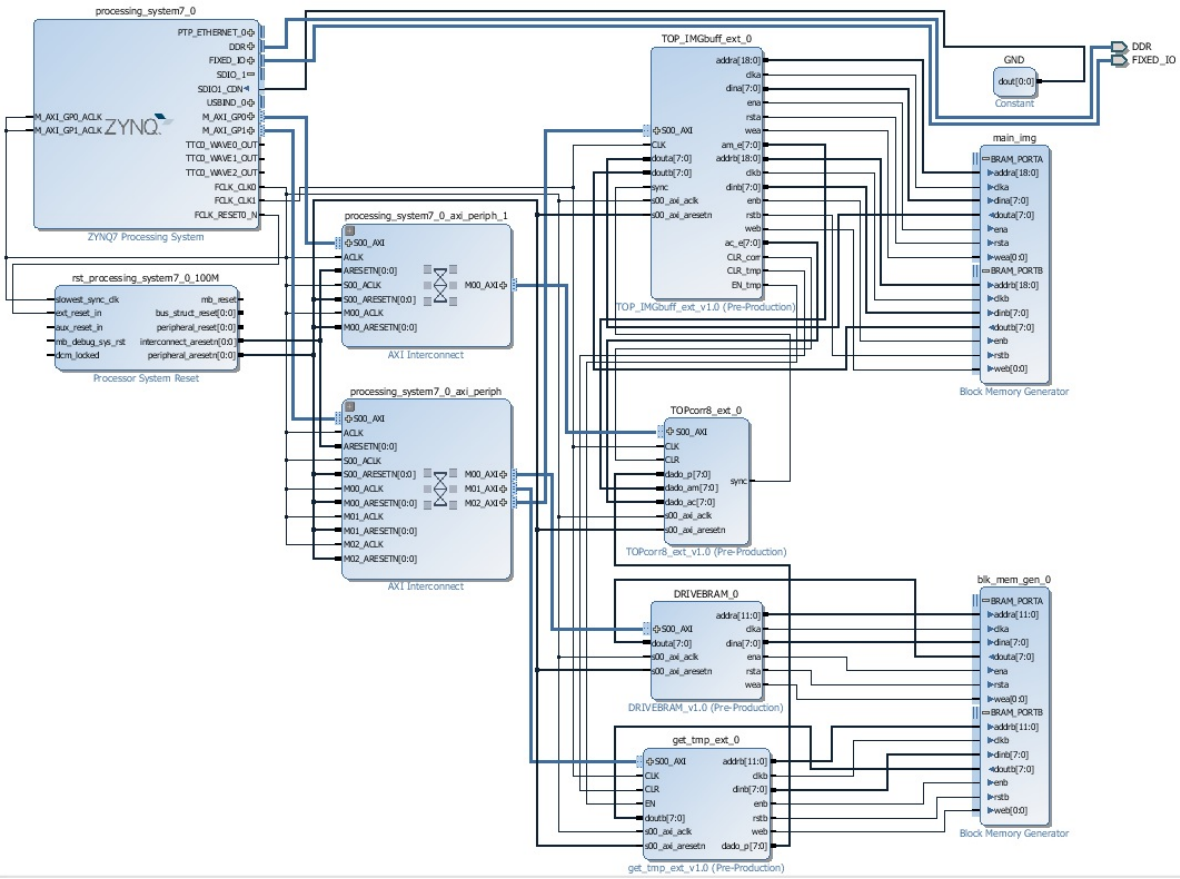


Figura 18: Implementação do sistema no Vivado

4.3.1 Coprocessador Dedicado

A Figura 19 apresenta a arquitetura proposta para o coprocessador. Este é responsável por realizar o cálculo da correlação entre duas imagens, conforme definido na Equação

4. Sua estrutura é projetada na forma de *pipeline*, onde cada um de seus três blocos corresponde a um dos seus três estágios. A cada transição de subida do sinal do *clock*, três informações são recebidas pelo coprocessador:

- `dado_p`: um pixel da imagem de referência (*template*), representado por 8 bits;
- `dado_ac`: um pixel da imagem a ser comparada, também de 8 bits;
- `dado_am`: um pixel da próxima imagem a ser comparada, de 8 bits.

Todas as imagens a serem comparadas foram definidas com 64x64 pixels, totalizando 4096 pixels que são representados por 4 KBytes. Os cálculos são realizados em cada bloco e passados para o próximo, a cada pulso de sincronismo. Esse pulso é gerado pelo componente Sincro a cada 4103 transições de subida do sinal do *clock*. Como saída, o coprocessador fornece o valor da correlação calculado (`result`), em complemento a 2, codificado em 32 bits, o flag indicando término (`flag_end`) e um sinal de erro (`error`) que indica que o resultado não é válido, sendo oriundo de uma divisão por zero.

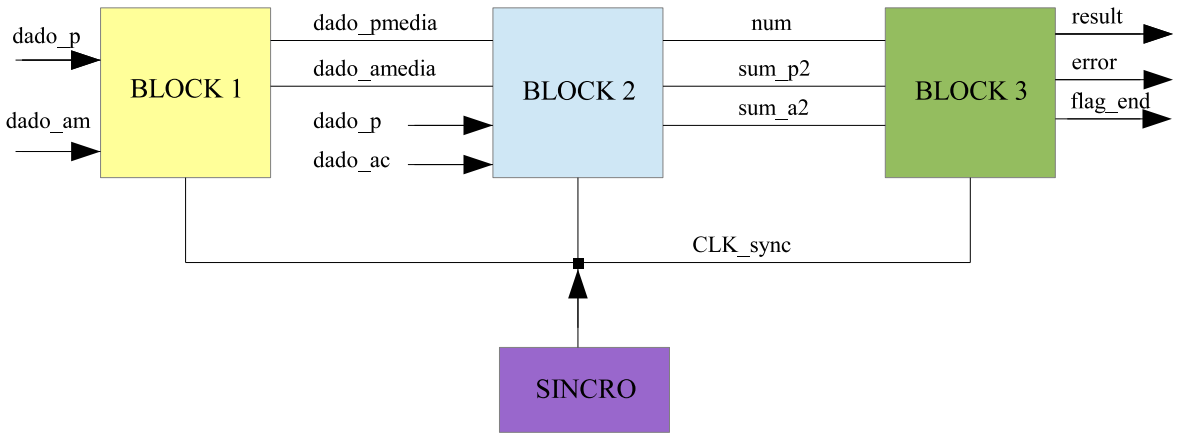


Figura 19: Macro-arquiteura do coprocessador

4.3.1.1 Macro-arquitetura do Bloco 1

A Figura 20 exibe o bloco 1 que forma o primeiro estágio do *pipeline* e é responsável por calcular a média dos pixels das duas imagens a serem comparadas. Possui registradores de saída que são carregados somente quando a tarefa do estágio é completada. Com pulso de sincronismo, somente o componente `media`, responsável por calcular a média da imagem a ser compara com o *template* é reiniciado.

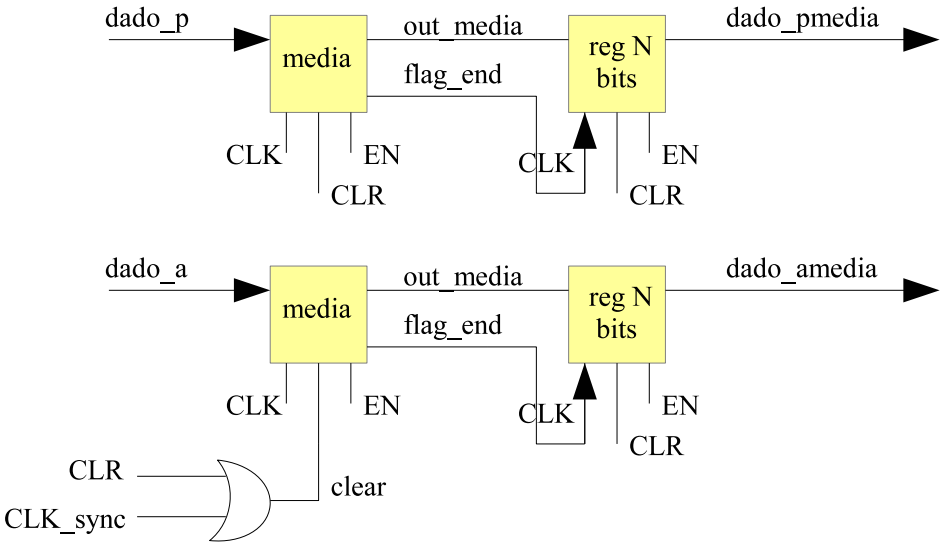


Figura 20: Arquitetura do Bloco 1

4.3.1.2 Macro-arquitetura do Bloco 2

A Figura 21 exibe a arquitetura do bloco 2 que forma o segundo estágio do *pipeline*. Este é responsável por calcular os 3 somatórios da correlação cruzada normalizada (Equação 4). É composto por 2 componentes `subt_A2` que realizam a subtração em complemento a2 dos pixels das imagens com as médias calculadas pelo bloco 1; por 3 componentes `mult_CLK` que realizam, em um pulso de *clock*, a multiplicação dos resultados oriundos dos componentes `subt_A2`; e por 3 componentes `soma_A2` que realizam os somatórios, em complemento a2, dos resultados das multiplicações provenientes dos componentes `mult_CLK`. Da mesma forma que no caso do bloco 1, o bloco 2 possui registradores de saída que são carregados somente quando a computação do estágio é completada. Com o pulso de sincronismo, os componentes `subt_A2`, `mult_CLK` e `soma_A2` são reiniciados.

4.3.1.3 Macro-arquitetura do Bloco 3

A Figura 22 exibe a arquitetura do bloco 3, compondo o terceiro estágio do *pipeline*. Este é responsável por calcular a multiplicação principal, a raiz quadrada e a divisão da equação da correlação cruzada normalizada (Equação 4). É composto pelo componente `mult_CLK` que, em um pulso de CLK, executa a multiplicação dos somatórios do denominador da Equação 4; pelo componente `SQRT` que calcula a raiz quadrada; e pelo componente `div_frac_A2` que realiza a divisão, proporcionando um resultado com a precisão de 2^{-24}. Este último componente é aquele que fornece os sinais de saída do coprocessador. A ope-

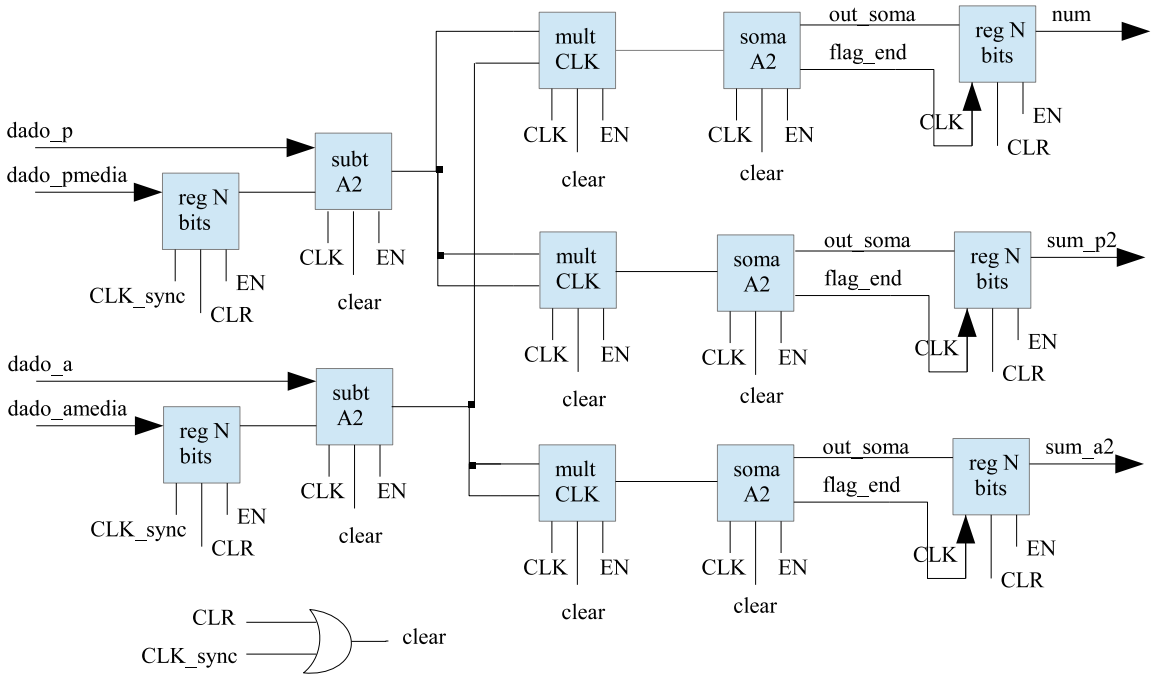


Figura 21: Arquitetura do Bloco 2

ração deste bloco é controlada por uma máquina de estado (FSM), coordenando a atuação dos seus componentes. Com o pulso de sincronismo, a máquina retorna ao seu estado inicial. Da mesma forma que os blocos 1 e 2, possui registradores de saída que são carregados somente quando a tarefa do estágio é completada.

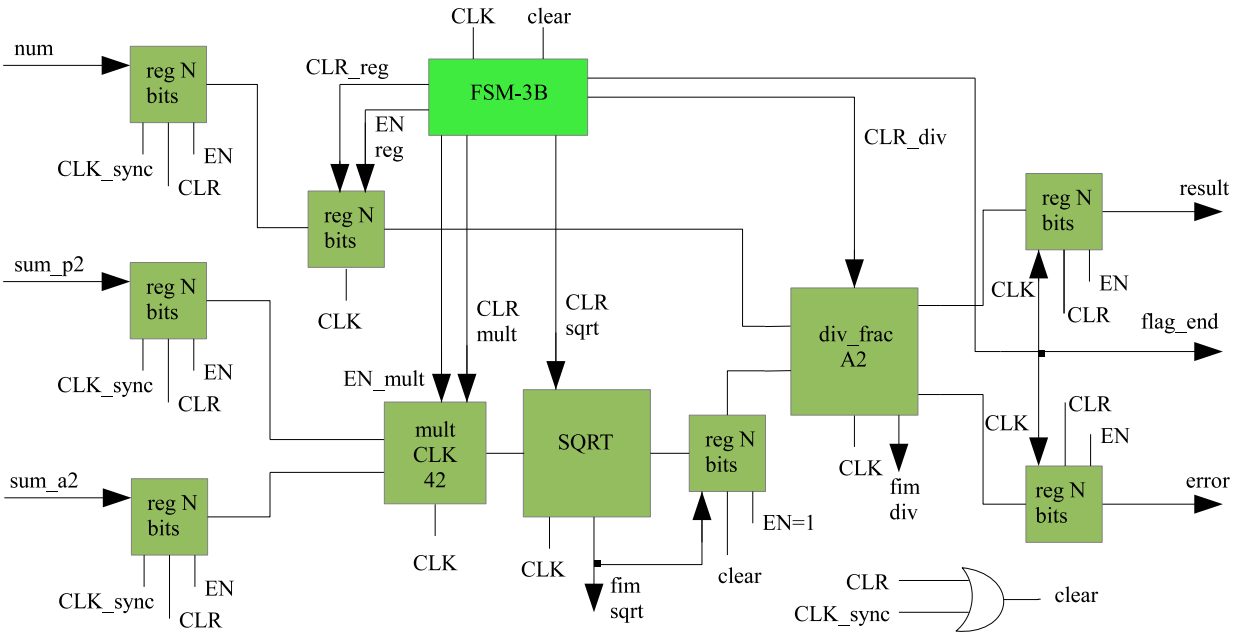


Figura 22: Arquitetura do Bloco 3

O método numérico *babylonian* foi utilizado para cálculo da raiz quadrada. Baseado nele, o componente `SQRT` do bloco 3 foi implementado, em hardware., utilizando um processo iterativo.

Ademais, o Algoritmo 1 foi utilizado para o cálculo da divisão. Esse algoritmo foi implementado, em hardware, para o desenvolvimento do componente `div_frac_A2`.

Algorithm 1 Divisão $Q = A/B$ com 24 bits de precisão fracional

$N_B := 0$
$RA_{\langle N-1 \ldots \frac{N}{2} \rangle} :=$ A
$RA_{\langle \frac{N}{2}-1 \ldots 0 \rangle} := 0$
enquanto $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} > B$ **faça**
  $RA :=$ deslocar RA para direita
  $N_B := N_B + 1$
fim enquanto
$Q := 0$
para $j = 1, N_B$ **faça**
  $RA :=$ deslocar RA para esquerda
  $Q :=$ deslocar Q para esquerda
  se $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} > B$ **então**
    $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} := RA_{\langle N-1 \ldots \frac{N}{2} \rangle} - B$
    $Q_0 := 1$
  senão
    $Q_0 := 0$
  fim se
fim para
para $j = 1, 24$ **faça**
  $RA :=$ deslocar RA para esquerda
  $Q :=$ deslocar Q para esquerda
  se $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} > B$ **então**
    $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} := RA_{\langle N-1 \ldots \frac{N}{2} \rangle} - B$
    $Q_0 := 1$
  senão
    $Q_0 := 0$
  fim se
fim para

4.3.2 Controladores das Memórias

Os blocos de memória dedicada `BRAM_TMP` e `BRAM_IMG` armazenam o *template* e a imagem principal, respectivamente. Estes foram implementados na lógica programável do chip Zynq (PL). O componente `BRAM_TMP` pode armazenar até 4096 pixels de 8 bits, totalizando 4K bytes (o que corresponde ao tamanho do *template*). Já o componente `BRAM_IMG` pode

armazenar até 573x463 pixels de 8 bits cada, totalizando 260K bytes. Como as bordas da imagem principal são completadas com zeros, o tamanho máximo desta imagem é 510x400 pixels.

Os controladores `GET_TMP` e `GET_IMG` são responsáveis por proporcionar o acesso às memórias dedicadas `BRAM_TMP` e `BRAM_IMG`, respectivamente e disponibilizar ao coprocessador os dados, no momento correto. O processo de acesso em leitura e escrita às memórias é sincronizado pelo sinal do clock (`CLK`) e pelo sinal de sincronismo (`CLK_sync`). Além dessa função principal, esses controladores de memória também permitem que o processador acesse as memórias dedicadas.

4.4 Considerações Finais do Capítulo

Neste Capítulo foi apresentada a placa *Smart Vision Development Kit*, plataforma portátil sobre o qual o projeto foi implementado. O sistema proposto foi apresentado em nível macro (de blocos funcionais) e consiste de um projeto integrado de software/hardware que utiliza um coprocessador dedicado, para cálculo do coeficiente de correlação, e execução da técnica PSO *global best* em software. Ademais, foi realizada uma análise inicial do projeto em termos de tempo de processamento. Foi possível identificar que o cálculo de PCC é a parte mais custosa do algoritmo, seguido pelo acesso a memória. Estes resultados motivaram a utilização do coprocessador dedicado e de uma memória local a esse coprocessador, em um sistema do tipo co-design.

O Capítulo seguinte descreve a arquitetura do hardware desenvolvido em nível micro (de unidades lógicas).

Capítulo 5

MICRO-ARQUITETURA

NO Capítulo anterior, o projeto para rastreamento de objetos por *template matching* e correlação foi proposto em uma plataforma portátil via um sistema integrado de software/hardware. A arquitetura do hardware foi descrita de forma macro, no nível de seus blocos funcionais.

Este Capítulo tem por objetivo descrever a arquitetura do hardware desenvolvido em nível micro. Os blocos funcionais, elaborados em VHDL, são apresentados no nível de unidades lógicas. A representação numérica utilizada foi a de ponto fixo, no qual o tamanho da parte fracionária varia de acordo a necessidade de cada unidade. Todos os componentes consideram o clock de subida.

Para tal, a Seção 5.1 apresenta e descreve os componentes que formam o Bloco 1. Na Seção 5.2, os componentes do Bloco 2 também são descritos. Também é realizada uma análise mais detalhada do componente `multCLK`, e de sua capacidade de realizar multiplicação em um único ciclo de clock. A Seção 5.3 apresenta e descreve os componentes que formam o Bloco 3, com ênfase em `DIV_fracA2` e `SQRT`. Uma análise detalhada da divisão e da raiz quadrada implementadas em circuitos digitais também é realizada. Na Seção 5.4, o componente `Sincro` e sua máquina de estados finitos associada são descritos. Por fim, na Seção 5.5, são realizadas considerações finais.

5.1 Micro-arquitetura do Bloco 1

O bloco 1, Figura 20, é formado pelos componentes `media` e `regN_bits`. Possui dados de entrada de 8 bits inteiros sem sinal, que representam a intensidade dos pixels das imagens (`dado_p` e `dado_a`). As saídas (*dado_pmedia* e *dado_amedia*) são sinais com 20 bits, sendo 8 bits mais significativos para parte inteira e 12 bits menos significativos para a parte

fracionária. O componente `regN_bits` é um simples registrador de 20 bits, com enable síncrono e clear assíncrono. O componente `media`, ilustrado na Figura 23, é composto por uma acumulador (que realiza as somas dos pixels) e um contador que desabilitada o acumulador após 4096 pulsos de clock. Esse valor equivale ao total de pixels do *template* (64x64).

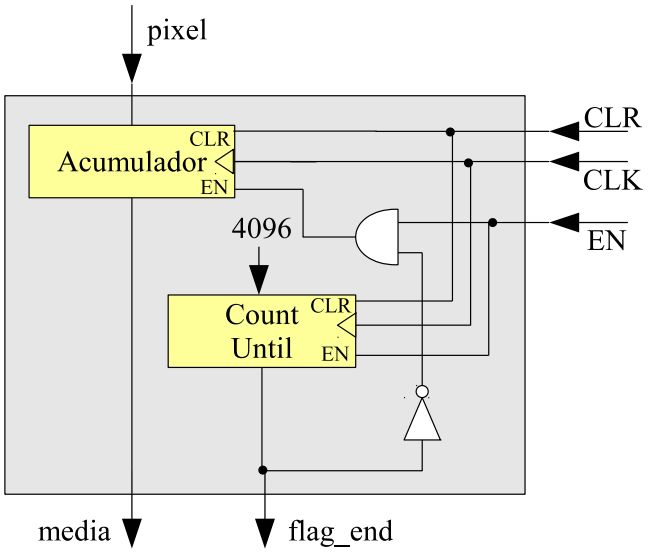


Figura 23: Componentes *media* e *soma A2*

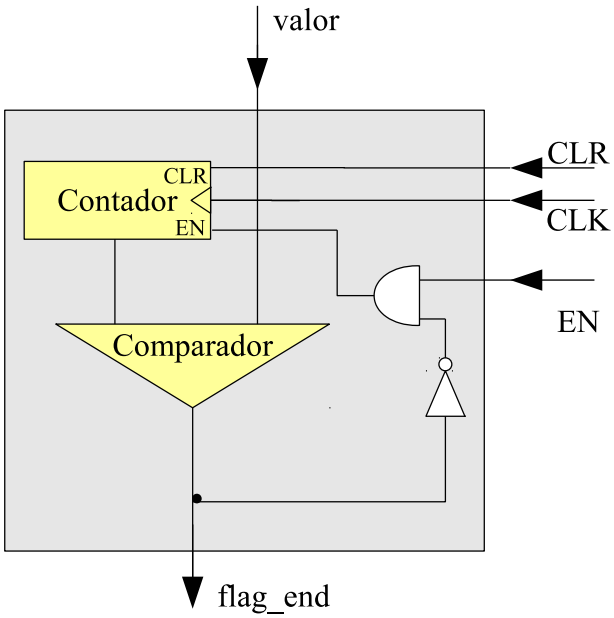


Figura 24: Componente *count until*

Após a soma dos pixels, é necessário dividir pelo total para achar a média. Dividir por 4096 equivale a deslocar a saída do acumulador em 12 bits para esquerda. Se considerarmos que os 12 bits menos significativos de saída são equivalentes a parte fracionária, nenhuma operação precisa ser realizada.

O componente `count_until`, Figura 24, é basicamente um contador de pulsos de clock que tem a contagem desabilitada quando atinge o *valor* especificado como entrada.

5.2 Micro-arquitetura do Bloco 2

O bloco 2, Figura 21, é formado pelos componentes `subt_A2`, `multCLK`, `somaA2` e `regN_bits`. Possui dados de entrada de 8 bits (`dado_p` e `dado_a` referentes aos pixels) e de 20 bits (*dado_pmedia* e *dado_amedia* referentes às médias calculadas no bloco 1). As saídas *num*, *sum_p2* e *sum_a2* representam números de 42 bits, em complemento A2. O bit mais significativo indica o sinal, os demais 29 bits mais significativos caracterizam a parte inteira e os 12 bits menos significativos caracterizam a parte fracionária.

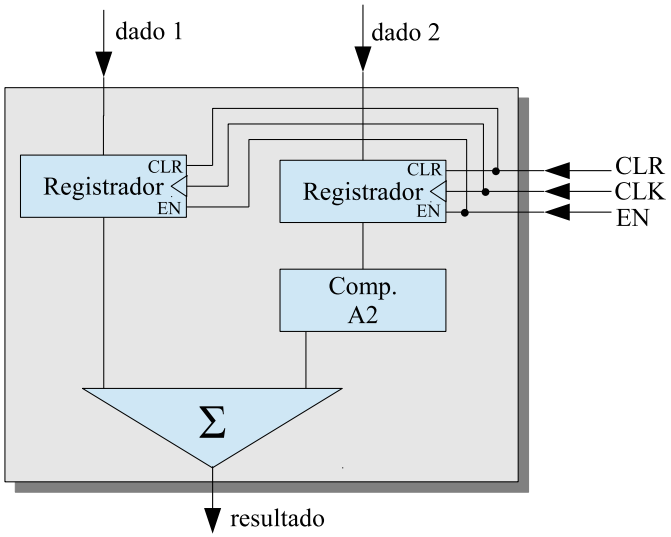


Figura 25: Componente *subt A2*

O componente `subt_A2`, ilustrado na Figura 25, realiza a subtração de dois números positivos de 20 bits, somando o primeiro com o complemento A2 do segundo. O componente `multCLK` é descrito em detalhes na Seção 5.2.2. Suas entradas são sinais de 21 bits, em complemento A2, cujo bit mais significativo indica o sinal, os demais 8 bits mais significativos caracterizam a parte inteira e os 12 bits menos significativos caracterizam a parte fracionária. A saída possui esse mesmo formato com 17 bits para parte inteira

e 12 bits para a parte fracionária. O componente `somaA2` realiza a soma de cada uma das multiplicações oriundas dos componentes `multCLK`. Apresenta a mesma estrutura do componente `media`, Figura 23, mas possui dado de entrada de 30 bits e saída com 42 bits.

5.2.1 Método de Multiplicação

Analisando a Equação 4, observa-se que as operações mais realizadas no cálculo da correlação são somatórios, subtrações e multiplicações. Somatórios e subtrações são facilmente implementadas em hardware e necessitam de somente um ciclo de clock para serem realizadas. Para que o cálculo da correlação, proposto em hardware, proporcione melhores resultados em termos de tempo de processamento, é imprescindível obter um método capaz de multiplicar dois números em um ciclo de clock.

A maneira mais fácil de realizar a multiplicação de dois números é repetir um deles (o multiplicando) pelo número de vezes do outro (o multiplicador). Quando implementado em hardware, esse processo pode se tornar muito lento pois o número de repetições dependerá do valor do multiplicador.

O método utilizado nesta dissertação aproveita a propriedade distributiva da multiplicação com características inerentes do sistema binário. Quando representados em binário, os números são compostos por múltiplos de 2. Por exemplo, $5 = 2^2 + 2^0 = 101_b$. Utilizando a propriedade distributiva, $9 \times 5 = 9 \times (2^2 + 2^0) = 9 \times 2^2 + 9 \times 2^0 = 36 + 9 = 45$. Por outro lado, multiplicar e dividir um número por múltiplos de 2, na base binária, equivale a andar com a vírgula (ou deslocar o número) para a esquerda e para direita, respectivamente.

A Figura 26 ilustra a multiplicação de 9×5, em binário, utilizando essas propriedades. Ou seja, multiplicar um número por outro equivale a realizar a soma de todas as multiplicações desse número por múltiplos de 2, que descrevem o multiplicador.

5.2.2 Multiplicação em um Pulso de Clock

Considere-se dois números binários de N bits e um circuito que é capaz multiplicar um desses números (o multiplicando) por 2^0, 2^1, 2^2 até 2^{N-1}. Se cada um dos N bits do outro número (o multiplicador) servirem para habilitar ou não tais multiplicações, ao se realizar a soma de todas as multiplicações, obtém-se o produto dos dois números.

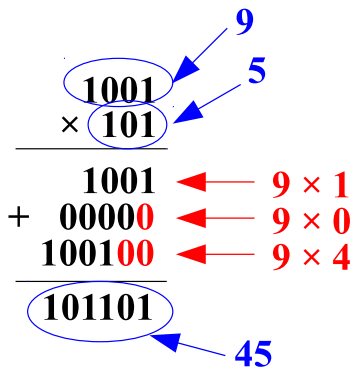


Figura 26: Multiplicação em binário

Baseado nessa ideia, o circuito da Figura 27 foi desenvolvido. Ele realiza, combinacionalmente, a multiplicação de dois números inteiros positivos. Para considerar números racionais com G bits nas casas fracionárias, é necessário transformá-los primeiramente para positivo, e também incluir componentes que realizem a multiplicação por 2^{-1}, 2^{-2} até 2^{-G}, para cada casa fracionária do multiplicador. Após a multiplicação, o resultado é convertido para positivo ou negativo, conforme os sinais dos números de entrada. O componente `multCLK`, Figura 28, foi desenvolvido dessa forma.

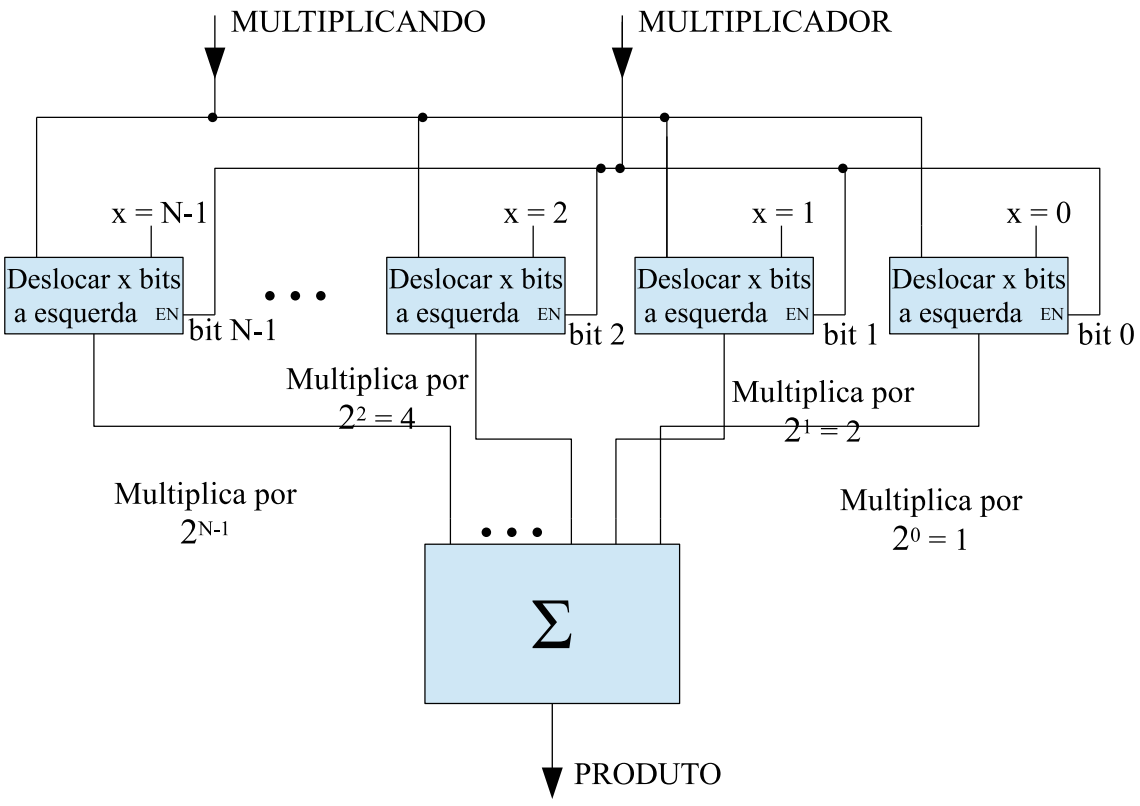


Figura 27: Circuito para multiplicar dois números inteiros positivos

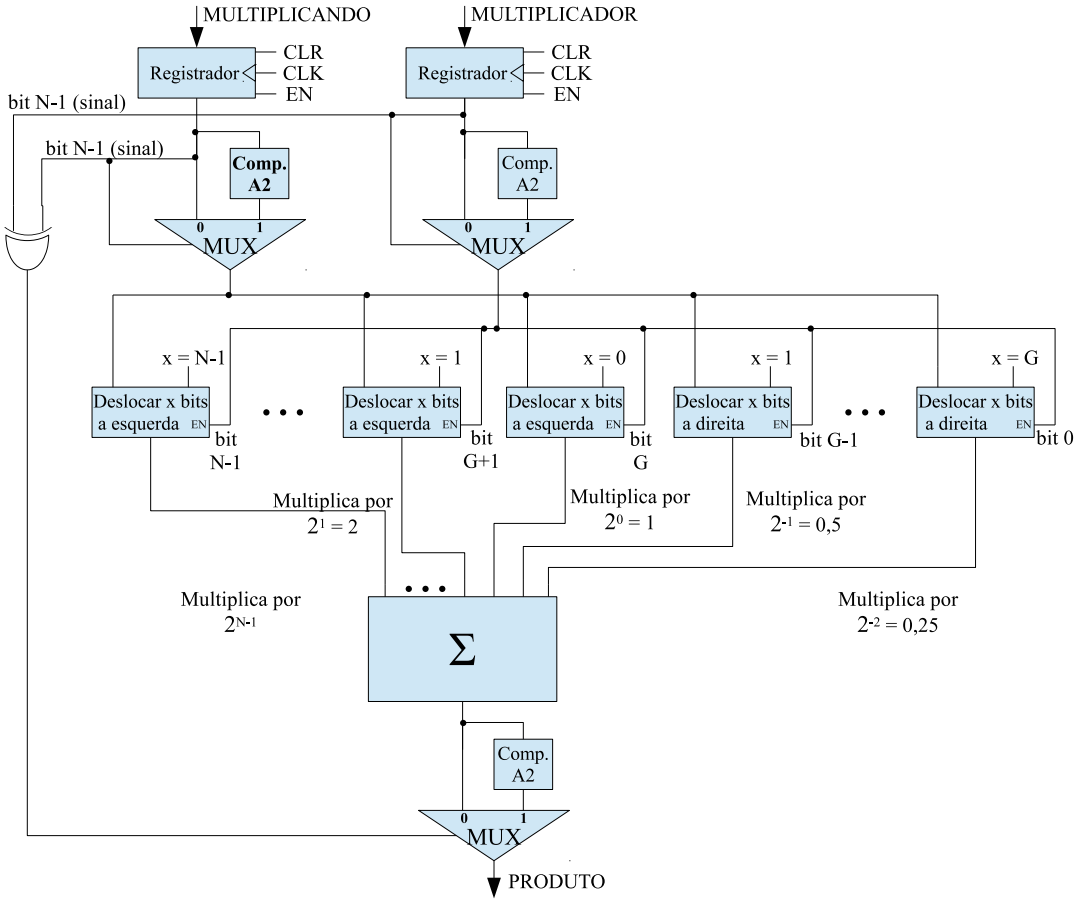


Figura 28: Componente *multCLK*

5.3 Micro-arquitetura do Bloco 3

O bloco 3, Figura 22, é formado pelos componentes `multCLK42`, `SQRT`, `div_fracA2` e `regN_bits`. Ele também possui uma máquina de estados que controla a operação de seus componentes. Possui dados de entrada de 42 bits (*num*, *sum_p2* e *sum_a2*) referentes às saídas do bloco 2. A saída do bloco 3 é o resultado final da correlação. É representado por um número de 32 bits onde o bit mais significativo indica o sinal, os demais 7 bits mais significativos caracterizam a parte inteira e os 24 bits menos significativos caracterizam a parte fracionária. Também apresenta como saída o bit $flag\_end$, que indica término do cálculo, e o bit *error*, que indica que o resultado não é válido e foi oriundo de uma divisão por zero.

O componente `multCLK42` é idêntico ao componente `multCLK` do bloco 2 mas possui dados de entrada de 42 bits (1 sinal + 29 inteiro + 12 fracionário) e saída com 72 bits (1 sinal + 59 inteiro + 12 fracionário). O componente `SQRT` é descrito em detalhes na Seção

5.3.2. Sua entrada é um sinal de 72 bits, em complemento A2, cujo bit mais significativo indica o sinal, os demais 59 bits mais significativos caracterizam a parte inteira e os 12 bits menos significativos caracterizam a parte fracionária. A saída corresponde a raiz quadrada da entrada e possui a mesma estrutura do sinal de entrada. Também apresenta como saída o bit *fim_sqrt*, que indica término do cálculo da raiz quadrada. O componente `div_fracA2` é descrito em detalhes na Seção 5.3.1. Sua entrada possui a mesma estrutura dos sinais de `SQRT` e as saídas equivalem aos sinais de saída *resultado* e *error* do próprio bloco 3. Também apresenta como saída o bit *fim_div*, que indica término da divisão.

O componente `FSM-3B` exerce um papel importantíssimo no bloco 3. Ele é o responsável pelo controle de cada um dos seus componentes de maneira que os cálculos sejam realizados nos momentos corretos, com os dados corretos. Ele monitora os sinais *clear*, *fim_sqrt* e *fim_div* e é basicamente composto por uma máquina de estados do tipo Moore. Uma máquina de estados finitos (*Finite State Machine* - FSM) é usada para modelar sistemas que variam entre um número finito de estados internos. As transições dependem do estado atual e de entradas externas. Uma das principais aplicações para uma FSM é atuar no controle de extensos sistemas digitais que dependem de sinais e comandos externos para controlar operações na via de dados (CHU, 2008). Um diagrama básico de uma FSM está exibido na Figura 29 e consiste de circuitos sequenciais regulares. É composto por um registrador de estado, uma lógica para o próximo estado e uma lógica para saída. As saídas Moore e Mealy são similares porém não são iguais. Uma FSM é dita do tipo Moore se as saídas dependem somente do estado que a máquina se encontra. Uma FSM é dita do tipo Mealy se suas saídas dependem do estado e de entradas externas (CHU, 2008).

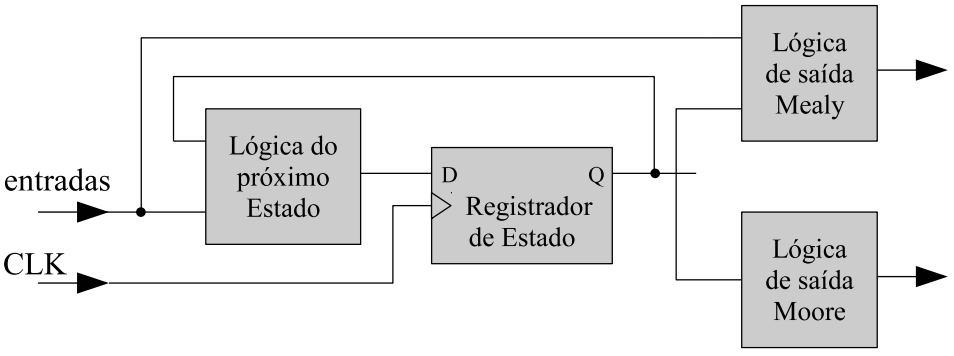


Figura 29: Diagrama em Blocos de uma Máquina de Estados Finitos

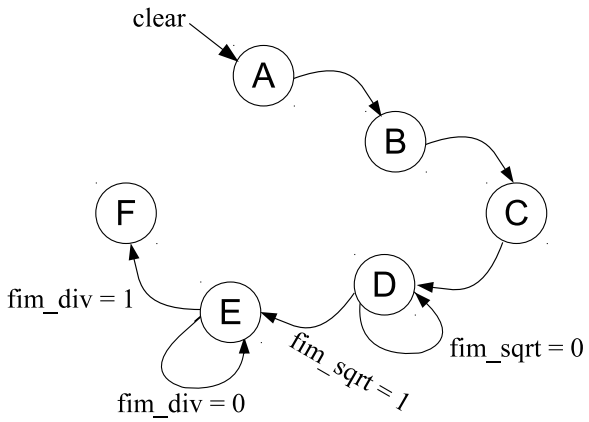


Figura 30: Diagrama de Estados da máquina FSM-3B

Tabela 9: Saídas associadas aos estados de FSM-3B

SAÍDAS	ESTADOS					
	A	B	C	D	E	F
CLR_reg	1	0	0	0	0	0
CLR_mult	1	0	0	0	0	0
CLR_sqrt	1	1	1	0	0	0
CLR_div	1	1	1	1	0	0
EN_reg	0	1	0	0	0	0
EN_mult	0	1	0	0	0	0
flag_end	0	0	0	0	0	1

A Figura 30 exibe o diagrama de estados e a Tabela 9 exibe as saídas associadas a cada estado da máquina `FSM-3B`. Inicialmente todos os componentes estão desabilitados. No estado B, o registrador do sinal *num* e o componente `multCLK` carregam suas entradas. No estado C, esses dois componentes são desabilitados e armazenam seus dados e cálculos. No estado D, o componente `SQRT` é então liberado para iniciar o cálculo da raiz quadrada. A máquina monitora o sinal *fim_sqrt* a fim de identificar o fim do cálculo e passar para o próximo estado. No estado E, o componente `div_fracA2` é liberado para executar a divisão. Quando o sinal *fim_div* vai para '1', indicando fim da divisão, a máquina passa para o último estado e seta o bit *flag_end*, indicando o final das tarefas do bloco 3 e carregando os registradores de saída. A máquina retorna ao estado inicial A com sinal *clear*.

5.3.1 Divisão em Hardware

Algoritmos de divisão podem ser divididos em duas classes, de acordo com seu operador iterativo. Na primeira classe esse operador é a subtração e os algoritmos são relativamente

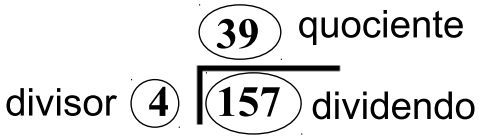
lentos. O método mais simples de se realizar uma divisão é realizar subtrações sucessivas do dividendo pelo divisor até que o resto seja menor que o divisor. A cada subtração, o quociente é incrementado. Este método é lento e o tempo de execução depende do tamanho dos operandos. Quanto maior for o dividendo e menor for o divisor, um maior número de subtrações deverão ser executadas e maior tempo é consumido.

A segunda classe é baseada na multiplicação e converge mais rapidamente. Apesar disso, são mais complexos e muita das vezes apresentam resultados por aproximações numéricas. Um exemplo é o método de Newton-Raphson.

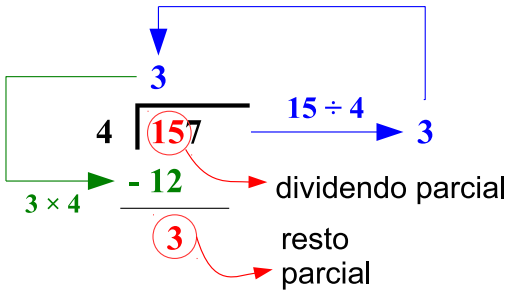
O método de divisão utilizado nesta dissertação é aquele difundido no ensino básico, baseado na subtração. Apesar de ser lento, esse método é de fácil implementação. A cada iteração, um dígito do quociente é calculado. O tempo máximo gasto é proporcional ao número de algarismos do dividendo. A Figura 31(a) ilustra a divisão de 157 por 4. Na Figura 31(b), o valor 15 é separado da parte mais significativa do dividendo. Esse valor é chamado de dividendo parcial, deve ter o mínimo de dígitos possível e deve ser maior ou igual a 4 (o divisor). Dividi-se então o dividendo parcial pelo divisor, resultando em 3. Esse resultado é o primeiro dígito do quociente. O número 3 então é multiplicado pelo divisor resultando no número 12. Para encontrar o resto parcial, o valor 15 inicialmente separado é então subtraído por 12, resultando em 3. Esse é o resto parcial. A divisão só acaba quando todos os dígitos são considerados. Assim, o último algarismo, o número 7, é concatenado ao resto parcial, Figura 31(c). Dividi-se 37 por 4 resultando em 9. O número 9 é o segundo dígito encontrado para o quociente. Multiplicando-se 9 por 4 encontra-se 36. O resto parcial anterior subtraído de 36 gera o novo resto, o valor 1. Como não há mais dígitos, esse é o resto final e a divisão inteira termina.

Para números racionais (aqueles que possuem vírgula), basta que as casas decimais sejam igualadas antes de proceder à divisão. Realiza-se os mesmos passos da divisão inteira considerando os dois números completos, como se não houvesse separação entre a parte inteira e a parte fracionária. Caso se deseje obter um quociente com casas decimais, as operações continuam até que se encontre resto zero. Para tal, os restos parciais são completados com zero após o término da divisão inteira. A Figura 32(a) ilustra esse processo para a divisão de 157 por 4 na base decimal.

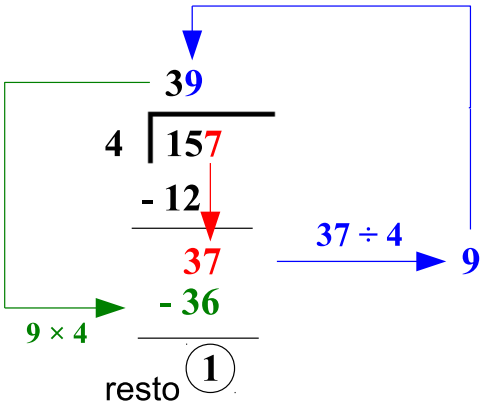
Na base binária, o princípio de divisão é o mesmo. As Figuras 33 e 32(b) ilustram os mesmos passos executados para base decimal, nessa base. Observa-se que cada dígito



(a) Divisão de 157 por 4



(b) Primeiros dividendo e resto parciais

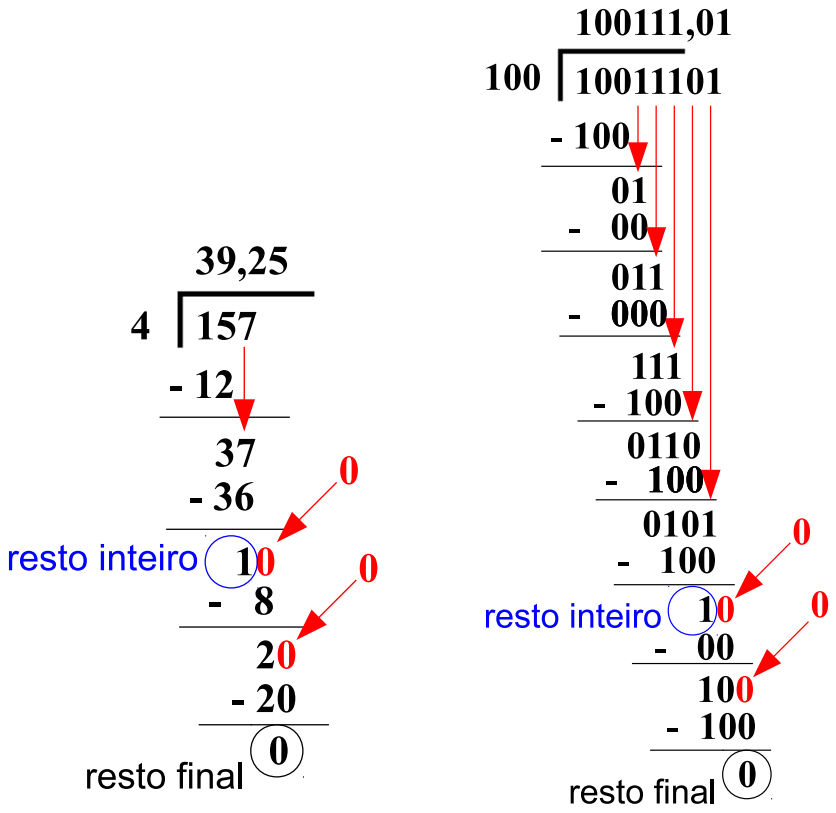


(c) Quociente e resto finais

Figura 31: Ilustração da divisão inteira de 157 por 4

do quociente é oriundo de uma comparação. Como a base é binária, o quociente só pode assumir dois valores: '0' ou '1'. Se o dividendo parcial for maior que o divisor, o dígito do quociente é '1', caso contrário, o dígito assume valor '0'. Para obtenção do resto parcial, é necessário subtrair do dividendo parcial, o valor do divisor multiplicado pelo dígito encontrado no quociente. Observa-se também, na Figura 33(b), que o valor a ser subtraído é sempre o próprio divisor (se o quociente for '1') ou zero (se o quociente for '0'). Essas duas características, inerentes ao sistema binário, facilitam a execução desta metodologia e também facilitam a sua implementação em hardware.

Considere-se um dividendo com N bits e um registrador com o dobro de bits. Os N bits mais significativos representam o dividendo parcial e os N bits menos significativos representam o restante do dividendo. A cada passo da divisão, a metade mais significativa é atualizada com o resto parcial e o registrador é deslocado para esquerda, a fim de concatenar o próximo dígito a ser considerado. Quando todos os bits do dividendo forem considerados (deslocados), a divisão termina e o resto final será o valor armazenado na parte MSB desse registrador. Baseado nessa ideia, o componente `DIV_resto` da Figura 34 foi elaborado. Ele é o responsável por gerar os dividendos parciais e restos da divisão. Possui um multiplex que atualiza os bits do dividendo parcial (mais significativos do

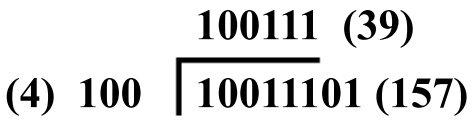


(a) Base decimal (b) Base binária

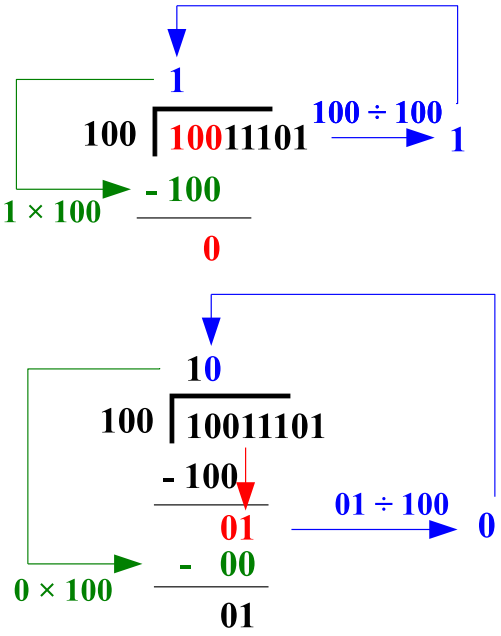
Figura 32: Ilustração da divisão fracionária de 157 por 4

registrador) conforme comparação do divisor com o próprio dividendo parcial. Se o dividendo parcial for maior ou igual ao divisor, a metade mais significativa do registrador é carregada com a subtração desse dividendo com divisor. Se o dividendo parcial for menor que o divisor, nenhum operação matemática é realizada e a metade menos significativa do registrador é carregada com o próprio dividendo parcial. O processo se repete até que todos os bits do dividendo sejam considerados. Esse momento é indicado pelo sinal *flag_end*, que trava o registrador por meio de sua entrada *enable*.

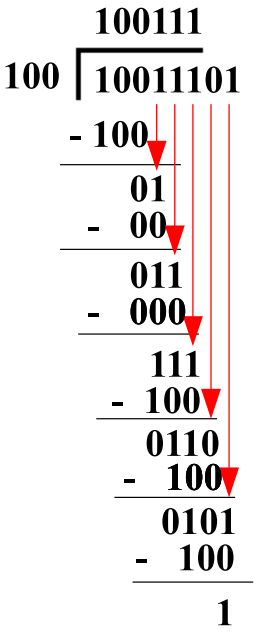
Para formação do quociente, o componente `DIV_quociente` foi desenvolvido e utiliza um registrador de N bits. A cada passo da divisão (representado por cada ciclo de clock), o registrador é deslocado a esquerda e carregado por meio do multiplexador. Se o dividendo parcial for maior ou igual ao divisor, o bit menos significativo do quociente é setado. Se o dividendo parcial for menor que o divisor, o bit menos significativo do quociente permanece zerado. Da mesma forma que no componente `DIV_resto`, o processo se repete até que todos os bits do dividendo sejam considerados. Esse momento é indicado por meio do sinal *flag_end*, que trava o registrador por meio de sua entrada *enable*.



(a) Divisão de 157 por 4



(b) Dois primeiros restos parciais



(c) Divisão inteira completa

Figura 33: Ilustração da divisão inteira de 157 por 4 em base binária

Para executar a divisão inteira, ainda resta considerar a preparação do dividendo para iniciar o processo. Conforme explicado anteriormente, o primeiro dividendo parcial precisa ter o mínimo de dígitos possível e deve ser maior ou igual ao divisor. Para tal, o componente `prep_DIV`, da Figura 34, foi elaborado. Ele possui um registrador de 2N bits que armazena o dividendo em seus N bits mais significativos e seus N bits menos significativos são zerados. O componente realiza deslocamentos a direita sucessivos nesse registrador até sua metade mais significativa seja menor ou igual ao divisor. Esse bits mais significativos irão compor o primeiro dividendo parcial quando forem carregados no registrador de `DIV_resto` e sofrerem o primeiro deslocamento a esquerda. O componente `prep_DIV`, ao preparar o dividendo, também realiza uma contagem dos deslocamentos efetuados. Esse valor é equivalente ao número de passos (dígitos) a serem considerados na divisão. Ele é carregado no componente `count_until`, responsável por gerar o sinal

flag_end. A Figura 34 exibe o diagrama do componente para realizar a divisão inteira de dois números racionais positivos. Os passos desta divisão podem ser ordenados conforme o Algoritmo 2. Um ponto importante a ser observado é que os componentes `DIV_resto` e `DIV_quociente` somente são liberados para operação quando o sinal *clear* é zerado, oriundo do sinal *flag_prep*, que indica fim da preparação do dividendo. O tempo máximo para executar a divisão é fixo e equivale a duas vezes o número de bits do dividendo.

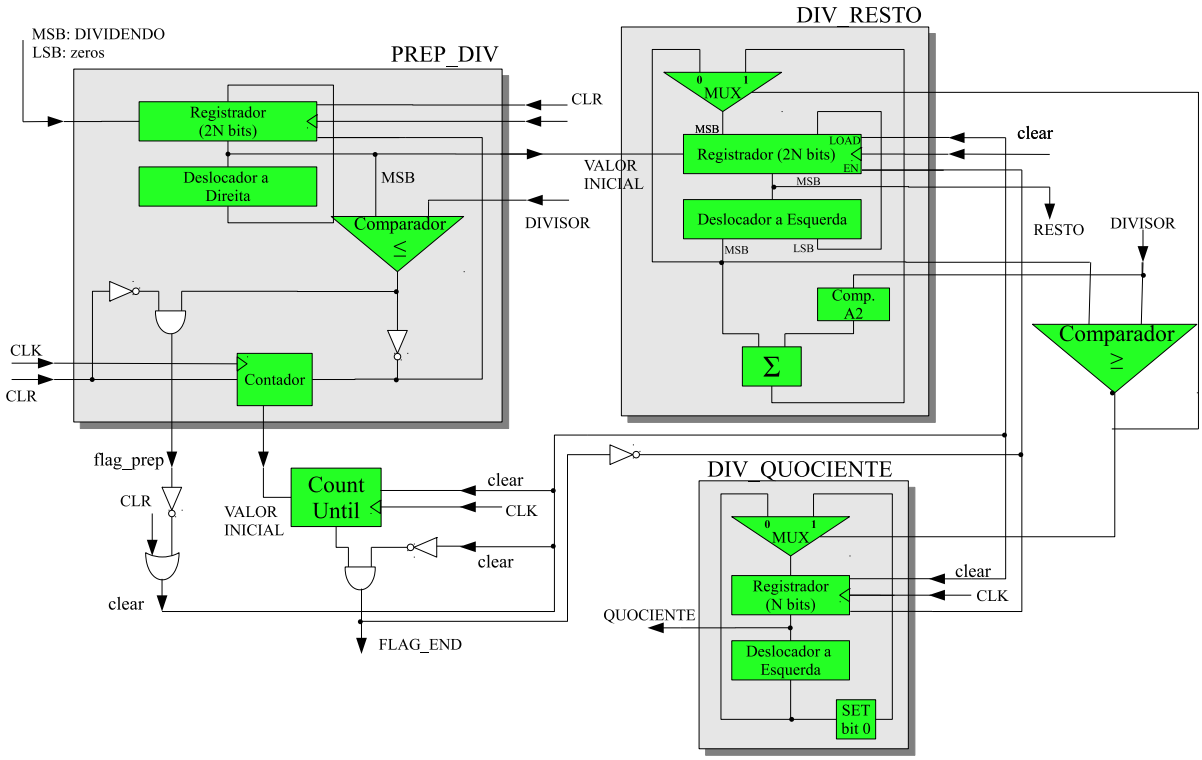


Figura 34: Diagrama do componente *DIV_inteiro*

Para obter números com dígitos fracionários (após a vírgula), foi elaborado o componente `DIV_frac`, Figura 35. Como exposto anteriormente, após a divisão inteira, os mesmos passos continuam a serem executados até que se encontre resto zero (ou até a precisão desejada). Para tal, os restos parciais são completados com zero a cada ciclo de clock. Utiliza-se o componente `DIV_inteiro` para se obter o resto inteiro e o quociente inteiro e os mesmos componentes `DIV_resto` e `DIV_quociente`. Os passos se repetem por 24 ciclos de clock, que equivalem a 24 dígitos após a vírgula. Essa é a precisão do quociente para esse componente. Seu funcionamento é definido conforme o Algoritmo 1.

Finalmente, o componente `DIV_fracA2` foi elaborado para tratar números racionais positivos e negativos. Ele trabalha com entradas e saídas em complemento A2 e utiliza o

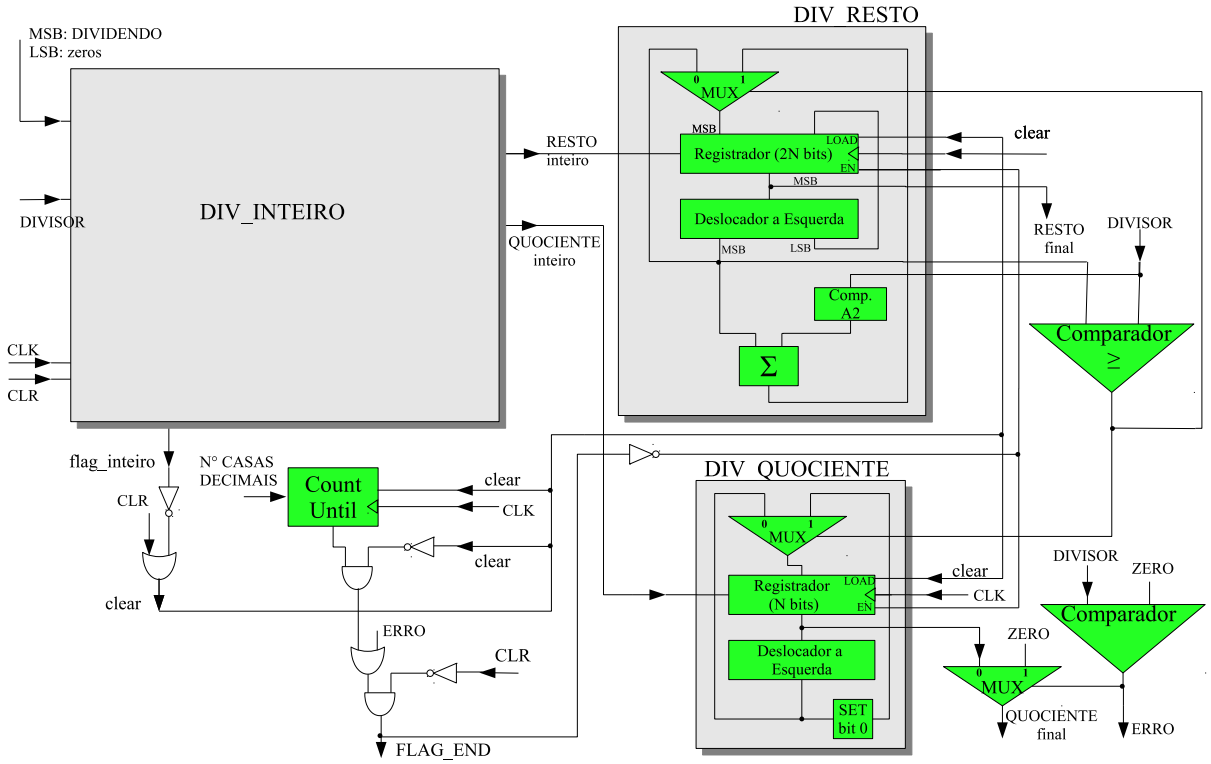


Figura 35: Diagrama do componente *DIV_frac*

componente `DIV_frac`. Ele basicamente transforma os números de entrada em positivos e conforme os sinais desses números, a saída é convertida para negativo ou não.

5.3.2 Raiz Quadrada

Para cálculo da raiz quadrada do bloco 3, o método numérico *babylonian* foi empregado. O método possui rápida convergência e utiliza um processo iterativo de M passos com base na Equação 11:

$$x_{i+1} = 0,5 \times (x_i + \frac{S}{x_i}), \tag{11}$$

onde x é o radical, S é o radicando ($x = \sqrt{S}$), x_i é o valor de x na iteração atual e x_{i+1} é o valor da próxima iteração. O número de iterações M é definido de acordo com a precisão do resultado requerida.

O método basicamente consiste de uma operação de divisão, uma de soma e uma divisão por dois. A Figura 36 exibe o componente `SQRT`, elaborado para executar o método em hardware. Possui um registrador que armazena o valor de x a cada iteração. O componente `DIV_frac mod` realiza a divisão do radicando pelo valor de x. Esse componente é uma versão modificada do componente `DIV_frac`. A dificuldade de utilizar o componente `DIV_frac` diretamente é que ele produz quocientes maiores que suas entradas, sobretudo

Algorithm 2 Divisão inteira $Q = A/B$

$N_B := 0$
$RA_{\langle N-1 \ldots \frac{N}{2} \rangle} :=$ A
$RA_{\langle \frac{N}{2}-1 \ldots 0 \rangle} := 0$
enquanto $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} > B$ **faça**
  $RA :=$ deslocar RA para direita
  $N_B := N_B + 1$
fim enquanto
$Q := 0$
para $j = 1, N_B$ **faça**
  $RA :=$ deslocar RA para esquerda
  $Q :=$ deslocar Q para esquerda
  se $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} > B$ **então**
    $RA_{\langle N-1 \ldots \frac{N}{2} \rangle} := RA_{\langle N-1 \ldots \frac{N}{2} \rangle} - B$
    $Q_0 := 1$
  senão
    $Q_0 := 0$
  fim se
fim para

por causa dos bits gerados pela parte fracionária. Devido à característica da Equação 11, o tamanho de x_{i+1} será sempre menor que x_i. Dessa forma, o componente `DIV_frac` pôde ser modificado para gerar uma saída com o mesmo tamanho das entradas.

O componente `SQRT` também possui um somador e um deslocador a direita. Dividir por dois, em binário, equivale a deslocar o número em um bit para direita. O componente `count_until` tem a função de indicar que o cálculo da raiz terminou. A cada iteração (ou ciclo de clock), seu contador é incrementado até que atinja um valor pré-definido. Neste momento, o sinal *fim_sqrt* é setado, juntamente com o *flag_end* de saída. Empiricamente, chegou-se a conclusão de que um valor inicial de 8 no registrador e 35 iterações atingem bons resultados com rápida convergência.

O comparador e o multiplexador tem o papel de lidar com radicando igual a zero, caso contrário o sistema ficaria em loop infinito (divisão por zero). Eles zeram a saída e ativam o *flag_end* de término.

Todo o processo é controlado pela máquina de estados finitos `FSM-SR`. É uma máquina do tipo Moore que controla o registrador, o contador e o componente `DIV_frac mod`. A Figura 37 exibe o diagrama de estados e a Tabela 10 exibe as saídas associadas a cada estado da máquina. Inicialmente, todos os componentes são reiniciados (*CLR_reg*, *CLR_div* e *CLR_count* em '1'). No estado B, os componentes ainda continuam desabi-

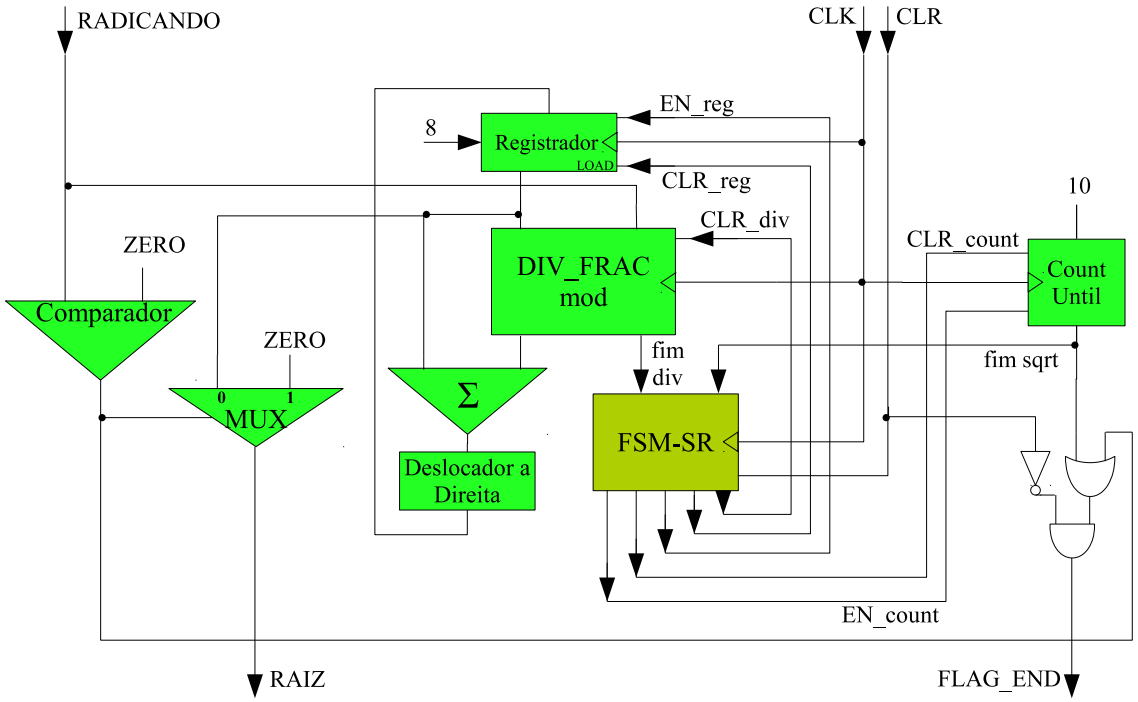


Figura 36: SQRT

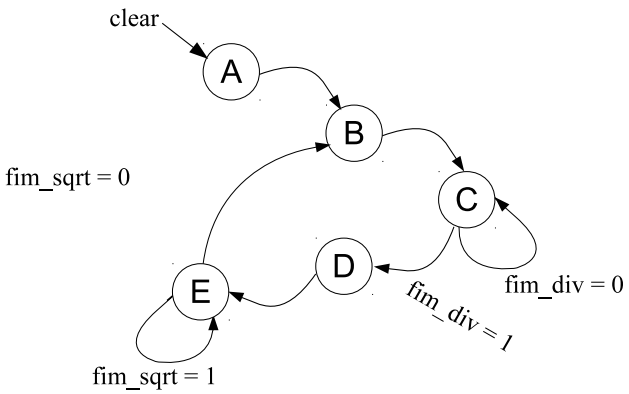


Figura 37: Diagrama de Estados da máquina FSM-SR

Tabela 10: Saídas associadas aos estados de FSM-SR

SAÍDAS	ESTADOS				
	A	B	C	D	E
CLR_reg	1	0	0	0	0
CLR_div	1	1	0	0	0
EN_reg	0	0	0	1	0
CLR_count	1	0	0	0	0
EN_count	0	0	0	1	0

litados (*EN_reg* e *EN_count* em '0') e a divisão ainda está reiniciada (*CLR_div* em '1'). No estado C, libera-se o cálculo para divisão (*CLR_div* em '0'). A máquina monitora o término da divisão por meio do sinal *fim_div*. Quando a divisão termina, a máquina vai para o estado D, onde o registrador e o contador são habilitados a carregar o novo valor de x e ser incrementado (*EN_reg* e *EN_count* em '1'), respectivamente. Chega-se então ao estado E onde o término do processo de radiciação é verificado. Caso o sinal fim_sqrt esteja em '1', o processo termina e fica em loop infinito no estado E. Caso contrário, a máquina retorna ao estado B e o processo de divisão é reiniciado.

5.4 Micro-arquitetura do Componente Sincro

O componente `Sincro`, Figura 38, tem a função de gerar o pulso de sincronismo para coordenação dos estágios do coprocessador. Ele é composto por um contador e por uma máquina de estados finitos do tipo Moore. A FSM-SY possui uma única saída, sinal *sync*, e monitora o flag do contador.

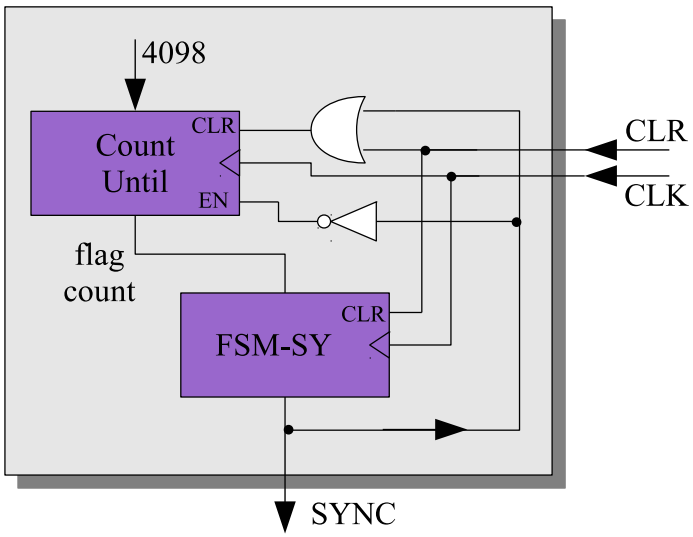


Figura 38: Componente Sincro

Tabela 11: Saídas associadas aos estados de FSM-SY

SAÍDA	ESTADOS				
	A	B	C	D	E
sync	0	1	1	1	1

O sinal *sync* permanece em zero até que o flag do contador seja setado, após 4098 pulsos de clock. A máquina então mantém o sinal sync setado por quatro estados (ou

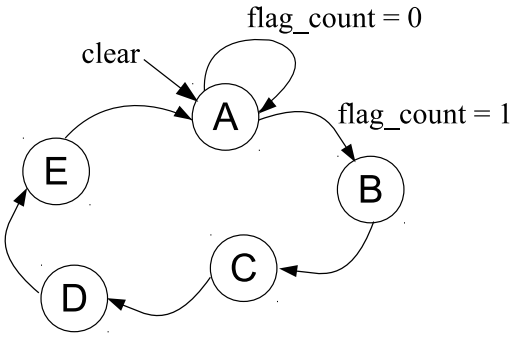


Figura 39: Diagrama de Estados da máquina FSM-SY

quatro períodos de clock). A Figura 39 exibe o diagrama de estados e a Tabela 11 exibe as saídas associadas a cada estado da FSM-SY.

O valor de 4098 pulsos de clock está associado ao tempo que o estágio mais lento (bloco 2) necessita para executar suas operações. Esse valor é a soma do processamento dos 4096 pixels de cada imagem, recebidos em cada clock, com os 2 ciclos necessários para que o estágio fique completo. O período total do sinal *sync* é de 4103 ciclos de clock.

5.5 Considerações Finais do Capítulo

Neste Capítulo, a arquitetura de hardware proposta foi descrita de forma micro, no nível de suas unidades lógicas. Os componentes que formam o coprocessador foram descritos, bem como detalhados seus funcionamentos. O funcionamento do componente `multCLK`, e sua capacidade de multiplicação em um ciclo de clock foram detalhados. Tal capacidade foi de vital importância para o desenvolvimento coprocessador, da forma como foi elaborado e implementado. Também foi dada ênfase a implementação dos componentes `DIV_fracA2` e `SQRT` que realizam as operações de divisão e radiciação em hardware, sem precisar de controle de software.

Capítulo 6

ANÁLISE DOS RESULTADOS

NESTE Capítulo são apresentados e analisados os resultados obtidos pela implementação do projeto proposto na plataforma portátil SVDK, descrito nos Capítulos 4 e 5.

A Seção 6.1 apresenta as métricas utilizadas no Capítulo. A Seção 6.2 expõe a forma como o PSO foi configurado e a escolha dos seus parâmetros. A Seção 6.3 avalia a qualidade do cálculo da correlação efetuado pelo coprocessador. A Seção 6.4 apresenta os resultados em termos de tempo de processamento, *speedup* e confiabilidade. Esta Seção também apresenta as imagens utilizadas como referência para os testes, bem como suas características e peculiaridades. A Seção 6.6 apresenta o resultado da implementação do sistema na placa SVDK, bem como as limitações observadas. Por fim, a Seção 6.7, realiza considerações finais.

6.1 Métricas Utilizadas

Com propósito de analisar os resultados, foram utilizados os medidores de desempenho citados na Seção 3.2.3.

A acurácia (A), Equação 8, equivale a qualidade da solução encontrada pelo PSO. Seu valor é calculado pela subtração do valor ótimo de PCC ($1,000$) com o melhor valor encontrado pelo PSO. Considerando que foi estabelecido o valor de 0,95 como um dos critérios de parada (além do número máximo de iterações), a acurácia ficou fixada em 0,05. Ou seja, todos os rastreamentos que convergirem, possuirão, obrigatoriamente, acurácia menor ou igual a 0,05.

A confiabilidade (C), Equação 9, equivale ao grau de convergência do algoritmo para uma solução com determinada qualidade. Seu valor é calculado pela divisão do

número de rastreamentos que atingiram acurácia menor ou igual a 0,05 pelo número de rastreamentos total. O resultado é expresso em termos de porcentagem.

A eficiência (E) se refere ao tempo necessário para obter a solução desejada. Ela foi computada, na análise, como o tempo de processamento consumido por um rastreamento do PSO. Esta medida está ligada diretamente a velocidade do processador e do coprocessador.

Além desses, o *speedup* também foi utilizado como medidor de desempenho. A lei de Amdahl define o *speedup* (S) como o ganho em desempenho que pode ser obtido ao melhorar determinada tarefa computacional, conforme Equação 12:

$$Speedup(S) = \frac{\text{tempo de execução da operação sem usar a melhoria M}}{\text{tempo de execução da operação usando a melhoria M}}. \quad (12)$$

Para efeitos de análise de resultados, foram consideradas as seguintes melhorias: HS (cálculo da correlação em hardware de modo sequencial) e HP (cálculo da correlação em hardware em modo *pipeline*).

6.2 Escolha dos Parâmetros do PSO

Com o objetivo de avaliar a qualidade e o desempenho do sistema proposto, foram utilizadas nove imagens de referência (*benchmarks*), oriundas de (COLLINS; ZHOU; TEH, 2005). Essas imagens, denominadas *pickup*, *truck*, *redcar*, *hollywood*, *carros*, *sedan*, *IRcar1*, *IRcar2* e *IRcar3* serão detalhadas na Seção 6.4. Para o PSO, essas imagens correspondem a função a ser otimizada, cujos valores variam de -1 a 1.

Inicialmente, o PSO foi configurado de forma empírica objetivando obter resultados aceitáveis. Esta configuração foi a seguinte:

- $PSO_{inicial}$: enxame de 20 partículas, $w = 1$, $c_1 = 1$, $c_2 = 1$ e número máximo de 10 iterações.

A Tabela 12 exibe a confiabilidade média para o rastreamento nas imagens de referência utilizando esses parâmetros iniciais. Devido ao caráter estocástico do PSO, os resultados devem ser analisados em termos estatísticos e, por isso, foram realizados 1000 rastreamentos para cada imagem, obtendo-se resultados médios.

Nota-se que a imagem *hollywood* obteve o pior desempenho. Por este motivo, esta imagem foi escolhida para se realizar a sintonia do PSO. Com a configuração $PSO_{inicial}$,

Tabela 12: Confiabilidade média para 1000 rastreamentos com parâmetros iniciais

	pickup	truck	redcar	hollywood	carros	sedan	IRcar1	IRcar2	IRcar3
C(%)	76,6	94,2	88,3	39,6	77,6	96,2	94,8	62,7	92,2

os três principais parâmetros da técnica (w, c_1 e c_2) foram variados individualmente de 0 a 2, com passo de 0,2, e a confiabilidade média para 1000 rastreamentos foi registrada. Os resultados encontram-se na Tabela 13 e nos gráficos das Figuras 40, 41 e 42. Observa-se nestes gráficos que a maior confiabilidade acontece para os seguintes parâmetros:

- PSO_{otimo}: enxame de 20 partículas, $w = 0,6$, $c_1 = 0,6$, $c_2 = 2$ e número máximo de 10 iterações.

No intuito de avaliar o desempenho com os novos parâmetros, foram realizados novos 1000 rastreamentos para as imagens de referência, cujos resultados encontram-se na Tabela 14. A confiabilidade para todas as imagens aumentou, ficando acima de 88,7%, exceto para a imagem *hollywood*, com 65,5%.

Tabela 13: Taxa de confiabilidade média para imagem *hollywood* (1000 rastreamentos)

$(w; c_1; c_2)$	C (%)	$(w; c_1; c_2)$	C (%)	$(w; c_1; c_2)$	C (%)
(0,0; 1,0; 1,0)	28,5	(1,0; 0,0; 1,0)	27,8	(1,0; 1,0; 0,0)	6,7
(0,2; 1,0; 1,0)	41,5	(1,0; 0,2; 1,0)	34,8	(1,0; 1,0; 0,2)	17,5
(0,4; 1,0; 1,0)	54,1	(1,0; 0,4; 1,0)	39,0	(1,0; 1,0; 0,4)	25,6
(0,6; 1,0; 1,0)	60,2	(1,0; 0,6; 1,0)	42,9	(1,0; 1,0; 0,6)	35,9
(0,8; 1,0; 1,0)	51,7	(1,0; 0,8; 1,0)	42,6	(1,0; 1,0; 0,8)	35,1
(1,0; 1,0; 1,0)	39,6	(1,0; 1,0; 1,0)	39,6	(1,0; 1,0; 1,0)	39,6
(1,2; 1,0; 1,0)	34,3	(1,0; 1,2; 1,0)	38,9	(1,0; 1,0; 1,2)	46,5
(1,4; 1,0; 1,0)	33,4	(1,0; 1,4; 1,0)	40,0	(1,0; 1,0; 1,4)	44,5
(1,6; 1,0; 1,0)	28,4	(1,0; 1,6; 1,0)	41,4	(1,0; 1,0; 1,6)	43,6
(1,8; 1,0; 1,0)	27,5	(1,0; 1,8; 1,0)	40,0	(1,0; 1,0; 1,8)	50,9
(2,0; 1,0; 1,0)	23,5	(1,0; 2,0; 1,0)	33,7	(1,0; 1,0; 2,0)	51,8

Tabela 14: Confiabilidade média para 1000 rastreamentos com parâmetros otimizados

	pickup	truck	redcar	hollywood	carros	sedan	IRcar1	IRcar2	IRcar3
C(%)	93,3	97,3	96,7	65,5	93,2	97,6	98,0	88,7	97,1

A sintonia do PSO não se limita somente a escolha dos parâmetros principais. É necessário também definir o número de partículas e o número máximo de iterações. Quanto maior o número de partículas e de iterações, maior a chance do PSO convergir.

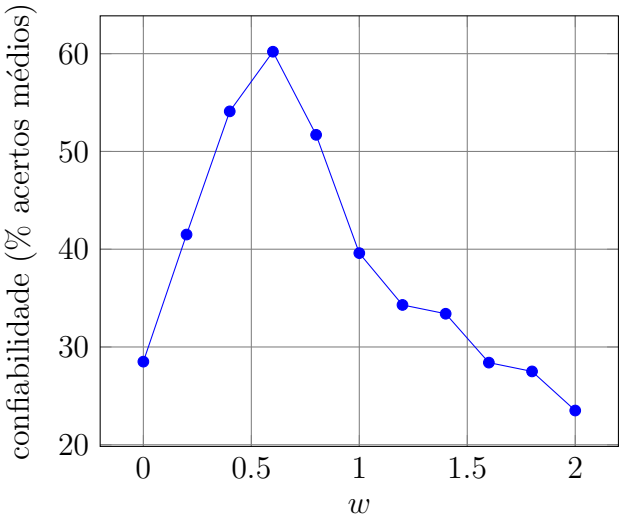


Figura 40: Variação de w para rastreamento na imagem *hollywood*

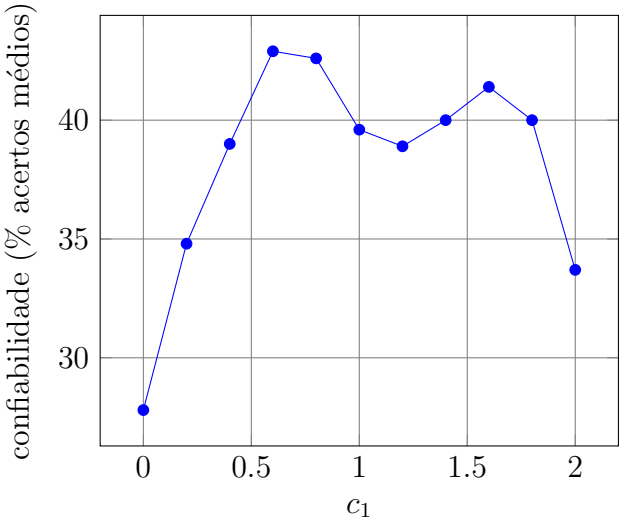


Figura 41: Variação de c_1 para rastreamento na imagem *hollywood*

Acontece que, em situações do mundo real, nem sempre se possui recursos computacionais e tempo para se realizar a busca indefinidamente. Um vídeo comum, de televisão por exemplo, possui taxa de 30 *frames*/segundo. Para que se possa tratar esse vídeo em tempo real, cada *frame* deve processado em, no máximo, $\frac{1}{30} = 0,03333$ s. Ou seja, o rastreamento realizado pelo PSO, somando o tempo de todas as suas iterações, deve ser menor que 33,33 ms.

Sabe-se que o tempo máximo para um rastreamento está associado ao número de partículas e ao número de iterações utilizadas. Caso não haja convergência, esse tempo pode ser estimado conforme Equação 13:

$$T_{MAX} = T_{IT} \times IT_{MAX}, \tag{13}$$

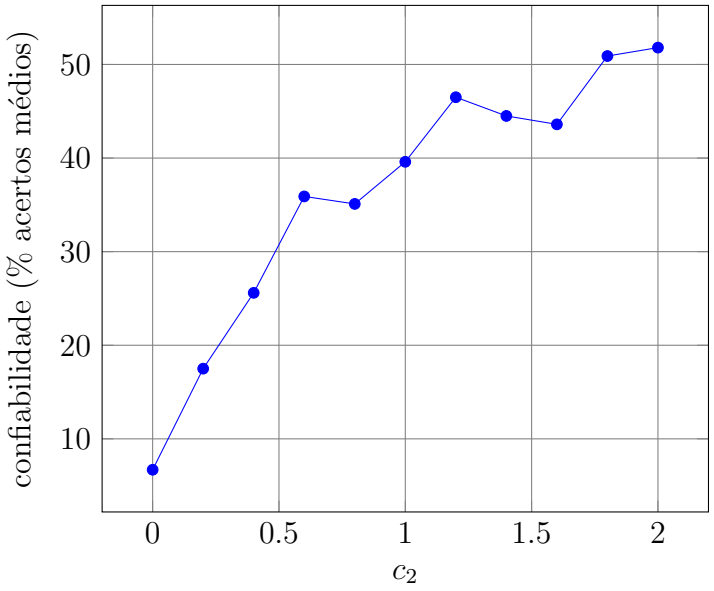


Figura 42: Variação de c_2 para rastreamento na imagem *hollywood*

Tabela 15: Variação do número de partículas e número máximo de iterações

#partículas	IT_{MAX}	T_{IT} (ms)	T_{MAX} (ms)	C (%)
10	18	1,9	34,2	56,4
11	16	2,06	32,96	57,5
12	15	2,23	33,45	59,1
13	14	2,39	33,46	58,6
14	13	2,56	33,28	58,8
15	12	2,72	32,64	61,8
16	11	2,88	31,68	62
17	11	3,06	33,66	63,1
18	10	3,22	32,2	64,6
19	9	3,39	30,51	56,4
20	9	3,55	31,95	62,1
21	8	3,72	29,76	53,3
22	8	3,88	31,04	57,2
23	8	4,05	32,4	59,3
24	7	4,21	29,47	51,1
25	7	4,38	30,66	53,6
30	6	5,2	31,2	48,2
35	5	6,03	30,15	38,9
40	5	6,86	34,3	40,4
45	4	7,69	30,76	27,4
50	4	8,52	34,08	28,1

onde T_{MAX} é o tempo máximo para um rastreamento, T_{IT} é o tempo de um iteração e IT_{MAX} é o número máximo de iterações estipulado. O tempo T_{IT} está diretamente ligado ao número de partículas.

No intuito de definir o número de partículas e o número máximo de iterações, que atendam às limitações de tempo, a Tabela 15 foi construída. Para obtenção dos resultados, o PSO foi implementado como um projeto integrado de software/hardware com o coprocessador funcionando em *pipeline*. A coluna 3 dessa tabela foi preenchida com o tempo consumido por uma iteração, a partir do número de partículas da coluna 1. Para cada número de partículas, foi estipulado um número máximo de iterações (coluna 2) no qual o tempo máximo do rastreamento não ultrapassasse $33,33 \pm 1$ ms. Esta é a limitação imposta pelo tempo real. A coluna 4 é o resultado da aplicação da Equação 13 e exibe o tempo máximo do rastreamento para cada configuração.

Finalmente, a qualidade de cada configuração partícula/iterações foi avaliada. Foram realizados 1000 rastreamentos para imagem *hollywood* e a confiabilidade foi registrada na coluna 5 da Tabela 15 e no gráfico da Figura 43. É possível identificar que a configuração com 18 partículas e 10 iterações máximas apresenta a melhor confiabilidade. Assim, a configuração do PSO, considerando aplicações em tempo real, pode ser definida como:

- PSO_{real}: enxame de 18 partículas, $w = 0.6$, $c_1 = 0.6$, $c_2 = 2$ e número máximo de 10 iterações.

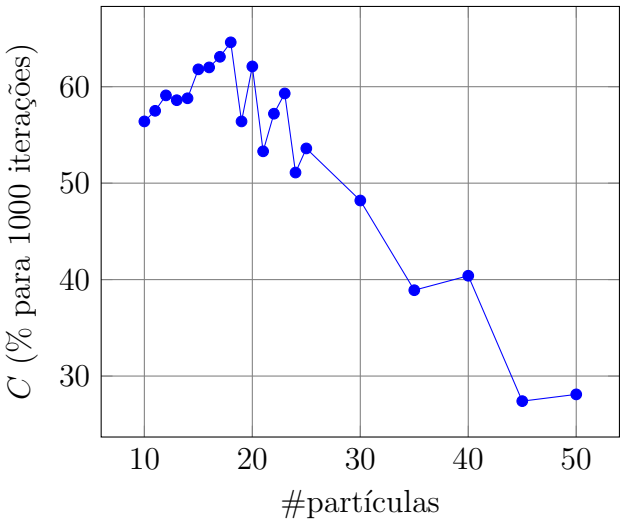
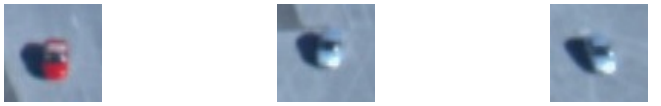


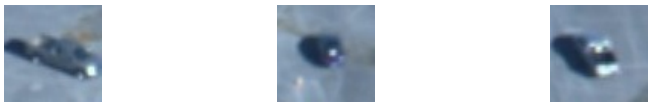
Figura 43: Variação do número de partículas e IT_{MAX} para imagem *hollywood*

6.3 Avaliação do Coprocessador

O coprocessador é um dos produtos desta dissertação e foi desenvolvido, em linguagem VHDL, para realizar o cálculo do coeficiente de correlação em hardware.



(a) Template (b) Carro1 (c) Carro2



(d) Carro3 (e) Carro4 (f) Carro5

Figura 44: Pedaços retirados da imagem *redcar*

Tabela 16: Valor de PCC para 5 imagens da Figura 44 com relação ao *template* de *redcar*

Cálculo por	Carro1	Carro2	Carro3	Carro4	Carro5
processador	0,286555	0,551740	0,321651	0,488303	0,512365
coprocessador	0,286254	0,551743	0,321650	0,488301	0,512365

Tabela 17: Valor do PCC calculado para 50 imagens de *redcar*

(y,x)	processador	coprocessador	(y,x)	processador	coprocessador
(272,359)	-0,070696	-0,070697	(24,178)	0,046019	0,046019
(122,423)	0,051468	0,051468	(381,417)	-0,147763	-0,147764
(312,311)	0,002391	0,002392	(397,63)	0,135572	0,135570
(129,220)	0,178111	0,178110	(21,306)	-0,183542	-0,183542
(399,296)	0,188550	0,188550	(127,164)	-0,291032	-0,291032
(77,90)	0,006302	0,006302	(217,423)	0,252708	0,252707
(331,417)	0,175863	0,175865	(36,34)	-0,150577	-0,150577
(106,457)	-0,165561	-0,165561	(208,493)	-0,184056	-0,184056
(219,220)	-0,054359	-0,054359	(394,167)	-0,064563	-0,064563
(311,409)	0,164047	0,164050	(201,103)	0,043871	0,043872
(87,405)	0,228194	0,228192	(71,8)	0,020039	0,020039
(112,450)	-0,119795	-0,119794	(76,328)	0,022579	0,022579
(196,87)	-0,124817	-0,124817	(365,412)	-0,138024	-0,138029
(232,67)	0,099645	0,099645	(235,76)	0,151061	0,151061
(224,376)	-0,173963	-0,173963	(24,221)	-0,106111	-0,106111
(214,146)	0,276293	0,276294	(61,398)	0,185553	0,185553
(190,448)	0,133338	0,133338	(316,347)	0,297266	0,297266
(116,375)	-0,059652	-0,059653	(390,122)	-0,060710	-0,060710
(104,15)	-0,076945	-0,076945	(54,504)	-0,297323	-0,297326
(204,389)	-0,209850	-0,209850	(113,88)	-0,229701	-0,229700
(170,129)	0,264429	0,264427	(291,244)	0,187798	0,187798
(282,448)	0,160680	0,160679	(395,433)	0,083750	0,083748
(118,271)	-0,155118	-0,155119	(1,104)	-0,212266	-0,212265
(113,190)	-0,092083	-0,092083	(115,11)	0,007360	0,007360
(65,404)	0,090310	0,090310	(300,152)	0,003998	0,003999

Esta Seção tem por objetivo avaliar o coprocessador, em termos de precisão de seus resultados. Primeiramente, comparou-se o valor do PCC calculado pelo coprocessador com aquele calculado, como referência, pelo processador. Para tal, foram retirados cinco pedaços da imagem *redcar*, Figuras 44(b) a 44(f), a serem comparados com seu *template*, Figura 44(a). Esses pedaços são pequenas imagens que se referem aos carros presentes em *redcar*. Os valores calculados encontram-se na Tabela 16, que diz respeito a 1 cálculo de PCC. Observa-se diferença de valores somente na sexta casa decimal.

A fim de gerar mais resultados para avaliação, foram extraídos 50 pedaços aleatórios da imagem *redcar*, de 64x64 pixels. Essas subimagens foram comparados com o *template*. A Tabela 17 exibe, nas colunas 1 e 4, os pontos centrais do local que as imagens foram retiradas. Os valores do PCC calculados pelo processador encontram-se nas colunas 2 e 5, e pelo coprocessador nas colunas 3 e 6. Observa-se que o cálculo realizado pelo coprocessador é bem preciso, possuindo uma diferença máxima de 0,000003 para a imagem (311,409). Essa precisão é aceitável para esta dissertação, que utiliza 3 casas decimais.

6.4 Avaliação do Desempenho

O principal objetivo desta dissertação é implementar um projeto, em um sistema portátil embarcado, capaz de rastrear imagens por *template matching* e correlação em tempo real. Dessa forma, é imprescindível que se faça uma comparação dos resultados, em termos de tempo de processamento, com e sem as melhorias propostas no projeto integrado de software/hardware. Para avaliar esse desempenho, foram utilizados seis cenários, do quais três baseados na busca exaustiva (referências) e três baseados no PSO *global best*:

1. BE_{SO}: Busca exaustiva implementada em software e executada pelo processador;
2. BE_{HS}: Busca exaustiva implementada em software com cálculo de PCC pelo coprocessador em modo sequencial;
3. BE_{HP}: Busca exaustiva implementada em software com cálculo de PCC pelo coprocessador funcionando em *pipeline*;
4. PSO_{SO}: PSO *global best* implementado em software somente e executado pelo processador;

5. PSO_{HS}: PSO implementado como um projeto integrado de software/hardware com o coprocessador funcionando em modo sequencial; e

6. PSO_{HP}: PSO implementado como um projeto integrado de software/hardware com o coprocessador funcionando em *pipeline*.

Para o PSO, foram utilizados os parâmetros PSO_{real}, encontrados na Seção 6.2, a saber:

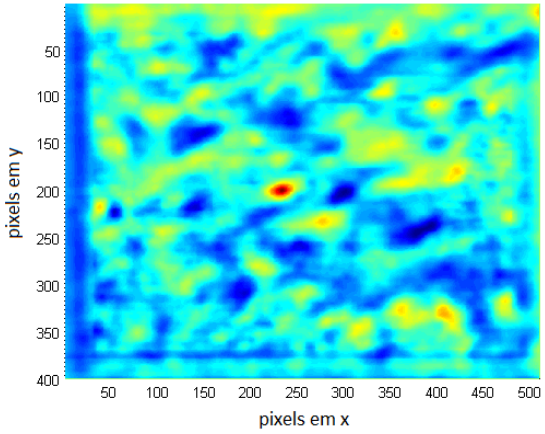
- PSO_{real}: enxame de 18 partículas, $w = 0,6$, $c_1 = 0,6$, $c_2 = 2$ e número máximo de 10 iterações.

Também foi estabelecida a velocidade máxima de 10 pixels/iteração, para cada partícula, e foi utilizada também uma técnica de memorização. Esta técnica é empregada para acelerar programas por meio do armazenamento dos resultados de funções custosas e pelo retorno desses resultados, quando as mesmas entradas ocorrerem novamente. Os valores de PCC calculados para cada partícula foram armazenados em uma memória local. Se uma partícula atingir posição que alguma outra partícula já tenha estado, o valor da correlação é reutilizado, ao invés de recalculado. Esse processo aproveita o trabalho já realizado anteriormente, acelerando o rastreamento.

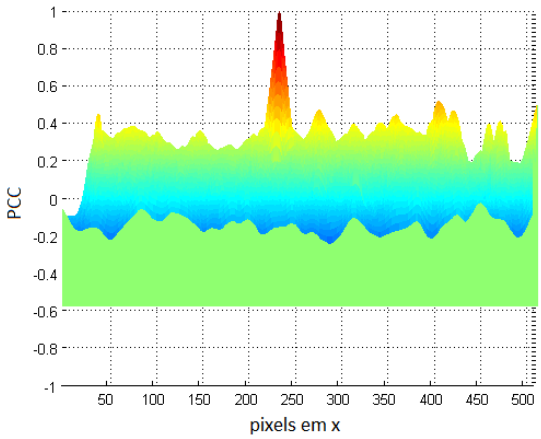
Em termos do espaço de busca, ao invés de se investigar a imagem inteira, foi definida uma janela. Em uma situação real de emprego, onde se deseja acompanhar um alvo pré-estabelecido por um atirador, por exemplo, espera-se que o alvo não mude sua posição abruptamente, e que permaneça nos arredores de sua posição no *frame* anterior. Desta forma, pode-se semear um ponto inicial para a busca e uma janela máxima, onde espera-se que o objeto esteja. Para o caso desta dissertação, estabeleceu-se uma variação máxima de 50 pixels por *frame*, ou seja uma janela de busca de 101x101 pixels. O ponto inicial ficou estabelecido como a própria posição do objeto.

O desempenho do sistema proposto foi avaliado a partir dos resultados obtidos para nove imagens de referência (*benchmarks*), oriundas de (COLLINS; ZHOU; TEH, 2005). Para o PSO, essas imagens correspondem a função a ser otimizada, cujos valores variam de -1 a 1. Os resultados encontram-se didaticamente separados nas próximas Seções.

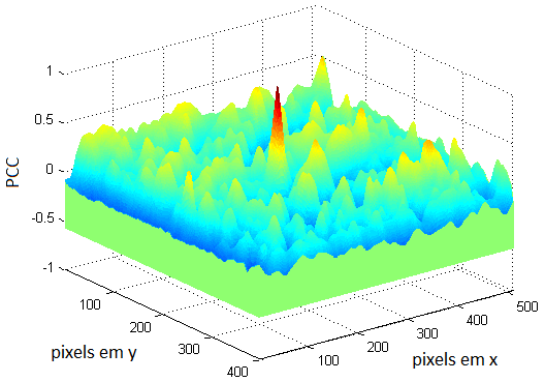
(a) Imagem *pickup* com *template* destacado



(b) Visão de cima da correlação



(c) Visão lateral da correlação



(d) Perspectiva da correlação

Figura 45: Imagem *pickup* e seu comportamento para correlação

6.4.1 Imagem *pickup*

A imagem chamada *pickup*, Figura 45(a), se refere ao *frame* 170, do *benchmark* Egtest05, que foi redimensionado para possuir 510x400 pixels. Foram retiradas 65 colunas de cada borda lateral e 40 linhas de cada borda horizontal, sem redução de escala. A Figura 45(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 45(a). As Figuras 45(b) e 45(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Observa-se um máximo global bem visível, na posição do *template*, e outros máximos locais abaixo do valor 0,5, espalhados ao longo da imagem.

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 18. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.

Tabela 18: Resultados de BE para imagem *pickup*

Cenário	Tempo (*ms*)	*Speedup*
BE_{SO}	23416,69	1
BE_{HS}	4106,64	5,70
BE_{HP}	1728,39	13,55

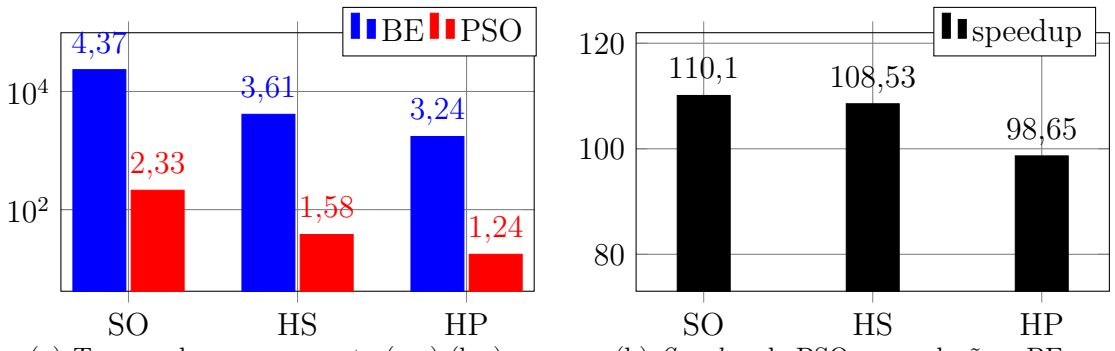
Tabela 19: Resultados de PSO médios para imagem *pickup* (1000 rastreamentos)

Cenário	#Iterações	Tempo (*ms*)	C (%)	*Speedup*
PSO_{SO}	5,48	212,68	95,90	1
PSO_{HS}	5,47	37,84	96,20	5,62
PSO_{HP}	5,68	17,52	92,90	12,14

Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 19. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

Para efeito de comparação, os gráficos de barras das Figura 46 foram produzidos. O gráfico da Figura 46(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 46(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,39 ms, registrando um *speedup* de 13,55, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento



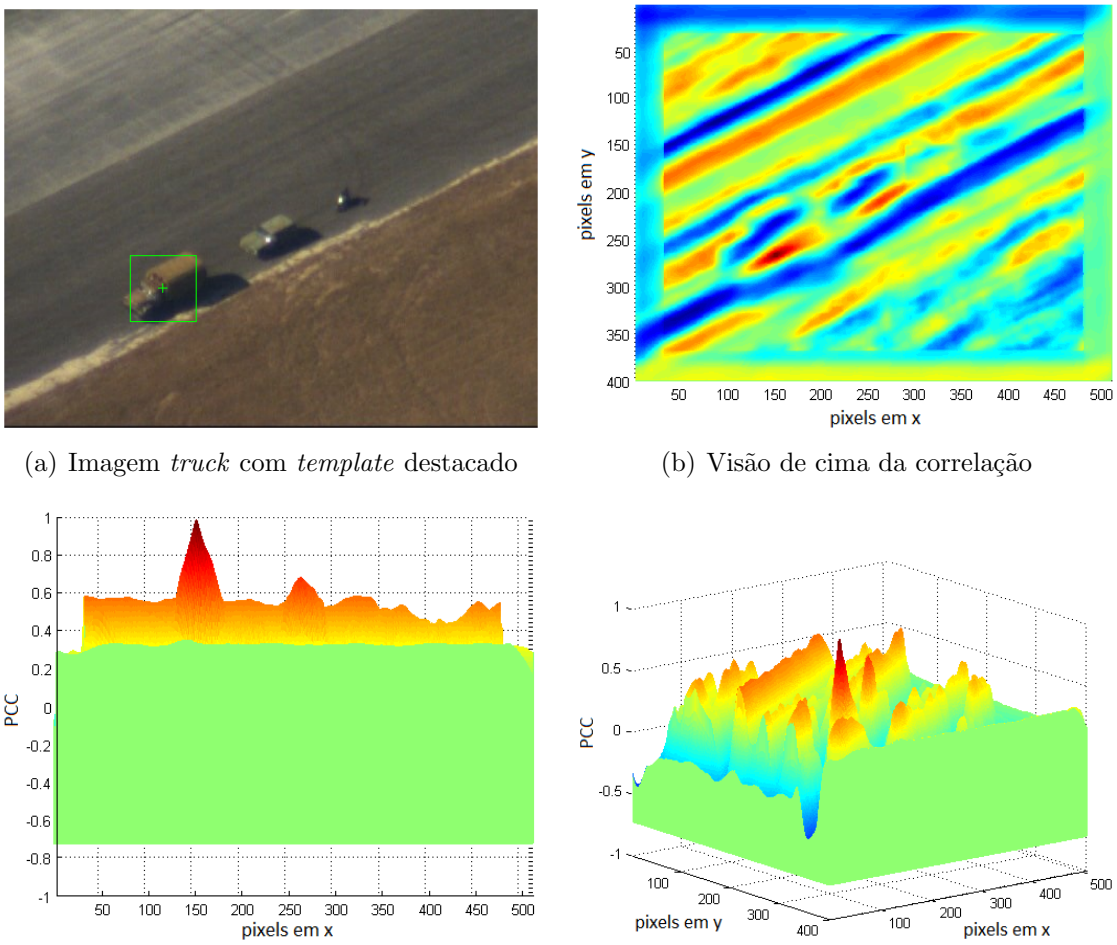
(a) Tempos de processamento (*ms*) (log)

(b) *Speedup* de PSO com relação a BE

Figura 46: Gráficos de barras para imagem *pickup*

em 17,52 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,14. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 110,1, 108,53 e 98,65 para as soluções SO, HS e HP, respectivamente.

6.4.2 Imagem *truck*



(a) Imagem *truck* com *template* destacado

(b) Visão de cima da correlação

(c) Visão lateral da correlação

(d) Perspectiva da correlação

Figura 47: Imagem *truck* e seu comportamento para correlação

A imagem chamada *truck*, Figura 47(a), se refere ao *frame* 2370, do *benchmark* Egtest03, que foi redimensionado para possuir 510x400 pixels. Foram retiradas 130 colunas da borda da direita e 80 linhas da borda superior, sem redução de escala. A Figura 47(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 47(a). As Figuras 47(b) e 47(c) se referem às vistas superior e lateral desse gráfico, respectivamente.

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 20. Os *speedup* dos cenários, em relação a BE_{SO} (busca

Tabela 20: Resultados de BE para imagem *truck*

Cenário	Tempo (*ms*)	Speedup
BE_{SO}	23416,53	1
BE_{HS}	4104,66	5,70
BE_{HP}	1728,39	13,55

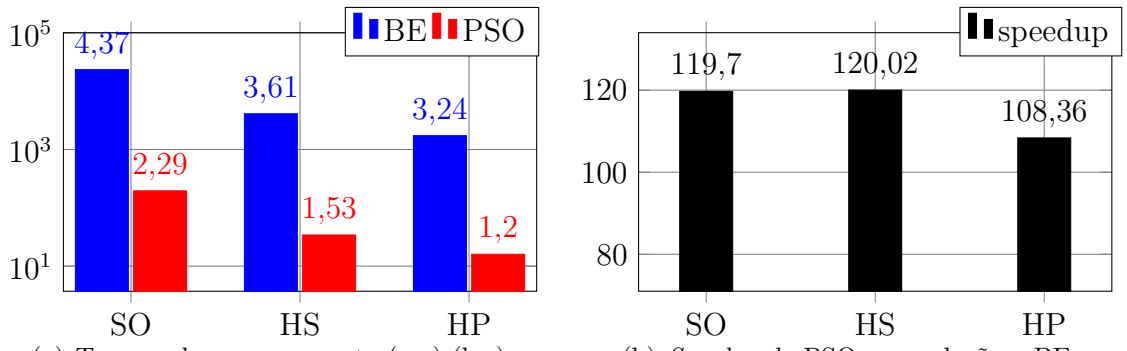
Tabela 21: Resultados de PSO médios para imagem *truck* (1000 rastreamentos)

Cenário	#Iterações	Tempo (*ms*)	C (%)	Speedup
PSO_{SO}	4,97	195,63	100,00	1
PSO_{HS}	4,88	34,20	99,70	5,72
PSO_{HP}	5,12	15,95	96,20	12,27

exaustiva somente por software), pode ser observado na coluna 3 dessa tabela. Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 21. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

Para efeito de comparação, os gráficos de barras da Figura 48 foram produzidos. O gráfico da Figura 48(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 48(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,39 ms, registrando um *speedup* de 13,55, com relação a



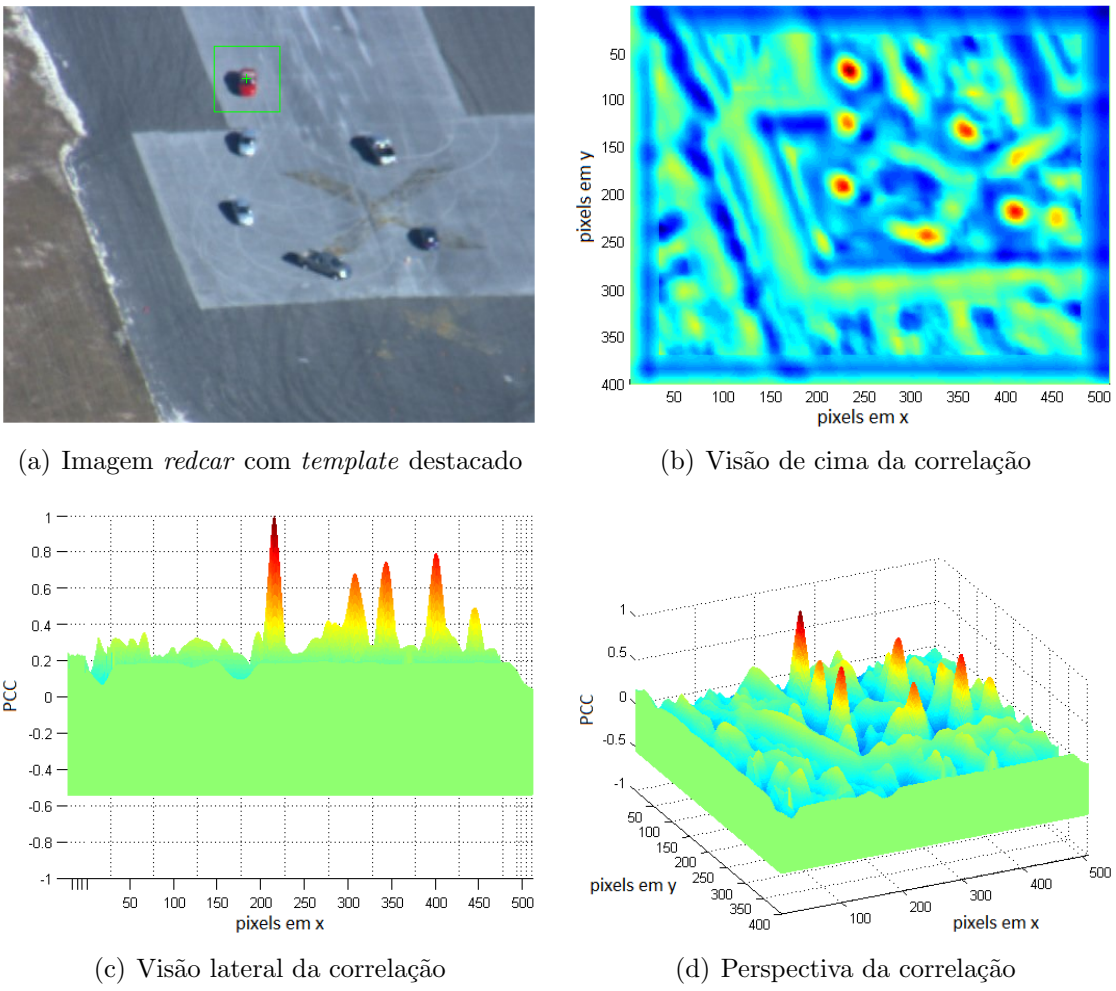
(a) Tempos de processamento (*ms*) (log)

(b) *Speedup* de PSO com relação a BE

Figura 48: Gráficos de barras para imagem *truck*

BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 15,95 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,27. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 119,7, 120,02 e 108,36 para as soluções SO, HS e HP, respectivamente.

6.4.3 Imagem *redcar*



(a) Imagem *redcar* com *template* destacado

(b) Visão de cima da correlação

(c) Visão lateral da correlação

(d) Perspectiva da correlação

Figura 49: Imagem *redcar* e seu comportamento para correlação

A imagem chamada *redcar*, Figura 49(a), se refere ao *frame* 108, do *benchmark* Egtest01, que foi redimensionado para possuir 510x400 pixels. Foram retiradas 130 colunas da borda da direita e 80 linhas da borda inferior, sem redução de escala. A Figura 49(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 49(a). As Figuras 49(b) e 49(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Observa-se máximos locais bem

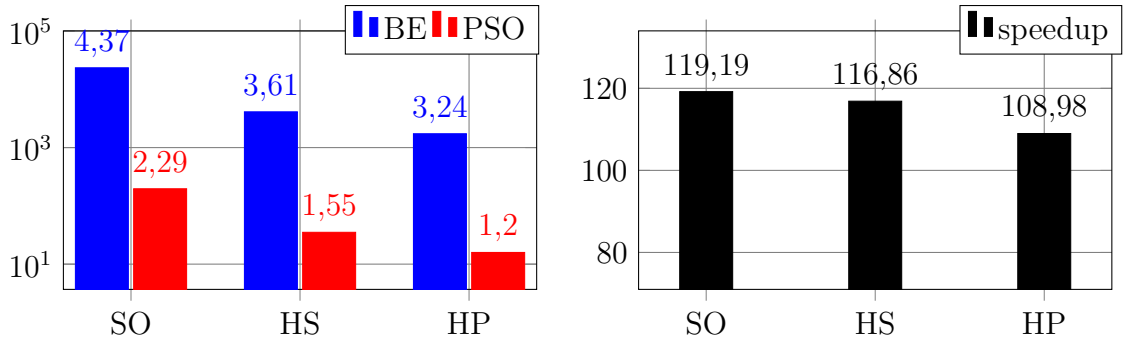
significativos, referentes aos outros carros presentes nessa imagem. Um deles chega atingir o valor de correlação de 0,8.

Tabela 22: Resultados de BE para imagem *redcar*

Cenário	Tempo (ms)	Speedup
BE_{SO}	23416,49	1
BE_{HS}	4108,77	5,70
BE_{HP}	1728,45	13,55

Tabela 23: Resultados de PSO médios para imagem *redcar* (1000 rastreamentos)

Cenário	#Iterações	Tempo (ms)	C (%)	Speedup
PSO_{SO}	5,00	196,47	99,10	1
PSO_{HS}	5,04	35,16	98,90	5,59
PSO_{HP}	5,08	15,86	97,50	12,39



(a) Tempos de processamento (ms) (log)

(b) *Speedup* de PSO com relação a BE

Figura 50: Gráficos de barras para imagem *redcar*

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 22. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.

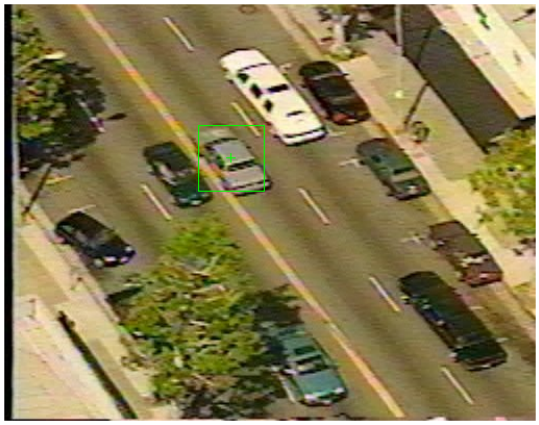
Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 23. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

Para efeito de comparação, os gráficos de barras da Figura 50 foram produzidos. O gráfico da Figura 50(a) compara os tempos de processamento consumidos por cada cenário,

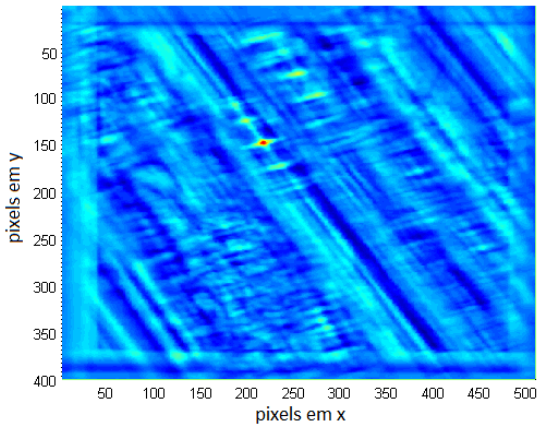
considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 50(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,45 ms, registrando um *speedup* de 13,55, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 15,86 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,39. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 119,19, 116,86 e 108,98 para as soluções SO, HS e HP, respectivamente.

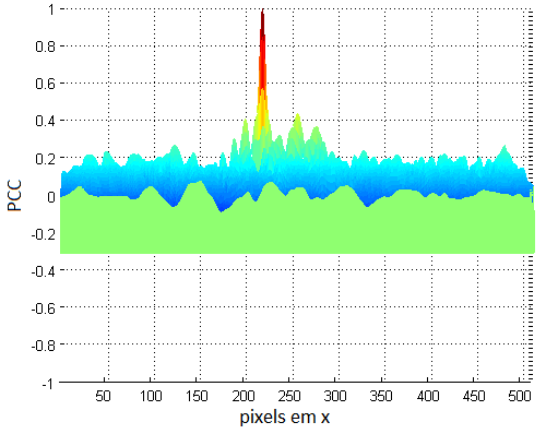
6.4.4 Imagem *hollywood*



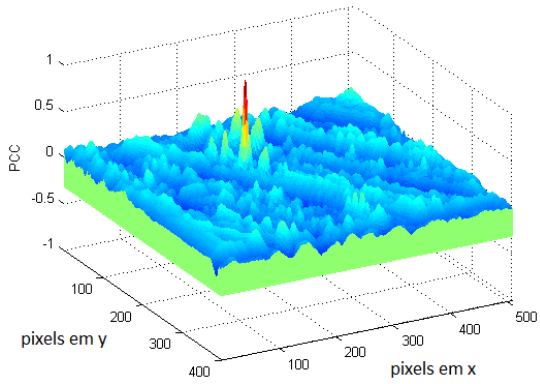
(a) Imagem *hollywood* com *template* destacado



(b) Visão de cima da correlação



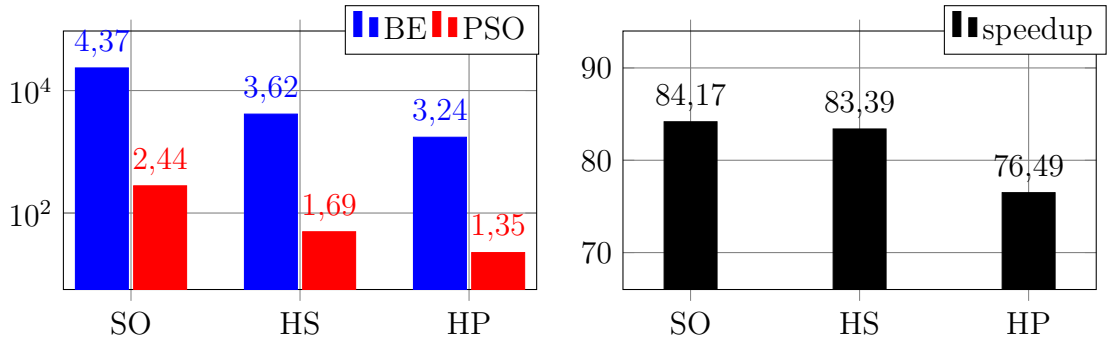
(c) Visão lateral da correlação



(d) Perspectiva da correlação

Figura 51: Imagem *hollywood* e seu comportamento para correlação

A imagem chamada *hollywood*, Figura 51(a), se refere ao *frame* 287, do *benchmark holywood*, que foi redimensionado para possuir 510x400 pixels. Foram retiradas 210



(a) Tempos de processamento (*ms*) (log) (b) *Speedup* de PSO com relação a BE

Figura 52: Gráficos de barras para imagem *hollywood*

colunas da borda da direita e 80 linhas da borda superior, sem redução de escala. A Figura 51(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 51(a). As Figuras 51(b) e 51(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Observa-se que essa imagem possui a maior parte dos pontos com valor de correlação abaixo de 0,2.

Tabela 24: Resultados de BE para imagem *hollywood*

Cenário	Tempo (*ms*)	Speedup
BE_{SO}	23416,39	1
BE_{HS}	4125,06	5,68
BE_{HP}	1728,77	13,55

Tabela 25: Resultados de PSO médios para imagem *hollywood* (1000 rastreamentos)

Cenário	#Iterações	Tempo (*ms*)	C (%)	Speedup
PSO_{SO}	7,75	278,22	65,40	1
PSO_{HS}	7,70	49,47	67,20	5,62
PSO_{HP}	7,71	22,60	64,60	12,31

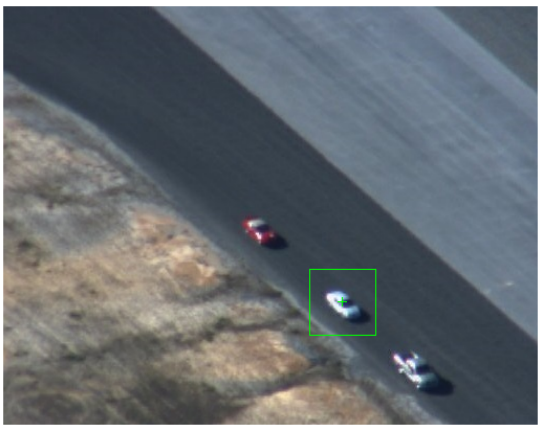
O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 24. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.

Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 25. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

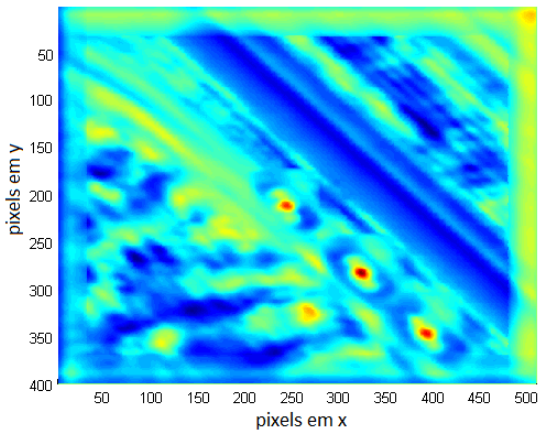
Para efeito de comparação, os gráficos de barras da Figura 52 foram produzidos. O gráfico da Figura 52(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 52(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,77 ms, registrando um *speedup* de 13,55, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 22,60 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,31. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 84,17, 83,39 e 76,49 para as soluções SO, HS e HP, respectivamente.

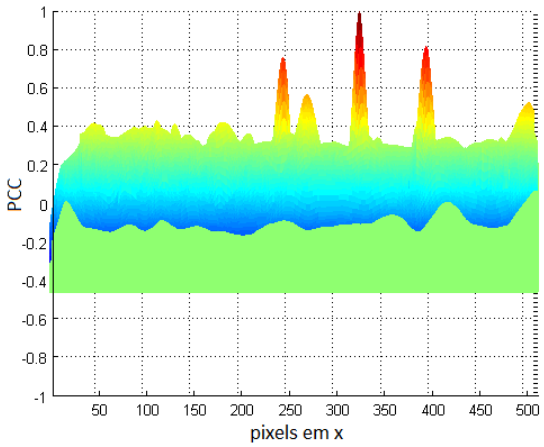
6.4.5 Imagem *carros*



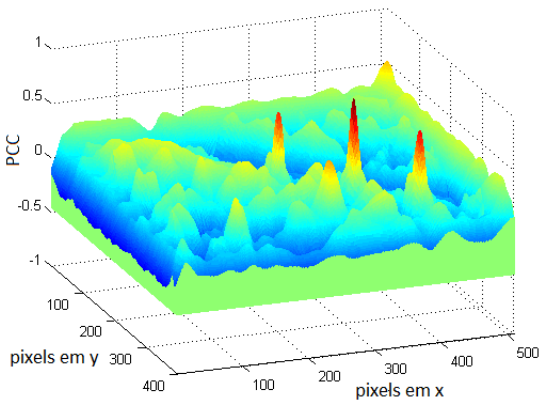
(a) Imagem *carros* com *template* destacado



(b) Visão de cima da correlação

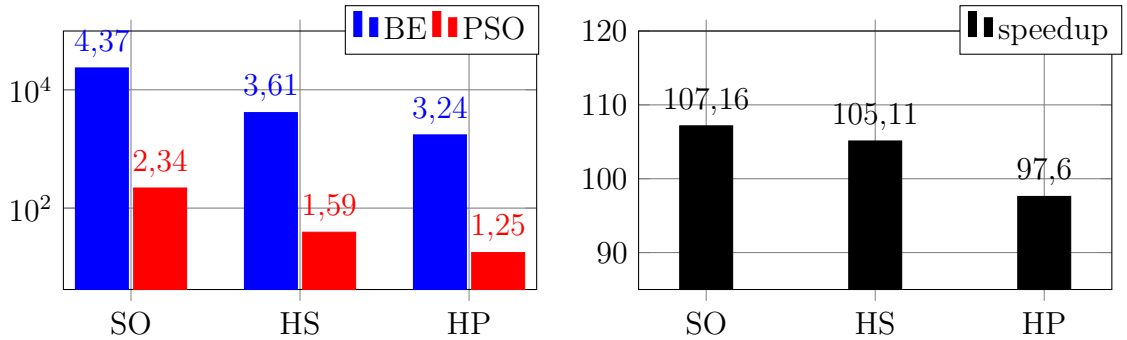


(c) Visão lateral da correlação



(d) Perspectiva da correlação

Figura 53: Imagem *carros* e seu comportamento para correlação



(a) Tempos de processamento (*ms*) (log) (b) *Speedup* de PSO com relação a BE

Figura 54: Gráficos de barras para imagem *carros*

A imagem chamada *carros*, Figura 53(a), se refere ao primeiro *frame* do *benchmark* Egtest02, que foi redimensionado para possuir 510x400 pixels. Foram retiradas 65 colunas de cada borda lateral e 40 linhas de cada borda horizontal, sem redução de escala. A Figura 53(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 53(a). As Figuras 53(b) e 53(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Observa-se dois máximos locais bem significativos, referentes aos outros carros presentes nessa imagem. O valor de PCC nesses pontos de máximos locais atinge valores próximos a 0,8 e 0,75.

Tabela 26: Resultados de BE para imagem *carros*

Cenário	Tempo (*ms*)	Speedup
BE_{SO}	23416,63	1
BE_{HS}	4114,15	5,69
BE_{HP}	1728,55	13,55

Tabela 27: Resultados de PSO médios para imagem *carros* (1000 rastreamentos)

Cenário	#Iterações	Tempo (*ms*)	C (%)	Speedup
PSO_{SO}	5,64	218,53	95,40	1
PSO_{HS}	5,66	39,14	94,80	5,58
PSO_{HP}	5,78	17,71	90,50	12,34

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 26. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.

Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 27. A coluna 2 exibe o número médio de iterações que os cenários

utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

Para efeito de comparação, os gráficos de barras da Figura 54 foram produzidos. O gráfico da Figura 54(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 54(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,55 ms, registrando um *speedup* de 13,55, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 17,71 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,34. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 107,16, 105,11 e 97,6 para as soluções SO, HS e HP, respectivamente.

6.4.6 Imagem *sedan*

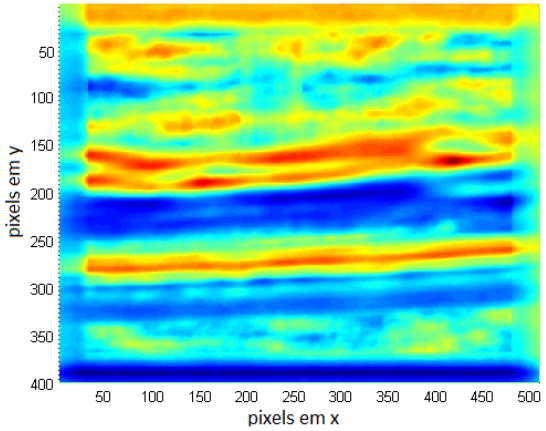
A imagem chamada *sedan*, Figura 55(a), se refere ao *frame* 1410, do *benchmark* Egtest04, que foi redimensionado para possuir 510x400 pixels. Foram retiradas 65 colunas de cada borda lateral e 80 linhas da borda superior, sem redução de escala. A Figura 55(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 55(a). As Figuras 55(b) e 55(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Nota-se que o gráfico da correlação possui muitos pontos com valores superiores a 0,6.

Tabela 28: Resultados de BE para imagem *sedan*

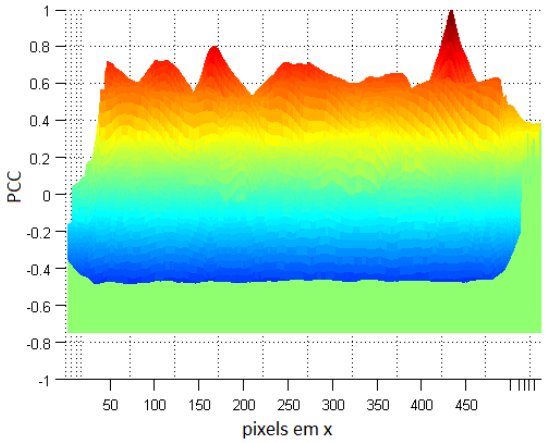
Cenário	Tempo (ms)	Speedup
BE_{SO}	23416,62	1
BE_{HS}	4119,59	5,68
BE_{HP}	1728,65	13,55

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 28. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.

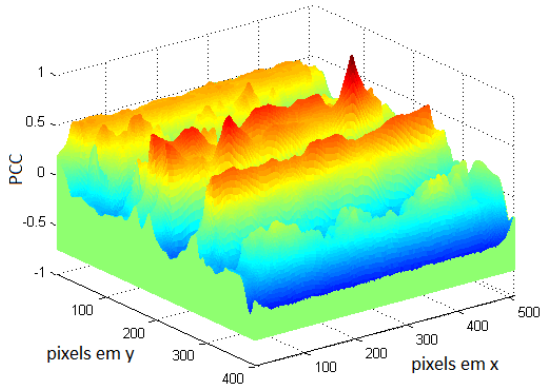
(a) Imagem *sedan* com *template* destacado



(b) Visão de cima da correlação



(c) Visão lateral da correlação



(d) Perspectiva da correlação

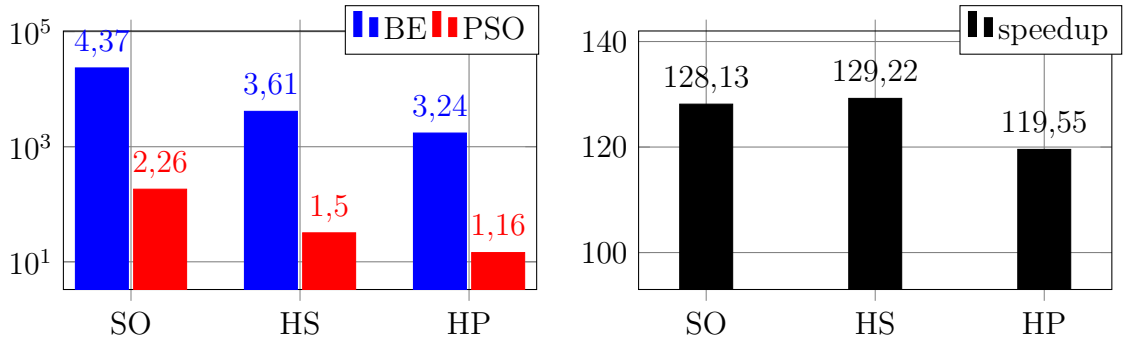
Figura 55: Imagem *sedan* e seu comportamento para correlação

Tabela 29: Resultados de PSO médios para imagem *sedan* (1000 rastreamentos)

Cenário	#Iterações	Tempo (ms)	C (%)	Speedup
PSO_{SO}	4,61	182,76	99,70	1
PSO_{HS}	4,51	31,88	99,20	5,73
PSO_{HP}	4,61	14,46	97,60	12,64

Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 29. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

Para efeito de comparação, os gráficos de barras da Figura 56 foram produzidos. O gráfico da Figura 56(a) compara os tempos de processamento consumidos por cada cenário,



(a) Tempos de processamento (*ms*) (log) (b) *Speedup* de PSO com relação a BE

Figura 56: Gráficos de barras para imagem *sedan*

considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 56(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,65 ms, registrando um *speedup* de 13,55, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 14,46 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,64. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 128,13, 129,22 e 119,55 para as soluções SO, HS e HP, respectivamente.

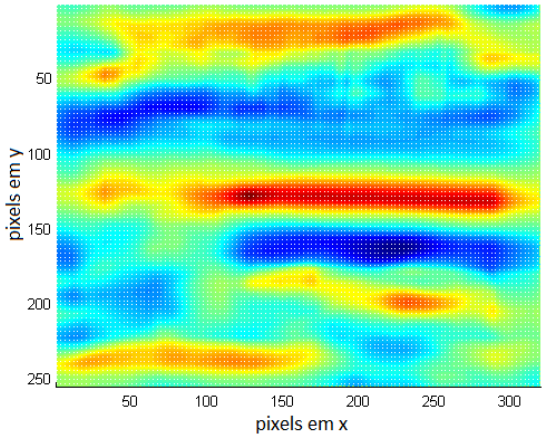
6.4.7 Imagem *IRcar1*

A imagem chamada *IRcar1*, Figura 57(a), se refere ao primeiro *frame* do *benchmark* pk-test01. Essa é uma imagem térmica (*infrared image*) com 320x256 pixels. Trabalhar com imagens térmicas, em termos de rastreamento, é a mesma coisa que trabalhar com imagens no espectro de frequências visual. A Figura 57(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 57(a). As Figuras 57(b) e 57(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Observa-se muitos pixels que possuem valores de correlação bem altos, próximos a 0,9.

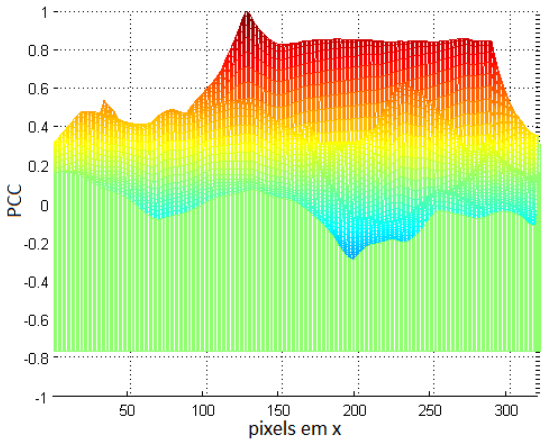
Tabela 30: Resultados de BE para imagem *IRcar1*

Cenário	Tempo (*ms*)	Speedup
BE_{SO}	23088,21	1
BE_{HS}	4135,85	5,58
BE_{HP}	1728,98	13,55

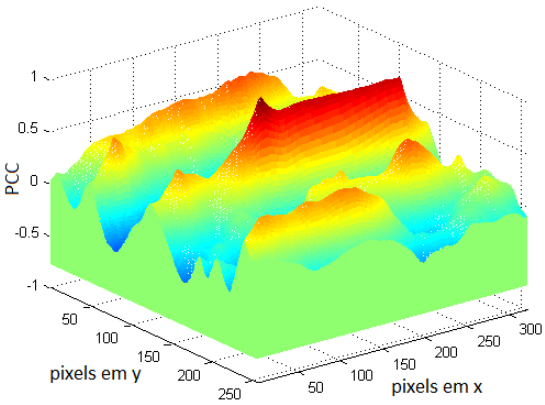
(a) Imagem *IRcar1* com *template* destacado



(b) Visão de cima da correlação



(c) Visão lateral da correlação



(d) Perspectiva da correlação

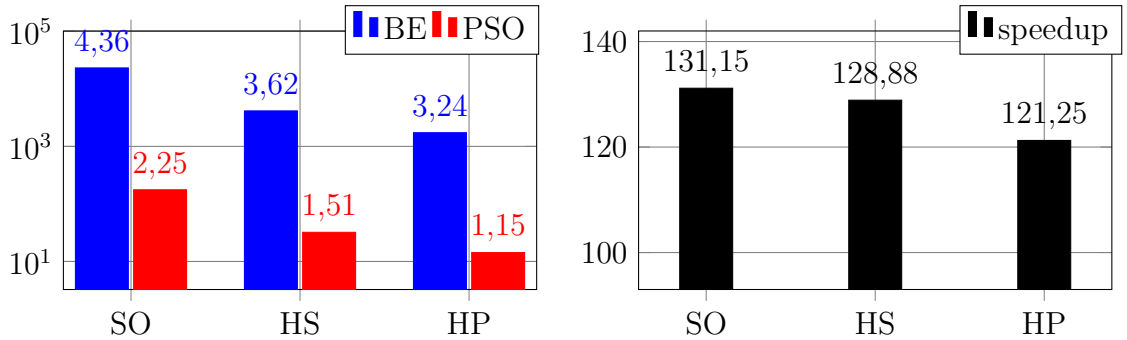
Figura 57: Imagem *IRcar1* e seu comportamento para correlação

Tabela 31: Resultados de PSO médios para imagem *IRcar1* (1000 rastreamentos)

Cenário	#Iterações	Tempo (ms)	C (%)	Speedup
PSO_{SO}	4,44	176,04	99,60	1
PSO_{HS}	4,53	32,09	99,70	5,49
PSO_{HP}	4,53	14,26	98,50	12,35

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 30. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.

Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 31. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO



(a) Tempos de processamento (*ms*) (log) (b) *Speedup* de PSO com relação a BE

Figura 58: Gráficos de barras para imagem *IRcar1*

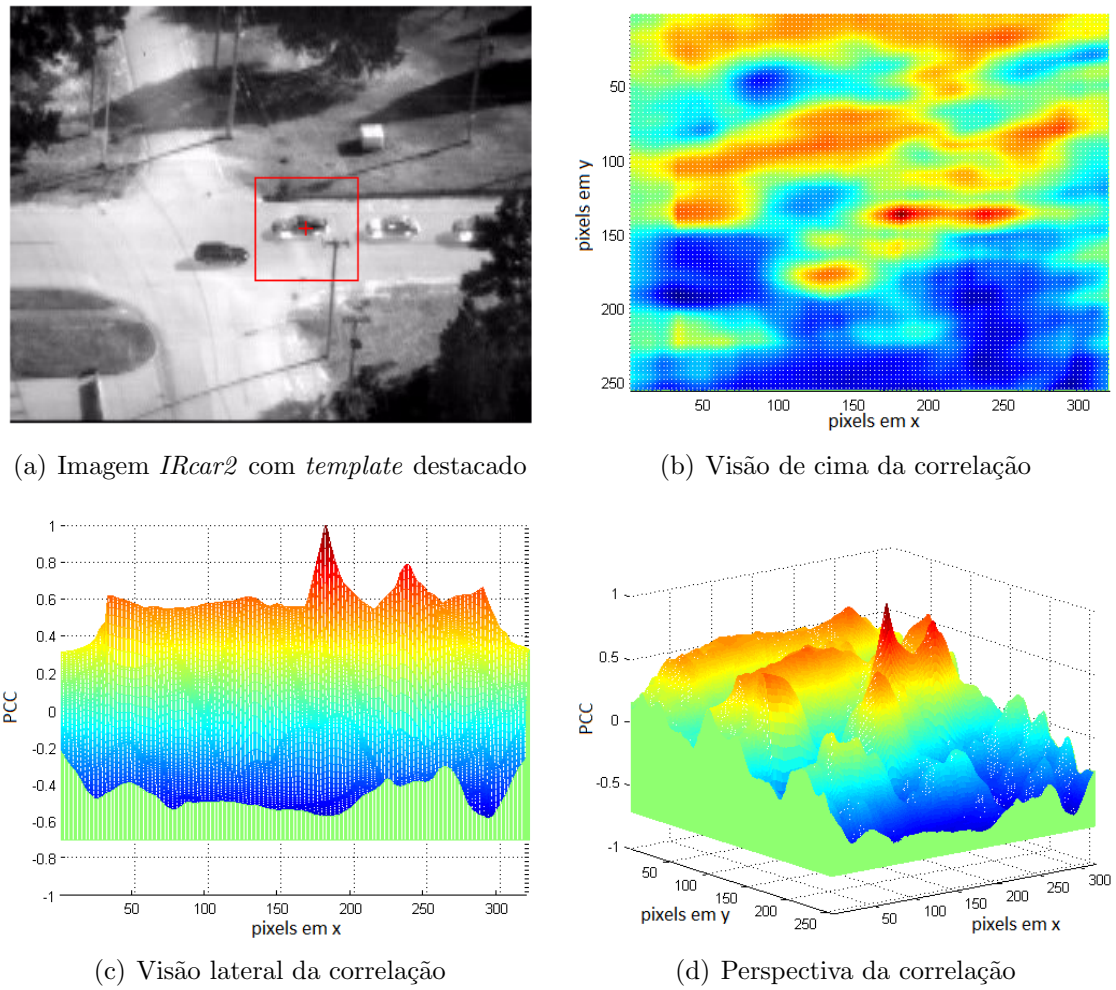
sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela. Para efeito de comparação, os gráficos de barras da Figura 58 foram produzidos. O gráfico da Figura 58(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 58(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,98 ms, registrando um *speedup* de 13,55, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 14,26 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,35. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 131,15, 128,88 e 121,25 para as soluções SO, HS e HP, respectivamente.

6.4.8 Imagem *IRcar2*

A imagem chamada *IRcar2*, Figura 59(a), se refere ao *frame* 635, do *benchmark* pktest02. Essa é uma imagem térmica (*infrared image*) com 320x256 pixels. Trabalhar com imagens térmicas, em termos de rastreamento, é a mesma coisa que trabalhar com imagens no espectro de frequências visual. A Figura 59(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 59(a). As Figuras 59(b) e 59(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Percebe-se muitos pixels que possuem valores de correlação relativamente altos, com um máximo local de 0,8.

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 32. Os *speedup* dos cenários, em relação a BE_{SO} (busca



(a) Imagem *IRcar2* com *template* destacado

(b) Visão de cima da correlação

(c) Visão lateral da correlação

(d) Perspectiva da correlação

Figura 59: Imagem *IRcar2* e seu comportamento para correlação

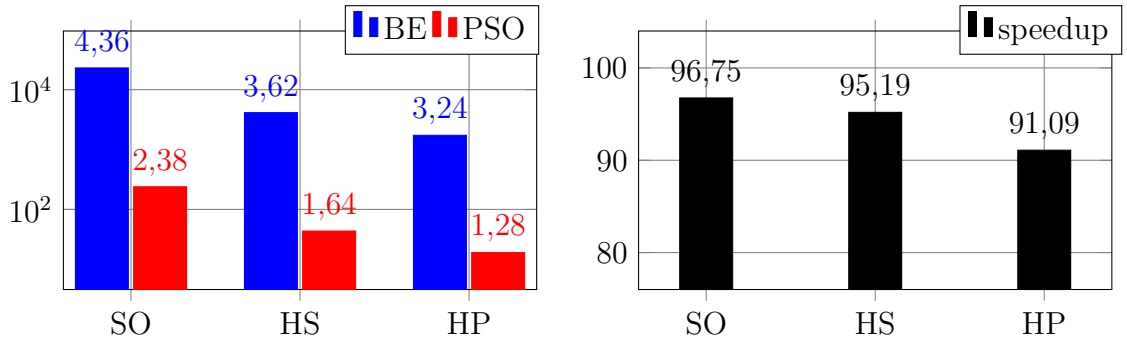
Tabela 32: Resultados de BE para imagem *IRcar2*

Cenário	Tempo (ms)	Speedup
BE_{SO}	23088,36	1
BE_{HS}	4134,27	5,58
BE_{HP}	1728,94	13,35

Tabela 33: Resultados de PSO médios para imagem *IRcar2* (1000 rastreamentos)

Cenário	#Iterações	Tempo (ms)	C (%)	Speedup
PSO_{SO}	6,24	238,64	91,50	1
PSO_{HS}	6,34	43,43	91,60	5,49
PSO_{HP}	6,21	18,98	89,40	12,57

exaustiva somente por software), pode ser observado na coluna 3 dessa tabela. Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 33. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna



(a) Tempos de processamento (*ms*) (log)

(b) *Speedup* de PSO com relação a BE

Figura 60: Gráficos de barras para imagem *IRcar2*

4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

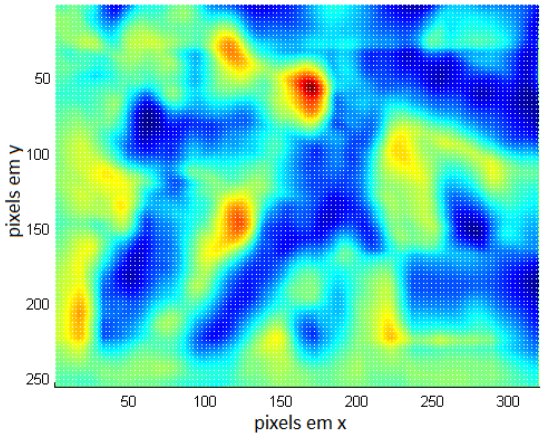
Para efeito de comparação, os gráficos de barras da Figura 60 foram produzidos. O gráfico da Figura 60(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 60(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,94 ms, registrando um *speedup* de 13,35, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 18,98 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,57. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 96,75, 95,19 e 91,09 para as soluções SO, HS e HP, respectivamente.

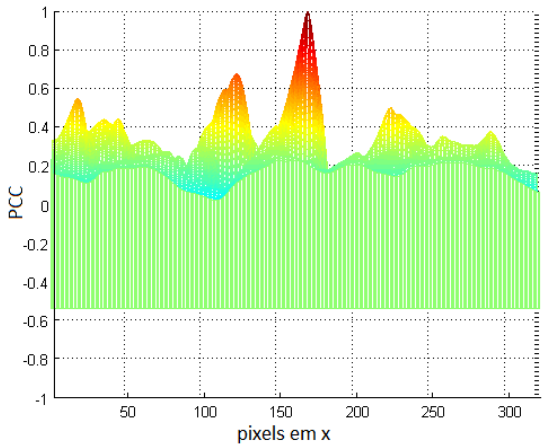
6.4.9 Imagem *IRcar3*

A imagem chamada *IRcar3*, Figura 61(a), se refere ao *frame* 1440, do *benchmark* pktest03. Essa é uma imagem térmica (*infrared image*) com 320x256 pixels. Trabalhar com imagens térmicas, em termos de rastreamento, é a mesma coisa que trabalhar com imagens no espectro de frequências visual. A Figura 61(d) exibe o gráfico, em três dimensões, do cálculo do PCC para todos os pixels da imagem, com relação ao *template* marcado na Figura 61(a). As Figuras 61(b) e 61(c) se referem às vistas superior e lateral desse gráfico, respectivamente. Nota-se alguns máximos locais no gráfico da correlação.

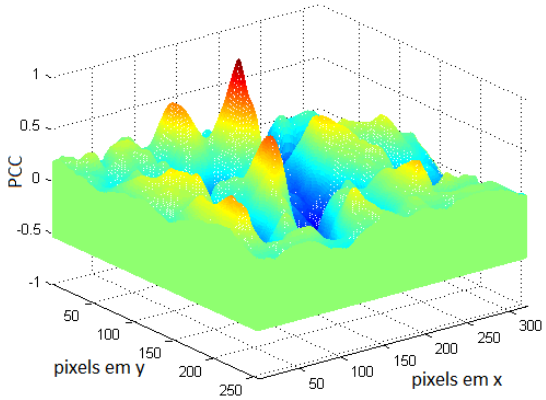
(a) Imagem *IRcar3* com *template* destacado



(b) Visão de cima da correlação



(c) Visão lateral da correlação



(d) Perspectiva da correlação

Figura 61: Imagem *IRcar3* e seu comportamento para correlação

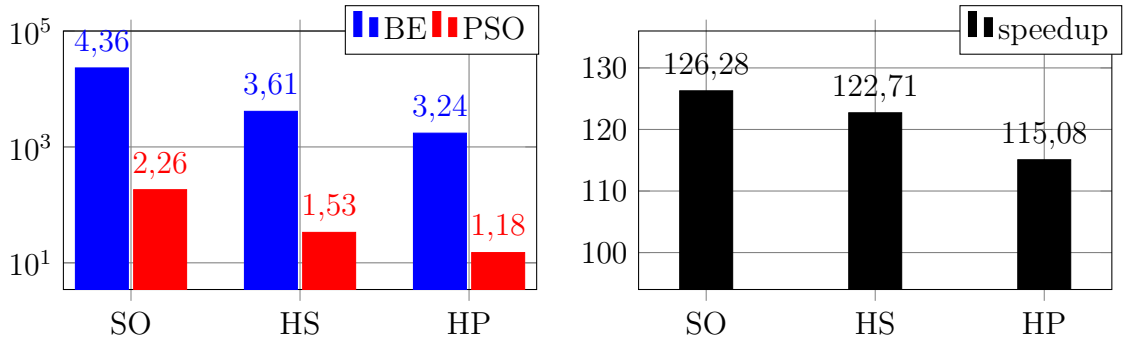
Tabela 34: Resultados de BE para imagem *IRcar3*

Cenário	Tempo (ms)	Speedup
BE_{SO}	23088,13	1
BE_{HS}	4113,18	5,61
BE_{HP}	1728,50	13,36

Tabela 35: Resultados de PSO médios para imagem *IRcar3* (1000 rastreamentos)

Cenário	#Iterações	Tempo (ms)	C (%)	Speedup
PSO_{SO}	4,65	182,83	98,60	1
PSO_{HS}	4,80	33,52	98,30	5,45
PSO_{HP}	4,81	15,02	96,90	12,17

O tempo de processamento consumido para os cenários de busca exaustiva, encontram-se na segunda coluna da Tabela 34. Os *speedup* dos cenários, em relação a BE_{SO} (busca exaustiva somente por software), pode ser observado na coluna 3 dessa tabela.



(a) Tempos de processamento (*ms*) (log) (b) *Speedup* de PSO com relação a BE

Figura 62: Gráficos de barras para imagem *IRcar3*

Os resultados médios para 1000 rastreamentos, utilizando os cenários com PSO, encontram-se na Tabela 35. A coluna 2 exibe o número médio de iterações que os cenários utilizaram. A coluna 3 registra o tempo médio consumido pelos cenários a cada rastreamento. A coluna 4 apresenta a confiabilidade de cada cenário, ou seja, a porcentagem de rastreamentos que convergiram, dentre os 1000. O *speedup*, em relação a PSO_{SO} (PSO sem auxílio do coprocessador), pode ser observado na coluna 5 dessa tabela.

Para efeito de comparação, os gráficos de barras da Figura 62 foram produzidos. O gráfico da Figura 62(a) compara os tempos de processamento consumidos por cada cenário, considerando as soluções somente software (SO), de hardware em modo sequencial (HS) e de hardware em modo *pipeline* (HP). O gráfico da Figura 62(b) relaciona o *speedup* do PSO em relação a BE, para cada uma dessas soluções.

Observa-se que o cenário BE_{HP} foi o melhor para busca exaustiva e conseguiu realizar o rastreamento em 1728,50 ms, registrando um *speedup* de 13,36, com relação a BE_{SO}. Já o cenário PSO_{HP} foi o melhor para o PSO e conseguiu realizar o rastreamento em 15,02 ms, na média. Com esse valor, o *speedup* observado, com relação a PSO_{SO}, foi de 12,17. Comparando o PSO com a busca exaustiva, os *speedup* verificados foram de 126,28, 122,71 e 115,08 para as soluções SO, HS e HP, respectivamente.

6.5 Comparação dos Resultados

Os resultados para as imagens de referência são muito parecidos, em termos de tempo de processamento. As Figuras 63 e 64 exibe esses resultados, em gráficos de barras.

Observa-se que os cenários que utilizam solução de hardware em modo sequencial apresentam melhores resultados do que aqueles que utilizam solução somente de software.

Da mesma forma, os cenários com hardware funcionando em *pipeline* são melhores do que aqueles com hardware funcionando em modo sequencial.

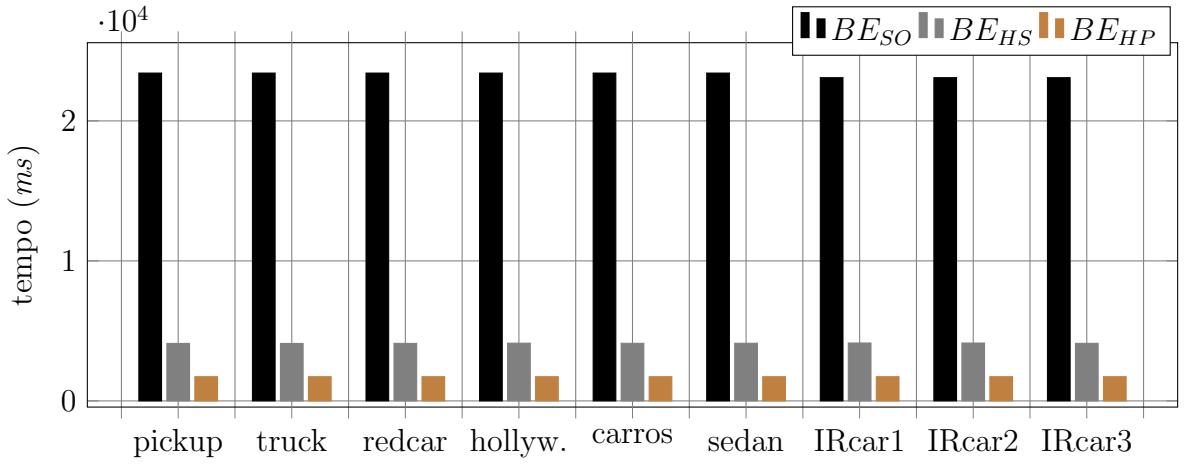


Figura 63: Tempo de processamento para as imagens de referência (BE)

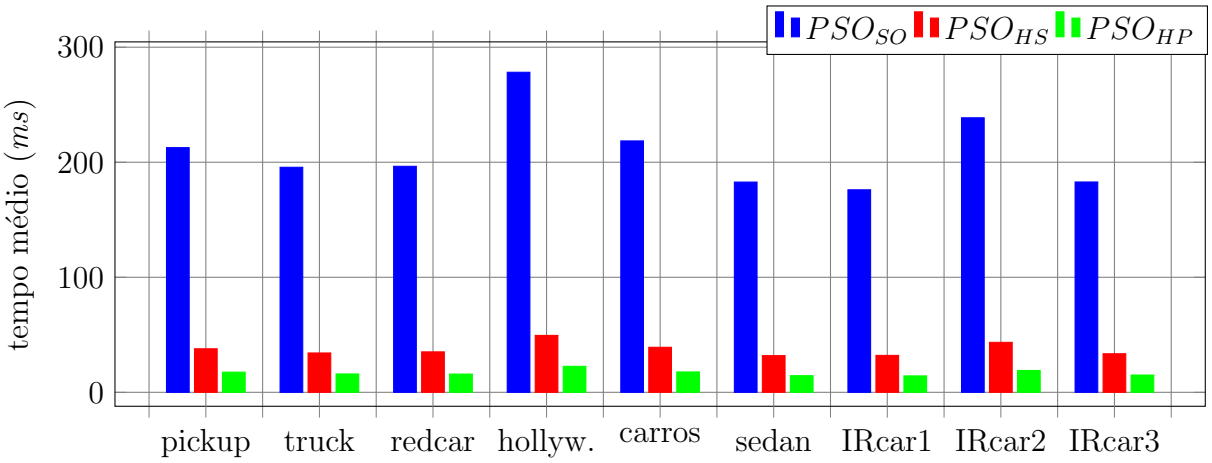


Figura 64: Tempo de processamento para as imagens de referência (PSO)

Tabela 36: *Speedup* de PSO_{HP} com relação aos outros cenários

Imagens	BE_{SO}	BE_{HS}	BE_{HP}	PSO_{SO}	PSO_{HS}
pickup	1336,57	234,40	98,65	12,14	2,16
truck	1468,12	257,35	108,36	12,27	2,14
redcar	1476,45	259,06	108,98	12,39	2,22
hollywood	1036,12	182,52	76,49	12,31	2,19
carros	1322,23	232,31	97,60	12,34	2,21
sedan	1619,41	284,90	119,55	12,64	2,20
IRcar1	1619,09	290,03	121,25	12,35	2,25
IRcar2	1216,46	217,82	91,09	12,57	2,29
IRcar3	1537,16	273,85	115,08	12,17	2,23

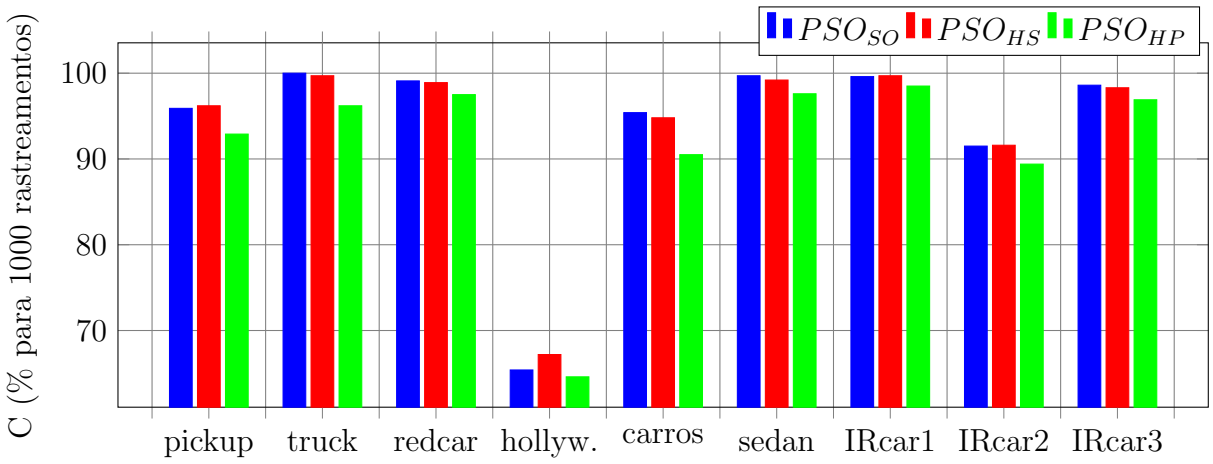


Figura 65: Confiabilidade para as imagens de referência

A utilização do coprocessador na busca exaustiva consegue diminuir o tempo de processamento para valores na casa de alguns segundos. A busca exaustiva é um método de força bruta que, mesmo com auxílio do processador, não consegue atingir o objetivo de processamento em tempo real. Confirma-se assim, a necessidade de utilização de técnicas inteligentes que otimizem o rastreamento.

A utilização do PSO somente em software consegue diminuir o tempo de processamento para valores na casa de algumas centenas de milissegundos, melhor do que qualquer cenário com BE. A união do PSO com o cálculo de PCC pelo coprocessador obtém os melhores resultados, sendo ainda a utilização do coprocessador em modo *pipeline* superior a sua utilização em modo sequencial. A Tabela 36 exibe um resumo, para todas as imagens de referência, com o *speedup* observado para PSO_{HP}, com relação aos outros cenários. No tocante a BE_{SO}, o maior *speedup* registrado foi de 1619,41, para a imagem *sedan*, e o menor foi de 1036,12, para a imagem *hollywood*. Considerando o cenário PSO_{SO}, o maior *speedup* registrado foi de 12,64, para a imagem *sedan*, e o menor foi de 12,14, para a imagem *pickup*. Esse resultado atende para aplicações em tempo real com robustez aceitável. Para melhorar ainda mais os resultados, o coprocessador pode trabalhar em frequências maiores, quando forem superadas as limitações da síntese.

O gráfico da Figura 65 resume os resultados, em termos de confiabilidade, para os cenários de PSO. Para a maior parte das imagens, a confiabilidade está acima de 89,4%, exceto para a imagem *hollywood*, que apresentou valor de 64,6%. Essa média confiabilidade se deve à característica da correlação associada a esta imagem. Observa-se na Figura 51(c) que o gráfico da correlação apresenta um pico muito estreito e muitos valores abaixo de

0,2. Esse perfil dificulta a orientação das partículas no espaço de busca. Uma possível solução para melhorar essa confiabilidade seria aumentar o número de partículas, porém, isso acarretaria um aumento no tempo de processamento de cada iteração, excedendo dos limites de processamento em tempo real estabelecidos.

6.6 Limitações do Hardware

A implementação na placa SVDK ocupou 11% de seus flip-flops, 39% de suas LUTs (*LookUp Table*), 25% dos buffers e 69% de suas BRAMs (*Block RAM*). Por limitações de tempo impostas na síntese, o coprocessador executou as tarefas com um clock de 25 MHz. Contudo, o processador ZYNQ pode fornecer para a lógica programável um clock de até 250 MHz.

Uma das maiores limitações foi o tamanho da memória BRAM para armazenar a imagem principal. Além da quantidade física disponível no chip, quanto maior essa memória, maior o tempo necessário para acessar os dados e, consequentemente, menor velocidade de trabalho. A quantidade máxima obtida, em termos de melhor custo benefício, foi a de 573x463 pixels de 1 byte, totalizando 260 Kbytes de dados. Esse tamanho pode armazenar uma imagem de, no máximo, 510x400 pixels, considerando que as bordas devem ser completadas com zeros (63 linhas e 63 colunas). Por este motivo, a maior parte das imagens de referência da Seção 6.4 tiveram suas dimensões alteradas para 510x400 pixels.

A memória para o *template* possui capacidade para armazenar 4096 dados de 1 byte, totalizando 4 Kbytes. Este tamanho equivale a imagens com 64x64 pixels e foi definido como um requisito de projeto. A posição do *template* na imagem principal é considerada aquela correspondente ao seu pixel central, conforme ilustrado na Figura 66.

Uma outra limitação, imposta pelo projeto, foi restringir o número de partículas em 50. A posição de cada partícula é armazenada em um registrador, no componente `GET_IMG`. Esses registradores são multiplexados para se obter o recorte de imagem referente a cada partícula, na imagem principal. Um aumento nesse número significa mudança na estrutura da entrada e aumento das conexões de entrada do multiplexador.

Todos os testes foram realizados considerando que as imagens avaliadas (tanto o *template* quanto a imagem principal) já haviam sido carregadas nas memórias do processador e do coprocessador, previamente. As imagens foram transmitidas para o processador

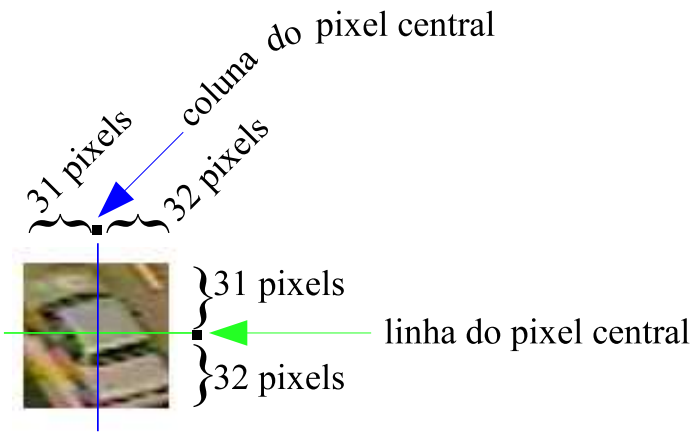


Figura 66: Dimensões e centro dos pixels nos *templates*

por meio do software MATLAB, utilizando porta serial e foram copiadas nas memórias BRAM pelo próprio processador, por meio da interface AXI4-lite. Para aplicações de vídeo, rastreamento em sucessivos *frames*, esses dois tipos de comunicação são lentos e precisariam ser modificados.

6.7 Considerações Finais do Capítulo

Neste Capítulo foram apresentados os resultados obtidos pela implementação da arquitetura proposta nos Capítulos 4 e 5 na placa SVDK. A qualidade do cálculo do coprocessador foi avaliada e apresentou boa precisão, menor que 0,000003.

A abordagem co-design proposta, com implementação do PCC em hardware e PSO em software, obteve os melhores resultados. O tempo médio de processamento consumido no rastreamento dos *templates* nas imagens é compatível com aplicações em tempo real (menor que 33 ms). Conseguiu-se acelerar esse tempo em até 12,64x, comparando-se com a solução somente de PSO em software, e em até 1619,41x, comparando-se com a busca exaustiva sem auxílio do coprocessador. A confiabilidade para este cenário ficou acima de 89,4% para a maior parte das imagens de referência, com exceção de *hollywood*, que apresentou confiabilidade de 64,6%.

Uma das maiores limitações na implementação foi o tamanho da memória BRAM para armazenar a imagem principal. Além da quantidade física disponível no chip, quanto maior essa memória, maior o tempo necessário para acessar os dados e, consequentemente, menor velocidade de trabalho. Tal fato contribuiu para impor limitações de tempo na síntese, e o coprocessador somente pôde executar as tarefas com um clock de 25 MHz.

Uma outra limitação, ligada ao projeto, foi a restrição do número de partículas para 50, no máximo.

O Capítulo seguinte finaliza este trabalho, abordando suas principais conclusões, bem como as contribuições mais relevantes da presente dissertação. Também serão tratadas direções para possíveis trabalhos futuros.

Capítulo 7

CONCLUSÕES E TRABALHOS FUTUROS

ESTE Capítulo completa esta dissertação, apresentando as principais conclusões obtidas com o desenvolvimento do presente trabalho e com a implementação do projeto proposto na placa SVDK. São apresentadas também possíveis melhorias e direções para trabalhos futuros.

7.1 Conclusões

Esta dissertação abordou a implementação da tarefa de *template matching* em um sistema embarcado, conferindo-lhe grande versatilidade, desempenho e possibilitando seu uso em equipamentos portáteis. Foram apresentados os conceitos básicos associados a detecção, classificação e rastreamento de objetos em imagens. Foi dada ênfase ao método de rastreamento por *template matching* e correlação cruzada normalizada.

Duas técnicas de inteligência computacional foram avaliadas para resolver o problema de otimização associado ao rastreamento: algoritmos genéticos e otimização por enxame de partículas (algoritmos *global best* e *local best*). Os dois algoritmos de PSO obtiveram melhores resultados que o algoritmo de GA. PSO *global best* foi melhor que *PSO local best*, em termos de eficiência (tempo de processamento) e confiabilidade.

No Capítulo 4 foi possível identificar que o cálculo de PCC é a parte mais custosa do rastreamento abordado, seguido pelo acesso a memória. Estes resultados motivaram a utilização de um projeto integrado de software/hardware que utiliza um coprocessador dedicado, para cálculo do coeficiente de correlação, uma memória local a esse coprocessador, e execução do PSO *global best* em software. Esta abordagem, chamada *co-design*,

é uma metodologia para desenvolver um sistema integrado, utilizando componentes de hardware e software, para atender requisitos de desempenho e limitações de custo.

Os resultados da implementação do projeto na placa SVDK apresentaram bom desempenho quando utilizando concomitantemente o PSO e o coprocessador no modo *pipeline*. O tempo médio de processamento consumido no rastreamento dos *templates* nas imagens é compatível com aplicações em tempo real (menor que 33 ms). Conseguiu-se acelerar esse tempo em até 12,64x, comparando-se com a solução somente de PSO em software, e em até 1619,41x, comparando-se com a busca exaustiva sem auxílio do coprocessador. A implementação do coprocessador também foi avaliada em termos de qualidade, constatando-se que o cálculo da correlação apresentou boa precisão, com erro inferior a 0,000003.

Uma das maiores limitações na implementação foi o tamanho da memória BRAM para armazenar a imagem principal. Além da quantidade física disponível no chip, quanto maior essa memória, maior o tempo necessário para acessar os dados e, consequentemente, menor velocidade de trabalho. Tal fato contribuiu para impor limitações de tempo na síntese, e o coprocessador somente pôde executar as tarefas com um clock de 25 MHz. Uma outra limitação, ligada ao projeto, foi a restrição do número de partículas para 50, no máximo.

A presente dissertação gerou a submissão de 5 artigos, do qual quatro foram aceitos para apresentação e publicação em congressos (TAVARES; NEDJAH; MOURELLE, 2015), (TAVARES; NEDJAH; MOURELLE, 2016a), (TAVARES; NEDJAH; MOURELLE, 2016b) e (TAVARES; NEDJAH; MOURELLE, 2016c). Além disso, os autores foram convidados a submeter uma versão estendida do artigo (TAVARES; NEDJAH; MOURELLE, 2015) para uma *special issue* do *International Journal of Swarm Intelligence Research* e uma do *International Journal of Bio-Inspired Computation*.

7.2 Trabalhos Futuros

Nesta Seção, são citadas algumas possíveis modificações do projeto proposto, com o intuito de melhorar seu desempenho. Também são levantadas propostas para futuros trabalhos na área de rastreamento de objetos e sistemas embarcados.

Uma primeira investigação a ser feita é a avaliação do hardware implementado em termos da limitação de frequência imposta na síntese. Os componentes deveriam ser inves-

tigados de forma a identificar qual deles é o mais crítico e propor soluções. Desta forma, o processador poderia trabalhar mais rápido e poderiam ser utilizadas mais partículas.

Uma segunda sugestão para futuros estudos é a avaliação de outras técnicas de inteligência computacional e de enxame para otimização do problema de rastreamento. O coprocessador desenvolvido calcula o PCC para os pixels de sua entrada, independente da técnica inteligente que está sendo utilizada. Para este componente, basta que sejam informados os pontos centrais no qual serão recortadas as imagens para serem comparadas com o *template*.

Uma outra opção de trabalho futuro é a modificação das interfaces de comunicação do processador com coprocessador e da placa SVDK com computador, de forma a permitir aplicações com vídeo.

Uma quarta sugestão é a pesquisa de soluções para problemas de *drift*, oclusão, mudanças de posição do objeto, iluminação e ruído, comuns em rastreamento de objetos por *template matching*.

REFERÊNCIAS

AHUJA, K.; TULI, P. Object recognition by template matching using correlations and phase angle method. *International Journal of Advanced Research in Computer and Communication Engineering*, v. 2, n. 3, 2013.

ALI, A.; KAUSAR, H.; KHAN, M. I. Automatic visual tracking and firing system for anti aircraft machine gun. In: IEEE. *Applied Sciences and Technology (IBCAST), 2009 6th International Bhurban Conference on*. Islamabad, Paquistão, 2009. p. 253–257.

AVNET. Picozed 7z015/7z030 system-on module hardware user guide, version 1.3. 2015.

BAY, H.; TUYTELAARS, T.; GOOL, L. V. Surf: Speeded up robust features. In: SPRINGER. *European conference on computer vision*. Berlim, 2006. p. 404–417.

BENFOLD, B.; REID, I. Stable multi-target tracking in real-time surveillance video. In: IEEE. *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*. Colorado, EUA, 2011. p. 3457–3464.

BLAKELOCK, J. H. *Automatic control of aircraft and missiles*. 2. ed. EUA: John Wiley & Sons, 1991.

BOWYER, K.; KRANENBURG, C.; DOUGHERTY, S. Edge detector evaluation using empirical roc curves. In: IEEE. *Computer Vision Image Understand.* Nova York, EUA, 2001. p. 77–103.

CANNY, J. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, n. 6, p. 679–698, 1986.

CASELLES, V.; KIMMEL, R.; SAPIRO, G. Geodesic active contours. In: IEEE. *Computer Vision, 1995. Proceedings., Fifth International Conference on*. Cambridge, MA, 1995. p. 694–699.

CHOI, H.; KIM, Y. Uav guidance using a monocular-vision sensor for aerial target tracking. *Control Engineering Practice*, Elsevier, v. 22, p. 10–19, 2014.

CHU, P. P. *FPGA prototyping by VHDL examples: Xilinx Spartan-3 version*. EUA: John Wiley & Sons, 2008.

COLLINS, R.; ZHOU, X.; TEH, S. K. An open source tracking testbed and evaluation web site: http://vision.cse.psu.edu/data/vivideval/datasets/datasets.html. *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*, v. 2, n. 6, p. 35, 2005.

COMANICIU, D.; MEER, P. Mean shift analysis and applications. In: IEEE. *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*. Kerkyra, 1999. v. 2, p. 1197–1203.

COMANICIU, D.; RAMESH, V.; MEER, P. Kernel-based object tracking. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 25, n. 5, p. 564–577, 2003.

DEORI, B.; THOUNAOJAM, D. M. A survey on moving object tracking in video. *International Journal on Information Theory (IJIT)*, v. 3, n. 3, p. 31–46, 2014.

DORIGO, M. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

ENGELBRECHT, A. P. *Fundamentals of computational swarm intelligence*. West Sussex, Inglaterra: John Wiley & Sons, 2006.

ENGELBRECHT, A. P. *Computational intelligence: an introduction*. 2. ed. West Sussex, Inglaterra: John Wiley & Sons, 2007.

FAN, B. et al. A robust template tracking algorithm with weighted active drift correction. *Pattern Recognition Letters*, Elsevier, v. 32, n. 9, p. 1317–1327, 2011.

FORLENZA, L. et al. Flight performance analysis of an image processing algorithm for integrated sense-and-avoid systems. *International Journal of Aerospace Engineering*, Hindawi Publishing Corporation, v. 2012, 2012.

GONZALEZ, R.; WOODS, R. *Digital Image Processing*. 3. ed. EUA: Pearson Prentice-Hall, 2008.

GONZALEZ, R.; WOODS, R.; EDDINS, S. *Digital Image Processing using Matlab*. EUA: Pearson Prentice-Hall, 2004.

GONZALEZ, R. C.; WOODS, R. E. *Processamento de imagens digitais*. Brasil: Edgard Blucher, 2000.

GREENSPAN, H. et al. Overcomplete steerable pyramid filters and rotation invariance. *Computer Vision and Pattern Recognition. IEEE Computer Society Conference on*, p. 222–228, 1994.

GREWE, L.; KAK, A. C. Interactive learning of a multiple-attribute hash table classifier for fast object recognition. *Computer Vision and Image Understanding*, Elsevier, v. 61, n. 3, p. 387–416, 1995.

HARALICK, R. M.; SHANMUGAM, B.; DINSTEIN, I. Textural features for image classification. *IEEE Transactions on systems, man, and cybernetics*, IEEE, p. 610–621, 1973.

HOLLAND, J. H. Adaptation in natural and artificial systems. *University of Michigan Press*, 1975.

HUTTENLOCHER, D. P.; NOH, J. J.; RUCKLIDGE, W. J. Tracking non-rigid objects in complex scenes. *Computer Vision. Proceedings., Fourth International Conference on*, p. 93–101, 1993.

KARABOGA, D. An idea based on honey bee swarm for numerical optimization. *Technical report TR06, Erciyes university*, 2005.

KASS, M.; WITKIN, A.; TERZOPOULOS, D. Snakes: Active contour models. *International journal of computer vision*, Holanda, v. 1, p. 321–331, 1988.

KENNEDY, J.; EBERHART, R. Particle swarm optimization. In: *IEEE International Conference on Neural Network*. EUA: [s.n.], 1995. v. 4, p. 1942–1948.

KORETSKY, G.; NICOLL, J.; TAYLOR, M. *A tutorial on electro-optical/infrared (EO/IR) theory and systems*. Alexandria, Virginia, 2013.

LAWS, K. *Textured image segmentation*. Tese (Doutorado) — University of Southern California, 1980.

LOWE, D. G. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, Springer, v. 60, n. 2, p. 91–110, 2004.

LUCAS, B. D.; KANADE, T. et al. An iterative image registration technique with an application to stereo vision. *IJCAI*, v. 81, p. 674–679, 1981.

MAHALAKSHMI, T. et al. Review article: an overview of template matching technique in image processing. *Research Journal of Applied Sciences, Engineering and Technology*, v. 4, n. 24, p. 5469–5473, 2012.

MALLAT, S. G. A theory for multiresolution signal decomposition: the wavelet representation. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 11, n. 7, p. 674–693, 1989.

MATTHEWS, I.; ISHIKAWA, T.; BAKER, S. The template update problem. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, IEEE, v. 26, n. 6, p. 810–815, 2004.

MIKOLAJCZYK, K.; SCHMID, C. An affine invariant interest point detector. *European conference on computer vision (ECCV)*, v. 1, p. 128–142, 2002.

MIRANDA, A. N. Pearson's correlation coefficient: A more realistic threshold for applications on autonomous robotics. *Computer Technology and Application*, David Publishing Company, Inc., v. 5, p. 69–72, 2014.

MITCHELL, M. Genetic algorithms: An overview. *Complexity*, Wiley Online Library, v. 1, n. 1, p. 31–39, 1995.

NARAYANA, M. *Automatic Tracking of Moving Objects in Video for Surveillance Applications*. Tese (Doutorado) — University of Kansas, 2007.

NEDJAH, N.; MOURELLE, L. de M. *Co-design for system acceleration: a quantitative approach*. Polonia: Springer Science & Business Media, 2007.

NEDJAH, N.; MOURELLE, L. de M. *Hardware for soft computing and soft computing for hardware*. Polonia: Springer, 2014.

NETO, A. de M. Pearson's correlation coefficient: A more realistic threshold for applications on autonomous robotics. *Computer Technology and Application*, David Publishing Company, Inc., v. 5, n. 2, 2014.

NGO, H. T. et al. Real-time video surveillance on an embedded, programmable platform. *Microprocessors and Microsystems*, Elsevier, v. 37, n. 6, p. 562–571, 2013.

NIXON, M. S.; AGUADO, A. S. *Feature extraction and image processing*. 1. ed. Great Britain: Academic Press, 2002.

OLIVER, N. M.; ROSARIO, B.; PENTLAND, A. P. A bayesian computer vision system for modeling human interactions. *IEEE transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 8, p. 831–843, 2000.

OLSON, T. L.; SANFORD, C. W. Real-time multistage IR image-based tracker. *AeroSense'99*, International Society for Optics and Photonics, p. 226–233, 1999.

PAPAGEORGIOU, C. P.; OREN, M.; POGGIO, T. A general framework for object detection. In: IEEE. *Computer vision, 1998. sixth international conference on*. Bombai, 1998. p. 555–562.

PERVEEN, N.; KUMAR, D.; BHARDWAJ, I. An overview on template matching methodologies and its applications. *International Journal of Research in Computer and Communication Technology*, v. 2, n. 10, p. 988–995, 2013.

PRAJAPATI, D.; GALIYAWALA, H. J. A review on moving object detection and tracking. *International Journal of Computer Application*, v. 5, n. 3, p. 168–175, 2015.

ROWLEY, H. A.; BALUJA, S.; KANADE, T. Neural network-based face detection. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 20, n. 1, p. 23–38, 1998.

SCHREIBER, D. Robust template tracking with drift correction. *Pattern recognition letters*, Elsevier, v. 28, n. 12, p. 1483–1491, 2007.

SENSORTOIMAGE. *SVDK Hardware User Guide, revision 1.1*. Schongau, Alemanha, 2015.

SHARMA, P.; KAUR, M. Classification in pattern recognition: A review. *International Journal of Advanced Research in Computer Science and Software Engineering*, v. 3, n. 4, p. 298–306, 2013.

SHI, J.; MALIK, J. Normalized cuts and image segmentation. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 8, p. 888–905, 2000.

SHI, Y.; EBERHART, R. A modified particle swarm optimizer. In: IEEE. *IEEE World Congress on Computational Intelligence*. Anchorage, AK, 1998. p. 69–73.

SINDHUJA, G.; RENUKA, D. S. M. A survey on detection and tracking of objects in video sequence. *International Journal of Engineering Research and General Science*, v. 3, n. 2, p. 418–426, 2015.

SIOURIS, G. M. *Missile guidance and control systems*. Nova York, EUA: Springer Science & Business Media, 2004.

STAUFFER, C.; GRIMSON, W. E. L. Learning patterns of activity using real-time tracking. *IEEE Transactions on pattern analysis and machine intelligence*, IEEE, v. 22, n. 8, p. 747–757, 2000.

SUN, J. A fast meanshift algorithm-based target tracking system. *Sensors*, Molecular Diversity Preservation International, v. 12, n. 6, p. 8218–8235, 2012.

TAVARES, Y. M.; NEDJAH, N.; MOURELLE, L. de M. Utilização de otimização por enxame de partículas e algoritmos genéticos em rastreamento de padrões. In: *Anais do 12º Congresso Brasileiro de Inteligência Computacional*. Curitiba, PR: ABRICOM, 2015. p. 1–6.

TAVARES, Y. M.; NEDJAH, N.; MOURELLE, L. de M. Embedded implementation of template matching using correlation and particle swarm optimization. In: SPRINGER. *International Conference on Computational Science and Its Applications*. Beijing, China, 2016. p. 530–539.

TAVARES, Y. M.; NEDJAH, N.; MOURELLE, L. de M. Hardware/software co-design system for template matching using particle swarm optimization and pearson's correlation coefficient. In: *Latin American Conference on Computational Intelligence*. Colômbia: ABRICOM, 2016.

TAVARES, Y. M.; NEDJAH, N.; MOURELLE, L. de M. Projeto integrado para rastreamento de padrões utilizando otimização por enxame de partículas e coeficiente de correlação de pearson. In: *21 Congresso Brasileiro de Automática*. Vitória, ES: SBA, 2016.

VINOD, G.; ANITTA, R. Implementation of FFT based automatic image mosaicing. *International Journal of Advanced Research in Electrical, Electronics and Instrumentation Engineering*, v. 2, n. 12, p. 6002–6009, 2013.

VIOLA, P.; JONES, M. J.; SNOW, D. Detecting pedestrians using patterns of motion and appearance. In: *IEEE International Conference on Computer Vision (ICCV)*. Nice, França: IEEE, 2003. v. 2, p. 734–741.

WATANABE, S. *Pattern recognition: human and mechanical*. Nova York, EUA: John Wiley & Sons, Inc., 1985.

WHITLEY, D. A genetic algorithm tutorial. *Statistics and computing*, Springer, v. 4, n. 2, p. 65–85, 1994.

XILINX. Ug585 zynq-7000 AP SOC technical reference manual, version 1.10. 2015.

YILMAZ, A.; JAVED, O.; SHAH, M. Object tracking: A survey. *Acm computing surveys (CSUR)*, v. 38, n. 4, p. 1–45, 2006.

YOUTUBE. *Rafale - Avião Caça de alta Tecnologia (Brasil)*. Agosto 2016. Disponível em: $<http://www.youtube.com/watch?v=e3wi\text{-}i\_hDVQ>$.