Pós-Graduação em Ciência da Computação

ÉRIKA SPENCER DE ALBUQUERQUE

DESENVOLVIMENTO DE UM MÓDULO PARA TEMPLATE MATCHING BASEADO EM ZNCC COM PROTOTIPAÇÃO EM FPGA.

Universidade Federal de Pernambuco
posgraduacao@cin.ufpe.br
www.cin.ufpe.br/~posgraduacao

RECIFE

2017

ÉRIKA SPENCER DE ALBUQUERQUE

"DESENVOLVIMENTO DE UM MÓDULO PARA TEMPLATE MATCHING BASEADO EM ZNCC COM PROTOTIPAÇÃO EM FPGA."

Trabalho apresentado ao Programa de Pós-graduação em Ciência da Computação do Centro de Informática da Universidade Federal de Pernambuco como requisito parcial para obtenção do grau de Mestre em Ciência da Computação.

Orientadora: *Edna Natividade da Silva Barros*

RECIFE
2017

Catalogação na fonte
Bibliotecário Jefferson Luiz Alves Nazareno CRB 4-1758

A345d Albuquerque, Érika Spencer de.
Desenvolvimento de um módulo para template matching baseado em ZNCC com prototipação em FPGA / Érika Spencer de Albuquerque – 2017.
97: fig., tab.

Orientadora: Edna Natividade da Silva Barros
Dissertação (Mestrado) – Universidade Federal de Pernambuco. CIn. Ciência da Computação, Recife, 2017.
Inclui referências.

1. Ciência da computação. 2. Hardware. 3. Visão por computador. I. Barros, Edna Natividade da Silva. (Orientador). II. Titulo.

004 CDD (22. ed.) UFPE-MEI 2017-194

Érika Spencer de Albuquerque

Desenvolvimento de um Módulo para Template Matching Baseado em ZNCC com Prototipação em FPGA

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de Pernambuco, como requisito parcial para a obtenção do título de Mestre em Ciência da Computação.

Aprovado em: 30/06/2017

BANCA EXAMINADORA

_____

Prof. Dr. Carlos Alexandre Barros de Mello
Centro de Informática / UFPE

_____

Prof. Dr. Elmar Uwe Kurt Melcher
Centro de Engenharia Elétrica e Informática / UFCG

_____

Profa. Dra. Edna Natividade da Silva Barros
Centro de Informática / UFPE
(Orientadora)

Dedico essa dissertação à minha mãe, que sempre apoiou os meus projetos.

Agradecimentos

Esse projeto não teria sido possível sem o apoio incondicional da minha família, devo agradecimentos especiais a minha mãe, Taciana, ao meu irmão, Henrique, e ao meu marido Roberto, obrigada por me apoiarem sempre, por sonharem junto comigo e também por suportarem as minhas ausências, a carga foi mais leve graças a vocês.

Agradeço a todo o suporte recebido da minha família pernambucana, obrigada por me apoiarem do início ao fim desse projeto, Adriana, Raul, Carolina, Márcio, Marcelo e Maria.

Agradeço a gerência e os conselhos valiosos de Antonyus Pyetro e João Paulo. Também não posso deixar de agradecer as pessoas que contribuíram diretamente para esse projeto, João Gabriel, Josivan, Severivo José, Djeefther, obrigada por enriquecerem esse trabalho.

Quero também agradecer as amizades que o LINCS me presenteou e que também contribuíram nessa jornada, Cecil, Rodrigo, Henrique, Hugo, Maria, Igor, Vanessa, Jefferson, Lucas, obrigada.

Finalmente quero agradecer aos mestres, essenciais a minha formação, em especial a minha orientadora a professora Edna por todos os conselhos e oportunidades que recebi, por ser um grande exemplo de professora pesquisadora e por cuidar tão bem dos alunos e do LINCS. Devo ainda um agradecimento aos professores do centro de Informática que tanto contribuíram na minha formação. Obrigada Professores Manuel, Abel, Carlos e Edna por todas as lições passadas.

"A essência do conhecimento consiste em aplicá-lo, uma vez possuído."

—CONFÚCIO

Resumo

Template matching ou casamento de padrões é um problema clássico de visão computacional, soluções para esse problema se aplicam a reconhecimento, detecção e rastreamento de objetos. O casamento de padrões consiste em buscar regiões de uma imagem fonte que mais se assemelham a uma imagem menor de referência (*template*). Uma abordagem para realizar essa busca baseia-se em comparar, através de uma medida de similaridade, a imagem de referencia com cada janela de mesma dimensão da imagem fonte. A métrica Correlação Cruzada Normalizada de Média Zero (ZNCC) é uma medida de similaridade amplamente utilizada em problemas de casamento de padrões devido a sua robustez a variações lineares de brilho e contraste. O principal desafio para o casamento de padrões, especialmente usando a métrica ZNCC é o alto custo computacional de calcular os valores de ZNCC referentes a cada janela de imagem. Há ainda, aplicações que requerem o casamento de padrões para múltiplos padrões (*templates*), como por exemplo, o rastreio de múltiplos objetos independentes ou de múltiplas poses do mesmo objeto , isso multiplica o custo computacional da operação, tornando difícil a obtenção de uma solução em tempo real. Esse trabalho propõe uma arquitetura de módulo em hardware com prototipação em FPGA que explora conceitos de paralelismo e pipeline para acelerar o cálculo da ZNCC entre uma imagem e múltiplos padrões. Resultados experimentais mostram que o módulo proposto chega a acelerar em 3x o tempo de processamento comparado às implementações em GPU e CPU. Além disso, o acelerador proposto alcança um dempenho de tempo real (32.13FPS) para o processamento de até 10 templates (Imagem 432x432 e template 72x144) (ALBUQUERQUE et al., 2016).

Palavras-chave: FPGA. Rastreamento de Objetos. ZNCC. Template Matching. Visão Computacional.

Abstract

Template matching is a well known computer vision problem. Its solutions can be aplied in object recognition, detection and tracking applications. An algorithm to solve template matching problem consists in looking for areas of an image that more closely resemble a smaller image of reference (template). Its operation is based on calculating the similarity or dissimilarity between the template and each region of image that it can overlay (image window). The metric Zero Mean Normalized Cross correlation (ZNCC) is a measure of similarity widely used in template matching problems due to its robustness to linear variations of brightness and contrast. The main disadvantage of the template matching technique, especially using ZNCC metric is the high computational cost of the calculation. Some applications, such as screening of multiple independent objects or multiple poses of the same subject require finding the best matching positions with multiple templates, which increases the computational cost of operation. A real time solution for multiple template matching is hard to be obtained. This paper proposes a coprocessor prototyped in FPGA that explores concepts of parallelism and pipeline to speed up the calculation of ZNCC between an image and multiple templates. Experimental results shows a 3x speedup comparing FPGA performance to implementations on GPU and CPU. Furthermore, the proposed accelerator achieves real-time performance (32.13FPS) for processing templates up to 10 (Image 432x432 and template 72x144) (ALBUQUERQUE et al., 2016).

Keywords: FPGA. Object Tracking. Computer Vision. Template Matching. Embedded Systems.

Lista de Figuras

Lista de Tabelas

Lista de Acrônimos

Sumário

1

INTRODUÇÃO

Esta dissertação tem como objetivo o desenvolvimento de um módulo em hardware descrito em *System Verilog* e prototipado em FPGA capaz de executar o cálculo da métrica Correlação Cruzada Normalizada com Média Zero (*ZNCC*) *Multitemplate* de forma eficiente, quando comparada a outras implementações. A motivação principal deste trabalho é disponibilizar o módulo em *System Verilog* para ser usado em diversas aplicações no campo da visão computacional, dentre essas aplicações o estudo de caso deste trabalho foi executar um algoritmo de rastreamento de objetos baseado em busca por *multiplos templates*, de forma eficiente, usando um sistema híbrido formado por CPU e FPGA. Este capítulo apresenta de uma forma geral os conceitos relacionados ao tema, *template matching*, a motivação do trabalho, os objetivos, os principais resultados e a organização do texto.

1.1 Motivação

Ver é descobrir através de imagens o que existe ao redor e onde cada coisa está *(MART, 1982)*. O processo de percepção visual que nos permite assimilar e interpretar o ambiente por meio de informações visuais é um processo complexo que envolve estruturas fisiológicas e habilidades cognitivas.

A emulação desse processo de percepção por sistemas computacionais é o campo de estudo da visão computacional. Essa área de pesquisa trouxe grandes avanços aos sistemas computacionais permitindo que estes extraiam informações a partir de imagens ou vídeos.

Extrair informações de imagens ou vídeo é uma importante etapa do processamento na análise de imagens médicas *(CHEN, 2014)*, de imagens espaciais *(HUMENBERGER et al., 2010)*, no auxílio à navegação em veículos autônomos *(BRISTEAU et al., 2011)*, além de ser fundamental em sistemas de segurança complexos *(JONES; PARAGIOS; REGAZZONI, 2012)* e em diversas outras aplicações.

Uma etapa recorrente na análise de imagens e vídeos estudada no campo da visão computacional, é a busca de um objeto conhecido em uma imagem. O casamento de padrões ou *Template matching* é o problema de localizar um objeto representado por uma imagem menor (*Template*) em uma cena, imagem maior.

Uma maneira de solucionar esse problema fazer uma aálise de similaridade por janela deslizante. Isto é, eleger uma métrica que quantifique a semelhança entre o padrão (*template*) e cada região da imagem com dimensões iguais as do template. Assim é possível escolher, quantitativamente, a janela de imagem mais parecida com o *template*.

Um problema comum com o uso dessa técnica é uma variação de iluminação e contraste entre a imagem de referência (*template*) e a imagem onde este está sendo buscado. Felizmente, para solucionar esse tipo de problema existem métricas de similaridade robustas a variações lineares de iluminação e contraste como a correlação normalizada com média zero (ZNCC) *(NARASIMHAN; NAYAR, 2003)*. O principal contraponto de escolher essa métrica é o alto custo computacional de calculá-la *(GHARAVI-ALKHANSARI, 2001)*.

O casamento de padrões por análise de similaridade é usado em sistemas de visão computacional em geral, dentre as aplicações de maior destaque tem-se o reconhecimento de digitais *(LINDOSO; ENTRENA, 2007)*, a detecção de objetos *(BENNAMOUN; MAMIC, 2012)*, o reconhecimento facial *(BRUNELLI; POGGIO, 1993)*, a detecção de defeitos em linha de produção *(TSAI; LIN, 2003)*, a visão estereoscópica *(QAYYUMA et al., 2015)* e o rastreamento de objetos*(MISHRA et al., 2013)*.

A detecção ou rastreamento de múltiplos objetos independentes, como a realizada por *YANG; DURAISWAMI; DAVIS (2005)* exige a busca simultânea por múltiplos *templates* . Essa busca por múltiplos *templates* também é usada para solucionar o problema de mudança de aparência durante o rastreamento, *(TATE; NORTHERN III, 2008)*, *(SANG; LIAO; YUAN, 2011)*, *(MAHMOOD; KHAN, 2010)*, *(CUI et al., 2007)*.

Sistemas de rastreamento ou de detecção de objetos em tempo real devem combinar alta precisão e baixo tempo de processamento *(KURUPPU et al., 2013)*. Erros de posição podem gerar um deslocamento da câmera ou da Região de Interesse (*ROI*) que podem ocasionar na perda do objeto procurado. Um processamento lento pode inviabilizar o rastreamento de objetos em movimento rápido.

A principal dificuldade de implementar algoritmos que necessitam de múltiplos resultados de *template matching* baseado em ZNCC é o tempo de processamento. O casamento de padrões baseado em ZNCC, com um padrão (template) já é considerado de alto custo computacional. Quando a abordagem é multitemplate, o tempo de processamento pode se tornar inviável para aplicações de tempo real.

Implementações paralelas usando hardware reconfigurável podem apresentar um desempenho semelhante ou até mesmo superior aos sistemas computacionais convencionais para aplicações específicas com a vantagem de serem mais facilmente embarcados em outros sistemas, muitas vezes com menor custo e menor consumo de energia.

Os trabalhos de *(CHEN et al., 2012)*, *(HASHIMOTO; ITO; NAKANO, 2013)*, *(SANG; LIAO; YUAN, 2011)*, *(TATE; NORTHERN III, 2008)* apresentam diferentes abordagens para o casamento de padrões usando tecnologias de Field Programmable Gate Array (*FPGA*) para prototipação. Os dois primeiros apresentam soluções para um único *template* e os dois últimos para múltiplos *templates*. Esse trabalhos indicam a viabilidade de acelerar a execução desse cálculo através da implementação em hardware com prototipação em *FPGA*. Contudo, nenhum desses trabalhos apresentou uma implementação eficiente para executar da métrica *ZNCC* de forma precisa para múltiplos *templates*.

1.2 Objetivos do trabalho

O objetivo deste trabalho foi desenvolver e implementar uma arquitetura de um módulo de hardware prototipada em FPGA para acelerar o casamento de padrões, com análise de similaridade por janela deslizante e mutiplus *templates* em paralelo, usando a métrica de correlação cruzada normalizada de média zero (ZNCC). Esse módulo deve permitir a execução em tempo real (acima de 30FPS) do cálculo das matrizes de resultados, sem diminuir a precisão dos resultados.

O objetivo principal deste trabalho foi alcançado através de alguns objetivos secundários essenciais ao desenvolvimento do projeto:

- Desenvolvimento de uma arquitetura para o cálculo da *ZNCC*, explorando paralelismo no processamento de cada frame.
- Implementação de um módulo em hardware para o cálculo da ZNCC entre múltiplos templates e uma imagem.
- Realização da verificação funcional módulo proposto.

- Execução de testes com o módulo proposto prototipado em *FPGA* e integrado com um micro processador em uma plataforma de prototipação.
- Comparação entre os tempos de processamento da arquitetura proposta com outras implementações do cálculo da ZNCC executadas em processador de propósito geral Unidade Central de Processamento (*CPU*) ou em uma Unidade Gráfica de Processamento (*GPU*) .

1.3 Organização da dissertação

Essa dissertação está organizada em sete capítulos. O segundo capítulo apresenta a fundamentação teórica necessária para compreensão do trabalho descrito neste documento. O terceiro capítulo apresenta os principais trabalhos relacionados da área, os trabalhos de base e os do estado da arte, que foram mais importantes para o projeto. O texto destaca, ainda, os aspectos mais relevantes de cada trabalho e faz uma comparação entre eles.

O quarto capítulo, descreve a arquitetura proposta por esse trabalho para o cálculo das matrizes de ZNCC multitemplate com a descrição interna de cada módulo e explicação detalhada do seu funcionamento. O quinto capítulo aborda aspectos práticos da implementação em FPGA.

O sexto capítulo apresenta os experimentos feitos usando a arquitetura aqui desenvolvida e os resultados obtidos, este capítulo apresenta os testes de qualidade e desempenho da arquitetura proposta. Finalmente, o sétimo capítulo elenca algumas conclusões e sugere trabalhos futuros que darão continuidade ao projeto.

2

FUNDAMENTAÇÃO TEÓRICA

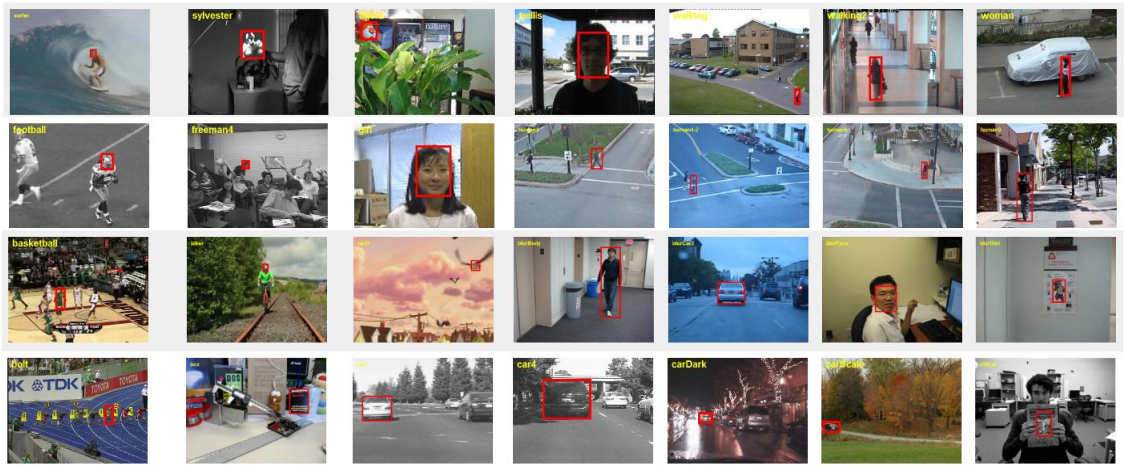
Neste capítulo são apresentadas as bases teóricas que deram origem ao trabalho. A compreensão dos conceitos detalhados neste capítulo facilita a leitura dos próximos capítulos. Inicialmente é apresentada uma visão geral do rastreamento de objetos em vídeo, em seguida há uma explicação sobre *template matching* por janela deslizante, as medidas de comparação entre imagens serão analisadas justificando a escolha da *ZNCC* neste projeto e finalmente é apresentada uma visão superficial do algoritmo de rastreamento que motivou a realização deste trabalho.

2.1 Rastreamento de objetos em vídeo

O rastreamento de objetos em vídeo é uma área importante da visão computacional, com aumento na disponibilidade de câmeras de vídeo de boa qualidade e baixo custo, cresceu a demanda por algoritmos de processamento automático de vídeo. Os algoritmos de rastreamento de objetos em vídeo atraem uma atenção especial devido a vasta gama de aplicações onde eles são necessários.

O Rastreamento por vídeo é o problema de estimar a trajetória de um objeto no plano da imagem enquanto ele se move ao redor da cena. A trajetória é o conjunto das coordenadas ao longo do tempo *(YILMAZ; JAVED; SHAH, 2006)*.

Figura 2.1: Rastreamento em vídeo



Fonte: *(WU; LIM; YANG, 2013)*

Algumas das aplicações mais populares do rastreamento em vídeo são: monitoramento de vídeos de segurança, monitoramento de tráfico com rastreamento de veículos, rastreamento de pedestres em ambientes externos, interação homem máquina por gestos, navegação automática de veículos com desvio de obstáculos.

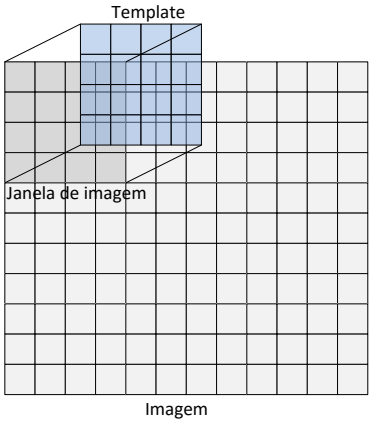
O rastreamento em vídeo também pode ser visto como a tarefa de localizar um ou mais objetos em cada *frame* do vídeo, pelas coordenadas dos pixels no *frame*. Neste trabalho o casamento de padrões foi abordado urilizando a análise de similaridade por janela deslizante para localizar o objeto, *template*, em cada *frame* , a aplicação alvo desse trabalho é o rastreamento de pedestres em ambientes externos.

Os principais desafios do rastreamento de pedestres em ambientes externos são as mudanças de iluminação e contraste, as quais os ambientes externos estão sujeitos, e as mudanças de pose do pedestre.

2.2 Casamento de padrões utilizando análise de similaridade por janela deslizante

A análise de similaridade por janela deslizante é muito usada em aplicações de visão computacional para encontrar a posição de um modelo, o *template*, em uma imagem maior. A busca consiste em deslizar o *template* sobre a imagem e encontrar a posição da imagem que o *template* mais se assemelha à janela de imagem sobreposta (Figura 2.2). Para cada posição do *template* sobre a imagem é calculada um resultado de métrica de comparação entre os pixels do *template* e os da janela de imagem, que este sobrepõe.

Figura 2.2: Ilustração de execução de Template matching: Imagem teplate e janela de imagem



A Figura 2.3 mostra um exemplo de resultado de análise de similaridade. O *template* destacado no canto superior esquerdo é deslizado, pixel a pixel, da direita para a esquerda e de cima para baixo, cobrindo todas as posições da imagem. A cada posição é calculado um resultado de similaridade entre o *template* e a janela de imagem que este sobrepõe. O conjunto desses resultados de similaridade é mostrado no gráfico do lado esquerdo. Cada ponto do gráfico ($R_{x,y}$) foi obtido por uma relação entre os pixels do *template* e os pixels da janela de imagem que tem origem nas coordenadas do ponto (x,y) e com dimensões iguais as do *template*.

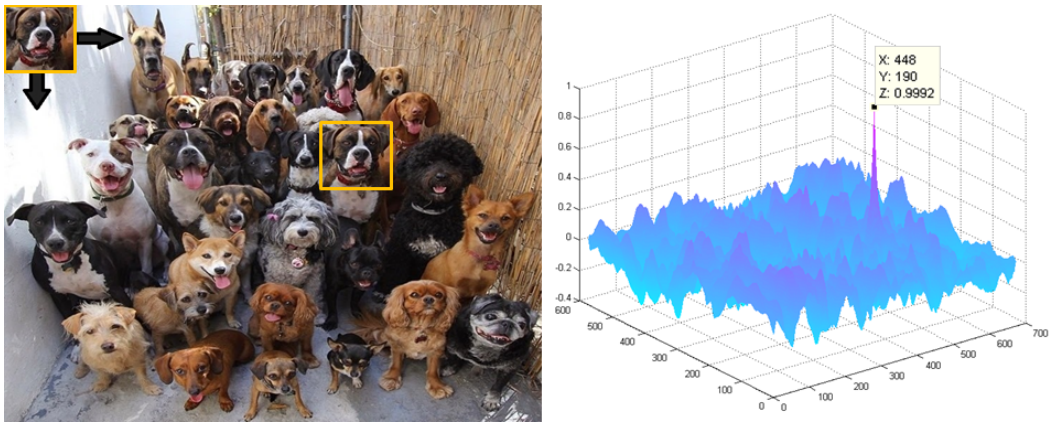
No gráfico está destacado o ponto de máximo, os valores de x e y indicam as coordenadas da origem da janela de imagem que mais se assemelha com o *template*, o valor 'z' indica o grau de similaridade entre a janela de imagem e o *template* em uma escala de -1 a 1.

A busca pelo objeto em cada *frame* é feita com base em um modelo do objeto, o *template*, mas a aparência do objeto muda ao longo do tempo e isso pode atrapalhar o rastreamento.

O rastreamento deve funcionar mesmo com mudança de iluminação o que é um desafio, principalmente se a métrica usada, para comparar o *template* com a janela de imagem, for muito sensível às variações de iluminação e contraste.

Outro obstáculo ao bom funcionamento do rastreamento é que o objeto é tridimensional, se move em um espaço tridimensional, e o vídeo é uma representação bidimensional da cena, assim o objeto pode mudar sua aparência mudando de pose. Considerar múltiplas representações bidimensionais do objeto pode aumentar a robustez do rastreamento.

Figura 2.3: (a) Exemplo de *template* (canto esquerdo superior) e imagem. (b) Gráfico mostrando valores de similaridade em cada posição da imagem.



A escolha de um limiar numérico na métrica de comparação para determinar se o *template* buscado se encontra na cena também é uma dificuldade de projeto. Um limiar muito restritivo pode resultar em muitos falsos negativos, quando o objeto está na cena e o rastreador considera que ela não está. Um limiar muito abrangente pode aumentar os falsos positivos, quando o objeto não está na cena, ou está sob oclusão, e o rastreamento passa a seguir um outro objeto da cena com algum grau de similaridade com o objeto buscado.

2.3 Métricas para comparação

A análise comparativa por janela deslizante gera uma matriz de resultados que é composta por valores quantitativos que representam a comparação entre os pixels do *template* e os da janela de imagem. As métricas para comparação podem ser dadas por medidas de similaridade ou de dissimilaridade. As métricas mais comuns para esse propósito são baseadas na soma das diferenças (dissimilaridade) ou na correlação cruzada (similaridade).

Esta seção apresenta as principais formas de calcular as métricas de comparação. Essas métricas foram analisadas para a escolha da métrica mais adequada a este trabalho. As fórmulas, aqui apresentadas, usam variáveis da imagem e do *template*, dessa forma vamos esclarecer a notação usada para facilitar a leitura.

Cada valor na matriz de resultados é um R(x,y), sendo ‘x’ o número da linha e ‘y’ o número da coluna, na matriz. O valor de cada pixel na imagem é dado por I(x,y), e no *template* é dado por T(x,y). Cada janela de imagem é uma região da imagem com origem na posição (x,y) e dimensões iguais às do *template*, como mostrado na Figura 4.2. As variáveis ‘i’ e ‘j’ são usadas para percorrer, respectivamente, as linhas e colunas, tanto no *template* como na janela de imagem. ‘N’ é o número total de pixels do *template* (N=i.j).

2.3.1 SOMA DOS QUADRADOS DAS DIFERENÇAS (*SSD*)

Uma das formas mais intuitivas de comparar duas imagens é subtrair uma imagem da outra, já que imagens iguais geram resultados nulos. A métrica *SSD*, mostrada na Equação 2.1, se baseia nesse conceito para calcular valores de dissimilaridade entre o *template* e cada janela de imagem, com a mesma dimensão do *template*, contidas na imagem fonte. Para calcular um resultado dessa matriz basta selecionar a janela de imagem na posição correspondente e subtrair, dos pixels dessa janela de imagem os respectivos pixels do*template*. Depois da subtração, eleva-se o erro obtido ao quadrado para que os valores sempre sejam positivos. O acumulado desses erros quadráticos é o resultado *SSD* na posição correspondente. Conforme exemplo mostrado na Figura 2.6.

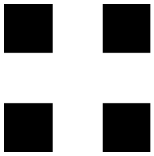
Quanto mais parecida for a janela de imagem com o *template* menor será o valor do resultado *SSD*, dado por R(x,y) na Equação 2.1.

$$R(x,y) = \sum_{i,j}^{N} (I(x+i, y+j) - T(i,j))^2 \tag{2.1}$$

Um exemplo com imagens binárias (intensidade dos pixels é '0' ou '1') de pequenas dimensões, será usado para entender melhor como as matrizes de resultado são obtidas. Nesse exemplo vamos buscar o *template* 3x3 mostrado na Figura 2.4. A busca será feita sobre a imagem 10x10 mostrada na Figura 2.5.

Usando imagens binárias há duas possibilidades de resultados para cada erro quadrático entre dois pixels, '0' se os pixels forem iguais e '1' se eles forem diferentes. Assim o valor de cada posição da matriz de resultados, dado pelo somatório dos erros, pode variar de '0' a 'N', sendo 'N' a quantidade de pixels do *template*. No exemplo 'N' é igual a 9. Os resultados '0' e 'N' correspondem, respectivamente, a uma janela de imagem exatamente igual ao *template* (dissimilaridade nula) e a janela de imagem completamente oposta ao *template* (dissimilaridade máxima).

Figura 2.4: Template binário 3x3



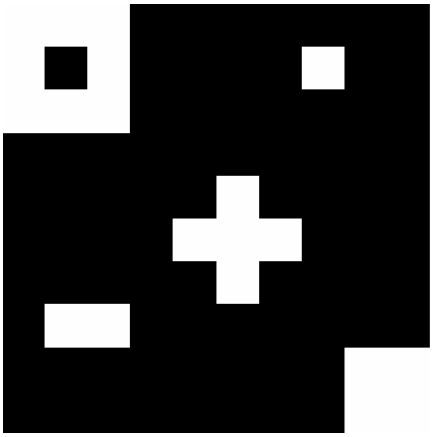
(a) Template: visualização gráfica

0	1	0
1	1	1
0	1	0

(b) Template: Matriz de pixels correspondentes

A Figura 2.6 mostra o cálculo de SSD entre a primeira janela de imagem e o *template*, correspondente a primeira posição da matriz de resultados. Seguindo o cálculo da direita para a esquerda e de cima para baixo até o trigésimo terceiro resultado (posição 5,5) encontramos a posição que possui a maior semelhança entre o *template* e a janela de imagem, como mostrado na figura 2.7

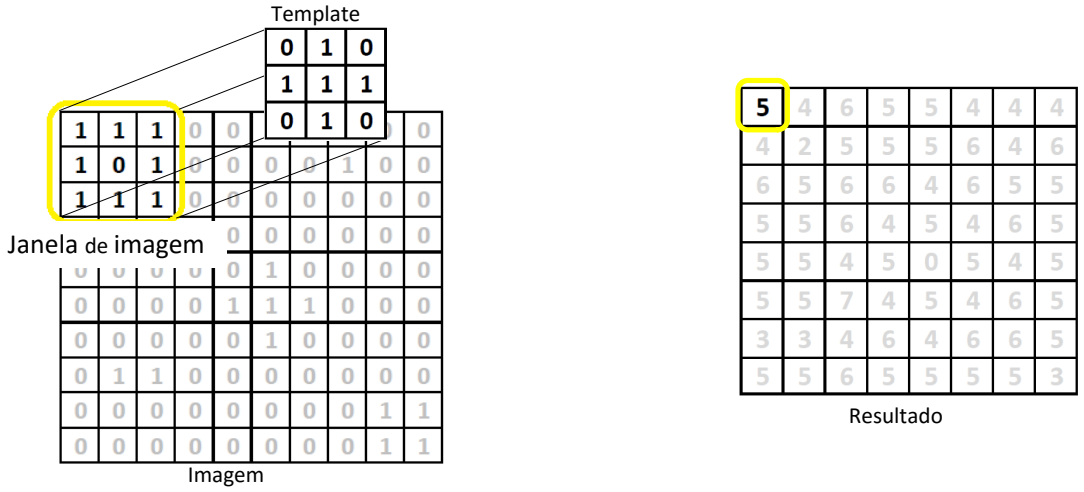
Figura 2.5: Imagem binária 10x10



1	1	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

(a) Imagem: visualização gráfica (b) Imagem: Matriz de pixels correspondentes

Figura 2.6: Cálculo da primeira posição da matriz de resultados



$$R(1,1) = (1-0)^2+(1-1)^2+(1-0)^2+(1-1)^2+(0-1)^2+(1-1)^2+(1-0)^2+(1-1)^2+(1-0)^2 = 5$$

Na Figura 2.8 pode-se ver a matriz de resultados completa e sua representação gráfica em escala de cinza, onde ‘0’ é representado pela cor preta e 9 pela cor branca. Uma vez obtida a matriz de resultados, basta localizar o mínimo da matriz para encontrar a posição da região da imagem que mais se assemelha ao *template*. O valor numérico de cada elemento da matriz representa o quão desigual a janela de imagem é do *template* em uma escala de ‘0’ a ‘N’.

O principal problema da métrica de comparação *SSD* é a grande variação na escala dos resultados. No caso de imagens binárias, essa escala depende apenas da quantidade de pixels do *template*. Quando as operações são feitas com imagens em escala de cinza, onde a intensidade de cada pixel é representada por um número inteiro de ‘d’ pixels, a escala dos resultados também

Figura 2.7: Cálculo da trigésima terceira posição da matriz de resultados

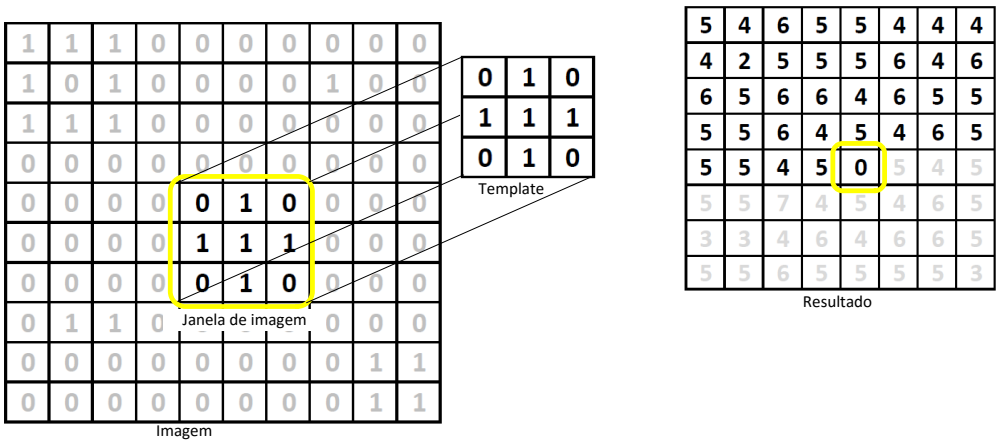
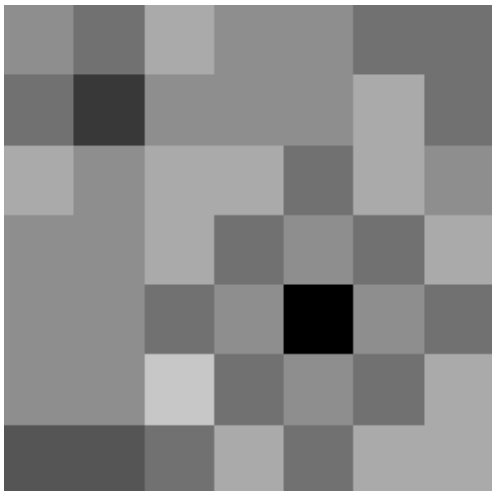


Figura 2.8: Resultado de SSD 7x7 em escala de cinza (escala entre 0 e 9)



(a) Resultado: visualização gráfica

5	4	6	5	5	4	4	4
4	2	5	5	5	6	4	6
6	5	6	6	4	6	5	5
5	5	6	4	5	4	6	5
5	5	4	5	0	5	4	5
5	5	7	4	5	4	6	5
3	3	4	6	4	6	6	5
5	5	6	5	5	5	5	3

(b) Resultado: Matriz numérica

é proporcional a $(2^d - 1)^2$. A Figura 2.9 mostra o resultado da métrica *SSD* para o caso em que imagem e *template* são representados com 8 Bits por pixel (*bpp*).

2.3.2 CORRELAÇÃO CRUZADA (*CC*)

A Equação 2.1, pode ser vista como:

$$R(x,y) = \sum_{i,j}^{N}(I(x+i,y+j))^2 - 2\sum_{i,j}^{N}(I(x+i,y+j)T(i,j)) + \sum_{i,j}^{N}(T(x+i,y+j))^2 \qquad (2.2)$$

Esse ponto de vista evidencia o termo da equação que realmente depende da relação entre imagem e *template*: $\sum_{i,j}^{N}(I(x+i,y+j)T(i,j))$. Esse termo é uma métrica de similaridade conhecida como *CC*, mostrada na Equação 2.3.

Essa medida estatística é obtida pelo acumulado dos produtos entre cada pixel de *template*

Figura 2.9: Exemplo de resultado de SSD entre imagem e *template* com 8 bits por pixel (bpp)

Imagem

255	255	255	0	0	0	0	0	0	0
255	0	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

0	255	0
255	255	255
0	255	0

Resultado SSD

325125	260100	390150	325125	325125	260100	260100	260100
260100	130050	325125	325125	325125	390150	260100	390150
390150	325125	390150	390150	260100	390150	325125	325125
325125	325125	390150	260100	325125	260100	390150	325125
325125	325125	260100	325125	0	325125	260100	325125
325125	325125	455175	260100	325125	260100	390150	325125
195075	195075	260100	390150	260100	390150	390150	325125
325125	325125	390150	325125	325125	325125	325125	195075

e o seu respectivo pixel na janela de imagem. Sendo uma medida de similaridade, quanto maior o resultado mais parecido é o*template* com a janela de imagem naquela posição.

$$R(x,y) = \sum_{i,j}^{N} I(x+i,y+j).T(i,j) \tag{2.3}$$

A Figura 2.10 mostra o cálculo da correlação cruzada entre a imagem da Figura 2.5 e o *template* da Figura 2.4. Observe que a posição onde o resultado de correlação cruzada é máximo (R=5) coincide com a posição onde a janela de imagem é igual ao *template* (quinta linha e quinta coluna).

Figura 2.10: Exemplo de resultado de CC entre imagem e *template*

Imagem

1	1	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

Template

0	1	0
1	1	1
0	1	0

Resultado CC

4	3	1	0	0	1	1	1
3	3	1	0	0	0	1	0
1	1	0	0	1	0	0	0
0	0	0	2	2	2	0	0
0	0	1	2	5	2	1	0
1	1	0	2	2	2	0	0
2	2	1	0	1	0	0	1
1	1	0	0	0	0	1	3

$$R(x,y) = \sum_{i,j}^{N} I(x+i,y+j).T(i,j)$$

A métrica *CC* apresenta problemas de escala semelhantes aos apresentados pela métrica *SSD*. A Figura 2.11 mostra o que acontece com a matriz de resultados quando as imagens de entrada são de 8 bpp.

A correlação cruzada também apresenta problemas de falso positivo em regiões muito iluminadas da imagem. Estas regiões tendem a produzir altos valores de correlação cruzada para

Figura 2.11: Exemplo de resultado de CC entre imagem e *template* quando a representação usa 8 bpp

Imagem

255	255	255	0	0	0	0	0	0	0
255	0	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

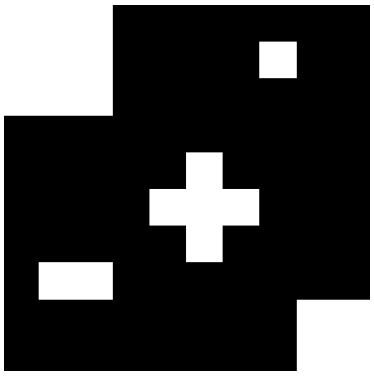
0	255	0
255	255	255
0	255	0

Resultado CC

260100	195075	65025	0	0	65025	65025	65025
195075	195075	65025	0	0	0	65025	0
65025	65025	0	0	65025	0	0	0
0	0	0	130050	130050	130050	0	0
0	0	65025	130050	325125	130050	65025	0
65025	65025	0	130050	130050	130050	0	0
130050	130050	65025	0	65025	0	0	65025
65025	65025	0	0	0	0	65025	195075

qualquer *template* confundindo o algoritmo de rastreamento. Para ilustrar esse problema, na imagem que vinha sendo usada como exemplo (Figura 2.5) foi trocado um pixel para que ela tivesse uma janela de imagem completamente branca (Figura 2.12).

Figura 2.12: Imagem de entrada modificada para exemplificar o problema com regiões claras da correlação cruzada



A Figura 2.13 mostra o que acontece com a matriz de resultados quando a imagem de entrada possui uma janela de imagem completamente branca, observe que a matriz de resultados apresenta dois máximos, um que a localização corresponde à localização do modelo buscado na imagem e o outro máximo na localização que corresponde à região clara da imagem.

Apesar desses problemas, vale a pena entender como essa métrica funciona para entender as métricas que derivam da correlação cruzada, as quais são descritas nas próximas subseções.

2.3.3 CORRELAÇÃO CRUZADA NORMALIZADA (*NCC*)

A grande variação na escala dos resultados é indesejável na hora de escolher um limiar de similaridade a partir do qual considera-se a janela de imagem suficientemente semelhante ao *template*. Isso faz da correlação cruzada normalizada uma métrica mais interessante nas aplicações de rastreamento que a métrica de correlação cruzada simples.

Figura 2.13: Exemplo de resultado de CC entre imagem e *template* quando a imagem possui uma região muito clara

Imagem

1	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

Template

0	1	0
1	1	1
0	1	0

Resultado CC

5	4	1	0	0	1	1	1
4	3	1	0	0	0	1	0
1	1	0	0	1	0	0	0
0	0	0	2	2	2	0	0
0	0	1	2	5	2	1	0
1	1	0	2	2	2	0	0
2	2	1	0	1	0	0	1
1	1	0	0	0	0	1	3

A forma de calcular os resultados da métrica *NCC* é muito parecida com a da métrica CC, mas os resultados são normalizados pela média geométrica entre os somatórios dos quadrados dos pixels do *template* e da imagem, como mostrado na Equação 2.4. A normalização poderia trazer um problema de divisão por zero caso todos os pixels da janela de imagem fossem nulos, mas convencionou-se que nesse caso o resultado recebe o valor zero.

$$R(x,y) = \frac{\sum_{i,j}^{N} I(x+i,y+j).T(i,j)}{\sqrt{\sum_{i,j}^{N}(I^2(x+i,y+j)).\sum_{i,j}^{N}(T^2(i,j))}} \tag{2.4}$$

As Figuras 2.14 e 2.15 mostram resultados da métrica *NCC* para imagens de entrada com 1 e 8 bits por pixel (*bpp*) respectivamente. Nesse exemplo foram usados o *template* da Figura 2.4 e a imagem da Figura 2.5. Observe que os valores numéricos dos resultados de NCC são independentes da quantidade de bpp das imagens de entrada.

Figura 2.14: Exemplo de resultado da métrica NCC entre imagem e *template* com 1 bpp

Imagem

1	1	1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

Template

0	1	0
1	1	1
0	1	0

Resultado NCC

0,63	0,60	0,26	0,00	0,00	0,45	0,45	0,45
0,60	0,77	0,32	0,00	0,00	0,00	0,45	0,00
0,26	0,32	0,00	0,00	0,45	0,00	0,00	0,00
0,00	0,00	0,00	0,52	0,45	0,52	0,00	0,00
0,00	0,00	0,45	0,45	1,00	0,45	0,45	0,00
0,32	0,32	0,00	0,52	0,45	0,52	0,00	0,00
0,63	0,63	0,45	0,00	0,45	0,00	0,00	0,32
0,32	0,32	0,00	0,00	0,00	0,00	0,32	0,67

A métrica *NCC* resolve o problema de escala uma vez que os resultados de *NCC* variam entre '-1' e '1' independente do tamanho do *template* e da quantidade de bpp da imagem e *template*. Isso possibilita a escolha de limiares fixos de similaridade genéricos, um determinado

Figura 2.15: Exemplo de resultado da métrica NCC entre imagem e *template* com 8 bpp

Imagem

255	255	255	0	0	0	0	0	0	0
255	0	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

0	255	0
255	255	255
0	255	0

Resultado NCC

0,63	0,60	0,26	0,00	0,00	0,45	0,45	0,45
0,60	0,77	0,32	0,00	0,00	0,00	0,45	0,00
0,26	0,32	0,00	0,00	0,45	0,00	0,00	0,00
0,00	0,00	0,00	0,52	0,45	0,52	0,00	0,00
0,00	0,00	0,45	0,45	1,00	0,45	0,45	0,00
0,32	0,32	0,00	0,52	0,45	0,52	0,00	0,00
0,63	0,63	0,45	0,00	0,45	0,00	0,00	0,32
0,32	0,32	0,00	0,00	0,00	0,00	0,32	0,67

valor de similaridade tem o mesmo significado para qualquer tamanho de *template* (N) e qualquer valor de profundidade de cor considerado (bpp). A NCC também é um pouco mais robusta para regiões muito iluminadas na imagem. No exemplo da Figura 2.16 a métrica NCC calculada na região totalmente branca tem um valor alto mas não chega a se confundir com o valor máximo da matriz de resultado, que é igual a 1.

Figura 2.16: Resultado de NCC entre imagem e *template* quando a imagem possui uma região muito clara

Imagem

1	1	1	0	0	0	0	0	0	0
1	1	1	0	0	0	0	1	0	0
1	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	0
0	0	0	0	1	1	1	0	0	0
0	0	0	0	0	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	1	1
0	0	0	0	0	0	0	0	1	1

Template

0	1	0
1	1	1
0	1	0

Resultado NCC

0,75	0,73	0,26	0,00	0,00	0,45	0,45	0,45
0,73	0,67	0,32	0,00	0,00	0,00	0,45	0,00
0,26	0,32	0,00	0,00	0,45	0,00	0,00	0,00
0,00	0,00	0,00	0,52	0,45	0,52	0,00	0,00
0,00	0,00	0,45	0,45	1,00	0,45	0,45	0,00
0,32	0,32	0,00	0,52	0,45	0,52	0,00	0,00
0,63	0,63	0,45	0,00	0,45	0,00	0,00	0,32
0,32	0,32	0,00	0,00	0,00	0,00	0,32	0,67

No contexto de rastreamento em ambientes externos, a cena está suscetível a variações de iluminação e contraste ao longo do tempo, a métrica NCC não é robusta a estas variações, o que pode ter efeitos negativos no algoritmo de rastreamento.

A Figura 2.18 ilustra o efeito da mudança de contraste na cena. Para exemplificar esse problema, o contraste do *template* usado nos exemplos anteriores (Figura 2.4) com 8bpp foi modificado até obter o *template* da Figura 2.17. A NCC foi calculada entre a imagem da Figura 2.13 e esse novo *template* com mudança de iluminação e contraste, perceba que neste exemplo a matriz de NCC indica que a janela de imagem branca é mais parecida com o *template* que a janela de imagem onde se encontra o *template* sem modificação.

Figura 2.17: Template com menor contraste 3x3

64	128	64
128	128	128
64	128	64

(a) Template: visualização gráfica (b) Template: Matriz de pixels correspondentes

Figura 2.18: Resultado de NCC entre imagem e *template* quando a imagem possui uma região muito clara e há uma diferença de iluminação e contraste entre imagem e *template*

Imagem

255	255	255	0	0	0	0	0	0	0
255	255	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

64	128	64
128	128	128
64	128	64

Resultado NCC

0,95	0,83	0,47	0,00	0,00	0,41	0,41	0,41
0,83	0,71	0,43	0,00	0,00	0,20	0,41	0,20
0,47	0,43	0,20	0,20	0,41	0,20	0,00	0,00
0,00	0,00	0,20	0,59	0,61	0,59	0,20	0,00
0,00	0,00	0,41	0,61	0,91	0,61	0,41	0,00
0,43	0,43	0,29	0,59	0,61	0,59	0,20	0,00
0,58	0,58	0,41	0,20	0,41	0,20	0,20	0,43
0,43	0,43	0,20	0,00	0,00	0,00	0,43	0,71

2.3.4 CORRELAÇÃO CRUZADA COM MÉDIA ZERO (*ZCC*)

A métrica conhecida como Correlação Cruzada com Média Zero (ZCC) se baseia na *CC* mas não usa diretamente o valor de intensidade dos pixels e sim o valor da diferença entre a intensidade do pixel e a média de valores na janela. Essa subtração evita que janelas de imagem muito claras resultem em valores de similaridade altos quando elas não são parecidas com o *template*.

A forma de calcular cada posição da matriz de resultados usando a métrica *ZCC* pode ser vista na Equação 2.5. O termo $\overline{I}(x,y)$ é a média da janela de imagem com origem na posição (x,y) (canto esquerdo superior), o termo $\overline{T}$ é a média de todos os pixels do*template*.

$$R(x,y) = \sum_{i,j}^{N} (I(x+i,y+j) - \overline{I}(x,y)).(T(i,j) - \overline{T}) \qquad (2.5)$$

A Figura 2.19 mostra o resultado da métrica ZCC entre uma imagem que contém uma janela totalmente branca e um *template* com mudança de iluminação e contraste. Observe que a região branca não gera um alto valor de *ZCC* e que o *template* buscado é encontrado mesmo com a mudança de iluminação e contraste. Apesar dessas vantagens, a métrica *ZCC* ainda apresenta a

desvantagem da grande variação na escala dos valores dos resultados.

Figura 2.19: Exemplo de resultado de ZCC entre imagem e *template* quando a imagem possui uma região muito clara e há uma diferença de iluminação e contraste entre imagem e *template*

Imagem

255	255	255	0	0	0	0	0	0	0
255	255	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

64	128	64
128	128	128
64	128	64

Resultado ZCC

0	10838	-10838	0	0	7225	7225	7225
10838	12644	-1806	0	0	-9031	7225	-9031
-10838	-1806	-9031	-9031	7225	-9031	0	0
0	0	-9031	5419	-3613	5419	-9031	0
0	0	7225	-3613	36125	-3613	7225	0
-1806	-1806	-18063	5419	-3613	5419	-9031	0
14450	14450	7225	-9031	7225	-9031	-9031	-1806
-1806	-1806	-9031	0	0	0	-1806	12644

Além das mudanças na escala dos resultados a métrica *ZCC* ainda é sensível a variações de iluminação e contraste. Observe a diferença entre os resultados da métrica *ZCC* na Figura 2.19 e os resultados da métrica *ZCC* na Figura 2.20, onde a única diferença entre as duas são os níveis de iluminação e contraste do *template*.

Figura 2.20: Exemplo de resultado de ZCC entre imagem e *template* quando a imagem possui uma região muito clara e o *template* e a imagem tem os mesmos níveis de iluminação e contraste

Imagem

255	255	255	0	0	0	0	0	0	0
255	255	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

0	255	0
255	255	255
0	255	0

Resultado ZCC

0	43350	-43350	0	0	28900	28900	28900
43350	50575	-7225	0	0	-36125	28900	-36125
-43350	-7225	-36125	-36125	28900	-36125	0	0
0	0	-36125	21675	-14450	21675	-36125	0
0	0	28900	-14450	144500	-14450	28900	0
-7225	-7225	-72250	21675	-14450	21675	-36125	0
57800	57800	28900	-36125	28900	-36125	-36125	-7225
-7225	-7225	-36125	0	0	0	-7225	50575

2.3.5 CORRELAÇÃO CRUZADA NORMALIZADA COM MÉDIA ZERO (*ZNCC*)

A métrica denominada Correlação Cruzada Normalizada de Média Zero (*ZNCC*) é mostrada na Equação 2.6. Esta combina as principais vantagens das métricas anteriores: é robusta às regiões muito claras da imagem, é robusta às variações lineares de brilho e contraste *(DI STEFANO; MATTOCCIA; TOMBARI, 2005)* e apresenta resultados normalizados.

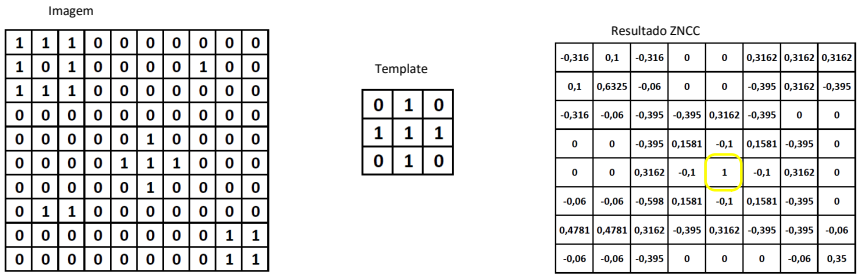
Os resultados da métrica são obtidos de acordo com a Equação 2.6, a qual se baseia na correlação cruzada, é normalizada entre -1 e 1 como a métrica *NCC*, e usa a diferença entre a intensidade de cada pixel e a média na janela de imagem como a métrica *ZCC*. Esta equação

também pode apresentar um problema de divisão por zero em regiões perfeitamente uniformes (desvio padrão nulo), nesse caso foi convencionado que o resultado recebe zero.

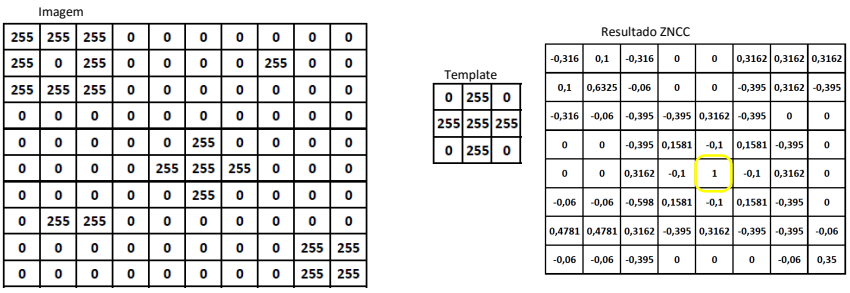
$$R(x,y) = \frac{\sum_{i,j}^{N}(I(x+i,y+j) - \overline{I}(x,y)).(T(i,j) - \overline{T})}{\sqrt{\sum_{i,j}^{N}(I(x+i,y+j) - \overline{I}(x,y))^2 . \sum_{i,j}^{N}(T(i,j) - \overline{T})^2}} \tag{2.6}$$

As Figuras 2.22(a), 2.22(b) e 2.22(a) mostram resultados da métrica *ZNCC* para imagem e *template*, variando a quantidade de bpp ou a iluminação e contraste, note que para essa mudanças os resultados se mantêm exatamente iguais, já que essa métrica é robusta a esse tipo de variação.

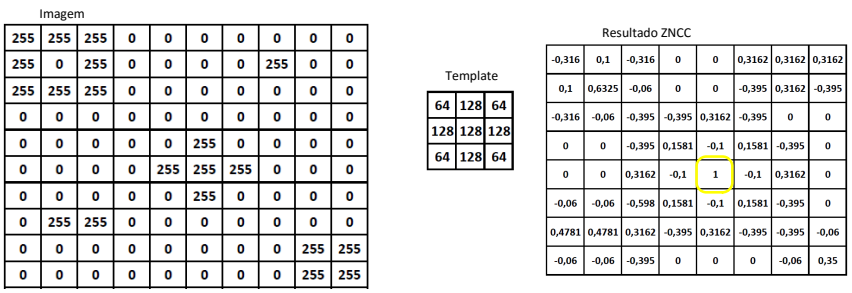
Figura 2.21: Exemplos da invariância da *ZNCC* a mudanças de quantidades de bpp e mudanças de iluminação e contraste



(a) Resultado de *ZNCC* entre imagem e *template* binários



(b) Resultado de *ZNCC* entre imagem e *template* com 8 bpp



(c) Resultado de *ZNCC* entre imagem e *template* com diferença de iluminação e contraste entre a imagem e o template

A Figura 2.22 mostra como se comporta o resultado da *ZNCC* quando existe uma região

muito clara na imagem, note que usando esta métrica, tal região não gera falsos positivos.

Figura 2.22: Exemplo da *ZNCC* quando a imagem possui uma região muito clara

Imagem

255	255	255	0	0	0	0	0	0	0
255	255	255	0	0	0	0	255	0	0
255	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	255	0	0	0	0
0	0	0	0	255	255	255	0	0	0
0	0	0	0	0	255	0	0	0	0
0	255	255	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	255	255
0	0	0	0	0	0	0	0	255	255

Template

64	128	64
128	128	128
64	128	64

Resultado ZCC

0	0,3162	-0,316	0	0	0,3162	0,3162	0,3162
0,3162	0,35	-0,06	0	0	-0,395	0,3162	-0,395
-0,316	-0,06	-0,395	-0,395	0,3162	-0,395	0	0
0	0	-0,395	0,1581	-0,1	0,1581	-0,395	0
0	0	0,3162	-0,1	1	-0,1	0,3162	0
-0,06	-0,06	-0,598	0,1581	-0,1	0,1581	-0,395	0
0,4781	0,4781	0,3162	-0,395	0,3162	-0,395	-0,395	-0,06
-0,06	-0,06	-0,395	0	0	0	-0,06	0,35

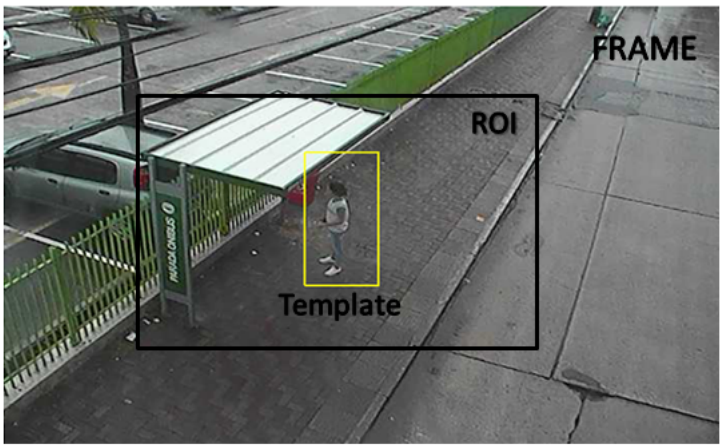
A métrica *ZNCC* foi a escolhida para este trabalho por apresentar os melhores resultados para o casamento de padrões por análise de similaridade por janela deslisante. No entanto, esta métrica possui alto custo computacional para ser calculada. Assim parte do objetivo desse trabalho é encontrar uma forma mais eficiente para calcular resultados da métrica *ZNCC* para aplicações de rastreamento.

2.4 Aplicação: Rastreamento baseado em ZNCC multitemplate

O algoritmo de rastreamento de *streams* de vídeo baseado em *templatematching* consiste em buscar, a cada novo*frame* , o*template* (estático) do objeto rastreado.

A busca é feita na região de interesse de cada *frame* , do inglês, Region of interest (ROI), esta engloba a posição do*template* no último*frame* e possíveis deslocamentos, para cima, para baixo, para a esquerda e para a direita. A Figura 2.23 mostra um exemplo de rastreamento com*frame* , *ROI* e *template*. Para a aplicação de rastreamento de pedestres foi considerada uma região de interesse com seis vezes a largura e três vezes a altura do *template*.

Figura 2.23: Rastreamento mostrando*frame* , *ROI* e*template*.



O Algoritmo 1 mostra o pseudocódigo de uma aplicação de rastreamento contínuo com o casamento de padrões que usa a métrica ZNCC para analisar a similaridade.

Algoritmo 1 Rastreamento contínuo de objetos, baseado em ZNCC

Passo 1: Receber um*frame* .
Passo 2: Usuário escolhe o objeto a ser rastreado.
Passo 3: Inicializar o*template* com a imagem escolhida pelo usuário.
Passo 4: Selecionar a *ROI* ao redor da última posição do*template*.
Passo 6: Converter a *ROI* para escala de cinza.
Passo 7: Calcular a matriz de resultado *ZNCC* entre a *ROI* e o *template*.
Passo 8: Obter o valor máximo na matriz dos resultados e sua posição.
Passo 9: Comparar o valor máximo com o limiar de aceitação, se for maior: atualizar a posição do objeto.
Passo 10: Voltar ao passo 4

Durante o rastreamento de um pedestre em ambiente externo podem ocorrer mudanças de iluminação, contraste e de pose. Mudanças de iluminação e contraste não têm grande impacto sobre os resultados da métrica *ZNCC* e portanto não causam problemas no Algoritmo de rastreamento, mas mudanças de pose podem gerar valores de similaridade cada vez menores e ocasionar a perda do pedestre que está sendo rastreado.

O algoritmo de rastreamento baseado em *ZNCC multitemplate* foi desenvolvido para contornar esse problema de mudança de pose. Ele utiliza um banco de *templates* com poses ligeiramente diferentes do pedestre sendo rastreado. Estes *templates* são simultaneamente buscados na imagem, diminuindo as chances de se perder a posição do pedestre por mudança de pose.

Uma política de substituição dos *templates* no banco, mantém o banco de*templates* atualizado à medida que o pedestre se movimenta.

Essa solução é adequada quando se considera que a posição e a aparência do pedestre não mudam abruptamente. Assim, é feita uma busca por 'm'*templates* do mesmo pedestre. O pseudocódigo deste algoritmo pode ser visto no Algoritmo 2.

Algoritmo 2 Rastreamento contínuo de objetos, usando *ZNCC* multitemplate

Passo 1: Receber um*frame* .
Passo 2: Usuário escolhe o objeto a ser rastreado.
Passo 3: Inicializar todos os*templates* do banco com uma cópia do indicado pelo usuário.
Passo 4: Selecionar a *ROI* ao redor da última posição do*template*.
Passo 6: Converter a *ROI* para escala de cinza.
Passo 7: Calcular as matrizes de resultados *ZNCC* entre a imagem da *ROI* e cada *template*.
Passo 8: Obter o valor máximo dentre os máximos das matrizes dos resultados e sua posição.
Passo 9: Comparar o valor máximo com o limiar de aceitação, se for maior: atualizar a posição do objeto e usar a política de substituição para substituir ou não um*template* do banco pela janela de imagem encontrada.
Passo 10: Voltar ao passo 4

As matrizes de resultados da métrica *ZNCC* contém valores normalizados entre -1 e 1.

Um limiar de aceitação foi configurado, para determinar se o objeto foi encontrado na cena. Se o maior valor dentre os resultados não ultrapassar esse limiar, é considerado que o objeto está sob oclusão, a última posição encontrada é mantida, e os*templates* permanecem inalterados.

Quando existe pelo menos um resultado que é maior que o limiar de aceitação, a posição do maior resultado é tomada como sendo a nova posição do objeto.

A janela de imagem pode ser adicionada ao banco de *templates* de acordo com a política de substituição. Essa política não é relevante para este trabalho, mas, em geral, o primeiro *template* do banco nunca é substituído. Para escolher o *template* a ser substituído se leva em conta o tempo passado desde a última substituição e os valores dos resultados para cada *template* .

A Figura 2.24, mostra o rastreamento em funcionamento usando 6 *templates*, o primeiro *template* do banco é o *template* original, enquanto os outros 5 *templates* foram obtidos ao longo do rastreamento e são atualizados com o tempo.

Figura 2.24: Rastreamento de pedestre usando 6*templates*.

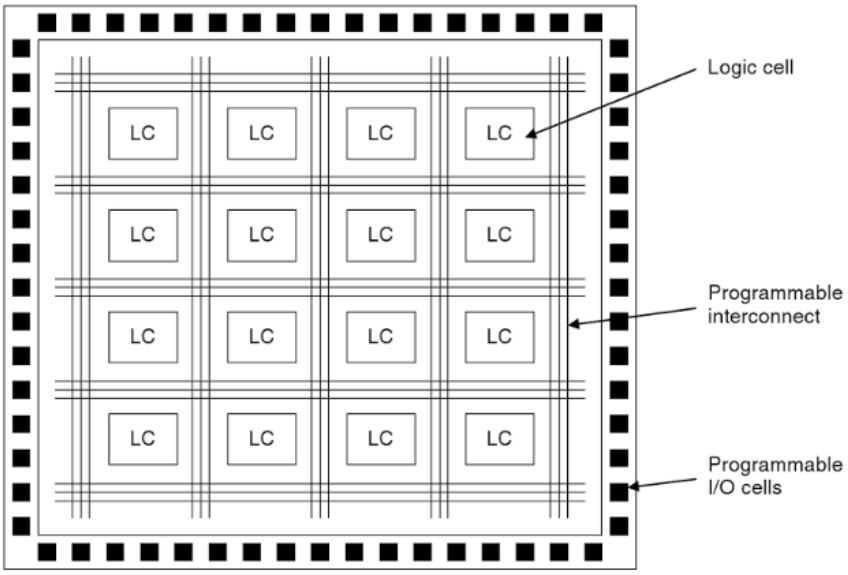
A etapa com o maior custo computacional do rastreamento é a etapa 7, que consiste em calcular a métrica *ZNCC* entre cada um dos *templates* do banco e a *ROI*, cada *template* gera uma matriz de resultados. Testes feitos no software de referencia mostram que essa etapa consome de 97% a 99% do tempo total de processamento.

Considerando o alto custo computacional e sua importância no rastreamento de pedestre, o objetivo desse trabalho é acelerar esta etapa através da implementação em hardware do módulo que calcula a similaridade de acordo com a *ZNCC* entre múltiplos *templates* e uma imagem.

2.5 FPGA

Os *FPGA*s, do inglês,*Field-Programmable Gate Arrays*, são assim chamados por serem um hardware passível de reconfiguração em campo. FPGAs são matrizes de blocos lógicos com conexões programáveis, como mostra a Figura 2.25.

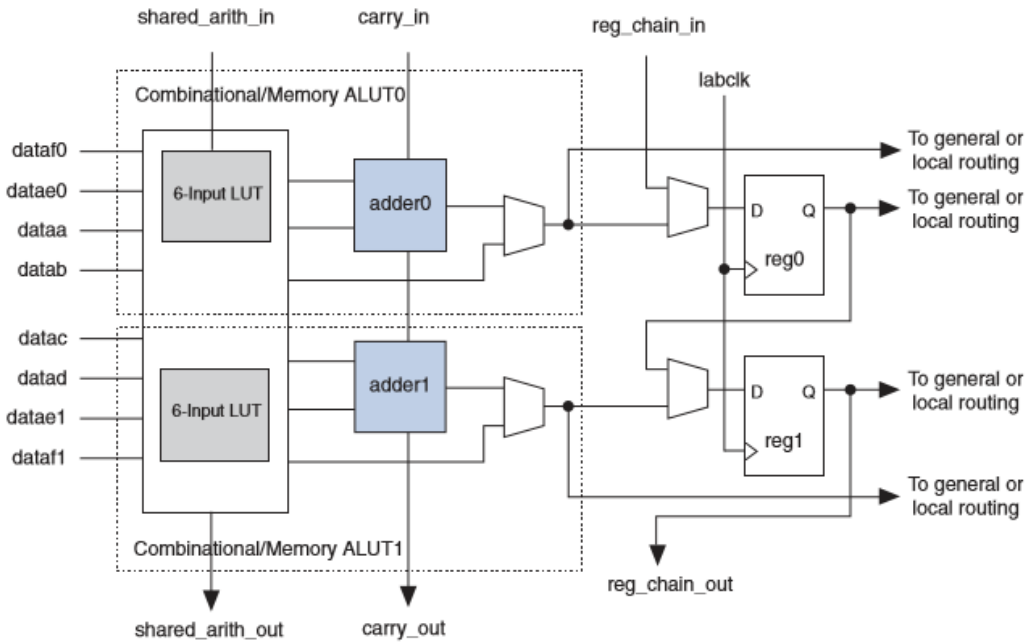
Figura 2.25: Arquitetura genérica de FPGA.



Fonte: Digital Systems Design with FPGAs and CPLDs *(GROUT, 2011)*.

Além dos blocos lógicos, de diversos tipos, elementos de roteamento e blocos de entrada e saída, mostrados na Figura 2.25, *FPGA*s modernos incluem blocos de memória e blocos dedicados de processamento digital de sinais (*DSP*). A Figura 2.26 mostra a estrutura interna de um bloco lógico com as funções lógicas implementadas por Look up Table (*LUT*) e registradores desempenhando o papel de elementos sequenciais.

Figura 2.26: Exemplo de bloco lógico contido na FPGA Stratix IV.

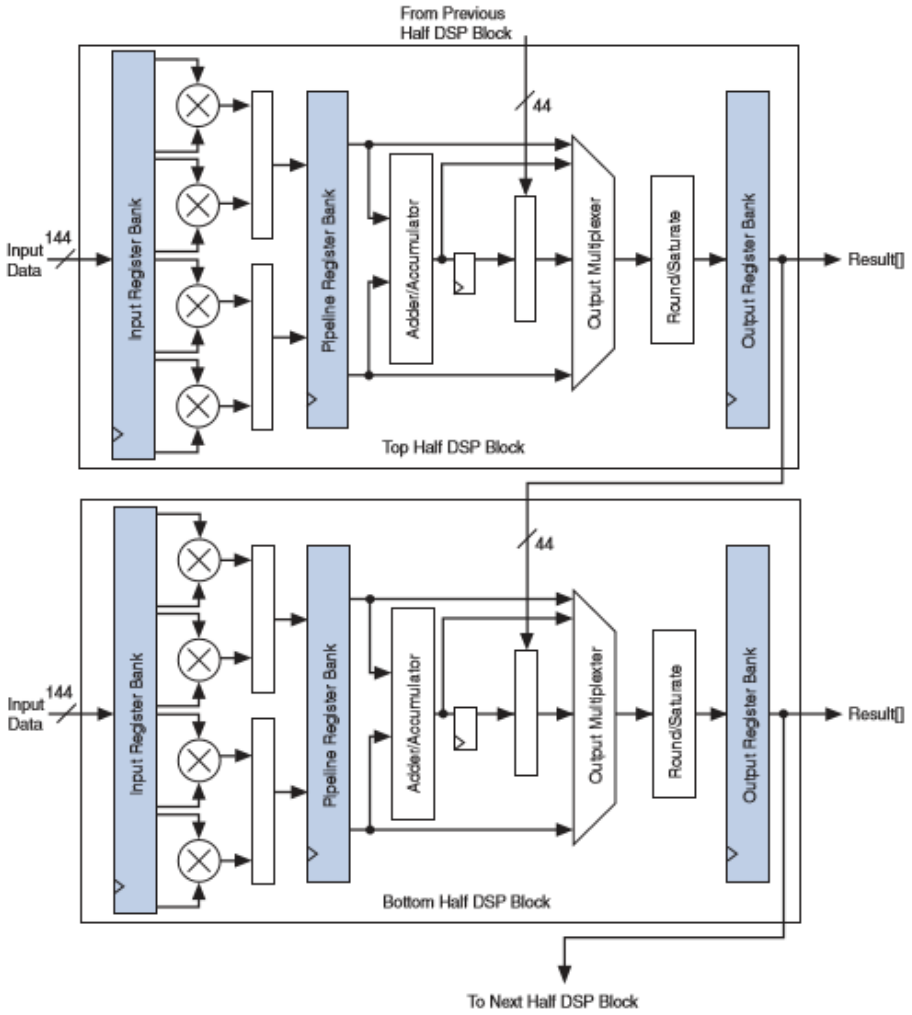


Fonte: Stratix IV Device Handbook *ALTERA (2015)*.

Os blocos *DSP* desempenham funções de multiplicadores e acumuladores, permitindo

diversas configurações. As funções implementadas por estes blocos podem ser realizadas por outros blocos lógicos, sem prejuízo de funcionalidade, mas em geral, de forma menos eficiente (uso de mais blocos e maior caminho crítico). A Figura 2.27 mostra um bloco DSP na FPGA Stratix IV.

Figura 2.27: Bloco DSP contido na FPGA Stratix IV.



Fonte: Stratix IV Device Handbook *ALTERA (2015).*

No contexto de sistemas computacionais baseados em FPGAs, o estudo de *(DIMOND; RACANIERE; PELL, 2011)* mostra que estes sistemas podem atingir desempenho de 31 a 37 vezes maiores e consumo de energia de 39 vezes menores, quando comparados a sistemas baseados em processadores de propósito geral (CPUs) de tamanho equivalente.

Os dispositivos FPGAs têm ganhado grande destaque no desenvolvimento de novas tecnologias. Eles são, em geral, a primeira solução de implementação no projeto de circuitos digitais *(KUON; TESSIER; ROSE, 2008)*. Adicionalmente FPGAs são muito usados para validar um circuito que será implementado como um Circuito integrado de aplicação específica (*ASIC*).

De forma geral, os circuitos integrados dedicados, do inglês, Application Specific Integrated Circuits (*ASIC*s), quando comparados à implementações *FPGA*, atingem maiores frequências

de operação, com menor dissipação de potência e podem apresentar um custo unitário reduzido para grandes escalas de produção.

Apesar disso, em alguns nichos com menor demanda de unidades e exigência de alguma flexibilidade do circuito, implementações em *FPGA* podem ser mais vantajosas. A computação de alta performance, ou High Performance Computing (*HPC*) tira grandes vantagens do uso de *FPGA*s, isto justifica a inclusão destes dispositivos em supercomputadores *(AWAD, 2009)*.

Dentre estes nichos destacamos os setores aeroespacial, médico, científico, financeiro, de bioinformática e de visão computacional *(ALTERA, 2016) (XILINX, 2016)*.

2.6 Conclusões

O objetivo deste capítulo foi apresentar os conceitos fundamentais para o entendimento do módulo desenvolvido nesta dissertação. Inicialmente foi apresentado o problema de rastreamento de objetos, o casamento de padrões aplicado ao rastreamento, o funcionamento da análise de similaridade por janela deslizante e algumas métricas de similaridade. Em seguida as principais métricas de comparação entre imagens foram analisadas com exemplos para identificar as vantagens e desvantagens de cada métrica, evidenciando a escolha da *ZNCC* para esse projeto. Por último foram apresentados os conceitos básicos do algoritmo de rastreamento baseado em *ZNCC multitemplate*, que motivou este trabalho.

3

TRABALHOS RELACIONADOS

Neste capítulo são apresentados os principais trabalhos que estão relacionados ao tema desta dissertação. Inicialmente, é apresentado o trabalho de *(LEWIS, 1995)*, que é um dos primeiros trabalhos para aceleração do cálculo da ZNCC e é o algoritmo implementado na função do Matlab. Em seguida, são apresentados os trabalhos que fazem a aceleração do cálculo da *ZNCC* via hardware *(CHEN et al., 2012)*, *(HASHIMOTO; ITO; NAKANO, 2013)*, *(SANG; LIAO; YUAN, 2011)*. No final do capítulo, alguns critérios qualitativos são usados para uma análise comparativa entre os trabalhos de estado da arte. A partir dessa análise são explicitadas as inspirações vindas de cada trabalho e possibilidades de melhoramento.

A literatura científica disponibiliza diversos trabalhos que tratam da aceleração do cálculo da *ZNCC*, neste capítulo são expostas algumas estratégias de implementação para realizar esse cálculo.

3.1 Fast normalized cross-correlation

O trabalho de *(LEWIS, 1995)* foi um dos primeiros trabalhos com o objetivo de acelerar o cálculo da ZNCC. Este trabalho é a base da função *normxcorr2*, do *MATLAB*.

O trabalho propõe acelerar as operações mais custosas para obter um resultado de ZNCC, isto é, a correlação, para obter o numerador, e o somatório da janela de imagem, usado para obter o denominador (Equação 3.1). No trabalho, o cálculo da correlação, foi feito no domínio da frequência, e o somatório da janela de imagem foi feito utilizando uma imagem integral da imagem original, procedimentos detalhados a seguir.

$$R(x,y) = \frac{\sum_{i,j}^{N}\left(I(x+i,y+j) - \overline{I}(x,y)\right).\left(T(i,j) - \overline{T}\right)}{\sqrt{\sum_{i,j}^{N}\left(I(x+i,y+j) - \overline{I}(x,y)\right)^2 . \sum_{i,j}^{N}\left(T(i,j) - \overline{T}\right)^2}} \tag{3.1}$$

O cálculo de uma correlação no domínio espacial ($\star$) é equivalente a uma multiplicação ($*$) no domínio da frequência, como mostra a Equação 3.2. Assim para obter a matriz com todos os resultados de correlação cruzada (cc) entre a imagem e o *template* , basta fazer o preenchimento das bordas do *template* com zeros para que ele fique com o mesmo tamanho da imagem, fazer a transformada de Fourier ($\mathscr{F}$) do *template* e da imagem, multiplicar os resultados e em seguida fazer a transformada de Fourier inversa para obter o resultado no domínio espacial.

A correlação cruzada entre uma imagem I com R.S pixels e um *template* T com O.P pixels produz uma matriz de resultados matR com (R-O+1).(S-P+1) elementos. No domínio espacial, a obtenção de cada elemento da matriz de resultados envolve O.P multiplicações, assim a complexidade computacional da correlação cruzada no domínio espacial é O.P.(R-O+1).(S-P+1).

O uso dessa propriedade para calcular a correlação cruzada no domínio da frequência, muda a complexidade de uma correlação de $o(O \cdot P \cdot (R-O+1) \cdot (S-P+1))$ para $o(R.S \cdot log(R.S))$.

$$\mathscr{F}\{f \star g\} = \mathscr{F}\{f\}^{*}\mathscr{F}\{g\} \tag{3.2}$$

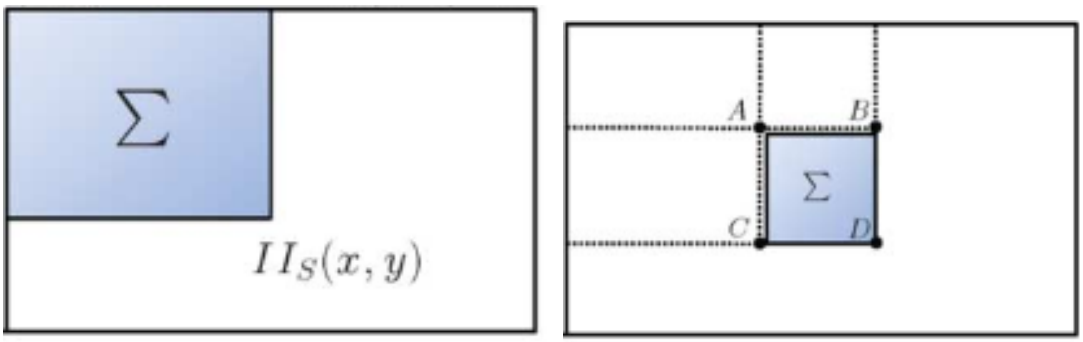
Imagem integral é uma matriz que permite o cálculo do somatório dos pixels de qualquer subconjunto da imagem original com apenas três operações, (Figura 3.1).

A partir de uma imagem original, calcula-se a imagem integral fazendo com que cada elemento desta (II_s(x,y)) seja a soma dos pixels de uma janela da imagem original que vai da origem (1,1) as coordenadas deste elemento (x,y) *(CROW, 1984)*.

Uma vez que a imagem integral está calculada, qualquer somatório dos pixels de uma região retangular da imagem original pode ser obtido com três operações sobre os quatro

elementos da imagem integral, localizados nas posições equivalentes aos cantos da região retangular (A, B, C e D): $S = II_s(D) + II_s(A) - II_s(B) - II_s(C)$.

Figura 3.1: (a) Valor do ponto II_s(x,y) da imagem integral, corresponde à soma das intensidades dos pixels da imagem original localizados a partir desta posição até a origem, região em destaque. (b) A soma da janela destacada pode ser calculada como: $S = II_s(D) + II_s(A) - II_s(B) - II_s(C)$, onde $II_s(A)$ é o valor da imagem integral na posição 'A'.



(a) (b)

Fonte: *(JUNG et al., 2010)*

O algoritmo apresentado nesse trabalho requer um pré-processamento para obter as imagens integrais e as imagens no domínio da frequência, e um pós processamento para obter o resultado no domínio espacial. Mesmo com os tempos de pré e pós processamento, dependendo das dimensões da imagem e do *template*, essa técnica reduz o tempo total de processamento para o cálculo da métrica *ZNCC* em software.

Pensando em uma abordagem para hardware, as transformadas de Fourier direta e inversa necessárias ao processamento usam muitas unidades funcionais complexas *(BARBOSA et al., 2015)*. Fazendo com que o cálculo da correlação no domínio espacial seja mais vantajoso para esse tipo de implementação. Além disso, por exigir um pré-processamento, essa estratégia de cálculo dificuta a exploração de paralelismo no hardware.

Assim o uso de imagens integrais é desnecessário já que as somas podem ser obtidas paralelamente ao cálculo da correlação, o uso de imagens integrais, em uma implementação paralela, restringiria o paralelismo já que exigiria o pré-processamento da imagem original.

3.2 Real-time FPGA-based template matching module for visual inspection application

O trabalho de *(CHEN et al., 2012)* propõe uma arquitetura baseada em *FPGA*, que explora paralelismo e pipeline para calcular uma métrica simplificada, inspirada na *ZNCC*.

A partir da Equação 2.6, que é a equação original da *ZNCC*, os autores observaram que uma parte do denominador depende apenas dos valores dos pixels do *template*, assim esse termo é uma constante em relação a (x,y). Multiplicar a métrica por um termo constante altera os valores de cada elemento da matriz mas não altera a ordem dos elementos entre si, o valor máximo continua na mesma posição da matriz

$$R(x,y) = \frac{\sum_{i,j}^{N}\left(I(x+i,y+j)-\bar{I}(x,y)\right).\left(T(i,j)-\overline{T}\right)}{\sqrt{\sum_{i,j}^{N}\left(I(x+i,y+j)-\bar{I}(x,y)\right)^2}.\cancel{\sqrt{\sum_{i,j}^{N}\left(T(i,j)-\overline{T}\right)^2}}}.\cancel{\sqrt{\sum_{i,j}^{N}\left(T(i,j)-\overline{T}\right)^2}} \quad (3.3)$$

A raiz quadrada no denominador foi eliminada, para simplificar o hardware, elevando ao quadrado o numerador e o denominador da fração, mais uma vez essa operação modifica os valores de cada elemento da matriz mas não a ordem dos elementos entre si. Essa operação é mostrada nas equações 3.4 3.5.

$$R(x,y) = \left(\frac{\sum_{i,j}^{N}\left(I(x+i,y+j)-\bar{I}(x,y)\right).\left(T(i,j)-\overline{T}\right)^2}{\sqrt{\sum_{i,j}^{N}\left(I(x+i,y+j)-\bar{I}(x,y)\right)^2}}\right)^2 \quad (3.4)$$

$$R(x,y) = \frac{\left(\sum_{i,j}^{N}\left(I(x+i,y+j)-\bar{I}(x,y)\right).\left(T(i,j)-\overline{T}\right)^2\right)^2}{\sum_{i,j}^{N}\left(I(x+i,y+j)-\bar{I}(x,y)\right)^2} \quad (3.5)$$

Além dessas modificações na fórmula, foram feitas manipulações algébricas para encontrar uma fórmula equivalente melhor adaptada a uma implementação em hardware, vamos acompanhar o passo a passo dessa manipulação nas equações abaixo:

Expandindo os produtos, tem-se:

$$R(x,y) = \frac{\left(\sum_{i,j}^{N}\left(T(i,j)\cdot I(x+i,y+j)-\overline{T}\cdot I(x+i,y+j)-T(i,j)\cdot\bar{I}(x,y)+\overline{T}\cdot\bar{I}(x,y)\right)\right)^2}{\sum_{i,j}^{N}\left(I^2(x+i,y+j)-2\cdot I(x+i,y+j)\bar{I}(x,y)+\left(\bar{I}(x,y)\right)^2\right)} \quad (3.6)$$

Os somatórios são distribuídos para obter:

$$R(x,y) = \frac{\left(\sum_{i,j}^{N}T(i,j)\cdot I(x+i,y+j)-\sum_{i,j}^{N}\overline{T}\cdot I(x+i,y+j)-\sum_{i,j}^{N}T(i,j)\cdot\bar{I}(x,y)+\sum_{i,j}^{N}\overline{T}\cdot\bar{I}(x,y)\right)^2}{\sum_{i,j}^{N}I^2(x+i,y+j)-\sum_{i,j}^{N}2\cdot I(x+i,y+j)\cdot\bar{I}(x,y)+\sum_{i,j}^{N}\left(\bar{I}(x,y)\right)^2} \quad (3.7)$$

termos constantes podem ficar fora dos somatórios:

$$R(x,y) = \frac{\left(\sum_{i,j}^{N}T(i,j)\cdot I(x+i,y+j)-\overline{T}\cdot\sum_{i,j}^{N}I(x+i,y+j)-\bar{I}(x,y)\cdot\sum_{i,j}^{N}T(i,j)+N\cdot\overline{T}\cdot\bar{I}(x,y)\right)^2}{\sum_{i,j}^{N}I^2(x+i,y+j)-2\cdot\bar{I}(x,y)\cdot\sum_{i,j}^{N}I(x+i,y+j)+N\cdot\left(\bar{I}(x,y)\right)^2} \quad (3.8)$$

Relembrando a definição de média, na janela de imagem,

$$\bar{I}(x,y) = \frac{1}{N}\sum_{i,j}^{N} I(x+i,y+j) \tag{3.9}$$

e no template,

$$\overline{T} = \frac{1}{N}\sum_{i,j}^{N} T(i,j) \tag{3.10}$$

Na Equação 3.8 vamos substituir o termo $\sum_{i,j}^{N} T(i,j)$ por $N \cdot \overline{T}$, os termos $\sum_{i,j}^{N} I(x+i,y+j)$ por $N \cdot \bar{I}(x,y)$:

$$R(x,y) = \frac{\left(\sum_{i,j}^{N} T(i,j)\cdot I(x+i,y+j) - N\cdot\overline{T}\cdot\bar{I}(x,y) - N\cdot\overline{T}\cdot\bar{I}(x,y) + N\cdot\overline{T}\cdot\bar{I}(x,y)\right)^2}{\sum_{i,j}^{N} I^2(x+i,y+j) - 2\cdot N\cdot\left(\bar{I}(x,y)\right)^2 + N\cdot\left(\bar{I}(x,y)\right)^2} \tag{3.11}$$

Simplificando os termos iguais, tem-se:

$$R(x,y) = \frac{\left(\sum_{i,j}^{N} T(i,j)\cdot I(x+i,y+j) - N\overline{T}\cdot\bar{I}(x,y)\right)^2}{\sum_{i,j}^{N} I^2(x+i,y+j) - N\bar{I}^2(x,y)}\,. \tag{3.12}$$

A equação 3.12 mostra a fórmula baseada na *ZNCC* usada por *(CHEN et al., 2012)* como métrica de similaridade. As modificações feitas na fórmula original da *ZNCC* ajudam a simplificar o hardware necessário para calcular a matriz de resultados. Os resultados obtidos com essa fórmula são diferentes dos resultados de *ZNCC* mas a ordem entre os resultados da matriz é igual. Assim, a posição de maior similaridade na imagem é a mesma para as duas métricas.

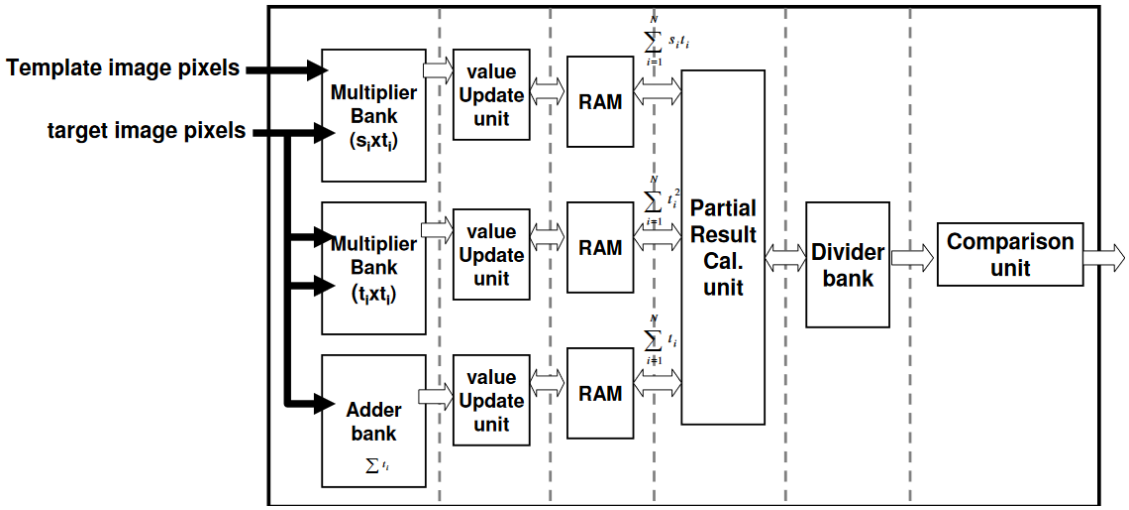
As modificações feitas na fórmula implicam que os valores dos resultados não são normalizados entre -1 e 1, e podem mudar com variações de contraste no *template*, isso dificulta a escolha de limiares de aceitação em algoritmos de busca, como foi mostrado no Capítulo 2.

Além da simplificação da fórmula foi executada uma subamostragem, de 1 pixel amostrado para 4 pixels de tamanho original, na imagem e no *template*, isso foi feito para reduzir a complexidade do problema. Usando essa proporção de 1 para 4, a quantidade de operações é reduzida em 16x. A subamostragem melhora o desempenho mas reduz a qualidade das imagens e pode introduzir erros no resultado.

O diagrama de blocos da arquitetura pode ser visto na Figura 3.2. Para reduzir o uso de banda na comunicação, blocos de memória interna foram instanciados facilitando o reuso de dados. O principal destaque dessa arquitetura é fazer o processamento totalmente em pipeline usando buffers circulares para a acumulação dos resultados parciais e obtenção do resultado final.

A Figura 3.2 mostra a arquitetura proposta para calcular os resultados da Equação 3.12 e selecionar o resultado máximo. Os primeiros três estágios do pipeline calculam e acumulam, sobre a janela de imagem, os resultados de correlação ($\sum IT$), de somatório dos pixels de

Figura 3.2: Arquitetura proposta por *CHEN et al. (2012)* para template matching.



Fonte: (CHEN et al., 2012)

imagem($\sum I$) e de somatório dos quadrados do pixels de imagem ($\sum I^2$). O quarto estágio computa o numerador e o denominador a partir dos resultados dos estágios anteriores. o *Divider bank* calcular os resultados R(x,y) e a *Comparison unit* seleciona o resultado máximo.

Os resultados do artigo mostram que a arquitetura proposta conseguiu um *speedup* que chega a 80x comparando com o mesmo cálculo sendo computado por software e atinge os requisitos mínimos para a aplicação.

3.3 Template matching using DSP slices on the FPGA

Em *HASHIMOTO; ITO; NAKANO (2013)* é apresentada uma arquitetura de hardware com prototipação em FPGA para acelerar o cálculo da *ZNCC* em imagens de baixa resolução. Esse trabalho conseguiu acelerar o algoritmo de busca em multiresolução proposto por *(UCHIDA; ITO; NAKANO, 2011)* em 3.66x comparando ao tempo de execução da mesma implementação na *GPU*, a taxa obtida foi 4273FPS para um template de 4x4 pixels.

Os autores analisaram o algoritmo proposto por *(UCHIDA; ITO; NAKANO, 2011)* e perceberam que a etapa crítica para o tempo de execução deste é a busca em baixa resolução, assim o coprocessador foi pensado para acelerar essa etapa e selecionar as janelas de imagem candidatas a gerar os maiores resultados de *ZNCC*. O tamanho da janela de imagem utilizado no trabalho é 4x4.

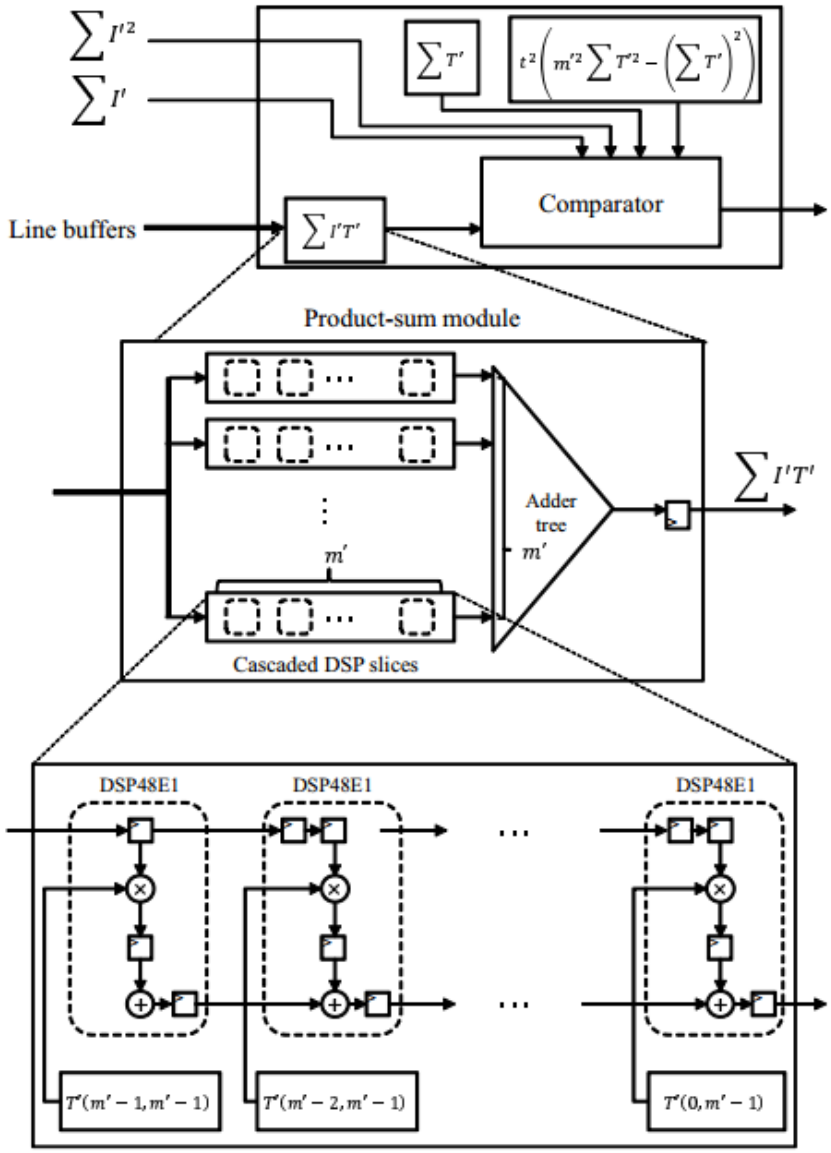
A Figura 3.3 mostra um esboço da arquitetura implementada. O trabalho fornece boas ideias de arquitetura dedicada ao casamento de padrões por análise comparativa. Os valores que dependem apenas do *template* são constantes para todas as janelas de imagem, por isso são calculados um vez por software e armazenados em registradores do hardware.

Os produtos da correlação cruzada entre todos os pixels do *template* e de uma janela

de imagem são feitos em paralelo usando blocos *DSP* no módulo *Product-sum module*, os somatórios são calculados usando somadores em árvore. A computação das janelas de imagem é feita em pipeline usando registradores de deslocamento. Um *buffer* de entrada fornece todas as linhas da janela de imagem em paralelo para as unidades de processamento.

A matriz de resultados não é calculada por essa arquitetura, o módulo *comparator* seleciona os resultados candidatos a gerar maior similaridade com base no numerador e denominador de cada resultado.

Figura 3.3: Arquitetura de processamento de *template matching* proposta por *HASHIMOTO; ITO; NAKANO (2013).*



Fonte: (HASHIMOTO; ITO; NAKANO, 2013)

O trabalho faz uso dos blocos *DSP* da FPGA, para realizar as multiplicações de forma

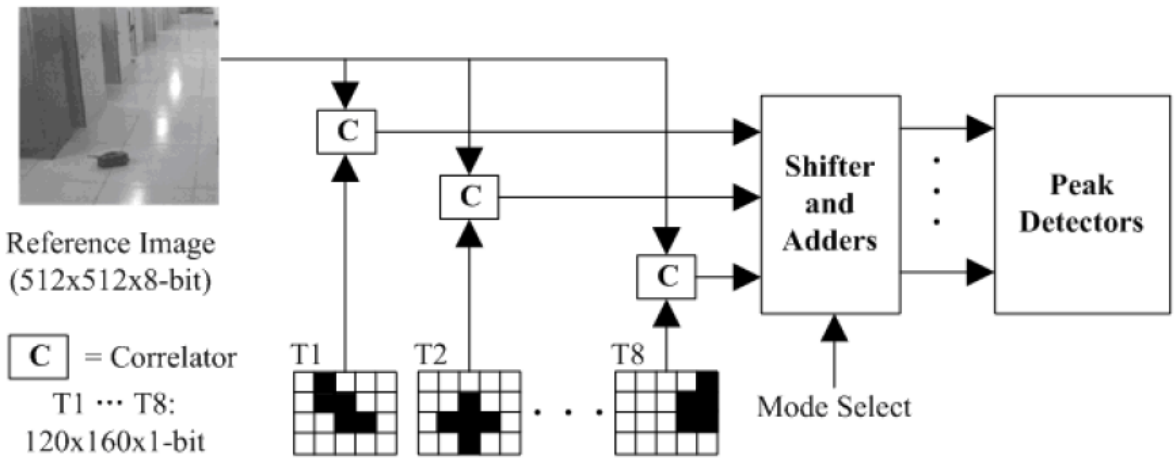
eficiente. A restrição ao uso de blocos *DSP* na síntese dos multiplicadores aumenta a frequência de operação mas pode limitar o tamanho do *template* e o tipo de *FPGA*, no qual a arquitetura será implementada. O ideal é que a arquitetura possa ser igualmente implementada usando blocos *DSP* ou blocos lógicos (ALUTs).

A limitação a *templates* pequenos não chega a ser negativa no trabalho de *(HASHIMOTO; ITO; NAKANO, 2013)* porque a aplicação alvo dele é acelerar por hardware apenas a etapa de busca em baixa resolução do algoritmo proposto por *(UCHIDA; ITO; NAKANO, 2011)*, as outras etapas do algoritmo utilizam *templates* de maiores resoluções mas são executadas em software.

Pensando em implementações mais gerais para serem usadas em outras aplicações, a arquitetura deve suportar *templates* de maiores resoluções, algumas modificações seriam necessárias para fazer parte do cálculo de forma serial e adequar a arquitetura às limitações de recursos do *FPGA*.

3.4 VLSI implementation of multiple large template-based image matching for automatic target recognition

Figura 3.4: Ilustração das multiplicações parciais que permitem a configuração pós síntese da quantidade de *template*s e resultados da operação.

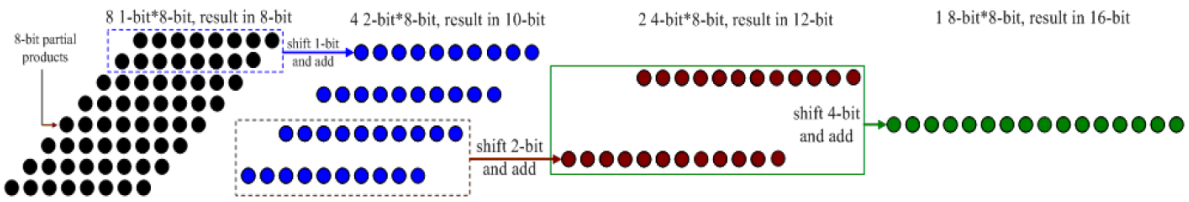


Fonte: *(SANG; LIAO; YUAN, 2011)*

No trabalho proposto por *(SANG; LIAO; YUAN, 2011)* foi desenvolvido um módulo implementado como ASIC para acelerar o cálculo da *NCC* multitemplate aplicada a acelerar a execução do algoritmo de Reconhecimento Automático de Alvos (*ATR*). Essa arquitetura, mostrada na Figura 3.4, pode ser configurada para calcular a *NCC* com apenas 1 *template* de 8 bits, com 2 *template*s de 4 bits, 4 *template*s de 2 bits ou 8 *templates* de 1 bit. Essa flexibilidade foi atingida usando multiplicações parciais, mostradas na 3.5.

Os autores conseguiram um speedup que chega a 31x comparado à implementação do mesmo cálculo em software. Com o intuito de deixar o Circuito Integrado (*CI*) configurável os

Figura 3.5: Ilustração das multiplicações parciais que permitem a configuração pós síntese da quantidade de *template*s e resultados da operação.



Fonte: *(SANG; LIAO; YUAN, 2011)*

elementos de processamento não podem ser construídos usando DSP, apenas blocos lógicos, na prática todas as operações são realizadas com 1 bit e os resultados são deslocados e adicionados no casos de 2,4 ou 8 bits.

3.5 Análise comparativa

Neste capítulo exploramos os principais trabalhos do estado da arte que solucionam o *template matching* com implementação em hardware.

Estes trabalhos foram selecionados por serem os que mais se aproximam de solucionar o problema de implementar um módulo para cálculo de ZNCC multitemplate em hardware.

Alguns aspectos devem ser destacados na comparação entre esses trabalhos. Nessa seção faremos uma breve análise comparativa entre os trabalhos baseada em oito aspectos mostrados na tabela 5.1.

O primeiro aspecto a ser destacado é o tipo de plataforma usada na implementação, no caso dos trabalhos elencados aqui dois deles usaram FPGA *(HASHIMOTO; ITO; NAKANO, 2013)* e *(CHEN et al., 2012)* e um *(SANG; LIAO; YUAN, 2011)* projetou um *ASIC*.

*ASIC*s tendem a ser mais adaptados ao problema, em geral apresentam menor consumo e melhor performance quando comparados ao mesmo design implementado em FPGA. Por outro lado designs implementados em *ASIC* são menos flexíveis. Um ASIC que execute a análise de similaridade por janela deslizante dificilmente será configurável em relação a dimensões de imagem e *templates* e quantidade de *templates*.

No trabalho proposto por *(SANG; LIAO; YUAN, 2011)* a quantidade de *templates* é um parâmetro configurável por software, para contornar o problema da flexibilidade, foram usadas multiplicações parciais para deixar a seleção escolher se a saída será um resultado de 16bits, ou dois resultados de 8bits, ou quatro resultados de 4bits ou oito resultados de 2bits.

Implementações em *FPGA* normalmente apresentam um custo por unidade mais alto e são mais ineficientes em consumo e performance mas possuem a grande vantagem de serem reconfiguráveis via síntese e apresentarem um baixo custo não retornável.

O segundo aspecto destacado na tabela 5.1 é se o design consegue uma taxa de processamento superior a 30FPS. Essa é a taxa de *streaming* vídeo considerada neste trabalho, se o

design consegue uma taxa superior a 30FPS podemos dizer que o processamento em *streaming* é sem perdas. Os três trabalhos expostos neste capítulo atingem taxas de processamento iguais ou superiores a 30FPS e por isso podem ser usados em aplicações *streaming*.

O terceiro aspecto destacado é se o design é capaz de processar múltiplos *templates* em paralelo. Este é um dos objetivos da arquitetura proposta por essa dissertação, processar múltiplos *templates* pode melhorar a acurácia do algoritmo ou permitir o rastreio de múltiplos objetos. Dentre os trabalhos estudados aqui, apenas o de *(SANG; LIAO; YUAN, 2011)* suporta múltiplos *templates* .

A quarta característica analisada é a métrica calculada, algumas métricas e suas particularidades foram expostas na seção 2 do capítulo de fundamentação teórica (Cap. 2). A escolha da métrica além de modificar a complexidade do cálculo, influencia na qualidade dos resultados, avaliada pela robustez a ruído e a variação de contraste. Os trabalhos de *(CHEN et al., 2012)* e *(HASHIMOTO; ITO; NAKANO, 2013)* apresentam melhor robustez a ruído e variação de contraste que o design proposto por *(SANG; LIAO; YUAN, 2011)*.

O sétimo ponto observado é se a métrica implementada entrega resultados normalizados entre -1 e 1, essa característica dos resultados é desejável para algoritmos de rastreio pois facilita a escolha de limiares de similaridade no algoritmo. As simplificações feitas no trabalho de *(CHEN et al., 2012)* deixam os resultados não normalizados e isso é uma desvantagem na comparação com os outros trabalhos.

O oitavo e último aspecto é o tamanho máximo de *template* suportado pelo design. O design de *(HASHIMOTO; ITO; NAKANO, 2013)* consegue processar uma janela de imagem completa em paralelo, e por isso consegue uma taxa de frames por segundo altíssima(4273FPS) porém o tamanho máximo de *template* suportado é 4x4 (16 pixels), que é muito pequeno para a maioria das aplicações.

Tabela 3.1: Quadro comparativo com os principais trabalhos do estado da arte.

Característica	CHEN	HASHIMOTO	SANG
Tipo do Acelerador	FPGA	FPGA	ASIC
Processamento do *Streaming* (>30FPS)	Sim(33,5FPS)	Sim(4273FPS)	Sim(75,5FPS)
Suporte a múltiplos *templates*	Não	Não	Sim
Métrica Calculada	Baseada em ZNCC	ZNCC	NCC
Robusto a variação de contraste e iluminação	Sim	Sim	Não
Resultados normalizados entre [-1 1]	Não	Sim	Sim
Tamanho de *template* suportado	80x80	4x4	120x160

4

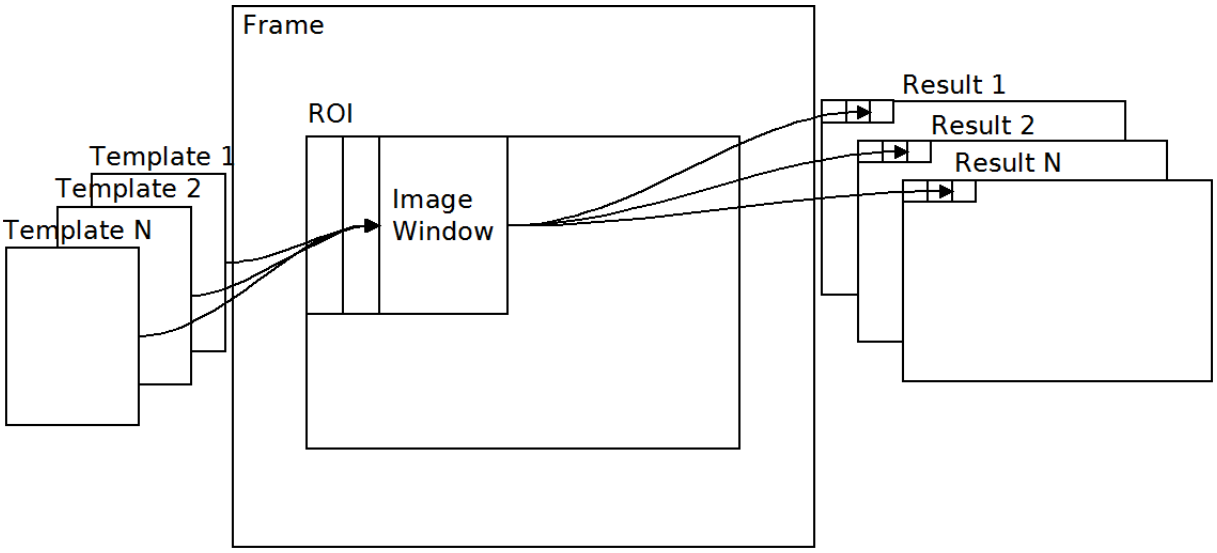
ARQUITETURA PROPOSTA PARA CÁLCULO DE ZNCC MULTITEMPLATE

O cálculo da métrica ZNCC entre uma imagem e múltiplos *templates* é uma operação de alto custo computacional. Considerando o algoritmo de rastreamento de objetos, estudo de caso deste trabalho, o cálculo da ZNCC representa 97% do tempo total de execução. Este capítulo propõe uma arquitetura implementada em FPGA para executar o cálculo da métrica ZNCC entre uma imagem e múltiplos *templates*. Essa arquitetura explora conceitos de paralelismo e pipeline para acelerar o cálculo.

No Capítulo de fundamentação teórica, vimos a fórmula matemática para calcular a ZNCC entre uma imagem e um *template* (Equação 4.1). O objetivo deste trabalho é acelerar o cálculo de múltiplas ZNCCs, obtidas a partir de uma imagem e múltiplos *templates* (Figura 4.3).

$$R(x,y) = \frac{\sum_{i,j}^{N}(I(x+i,y+j) - \overline{I}(x,y)).(T(i,j) - \overline{T})}{\sqrt{\sum_{i,j}^{N}(I(x+i,y+j) - \overline{I}(x,y))^2 . \sum_{i,j}^{N}(T(i,j) - \overline{T})^2}} \tag{4.1}$$

Figura 4.1: Ilustração do cáculo da ZNCC multitemplate.



Nesse cálculo, todos os *templates* são simultaneamente comparados com a mesma imagem. Vejamos um passo a passo algorítmico para calcular as ZNCCs entre múltiplos *templates* e uma imagem:

O primeiro passo é calcular as constantes relativas aos *templates*, esse valores não dependem da posição (x,y) da janela de imagem, assim eles são calculados apenas uma vez e usados no cálculo de todos os elementos da matriz de resultados. Eles são dados pelas médias ($\overline{T}$) e pelos desvios padrão ($\sqrt{\sum_{i,j}^{N}(T(i,j) - \overline{T})^2}$) de cada *template*.

O segundo passo é selecionar a janela de imagem que será usada para calcular as posições (1,1) de todas as matrizes de resultados, essa primeira janela de imagem tem as mesmas dimensões do *template* (OxP) e canto esquerdo superior na posição (1,1).

A partir da janela de imagem selecionada calcula-se a média dessa janela $\overline{I}$ e o desvio padrão ($\sqrt{\sum_{i,j}^{N}(I(i,j) - \overline{I})^2}$).

O produto entre o desvio padrão do *template* e o desvio padrão da janela de imagem, é o denominador da Equação 4.1. Assim, para cada *template* e seu respectivo desvio padrão, obtemos o denominador de cada resultado.

O numerador de cada resultado é a correlação cruzada de média zero entre a imagem e o *template* correspondente ao resultado. Para obtê-lo vamos fazer o produto entre cada pixel da janela de imagem, subtraído da média $\overline{I}$, e cada pixel do *template*, subtraído da média $\overline{T}$. O

acumulado desses produtos é o numerador da Equação 4.1, calculamos o numerador referente a cada *template*.

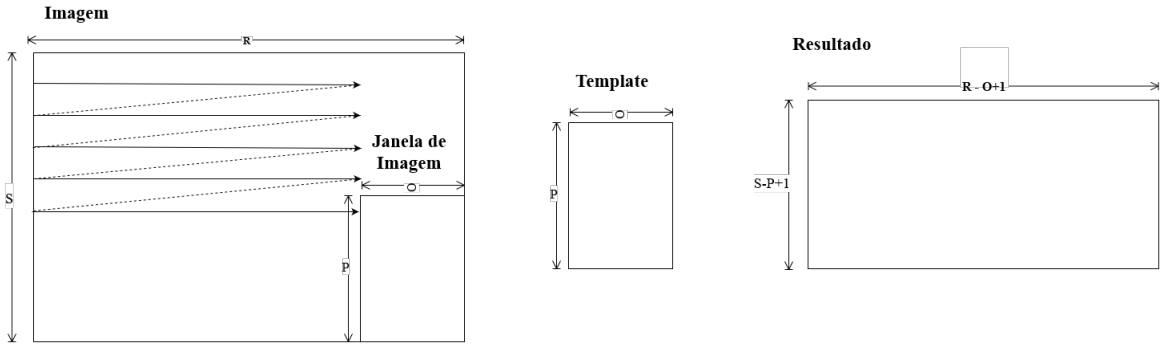
Nesse ponto temos todos os numeradores e denominadores necessários para calcular os elementos das posições (1,1) de cada resultado. Efetuando as divisões, começamos o preenchimento das matrizes de resultados.

O próximo passo é deslocar a seleção da janela de imagem de '1' pixel à direita, na imagem. A nova janela de imagem continua tendo as mesmas dimensões (OxP), e canto esquerdo superior na posição (1,2) da imagem.

Fazendo os cálculos descritos acima, usando os pixels da nova janela de imagem obteremos os elementos (1,2) das matrizes de resultados.

A seleção da janela de imagem é deslocada de um pixel à direita até atingir o limite da imagem à direita, como mostra a Figura 4.2, então, a seleção volta à primeira coluna, deslocada de um pixel para baixo. Ao se atingir o limite inferior direito da imagem, obtemos os elementos do canto direito inferior nas matrizes de resultados e finalizamos o cálculo.

Figura 4.2: Casamento de padrões: Ordem do cálculo, dimensões das entradas e da saída



4.1 Estratégia de cálculo proposta

O passo a passo apresentado acima para calcular as matrizes de resultados da métrica ZNCC entre imagem e *templates*, não é o mais adequado para explorar os recursos do hardware e obter um bom desempenho. A necessidade de calcular a média da janela de imagem antes de calcular o numerador dos resultados, nos obriga a fazer duas iterações sobre cada janela de imagem, uma para calcular a média e outra para calcular a correlação cruzada de média zero.

Assim, a Equação 4.1 foi manipulada algebricamente, para obter uma fórmula mais adequada para uma implementação em hardware, a seguir descrevemos o passo a passo para a obtenção dessa fórmula:

Expandindo o numerador e fazendo a distribuição das somas na Equação 4.1, tem-se:

$$\sum_{i,j}^{N} T(i,j)\cdot I(x+i,y+j) - \sum_{i,j}^{N} \overline{T}\cdot I(x+i,y+j) - \sum_{i,j}^{N} T(i,j)\cdot \bar{I}(x,y) + \sum_{i,j}^{N} \overline{T}\cdot \bar{I}(x,y)$$

Retirando os termos constantes, em relação a 'i' e 'j', de dentro dos somatórios, ficamos com a expressão:

$$\sum_{i,j}^{N} T(i,j)\cdot I(x+i,y+j) - \overline{T}\cdot\sum_{i,j}^{N} I(x+i,y+j) - \bar{I}(x,y)\cdot\sum_{i,j}^{N} T(i,j) + \overline{T}\cdot\bar{I}(x,y)\cdot\sum_{i,j}^{N} 1$$

Sabendo-se que o somatório dos pixels de uma região de 'N' pixels é 'N' vezes o valor médio dos pixels nessa região (Equação 4.2), ou seja,

$$\sum_{i,j}^{N} T(i,j) = N\overline{T} \tag{4.2}$$

ficamos com,

$$\sum_{i,j}^{N} T(i,j)\cdot I(x+i,y+j) - \overline{T}\cdot N\cdot\bar{I}(x,y) - \bar{I}(x,y)\cdot N\cdot\overline{T} + \overline{T}\cdot N\cdot\bar{I}(x,y)$$

Somando os termos semelhantes na expressão, obtemos

$$\sum_{i,j}^{N}(T(i,j)\cdot I(x+i,y+j)) - N\overline{T}\cdot\bar{I}(x,y)\ . \tag{4.3}$$

Agora que o numerador foi simplificado, faremos algo semelhante com o denominador da Equação 4.1. Expandindo os quadrados no denominador:

$$\sqrt{\sum_{i,j}^{N}(T^2(i,j) - 2\cdot\overline{T}\cdot T(i,j) + \overline{T}^2)\cdot\sum_{i,j}^{N}(I^2(x+i,y+j) - 2\cdot I(x+i,y+j)\bar{I}(x,y) + \bar{I}^2(x,y))} \tag{4.4}$$

Distribuindo as somas,

$$\sqrt{(\sum_{i,j}^{N} T^2(i,j) - \sum_{i,j}^{N} 2\overline{T}T(i,j) + \sum_{i,j}^{N}\overline{T}^2)\cdot(\sum_{i,j}^{N}(I^2(x+i,y+j) - \sum_{i,j}^{N} 2I(x+i,y+j)\bar{I}(x,y) + \sum_{i,j}^{N}\bar{I}^2(x,y))}$$

Retirando as constantes para fora dos somatórios,

$$\sqrt{\sum_{i,j}^{N} T^2(i,j) - 2\overline{T}\sum_{i,j}^{N} T(i,j) + \overline{T}^2\sum_{i,j}^{N} 1)(\sum_{i,j}^{N} I^2(x+i,y+j) - 2\bar{I}(x,y)\sum_{i,j}^{N} I(x+i,y+j) + \bar{I}^2(x,y)\sum_{i,j}^{N} 1)}$$

Como foi feito no numerador, vamos substituir a soma dos pixels ao longo de uma região de 'N' pontos por 'N' vezes a o valor médio dos pixels da região.

$$\sqrt{\sum_{i,j}^{N} T^2(i,j) - 2N\overline{T}\overline{T} + \overline{T}^2 N)(\sum_{i,j}^{N} (I^2(x+i,y+j) - 2N\bar{I}(x,y)\bar{I}(x,y) + \bar{I}^2(x,y)N)}$$

Segue,

$$\sqrt{\sum_{i,j}^{N} T^2(i,j) - N\overline{T}^2} \cdot \sqrt{\sum_{i,j}^{N} I^2(x+i,y+j) - N\bar{I}^2(x,y)} \tag{4.5}$$

Combinando o numerador (4.3) e o denominador (4.5):

$$\frac{\sum_{i,j}^{N}(T(i,j)\cdot I(x+i,y+j)) - N\overline{T}\cdot\bar{I}(x,y)}{\sqrt{\sum_{i,j}^{N} T^2(i,j) - N\overline{T}^2}\cdot\sqrt{\sum_{i,j}^{N} I^2(x+i,y+j) - N\bar{I}^2(x,y)}}. \tag{4.6}$$

Para deixar a implementação do sistema mais clara, vamos substituir os valores médios, $\bar{I}(x,y)$ e $\overline{T}$, por suas expressões em função de somatórios, $\frac{1}{N}\sum_{i,j}^{N} I(x+i,y+j)$ e $\frac{1}{N}\sum_{i,j}^{N} T(i,j)$, respectivamente, resultando na Equação 4.7.

$$R(x,y) = \frac{N\sum_{i,j}^{N}(T(i,j)\cdot I(x+i,y+j)) - \sum_{i,j}^{N} T(i,j)\cdot\sum_{i,j}^{N} I(x+i,y+j)}{\sqrt{N\sum_{i,j}^{N} T(i,j)^2 - (\sum_{i,j}^{N} T(i,j))^2}\cdot\sqrt{N\sum_{i,j}^{N} I(x+i,y+j)^2 - (\sum_{i,j}^{N} I(x+i,y+j))^2}} \tag{4.7}$$

Uma manipulação algébrica semelhante a esta pode ser vista no trabalho de (CHEN et al., 2012), ela foi mostrada no Capítulo 3. A principal diferença entre a Equação 3.12 e a Equação 4.7 é que esta mantém os resultados numericamente iguais aos resultados da ZNCC, enquanto aquela mantém apenas a ordem dos resultados.

Comparando a métrica proposta por (CHEN et al., 2012), o cálculo da métrica de similaridade usando a equação 4.7 demanda uma operação de raiz quadrada que aumenta o esforço computacional, no entanto esta equação fornece resultados normalizados e com média zero. Vimos nos capítulos precedentes que essa característica dos resultados facilita a escolha de limiares de aceitação no algoritmos que utilizam a ZNCC para identificação.

Essa é a forma da ZNCC implementada neste trabalho. Deve-se observar que esta fórmula (4.6) é mais adequada a uma implementação em hardware que a fórmula original da ZNCC 2.6, já que esta não exige o cálculo prévio das médias das janelas de imagem, esses valores podem ser calculados paralelamente à correlação.

A Equação 4.7 mostra que para obter cada resultado R(x,y) são necessários três resultados de somatórios de valores de pixels, esses somatórios variam com (x,y) e portanto devem ser computados para cada posição, são eles: $\sum_{i,j}^{N} I(x+i,y+j)$, $\sum_{i,j}^{N} I(x+i,y+j)^2$ e $\sum_{i,j}^{N} T(i,j)\cdot I(x+i,y+j)$.

Além desses somatórios, a fórmula para calcular cada resultado usa o tamanho do

template N, a soma dos pixels do *template* $\sum_{i,j}^{N} T(i,j)$ e o desvio padrão entre os pixels do *template* $\sqrt{N\sum_{i,j}^{N} T(i,j)^2 - (\sum_{i,j}^{N} T(i,j))^2}$. Estas são constantes do *template*, invariantes com (x,y) e por isso podem ser calculadas uma única vez pelo software sem atrapalhar o tempo de execução do algoritmo.

Ao analisar a equação 4.7 obtemos um novo passo a passo para obter as matrizes de resultados de ZNCC. Aplicaremos esse passo a passo ao seguite exemplo numérico para facilitar o entendimento do leitor:

Tabela 4.1: Exemplo imagem 3x3 pixels

1	0	3
4	5	2
4	5	1

Tabela 4.2: Exemplo template 2x2 pixels

5	2
5	1

O primeiro passo é calcular as constantes relativas aos *templates*, elas serão calculadas pelo software e disponibilizadas ao hardware. Usando como exemplo o template mostrado na tabela 4.2, temos:

$$\sum_{i=0,j=0}^{1,1} T(i,j) = 5+2+5+1 = 13 \tag{4.8}$$

$$\sqrt{N \cdot \sum_{i=0,j=0}^{1,1} (T(i,j))^2 - (\sum_{i=0,j=0}^{1,1} T(i,j))^2} = \sqrt{4 \cdot (25+4+25+1) - (13)^2} = \sqrt{51} = 7,14 \tag{4.9}$$

O segundo passo é, para cada posição (x,y), calcular os somatórios relativos a janela de imagem, ou seja, $\big(\sum_{i,j}^{N} I(x+i,y+j)$ e $\sum_{i,j}^{N} I(x+i,y+j)^2)\big)$, e os valores das correlações cruzadas, dados pela expressão: $\big(\sum_{i,j}^{N} T(i,j) \cdot I(x+i,y+j)\big)$, para cada *template*. Esses somatórios são independentes e podem ser calculados em paralelo.

No exemplo, façamos os cálculos dos somatórios e da correlação na posição (x=0,y=0):

$$\sum_{i=0,j=0}^{1,1} I(0+i,0+j) = 1+0+4+5 = 10 \tag{4.10}$$

$$\sum_{i=0,j=0}^{1,1} (I(0+i,0+j))^2 = 1+0+16+25 = 42 \tag{4.11}$$

$$\sum_{i=0,j=0}^{1,1} {}^{1,1}T(i,j) \cdot I(0+i,0+j) = 5+0+20+5 = 30 \tag{4.12}$$

Na imagem exemplo da tabela 4.1 existem quatro janelas de imagem, com origem nas posições: (0,0), (0,1), (1,0) e (1,1). Os resultados do segundo passo para todas as posições, geram as seguintes tabelas:

$\sum_{i=0,j=0}^{1,1} I(x+i,y+j) =$

10	10
18	13

$\sum_{i=0,j=0}^{1,1} (I(x+i,y+j))^2 =$

42	38
82	55

$\sum_{i=0,j=0}^{1,1} I(x+i,y+j) \cdot T(i,j) =$

30	33
55	55

O terceiro e último passo é aplicar as constantes relativas aos *templates* e os resultados parciais obtidos na etapa anterior à equação 4.7 para obter os resultados finais R(x,y). Chamaremos essa etapa de reduções aritméticas.

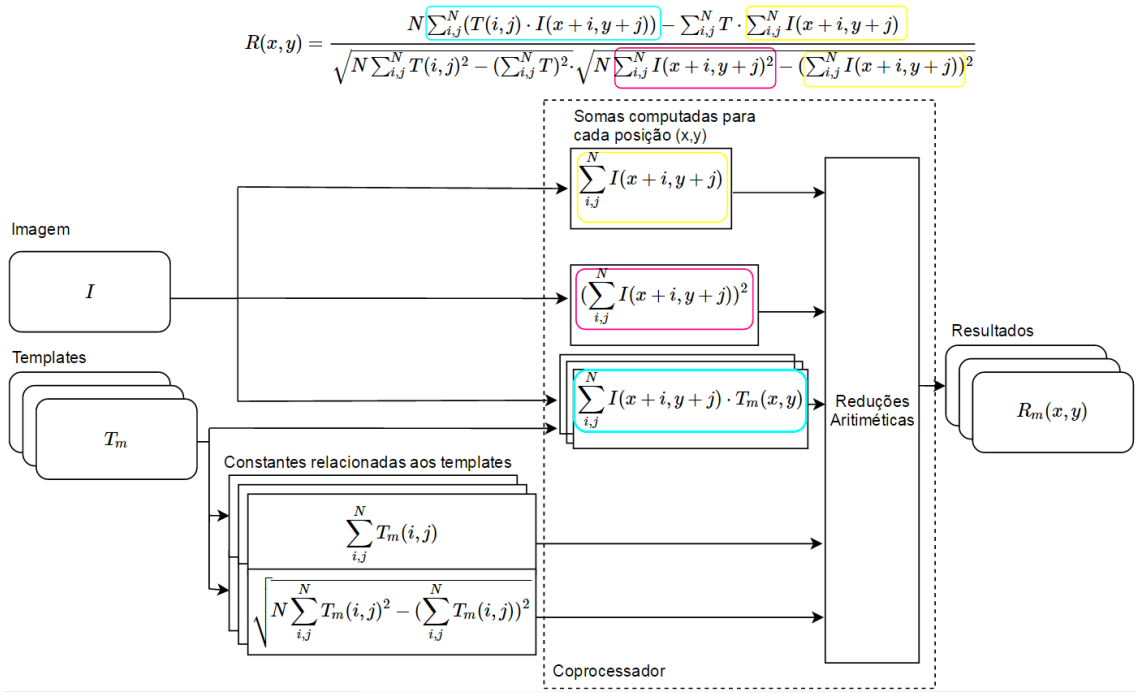
Aplicando as reduções aritméticas ao exemplo, temos:

$$R = \frac{4 \cdot \begin{bmatrix} 30 & 33 \\ 55 & 55 \end{bmatrix} - 13 \cdot \begin{bmatrix} 10 & 10 \\ 18 & 13 \end{bmatrix}}{7{,}14 \cdot \sqrt{4 \cdot \begin{bmatrix} 42 & 38 \\ 82 & 55 \end{bmatrix} - \begin{bmatrix} 10^2 & 10^2 \\ 18^2 & 13^2 \end{bmatrix}}} =$$

-0,17	0,04
-0,98	1

A Figura 4.3 mostra uma estratégia para o cálculo de matrizes de resultados da métrica ZNCC multitemplate, entre uma imagem e múltiplos *templates*. As entradas do sistema são a imagem e o conjunto de *templates*. A partir do conjunto de *templates*, são calculadas, em software, as constantes que servem para o cálculo de todos os resultados $R_l(x,y)$.

Figura 4.3: Mapeamento do cáculo da ZNCC multitemplate.



Resta ao hardware computar, para cada posição (x,y), as somas não constantes (soma de I, soma de I^2), as correlações (somas de IT) e realizar reduções aritméticas de acordo com a

Equação 4.7 para obter os resultados $R(x,y)$. Observe que as etapas que envolvem valores dos *templates* são paralelizadas para o processamento de múltiplos *templates*.

No exemplo mostrado as dimensões da imagem e do *template* são muito pequenas quando comparadas as dimensões de imagens utilizadas em aplicações reais. Imagens com dimensões geram matrizes de resultados parciais muito extensas para serem armazenadas na memória interna do FPGA.

A estratégia para calcular os resultados de ZNCC de forma eficiente, objetivando maximizar o reuso de dados e minimizar a ocupação da memória interna, é calcular os resultados linha por linha. Quando uma linha de resultados é calculada, esses valores são enviados para a saída, liberando espaço na memória interna para processar a próxima linha.

O cálculo de uma linha da matriz de resultados é feito utilizando todos os pixels do *template* e os pixels de uma faixa da imagem, com largura igual à largura da imagem e altura igual à altura do *template*, como as faixas mostradas em destaque na Figura 4.4.

O processamento é feito na sequência das linhas da imagem, os pixels da imagem vão sendo processados da direita para a esquerda gerando resultados por pixel (I, I^2 e IT), quando uma quantidade de pixels igual a largura do *template* é processada, esses resultados por pixel são somados, gerando resultados por linha ($\sum I/linha$, $\sum I^2/linha$ e $\sum IT/linha$).

A seleção se desloca, gerando os resultados por linha da janela de imagem vizinha à direita. Quando a seleção atinge o final da linha de imagem, o próximo pixel a ser processado é o primeiro da linha imediatamente abaixo. O processo recomeça, mas dessa vez os resultados por linha serão acumulados aos resultados da linha anterior.

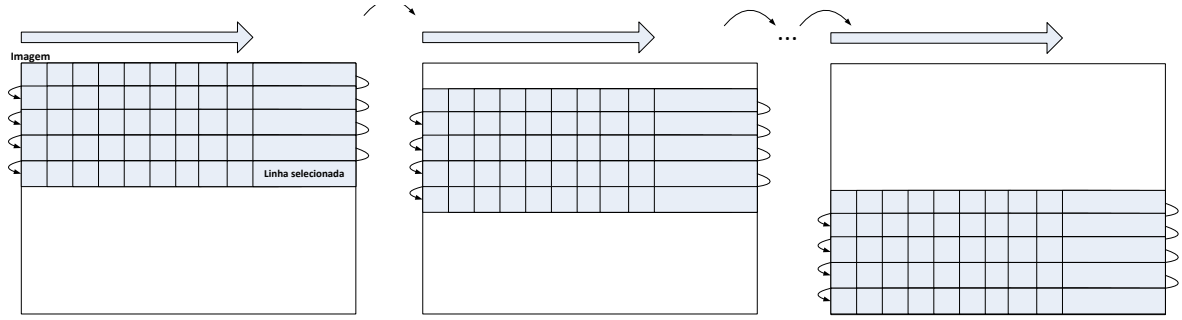
Quando a seleção atinge a última linha da faixa de resultados, começam a ser gerados resultados por janela de imagem, que já podem ser aplicado a Equação 4.7 para gerar a primeira linha de resultados de ZNCC.

Então, a próxima faixa da imagem, deslocada de um pixel para baixo em relação a anterior é processada para gerar a próxima linha de resultados. O processo se repete até que a matriz completa de resultados esteja calculada.

A Figura 4.4 ilustra o deslocamento da seleção de linha ao longo da imagem. Cada quadro mostra uma faixa de imagem selecionada, com os deslocamentos da esquerda para a direita e de cima para baixo até obter os somatórios de todas as janelas de imagem horizontalmente vizinhas. O processo é repetido para as janelas de imagem posicionadas em cada linha até o final da imagem.

Essa seção mostrou o passo a passo para calcular a ZNCC multitemplate, um exemplo reduzido foi mostrado para esclarecer as etapas de cálculo da ZNCC. Nas próximas sessões vamos compreender a arquitetura do hardware para realizar essa tarefa, e o funcionamento de cada módulo, o exemplo apresentado nessa seção será reutilizado para explicar o funcionamento desses módulos.

Figura 4.4: Ilustração do deslocamento da seleção de linha na imagem.

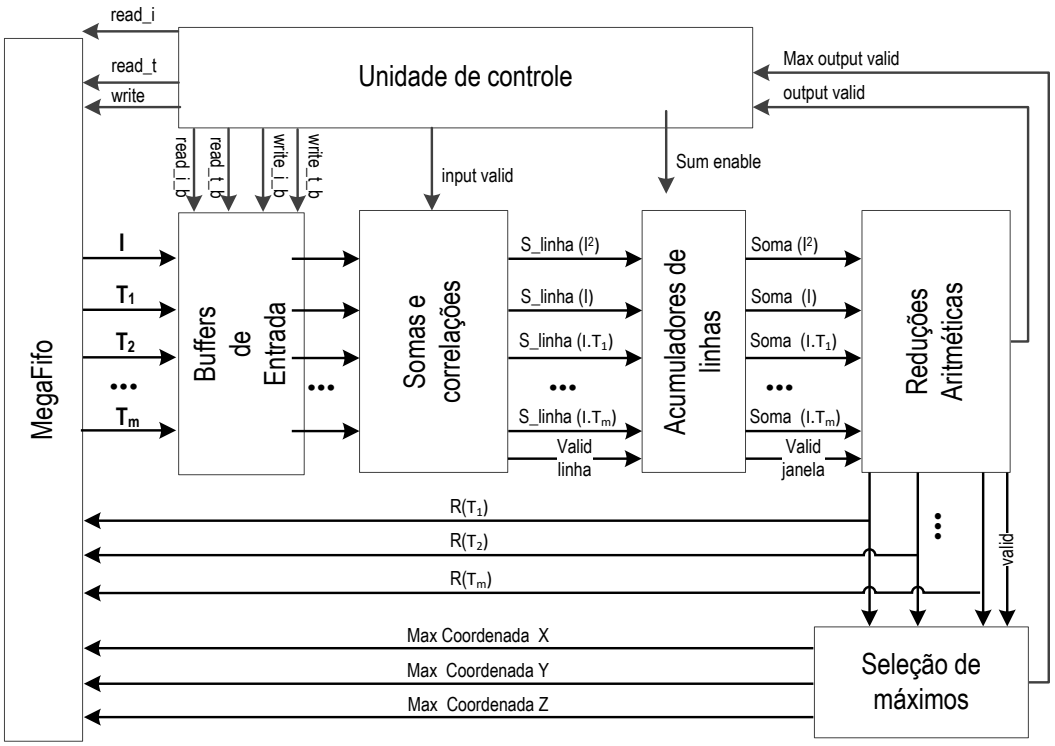


4.2 Visão geral da arquitetura

Esta seção mostra uma visão geral da arquitetura do coprocessador, e uma explicação sucinta sobre cada módulo que a compõe. A estrutura e uma explicação detalhada do funcionamento de cada módulo estão descritas nas próximas seções deste capítulo.

A Figura 4.5 mostra o diagrama de blocos da arquitetura do coprocessador dedicado ao cálculo da ZNCC multitemplate.

Figura 4.5: Visão geral do coprocessador ZNCC multitemplate.



O módulo MegaFifo, é específico para placas da fabricante GIDEL e é mostrado em mais detalhes no próximo capítulo, por enquanto, basta saber que ele é gerado usando o software procWizard GIDEL e serve para gerenciar a comunicação entre o *host* CPU e o FPGA, usando o

barramento PCIexpress.

O módulo *Buffers* de entrada é formado por FIFOs circulares, e tem a função de armazenar entradas (pixels de imagem e dos *templates*) que são reutilizadas ao longo do processamento pelo coprocessador ZNCC.

A computação das somas (soma de I , soma de I^2) e das correlações (soma de IT_x) é realizada pela interação entre dois blocos de processamento, o módulo denominado somas e correlações realiza a soma dos valores ao longo de uma linha, o módulo denominado acumuladores de linhas acumula os resultados das linhas para obter os somatórios por janela de imagem. Esses somatórios por janela de imagem serão utilizados pelo módulo de reduções aritméticas para obter os resultados R(x,y).

O módulo de reduções aritméticas usa as somas armazenadas no acumuladores de linhas para calcular os resultados finais de ZNCC, de acordo com a Equação 4.7.

O módulo seleção de máximos, foi construído usando comparadores, ele calcula e armazena os valores e posições dos máximos em cada matriz de resultados. Esse módulo é útil para otimizar a comunicação. Se o algoritmo do *host* só usa os valores e posições dos máximos da cada matriz de resultados, o tempo de comunicação pode ser reduzido fazendo a leitura desses valores máximos e evitando a leitura de todos os valores das matrizes de resultados.

A unidade de controle gerencia as leituras e escritas no módulo megaFifo e o processamento através do caminho de dados no FPGA.

A arquitetura desenvolvida nesse trabalho é parametrizada, os parâmetros podem ser configurados em uma etapa pré síntese pelo usuário.

Os parâmetros da arquitetura são tratados de forma simbólica neste capítulo, essa abordagem deixa a arquitetura mais genérica que a abordagem numérica, para facilitar a compreensão do leitor, vamos esclarecer quais são os parâmetros da arquitetura. A Tabela 4.3 mostra todos os parâmetros da arquitetura. Alguns valores são função de outros parâmetros, como o tamanho total de cada *template* ou a largura das matrizes de resultados. Outro valores nem aparecem na lista por serem exatamente iguais a um parâmetro já listado, como as dimensões da janela de imagem, que são exatamente iguais as dimensões dos *templates*

Tabela 4.3: Tabela com os parâmetros da arquitetura

Parâmetro	Nome
m	Quantidade de *templates*
O	Largura dos *templates*
P	Altura dos *templates*
N	Quantidade total de pixels de cada *template*(OxP)
R	Largura da imagem
S	Atura da imagem
U	Largura das matrizes de resultados (R-O+1)
V	Altura das matrizes de resultados (S-P+1)
L	Quantidade total de pixels de cada Matriz de resultados (UxV)

As seções seguintes contém descrições mais detalhadas de estrutura e funcionamento de cada módulo.

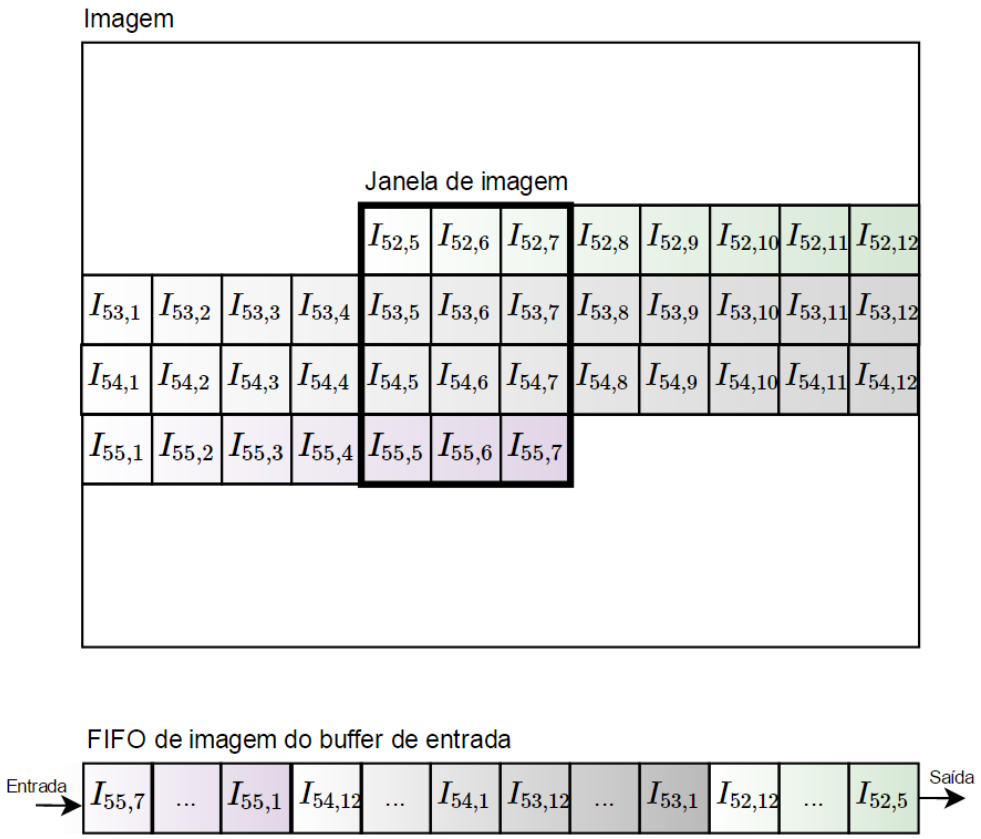
4.3 Buffers de entrada

Os resultados de ZNCC são calculados na mesma ordem dos pixels de imagem, da esquerda para a direita nas linhas e de cima para baixo nas colunas. Quando a primeira linha dos resultados está sendo calculada todos os pixels da imagem e dos *templates* estão vindo diretamente da entrada. A partir da segunda linha dos resultados, os valores usados para calcular a primeira linha de resultado são reaproveitados.

O módulo *Buffers* de entrada instancia FIFOs realimentadas para armazenar esses dados a serem reaproveitados. As FIFOs utilizadas para armazenar os *templates* possuem um controle simples, uma vez preenchida com os pixels dos *templates* vindos da megaFIFO, elas são continuamente realimentadas com os dados da saída, já que todos os pixels dos *templates* são reutilizados do inicio ao fim do processamento.

A FIFO usada para armazenar os pixels de imagem, sempre descarta a primeira linha dos pixels de saída, pois esses pixels não são reutilizados no processamento.

Figura 4.6: Ilustração dos pixels armazenados na FIFO de imagem do módulo *buffers* de entrada, durante o cálculo da ZNCC.



A Figura 4.6 ilustra os dados armazenados na FIFO de imagem do *buffer* de entrada enquanto as linhas de número 52 dos resultados estão sendo calculadas. Os pixels da linha 52 da

imagem são lidos da FIFO, mas não realimentam essa FIFO já que esses pixels não são mais necessários. Os pixels das linhas 53 e 54 são lidos e realimentados na FIFO, pois eles ainda serão reutilizados no cálculo da próxima linha de resultados (linha 53). Os da linha 55 são lidos da entrada e escritos na FIFO para serem reutilizados. Dessa forma, cada pixel só é lido uma vez, da entrada do módulo, e não é necessário armazenar a imagem completa no buffer de entrada, isso otimiza o uso de memória na FPGA.

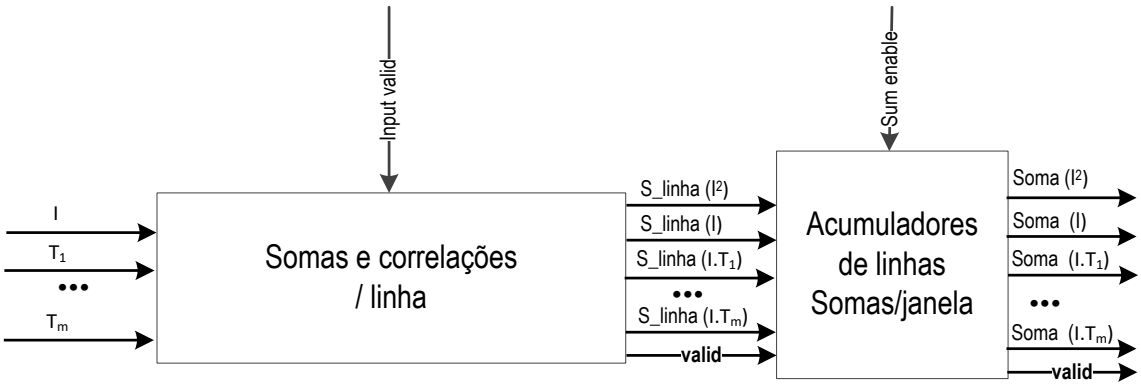
As FIFOs de *templates* do buffer de entrada podem ser vistas como buffers circulares, cada linha é lida e realimentada nessas FIFOs.

4.4 Somas

A computação das somas é uma etapa crítica do cálculo, para calcular 'm' matrizes de ZNCC de tamanho 'L' entre a imagem e os 'm' *templates* de tamanho 'N', são realizadas $L(mN+2N)$ operações de acumulação. Essa é a etapa de maior impacto no tempo total de processamento.

O cálculo das somas envolve disponibilizar os dados a serem somados, que são os valores dos pixels da imagem (I), os quadrados destes (I^2) e os produtos imagem vs *template* ($IT_1, ..., IT_m$), o que demanda recursos lógicos e de armazenamento. Considerando que as dimensões dos *templates* podem ser grandes e que os recursos disponíveis em FPGA são limitados, fazer todo o processamento de uma janela de imagem em paralelo pode limitar muito as dimensões de imagem e *template* suportadas. Para contornar esse problema, o processamento é feito por linha e esses resultados são acumulados para obter os resultados por janela de imagem. A Figura 4.7 mostra os dois módulos envolvidos para a obtenção das somas por janela de imagem.

Figura 4.7: As somas são computadas por linha e esses resultados são acumulados para obter os valores por janela de imagem



Os valores dos pixels são disponibilizados sequencialmente pelo módulo *buffers* de entrada para a etapa de somas. A partir dos pixels de entrada, os dados a serem somados devem

ser disponibilizados aos somadores em árvore. A estratégia para disponibilizar todos os valores relativos a uma linha da janela de imagem, e em seguida todos os valores relativos a uma linha da janela de imagem vizinha é aproveitar a ordem de cálculo da janela deslizante, que naturalmente se encaixa em um vetor sistólico (GUPTA; GUPTA, 2007).

Vetor sistólico é uma estrutura de processamento escalável, bem adaptada a implementações em hardware, que permite paralelizar operações mesmo quando as entradas chegam serialmente (BRENT; KUNG, 1984). Um vetor sistólico é composto por Elementos de Processamento (*PE*)s que calculam e armazenam resultados parciais e em seguida propagam alguns desses resultados no vetor de *PE*s.

O vetor sistólico para calcular todos os valores de uma linha a serem somados é um vetor com 'O' células de processamento, cada célula corresponde a uma coluna 'j' da janela de imagem de largura O. Os valores dos pixels de imagem são propagados na estrutura e os valores a serem somados são disponibilizados para os somadores em árvore.

No início do processamento de uma linha, pixels da imagem e dos *templates* são simultaneamente carregados no vetor de processamento, quando esse vetor está preenchido, os pixels de *template* ficam estáticos e apenas os pixels de imagem vão sendo deslocados até que o último pixel da linha da imagem seja processado.

Os pixels de imagem que entram sequencialmente, vão passando pelo vetor de células de processamento, cada célula gera os resultados da sua respectiva posição 'j', otimizando a geração de resultados por ciclo.

Nas subseções a seguir, vemos como os resultados parciais são gerados, somados e acumulados. Inicialmente, vamos explorar o módulo que realiza as somas e correlações nas linhas, entendendo como funcionam os somadores em árvore e a linha de correlação composta por células de correlação. Em seguida, vamos ver como os acumuladores de linhas funcionam para obter as somas e correlações por janela de imagem.

4.4.1 SOMAS E CORRELAÇÕES POR LINHA

A etapa de somas e correlações por linha é realizada por uma instância de um módulo linha de correlação e m+2 instâncias de somadores em árvore, como mostra a Figura 4.8. Essa é a estrutura necessária para calcular: soma de I, soma de I^2 e as correlações cruzadas entre imagem e *templates*, soma de IT_1 a soma de IT_m, valores por linha em uma aplicação com m *templates*.

A Figura 4.9 retoma o exemplo mostrado nas tabelas 4.1 e 4.2 mostrando a passagem da primeira linha de imagem pelo módulo de somas e correlações. Observe que nesse exemplo os somadores em árvore são simples somadores de duas entradas e uma saída com um ciclo de latência, a vantagem de utilizar somadores em árvore em relação a somadores em cascata só será percebida para *templates* com largura maior ou igual a 4 pixels.

Figura 4.8: Diagrama de blocos do módulo *template* datapath.

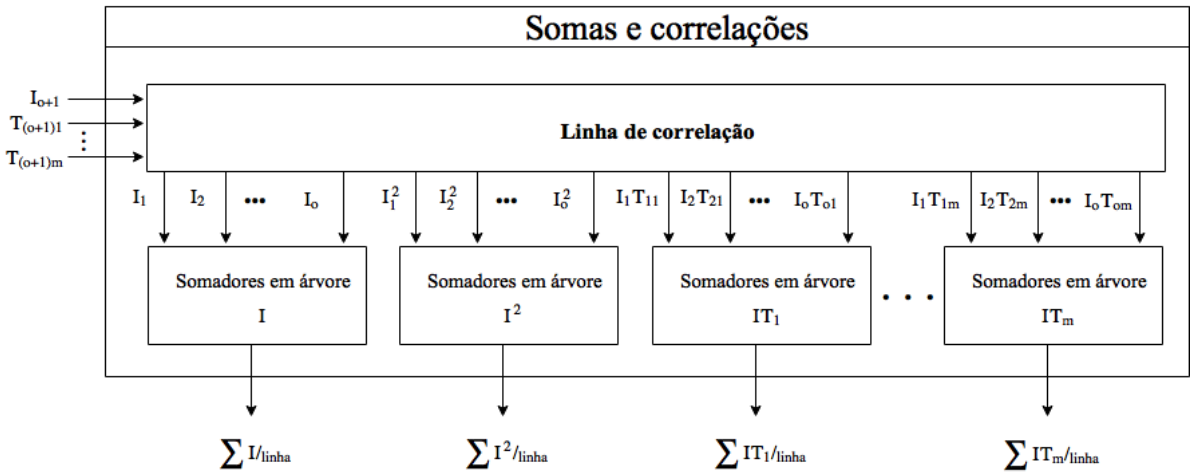
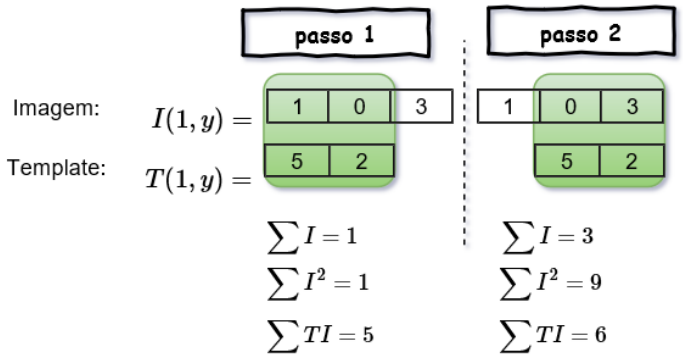


Figura 4.9: Exemplo reduzido: Somas e correlações da primeira linha da imagem.



4.4.1.1 Linha de correlação

A linha de correlação é um vetor sistólico que serve para disponibilizar em paralelo todos os resultados parciais de uma linha a serem somados.

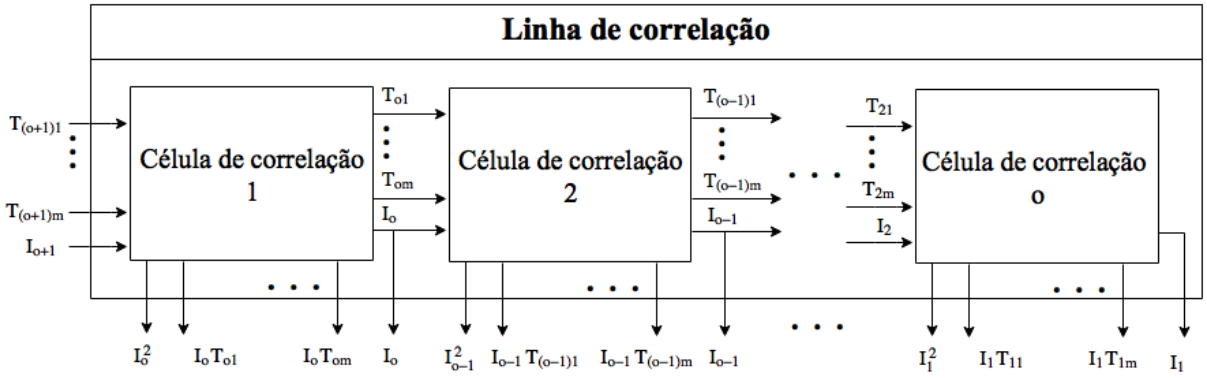
As unidades de processamento são as células de correlação. O valor parâmetro 'O' (largura do *template*/janela de imagem) é a quantidade de unidades de processamento necessárias em uma linha. Esse parâmetro pré síntese é usado para gerar as células de correlação instanciadas em uma linha de correlação.

Quando o processamento de uma linha da imagem se inicia, a linha de correlação vai sendo preenchida com os valores dos pixels da imagem e dos *templates* vindos da entrada, quando 'O' pixels de imagem entram no módulo, todos os valores necessários para calcular as somas parciais, relativas a primeira linha de imagem para a posição (1,1) dos resultados (I, I^2 and IT_1 to IT_m), ficam simultaneamente disponíveis na saída, como mostra a Figura 4.10. A partir desse ciclo, apenas os pixels de imagem continuam a se propagar pela linha de correlação.

A Figura 4.10 mostra um diagrama esquemático de uma linha de correlação; observe que uma linha de correlação contém 'O' células de correlação, e cada célula de correlação calcula

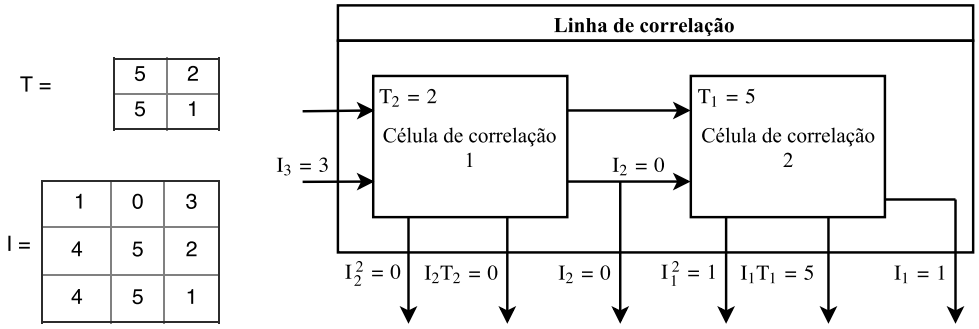
'm' produtos imagem template.

Figura 4.10: Linha de correlação para 'm' *templates* de largura 'O'.



Em regime de funcionamento cada novo pixel de imagem na entrada($I(x,y+1)$), gera um novo conjunto de resultados parciais que serve para calcular as somas parciais da janela de imagem na posição (x,y+1). O deslocamento dos pixels de imagem entre as células da linha de correlação faz com que resultados parciais vizinhos sejam disponibilizados, a cada ciclo, em sequência.

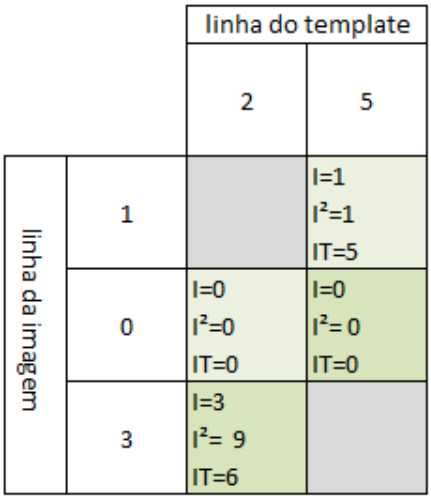
Figura 4.11: Exemplo de linha de correlação em funcionamento.



No exemplo da Figura 4.11, uma linha de correlação tem apenas duas células (O=2). Quando o circuito for ativado, as primeiras entradas da primeira célula de correlação são os pixels I(1,1) e T(1,1), a linha de correlação começa a ser preenchida. No segundo ciclo de relógio, as entradas da primeira célula de correlação seriam os pixels da primeira linha e segunda coluna: I(1,2)=0 e T(1,2)=2, as entradas da segunda célula de correlação seriam as saídas da primeira, I=1 e T=5. Depois de um ciclo de relógio as saídas da linha seriam: $I_2 = 0$, $I_1 = 2$, $(I_2)^2 = 0$, $(I_1)^2 = 1$ e $I_2T_2 = 0$, $I_2T_2 = 5$. Esse primeiro conjunto de resultados parciais teve uma latência de dois ciclos, mas o o próximo conjunto de resultados parciais fica pronto no terceiro ciclo, $I_2 = 3$, $I_1 = 0$, $(I_2)^2 = 9$, $(I_1)^2 = 0$ e $I_2T_2 = 6$, $I_2T_2 = 0$. A linha de correlação tem latência 'O' *throughput* '1'.

A Figura 4.12 mostra o exemplo da Figura 4.11 deixando mais claro o seu funcionamento como vetor sistólico. O primeiro e o último resultados não são resultados válidos, pois a linha de correlação está incompleta. Os elementos das células na primeira diagonal são os primeiros resultados válidos e são obtidos simultaneamente. Em seguida são calculados os elementos das células destacadas na segunda diagonal.

Figura 4.12: Exemplo de linha de correlação em funcionamento do ponto de vista de um vetor sistólico



4.4.1.2 Célula de correlação

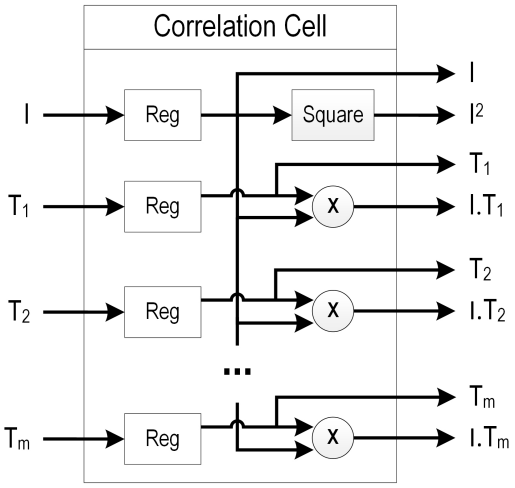
A célula de correlação é a unidade básica deste processamento, ela está representada na Figura 4.13. As entradas desta célula são: um pixel de imagem (I) e um vetor de m pixels de *template* (T_1 ... T_m). As saídas desta célula são: o pixel de imagem (I), o quadrado do pixel de imagem (I^2), o vetor de *templates* (IT_1 ... IT_m) e o vetor de correlações (IT_1 ... IT_m). Os valores das saídas são atualizados um ciclo depois dos valores das entradas.

Observe que os valores I, I^2 e IT_1 ... IT_m são resultados parciais para calcular as somas da Equação 4.7. Os valores I e T_1 ... T_m são usados para alimentar a próxima célula de correlação.

No exemplo das tabelas 4.1 e 4.2, as primeiras entradas de uma célula de correlação seriam os pixels da primeira posição, I(1,1)=1 e T(1,1)=5, depois de um ciclo as saídas da célula seriam: I=1, I²=1, T=5 e IT=5.

Além dos sinais de dados, mostrados na Figura, a célula de correlação possui um sinal de entrada *valid_in* e um sinal de saída *valid_out*, estes são sinais de controle, propagados entre células de correlação.

Figura 4.13: Célula de correlação para 'm' *templates*.



4.4.1.3 Somadores em árvore

Os resultados da linha de correlação precisam ser somados. Somadores em árvore são usadas para realizar essa tarefa de forma eficiente. A estrutura de árvore segue a estratégia dividir para conquistar (CORMEN, 2009).

A vantagem de utilizar somadores em árvores é que a latência de processamento é log_2 da quantidade de entradas a serem somadas, enquanto a latência de um somador em cascata é igual a quantidade de entradas menos um.

A Figura 4.14 ilustra um somador em árvore com 'O' entradas (e), uma saída (S) e $log_2 O$ camadas de somadores.

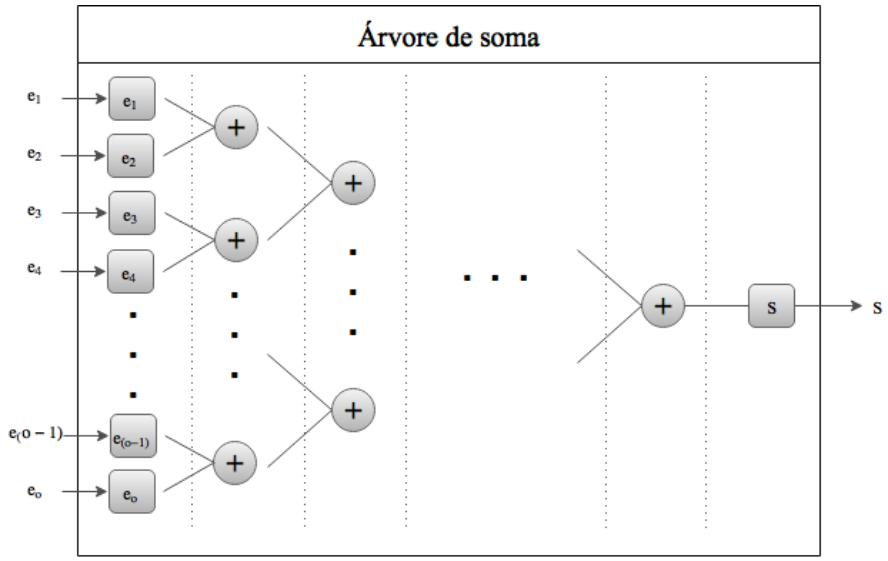
O funcionamento dos somadores em árvore é baseado em fatorar recursivamente a soma de 'O' fatores em uma soma de $\frac{O}{2}$ resultados de somas de dois fatores. A soma dos $\frac{O}{2}$ resultados é a soma de $\frac{O}{4}$ somas de dois fatores e assim sucessivamente até que restem apenas dois fatores a serem somados.

A quantidade de entradas de cada somador, instanciado aqui, é igual a largura do *template* dada pelo parâmetro 'O'. A geração dos somadores é feita utilizando recursão, isto é, cada somador em árvore de n entradas instancia uma camada de n/2 somadores e um somador em árvore com n/2 entradas, a condição de parada da recursão é quando o somador em árvore tem apenas duas entradas, nesse caso ele é um somador simples.

A latência para obter o resultado é função da quantidade de entradas ($log_2 O$), um sinal de *valid* é propagado pela árvore, com a mesma latência do resultado.

Para integrar as saídas da linha de correlação com as entradas da árvore de soma os dados são reorganizados em vetores, e cada vetor de dados será a entrada de uma árvore de soma como mostra a Figura 4.8.

Figura 4.14: Somador em árvore com ‘O’ entradas.

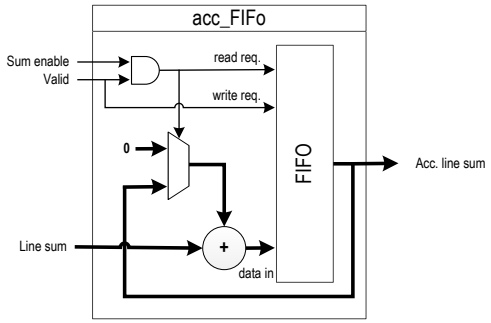


4.4.2 ACUMULADORES DE LINHAS

As saídas do módulo denominado somas e correlações são todas as somas parciais, por linha, necessárias para calcular a ZNCC. As somas completas, por janela de imagem, podem ser obtidas com o acumulado de uma série de ‘P’ de somas por linha. As somas parciais de cada uma das ‘P’ linhas da janela de imagem vão sendo acumulados em um módulo acc_FIFO.

A Figura 4.15 mostra o diagrama de blocos do módulo acc_FIFO, responsável por acumular os resultados das somas das linhas até obter os resultado de uma janela de imagem.

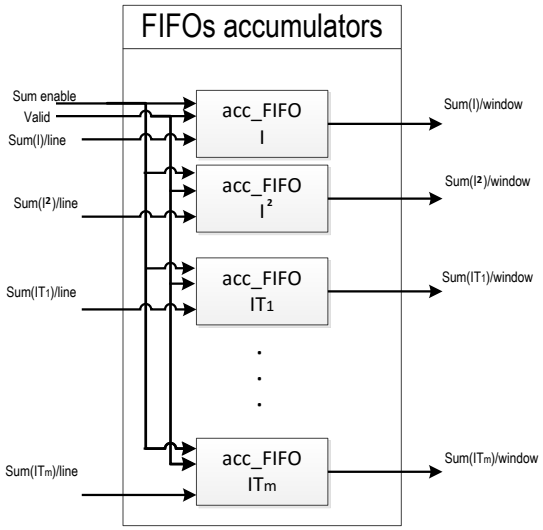
Figura 4.15: Acumulador FIFO para cálculo de somas em uma janela de correlação.



O módulo acc_FIFO é instanciado ‘m+2’ vezes no acumuladores de linhas, um para acumular os resultados da soma de I por linha, um para acumular os resultados da soma de I^2 por linha e m para acumular os resultados das m correlações, ou seja, as somas de IT_1 por linha, de IT_2 por linha até a soma de IT_m por linha, como mostra a Figura 4.16.

O módulo acumuladores de linhas representa o módulo com maior impacto no uso de memória interna do FPGA, para otimizar este uso os resultados são calculados linha a linha, desta forma a FIFO interna só precisa armazenar uma linha de somas parciais por vez, assim o

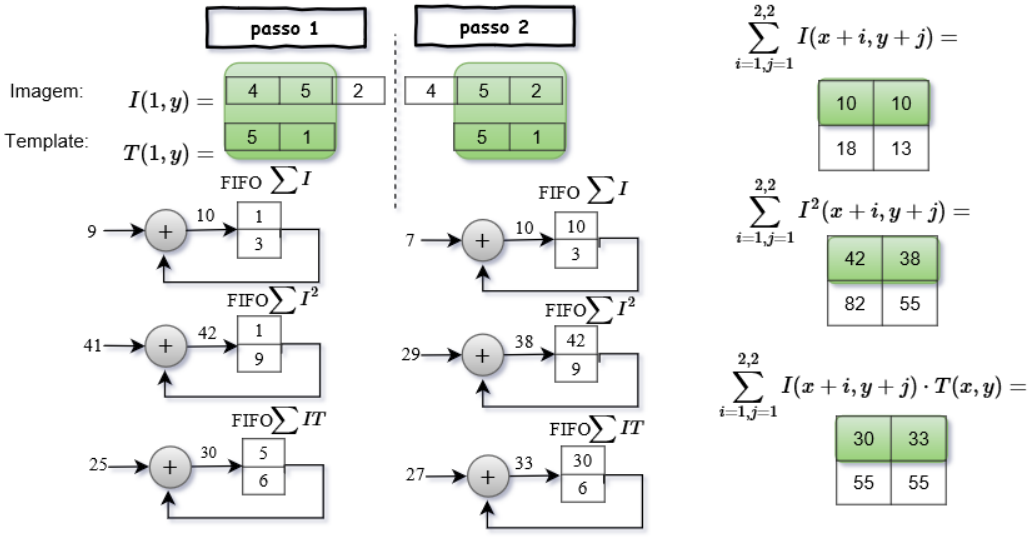
Figura 4.16: Estrutura interna do módulo acumuladores de linhas.



tamanho de cada FIFO é igual a largura das matrizes de resultados ‘R-O+1’.

Quando a última linha de uma janela de imagem está passando pela última linha dos *templates*, os resultados que estão sendo acumulados são liberados, pelo controle, para o módulo de reduções aritméticas, assim ao final do cálculo de uma linha de resultados, o modulo acumuladores de linhas reusa os espaços das FIFOs para acumular as somas parciais para calcular a próxima linha de resultados.

Figura 4.17: Exemplo: módulo acumuladores de linhas em funcionamento.



A Figura 4.17 mostra o preenchimento das FIFOs para o exemplo trabalhado neste capítulo (tabelas 4.1 e 4.2). A primeira coluna, mostra as três FIFOs preenchidas com os valores das somas da primeira linha, calculados na Figura 4.9. Nesse passo, o valor da primeira posição

da FIFO foi lida e somada com o resultado da linha atual.

A segunda coluna da Figura 4.18 mostra que o valor calculado no passo anterior foi escrito na FIFO e o segundo valor está sendo processado. A terceira coluna destaca a primeira linha de somas por janela de imagem, valores que estarão armazenados no acumuladores de linhas para o cálculo da primeira linha de resultados pelo módulo de reduções aritméticas.

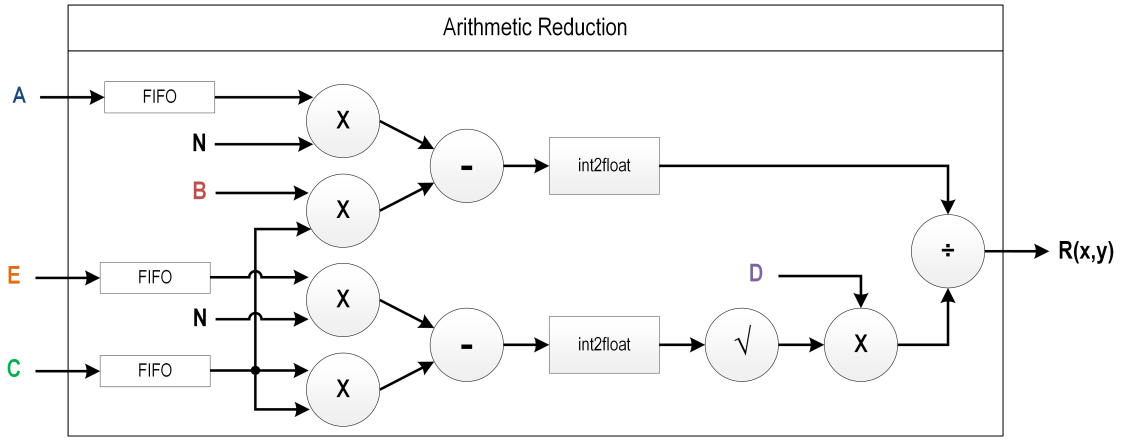
No exemplo, cada FIFO só tem duas posições, já que para essa imagem só existem duas janelas de imagem horizontalmente vizinhas.

4.5 Módulo de reduções aritméticas

O módulo de reduções aritméticas recebe os resultados das somas ($\sum I$, $\sum I^2$ e $\sum T_1 I$ a $\sum T_m I$) e a partir deles calcula o resultado final da ZNCC R(x,y). Os cálculos foram implementados em um pipeline de 6 grandes estágios: multiplicações, subtrações, conversões inteiro em ponto flutuante, radiciação, multiplicação e divisão, esses módulo operacionais são gerados utilizando a plataforma MegaWizard da Altera. A Figura 4.18 mostra a estratégia de cálculo das reduções aritméticas.

Figura 4.18: Mapeamento da equação em diagrama de blocos de implementação.

$$\frac{\overbrace{N\sum_{i,j} T(i,j)I(x+i,y+j)}^{A} - \overbrace{\sum_{i,j} T(i,j)}^{B}\overbrace{\sum_{i,j} I(x+i,y+j)}^{C}}{\underbrace{\sqrt{N\sum_{i,j} T^2(i,j) - \sum_{i,j}^{2} T(i,j)}}_{D}\underbrace{\sqrt{N\sum_{i,j} I^2(x+i,y+j)}}_{E} - \underbrace{\sum_{i,j}^{2} I(x+i,y+j)}_{C}}$$

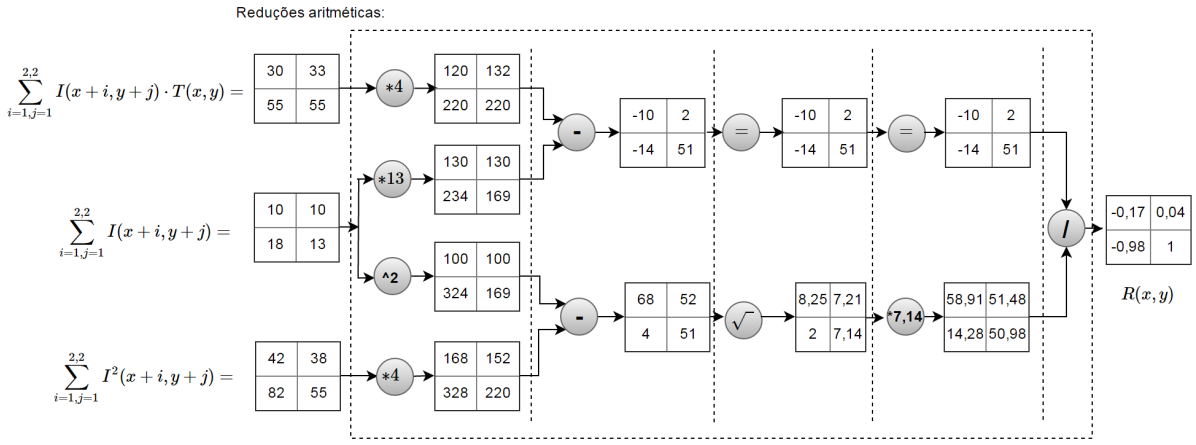


Cada módulo operacional tem estágios internos de pipeline. A latência total dos módulos funcionais do numerador é menor que a latência total do denominador, por isso existem barreiras de sincronização de dados antes do módulo de divisão. Além de barreiras temporais para sincronizar o numerador com o denominador, um sinal de *valid* é propagado junto com os dados pelo pipeline para garantir a validade dos resultados.

Os valores ‘A’, ‘B’ e ‘D’ da Figura 4.18 são, na prática, vetores de valores quando se trata do cálculo com múltiplos *templates*. Em uma arquitetura configurada para ‘m’ *templates* temos ‘m’ valores distintos de ‘A’ dados por $\sum_{i,j}^{N} T_1(i,j) \cdot I(x+i,y+j)$, $\sum_{i,j}^{N} T_2(i,j) \cdot I(x+i,y+j)$, ..., $\sum_{i,j}^{N} T_m(i,j) \cdot I(x+i,y+j)$, ‘m’ valores de ‘B’ dados por $\sum_{i,j}^{N} T_1(i,j)$, $\sum_{i,j}^{N} T_2(i,j)$, ..., $\sum_{i,j}^{N} T_m(i,j)$ e ‘m’ valores de ‘D’ dados por $\sqrt{N\sum_{i,j}^{N} T_1^2(i,j) - (\sum_{i,j}^{N} T_1(i,j))^2}$, $\sqrt{N\sum_{i,j}^{N} T_2^2(i,j) - (\sum_{i,j}^{N} T_2(i,j))^2}$, ..., $\sqrt{N\sum_{i,j}^{N} T_m^2(i,j) - (\sum_{i,j}^{N} T_m(i,j))^2}$. Assim, cada operação que envolve um desses valores vetoriais é composta por ’m’ instancias do mesmo operador.

A Figura 4.19, retoma o exemplo trabalhado ao longo deste capítulo (tabelas 4.1 e 4.2) com as etapas de cálculo dentro deste módulo. Na figura a cada passo é mostrada a matriz completa de resultados apenas para o leitor acompanhar o passo a passo do módulo de reduções aritméticas no hardware. As posições de cada matriz não estão sendo calculadas em paralelo, como a figura pode deixar entender, mas em sequência como foi explicado no texto. O módulo de reduções aritméticas utiliza resultados de somas calculados em um passo anterior e armazenados em FIFOs. Nesse exemplo, o símbolo de igualdade (=) está sendo usado para representar uma barreira temporal que garante que o numerador e o denominador cheguem simultaneamente no módulo de divisão.

Figura 4.19: Passo a passo de cada valor do exemplo dentro do módulo de reduções aritméticas

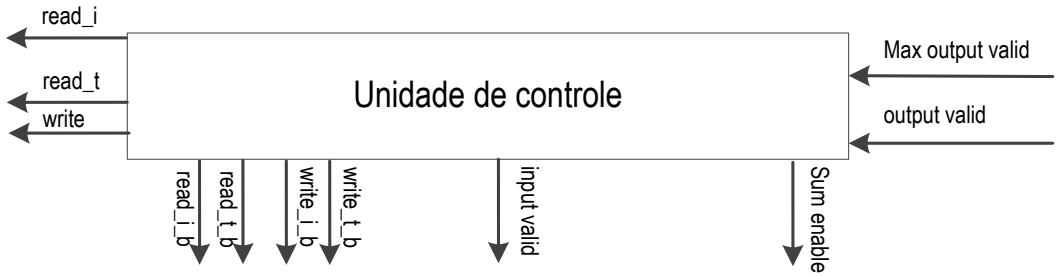


4.6 Unidade de controle

A unidade de controle centraliza as funções de leitura e escrita no megaFifo e nos *buffers* de entrada, e gerencia do processamento de dados, desencadeando as operações de soma, habilitando a acumulação dos resultados parciais e o envio dos resultados finais para o *host*. A Figura 4.20 mostra as entradas e saídas desse módulo.

Esta unidade foi implementada usando uma Máquina de Estados Finita (*FSM*) do tipo Moore de 14 estados. A simplicidade dessa máquina de estados só é possível graças ao uso do sinal de *valid* que caminha junto com os dados facilitando o controle (Figura 4.5). A unidade de

Figura 4.20: Unidade de controle



controle desencadeia uma série de ações que vão ocorrer com os dados válidos caminhando pelo pipeline.

Dos 14 estados mencionados, três são dedicados à comunicação com o PROCmegaFIFO. Considerando o controle do processamento dos dados, essa unidade comanda basicamente três ações: Ler imagem, ler *templates* e escrever resultados.

Os pixels de imagem são lidos quase o tempo todo de processamento, enquanto ocorre o processamento das primeiras janelas de imagem, aquelas que geram a primeira linha de resultados, os pixels são lidos diretamente da entrada. Nas janelas de imagem seguintes, a maior parte dos pixels já está armazenada em *buffers*, e são lidos dos *buffers* de entrada, exceto os da última linha de cada janela de imagem.

Os pixels de *templates* são lidos da entrada durante o processamento da primeira linha de resultados, no processamento das próximas janelas todos os *templates* já estão armazenados e são lidos dos *buffers*. Diferente dos pixels de imagem, os pixels dos *templates* não são lidos durante boa parte do tempo de processamento, pois eles ficam fixos na linha de correlação enquanto os pixels de imagem são deslocados.

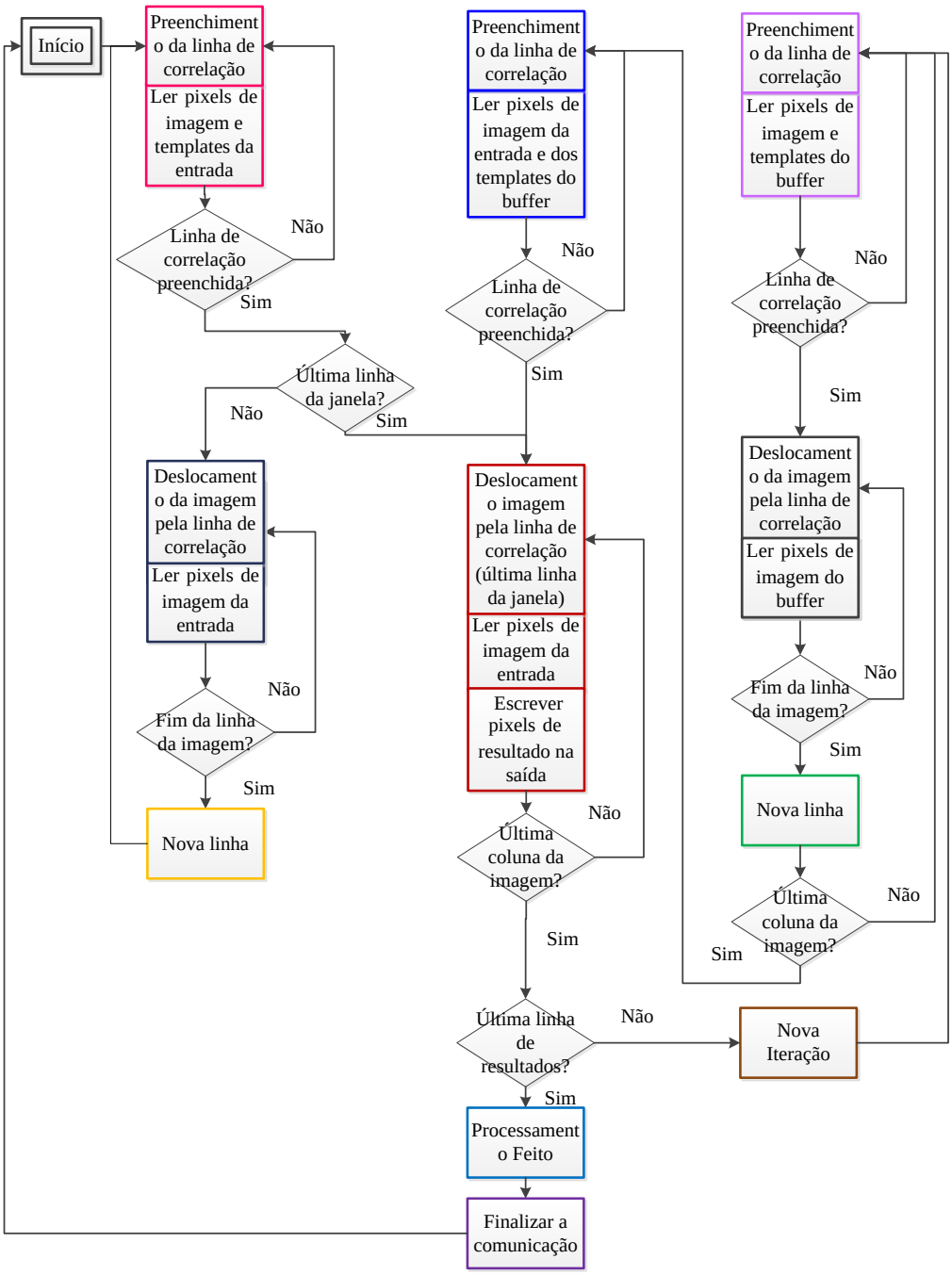
Os pixels dos resultados são escritos na saída. Uma linha de pixels de resultados é obtida a cada ‘P’ linhas de imagem processadas.

A Figura 4.22 contém o diagrama de estado da unidade de controle, as tarefas desempenhadas pelo controle podem ser resumidas no fluxograma da Figura 4.21, para facilitar a compreensão as cores usadas na Figura 4.21 indicam os estados da *FSM* representada na Figura 4.22.

O preenchimento da linha de correlação ocorre no início do processamento de cada linha da imagem, essa etapa corresponde aos ‘O’ ciclos necessários para preencher todas as células da linha de correlação com valores da imagem e do *template*. Existem 3 estados onde ocorre o preenchimento da linha de correlação.

No início do processamento, quando a primeira linha de resultados está sendo calculada, a linha de correlação é preenchida a partir dos dados imagem e *template* vindos diretamente do

Figura 4.21: Fluxograma da unidade de controle.

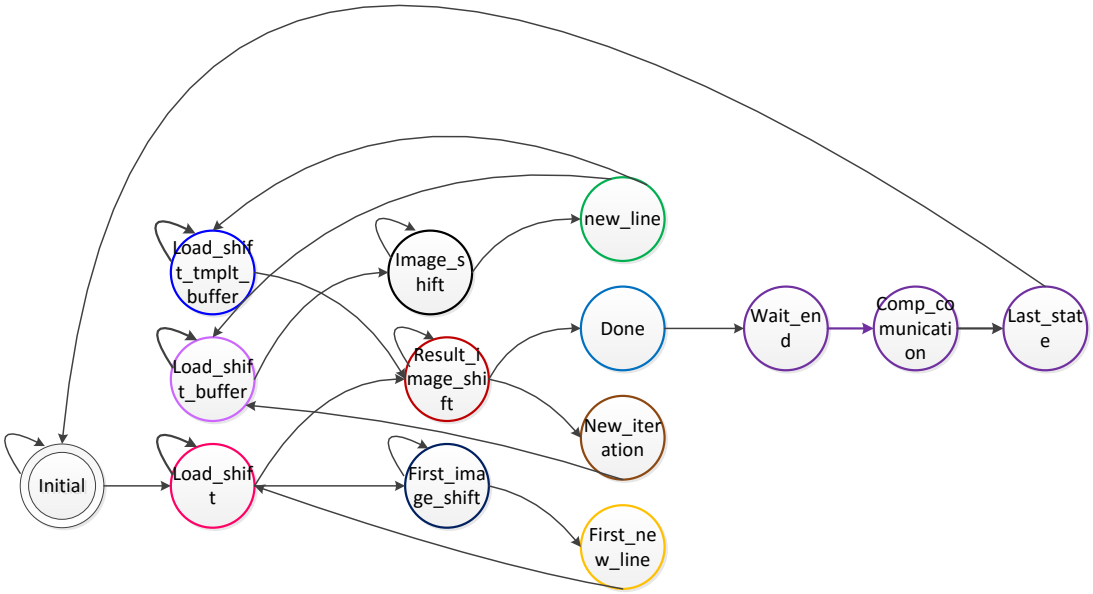


host através do módulo megaFifo, esse estado corresponde ao processamento pela primeira vez, dos pixels da região de imagem destacada na Figura 4.23(a).

A partir da segunda linha de resultados, todos os pixels de *template* e parte dos pixels da imagem já estão armazenados nos *buffers* de entrada, nesse caso eles são lidos desses *buffers* e não da megaFifo. Os pixels de imagem da região destacada na Figura 4.23(b) já estão armazenados nos *buffers* de entrada durante o processamento da segunda linha de resultados.

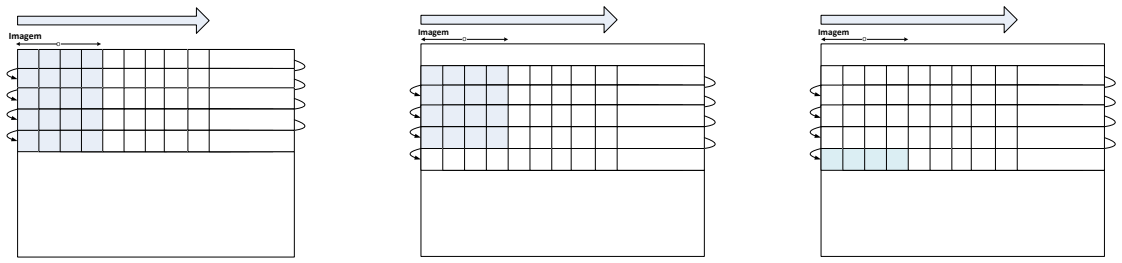
Durante o preenchimento da linha de correlação da última linha de uma janela de imagem, os pixels de imagem não estão armazenados nos *buffers* de entrada já que eles não

Figura 4.22: FSM da unidade de controle.



foram processados na iteração anterior, esses pixels estão em destaque na Figura 4.23(c) e são lidos diretamente da entrada megaFifo. Enquanto isso, os pixels de *template* já estão armazenados nos *buffers* de entrada.

Figura 4.23: pixels de imagem para etapas de preenchimento de linha de correlação



(a) 1^a iteração (b) 2^a iteração, pixels armazenados (c) 2^a iteração, pixels não armazenados

Após o preenchimento de uma linha de correlação os pixels de *template* ficam congelados enquanto os pixels de imagem são deslocados até o final da linha de correlação.

Existem três estados onde ocorre o deslocamento da imagem pela linha de correlação. Na primeira iteração, não há pixels de imagem armazenados, todos são lidos do megaFifo. Quando a linha deslocada é a última linha de uma iteração, também não há pixels de imagem armazenados. Nesse caso, existe a particularidade de que ao final dessa etapa inicia-se uma nova iteração. O terceiro tipo de deslocamento de imagem ocorre a a partir da segunda iteração em linha que já foram processadas, nesse caso os pixels de imagem são lidos do *buffer* de entrada.

Ao fim do deslocamento da última linha de imagem pela linha de correlação, os últimos resultados do frame são calculados.

4.7 Módulo de seleção dos máximos

O módulo seleção de máximos, é construído por comparadores, ele calcula e armazena os valores e posições dos máximos em cada matriz resultados. As saídas produzidas por esse módulo podem servir para otimizar o uso da banda de comunicação. Caso o algoritmo do *host* só use os valores de máximo e não a matriz completa de resultados, a quantidade de leituras feitas pelo *host* é reduzida em 'L' vezes.

Esse design foi pensado para uma aplicação *stream*, assim a unidade de controle só para se o *host* parar de demandar ZNCCs, nesse caso a máquina fica parada no estado inicial, ou se ocorrer falta de dados, nesse segundo caso todas as variáveis de estado congelam e uma bolha é inserida no pipeline dos dados, até o fim da falta de dados ou a solicitação de reinicialização.

4.8 Conclusões

Este capítulo apresentou a organização interna do módulo ZNCC multitemplate. Cada bloco constitutivo foi apresentado, bem como seu funcionamento e a interação entre blocos. A arquitetura apresentada nesse capítulo é genérica para qualquer tipo de FPGA.

O quinto capítulo deste trabalho apresenta aspectos práticos e específicos da implementação FPGA, realizada nesse trabalho. Ele contém detalhes sobre a geração do megaFifo, uso de recursos da placa e integração hardware/software.

5

IMPLEMENTAÇÃO FPGA

Este capítulo aborda os aspectos práticos da construção do sistema para rastreamento de objetos. Os principais tópicos explorados são: a metodologia utilizada no desenvolvimento do projeto, os dispositivos utilizados, a comunicação hardware/software, as configurações pré síntese e resultados de síntese.

5.1 Dispositivos

Esse trabalho foi implementado no kit de desenvolvimento Gidel Proc-Star IV. Essa plataforma possui quatro FPGAs Stratix IV (EP4SE530H35C2) disponíveis com uma memória DDR2 512MB conectada a cada FPGA, o sistema desenvolvido neste trabalho utiliza apenas um FPGA.

A *CPU* utilizada para rodar a aplicação foi um core i7, 3.3Ghz, 16GB de memória RAM, essa CPU poderia ser substituída por um processador embarcado sem perdas significativas para a performance do sistema, a única condição é que o processador tenha uma porta PCI-express para a comunicação hardware/software.

A comunicação entre *CPU* e *FPGA* acontece via barramento PCI-express usando a memoria DDR2 conectadas ao *FPGA* para sincronizar os dados.

5.2 Metodologia de projeto

Durante este trabalho de mestrado foi empregado o fluxo de desenvolvimento de projetos hardware mostrado na Figura 5.1.

O Primeiro passo é o estudo do algoritmo para planejar como o sistema irá funcionar. Nesse caso, a equação foi simplificada e os termos contantes referentes unicamente ao *template* foram isolados para serem calculados via software.

Em seguida foi elaborado um projeto de arquitetura de fluxo de dados. Nessa etapa foi decidida a ordem que os cálculos seriam feitos; o que seria executado em paralelo, pipeline ou em sequência; como seria a organização dos dados intermediários e como os dados de entrada poderiam ser reaproveitados.

Um modelo de referência foi elaborado e implementado em Matlab, seguindo o mesmo fluxo de dados pensado na etapa anterior, para testar a viabilidade do projeto. A partir desse projeto foi elaborada a arquitetura estrutural, com descrições de entradas e saídas e funcionalidades de cada módulo do design.

Baseando-se nessa especificação, cada módulo foi descrito em Linguagem de Descrição de Hardware (*HDL*) Verilog. Paralelamente a isso, foi elaborado um ambiente de verificação para cada módulo isolado e uma abordagem bottom-up foi usada para o design e verificação dos módulos integrados.

Cada módulo foi testado isoladamente, e a cada etapa de integração, um novo teste geral era executado. A verificação foi feita usando o ModelSim.

O teste final foi feito com sequências de 1000 *frames*, gerados aleatoriamente comparando os resultados da arquitetura aos do modelo de referência.

A cada etapa do desenvolvimento, o design foi verificado e sintetizado, usando o Quartus II, para verificar o uso de recursos do *FPGA* e os caminhos críticos do design. Algumas voltas a passos anteriores foram necessárias, seja para ajustar funcionalidades, seja para reduzir o uso de

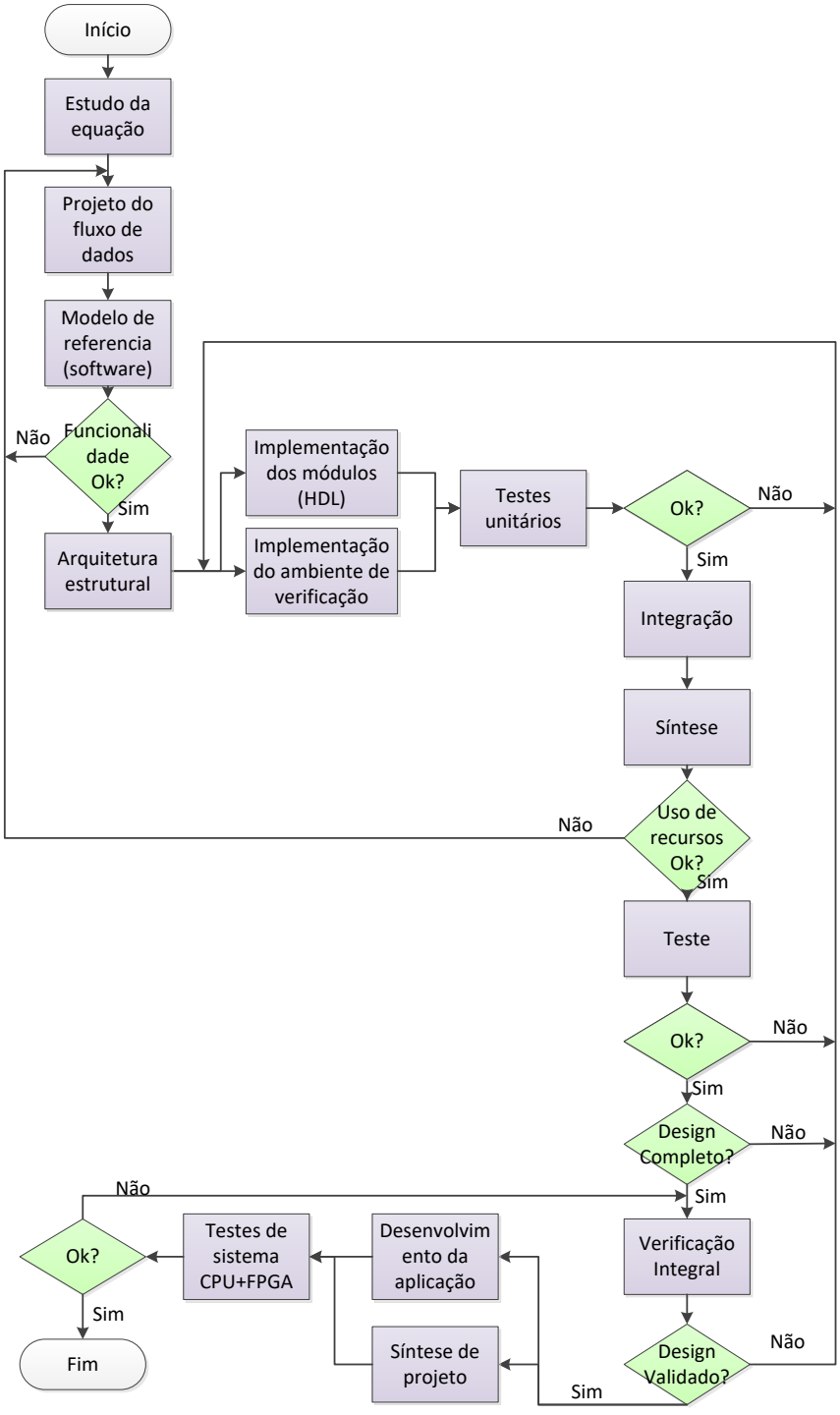


Figura 5.1: Fluxo de desenvolvimento de projeto

recursos.

A abordagem *bottom up* usada para o desenvolvimento e a verificação dos módulos permitiu encontrar grande parte dos erros em escalas menores do desenvolvimento, quando é mais fácil isolá-los e repará-los.

Tendo o design completo e verificado a aplicação do *host* foi desenvolvida em OpenCV e o design implementado na FPGA. Nesta etapa, foram feitos os testes completos do sistema, já com o *host CPU* e o acelerador *FPGA*, para esses testes foram utilizados vídeos com tamanhos de 300 e 10000 frames. A arquitetura também foi testada variando os parâmetros pré síntese,a quantidade de *templates* foi variada de 1 a 10, e as dimensões de imagem testadas foram 432x432 e 216x216.

5.3 MegaFifo e a integração hardware software

A biblioteca Gidel PROCWizard GIDEL (2010) disponibiliza alguns módulos Propriedade Intelectual (*IP*) para para facilitar a integração hardware/software: PROCMegaFIFO, PROC-MegaDelay e PROCMultiPort. O PROCMegaFIFO IP foi projetado para aplicações em *stream*, assim ele é adequado para a aplicação de rastreamento em tempo real deste trabalho.

O PROCMegaFIFO usa o barramento PCIexpress para comunicação e a memória DDR2 para sincronizar os dados vindos da *CPU* para o *FPGA* e do *FPGA* para a *CPU*, já que os dois dispositivos trabalham em frequências diferentes. A Figura 5.2 mostra um diagrama de blocos do sistema.

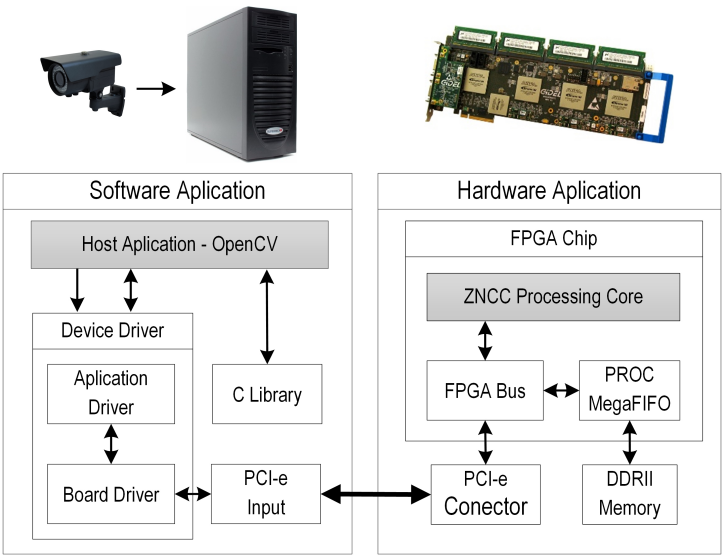


Figura 5.2: Visão geral do sistema para rastreamento de objetos acelerado por *FPGA*.

Neste projeto, foram sintetizadas três instâncias do *IP* core PROCMegaFIFO (GIDEL, 2008). Uma SimpleFIFO para os pixels de imagem, uma para o vetor de pixels dos templates e uma para o vetor de pixels dos resultados. Cada simpleFIFO tem o tamanho de dados limitado a 256 bits, assim uma única porta pode enviar até 32 templates de 8bits ou 8 resultados de 32bits. Quando o design usa mais de 8 templates, deve-se acrescentar uma simpleFIFO para transmitir os resultados extras.

Muitas vezes, as aplicações do casamento de padrões só utilizam as informações do ponto de melhor casamento, sua posição e o valor de similaridade. Assim, o módulo desenvolvido neste

trabalho também envia os valores e as coordenadas do máximo de cada matriz de resultados. Essa porta é útil para reduzir a quantidade de dados transmitidos e consequentemente o tempo de comunicação.

Tabela 5.1: Uso de recursos do FPGA apenas para o PROCmegaFIFO.

Combinational ALUTs	8,302 / 424,960 (2 %)
Dedicated logic registers	9,299 / 424,960 (2 %)
Total pins	726 / 744 (98 %)
Total block memory bits	464,912 / 21,233,664 (2 %)

A Tabela 5.1 mostra o uso de recursos apenas das instâncias do PROCmegaFIFO usadas neste projeto. É sempre um ponto de discussão saber quando vale a pena o uso desse tipo de *IP* core, já que ele utilizada recursos do *FPGA* que podem ser críticos em alguns projetos. Neste projeto, o uso deste *IP* foi vantajoso, ele permitiu que os esforços fossem concentrados no processamento dos dados facilitando a leitura e escrita das entradas e dos resultados.

5.4 Sínteses e resultados

Para validar a implementação do módulo proposto foi usada como estudo de caso a aplicação de rastreamento de pedestres, conforme discutido na Seção 2.4 deste trabalho. Nessa aplicação, a imagem é a região onde se busca o *template* (ROI): essa região tem 432x432 pixels, a dimensão dos *templates* é adaptada à busca de pedestres e é 72x144 pixels. A quantidade de *templates* foi variada de um a dez *templates* para a análise da taxa frames por segundo (*fps*) obtida em cada caso.

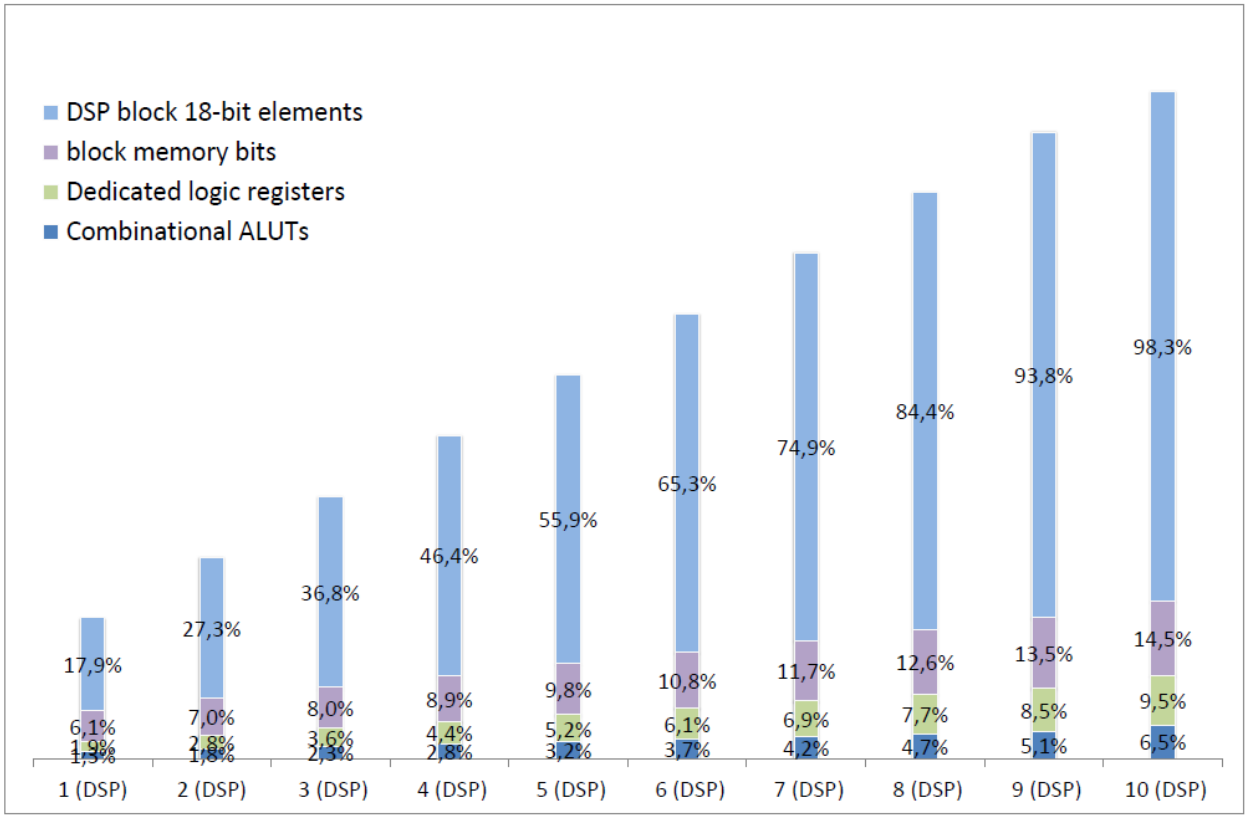
Os multiplicadores são operadores com várias instanciações nessa arquitetura, já que cada célula de correlação instancia pelo menos dois multiplicadores. Em um módulo sintetizado para um *template* cada célula de correlação instancia um multiplicador para calcular o quadrado do pixel de imagem (*I*) e um multiplicador para calcular o produto entre o pixel de imagem e o pixel do *template* (*IT*), de forma mais genérica cada célula de correlação instancia um multiplicador para calcular I² e 'm´ multiplicadores para calcular IT.

5.4.1 SÍNTESE PADRÃO

A síntese padrão do *Quartus Prime 15.1* usa um bloco DSP 18 bits para construir cada um desses multiplicadores. Aumentando o parâmetro quantidade de *templates*, mais multiplicadores são instanciados por cada célula de correlação, isso se reflete no uso de blocos DSP mostrado no relatório de síntese.

A Figura 5.3 mostra o uso dos principais recursos disponíveis no FPGA de acordo com a quantidade de *templates* configurada. Observe que quando o design é parametrizado para suportar 10 *templates* 98% dos DSPs disponíveis são utilizados, nesse caso a ferramenta não

Figura 5.3: Uso de recursos do FPGA, por quantidade de *templates*, usado multiplicadores escolhidos na síntese padrão construídos usando blocos DSP

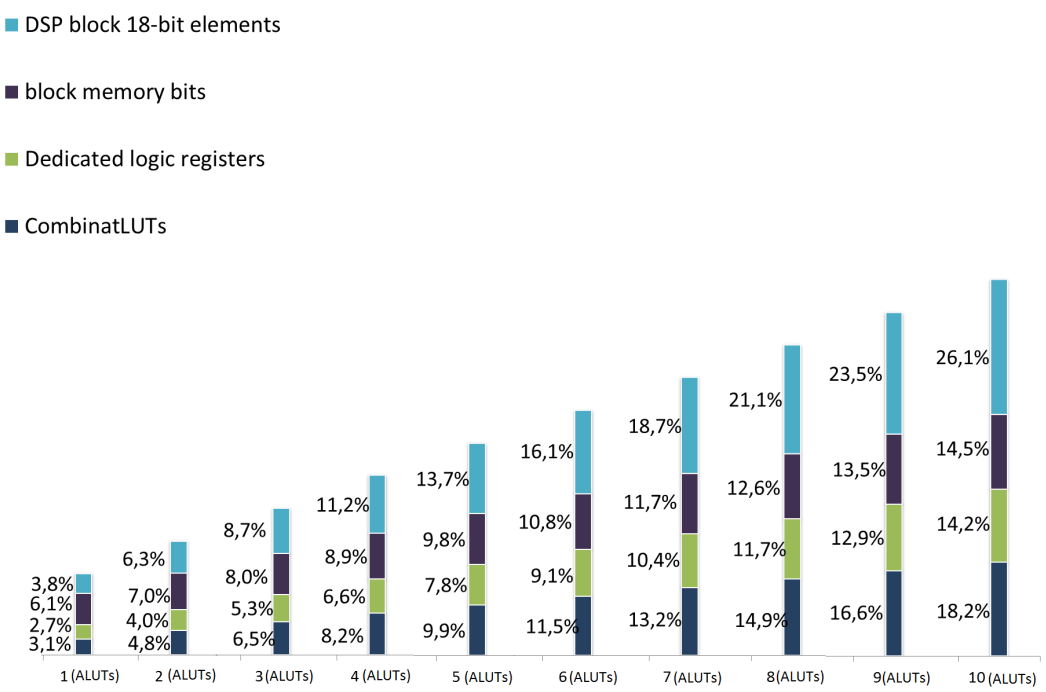


consegue realizar a etapa de roteamento da síntese, essa etapa se torna cada vez mais complexa com o aumento da ocupação no FPGA.

5.4.2 SÍNTESE PRIORIZANDO O USO DE LÓGICA COMBINACIONAL

Para deixar o design mais flexível em relação a quantidade de *templates* uma nova célula de correlação foi implementada, forçando a escolha blocos de lógica combinacional (ALUTs) para construir os multiplicadores, nesse caso, cada multiplicador IT utiliza 80 desses blocos, enquanto os multiplicadores com duas entradas iguais (I^2) utilizando 40 desses blocos. A principal diferença é que blocos de lógica combinacional são mais abundantes nas placas que os blocos DSP, assim o o módulo que usa blocos combinacionais na construção dos multiplicadores é sintetizado com folga no FPGA stratix IV mesmo quando configurado para processar 10 *templates*. A Figura 5.5 mostra o uso de recursos o uso dos principais recursos disponíveis no FPGA de acordo com a quantidade de *templates*, quando se força o uso de blocos ALUTs para construir os multiplicadores.

Figura 5.4: Uso de recursos do FPGA por tipo de multiplicador usado(DSP ou ALUTs) e por quantidade de templates

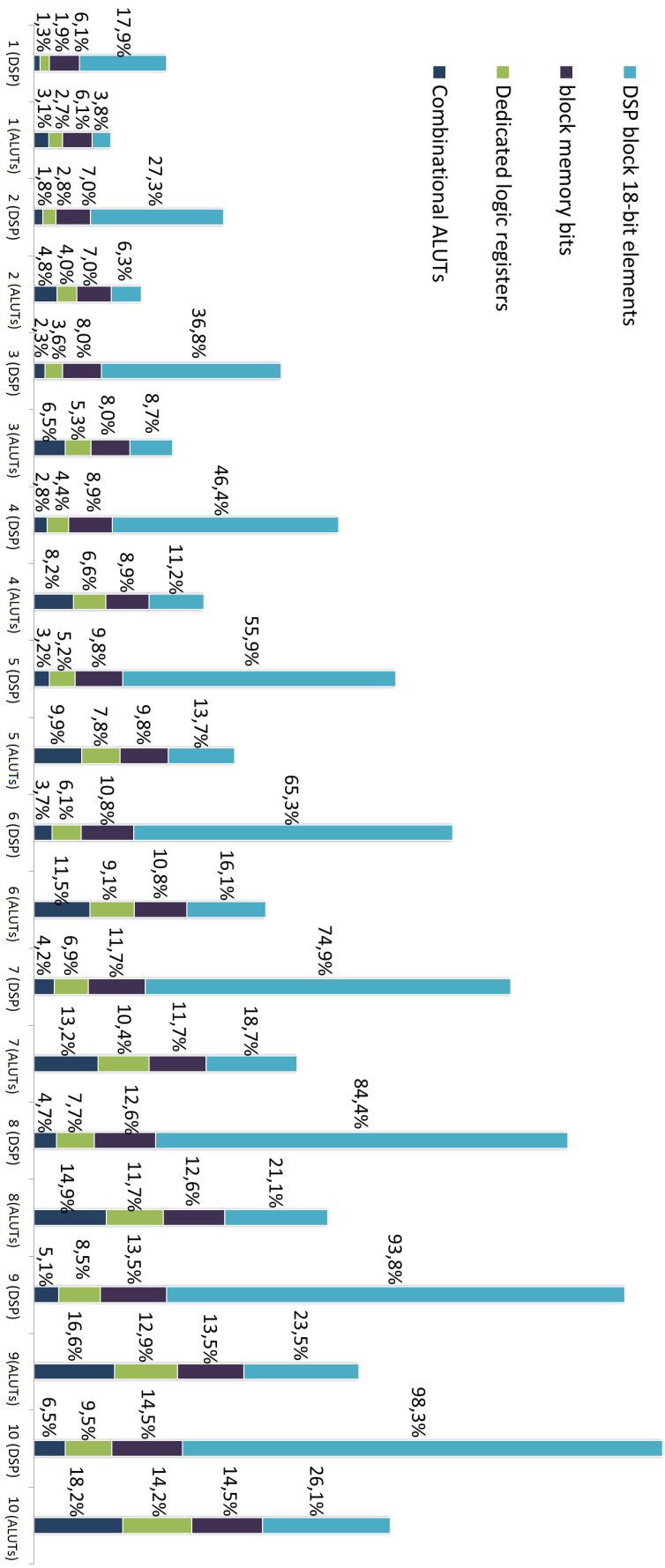


5.4.3 TIPO DE MULTIPLICADOR VIA PARÂMETRO DE SÍNTESE

Durante o desenvolvimento do projeto, foi percebido que a quantidade de blocos DSP disponíveis no *FPGA* poderia ser um fator limitante, tanto para aumentar a quantidade de *templates*, quanto para portar o design para outras plataformas com menos blocos DSP disponíveis. Ao mesmo tempo, dependendo do tipo de FPGA e da quantidade de templates escolhida pelo usuário, pode ser vantajoso o uso de blocos DSP na construção dos multiplicadores. Assim, o tipo dos multiplicadores das células de correlação foram modificados para ter duas opções de síntese: o usando blocos DSP, cada multiplicador usa um bloco DSP 18bits ou usando blocos lógicos (ALUTs), cada multiplicador usa 80 blocos lógicos, no caso dos multiplicadores com duas entradas iguais (quadrado) esse uso cai para 40 blocos lógicos.

A Figure 5.3 mostra o uso de recursos *FPGA* para cada opção de síntese, multiplicadores construídos usando DSP ou ALUT, e para quantidades de *templates* que variam de 1 a 10. Note que o recurso DSP chega ao limite quando o design que usa esse recurso nos multiplicadores é sintetizado para 10 *templates*.

Figura 5.5: Uso de recursos do FPGA por tipo de multiplicador usado(DSP ou ALUTs) e por quantidade de templates



6

EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados os experimentos feitos com a arquitetura em FPGA para o cálculo da ZNCC multitemplate. Foram feitos testes para atestar a qualidade dos resultados, comparando os valores fornecidos pelo FPGA com aqueles obtidos pelo software de referência. Em seguida, são mostrados os testes de desempenho, comparando o tempo de execução usando a plataforma baseada em FPGA com os tempos do mesmo algoritmo sendo executado apenas na CPU ou na CPU com GPU.

Nessa etapa de desenvolvimento do projeto, foram realizados tanto testes para avaliar o desempenho quanto para validar a precisão dos resultados de *ZNCC* obtidos com a arquitetura proposta, apresentada no Capítulo 4.

Primeiro foram realizados os testes de corretude. Eles foram feitos para garantir que os resultados obtidos usando o módulo proposto não se desviam muito dos resultados obtidos por um software largamente testado, para isso foi usada a métrica de Relação Sinal Ruído (*SNR*).

O testes de desempenho foram feitos medindo a taxa *fps* média, obtida com a execução de 1000 *frames*, utilizando a implementação do módulo proposto em FPGA e implementações do cálculo da *ZNCC* em CPU e GPU. As implementações em CPU e GPU são usadas para comparar o desempenho obtido com a arquitetura proposta e implementada em FPGA.

Valores de precisão e sobreposição foram medidos usando o algoritmo de rastreamento, aplicado ao benchmark desenvolvido por WU; LIM; YANG (2013) para tentar encontrar a quantidade ótima de templates.

Os testes feitos nesta seção consideram o estudo de caso de rastreamento de pedestres para configurar o conjunto de parâmetros do hardware:

- Número de colunas da imagem: R=432.
- Número de linhas da imagem: S=432.
- Número de colunas do template: O=72.
- Número de linhas do template: P=144.

A quantidade de *templates* foi variada de 1 a 10 para todos os testes.

6.1 Teste de corretude

Testes funcionais foram feitos com o módulo proposto para avaliar a precisão numérica dos resultados. Os resultados de ZNCC calculados no FPGA foram comparados aos resultados produzidos pelo software de referência utilizando as mesmas entradas de imagens e *templates*.

A avaliação da precisão dos resultados foi feita através da relação sinal ruído (*SNR*) mostrada na Equação 6.1.

$$SNR = 10log_{10}\frac{P_{sinal}}{P_{\text{ruído}}} \quad (dB) \tag{6.1}$$

Na equação 6.1, P_{sinal} é a potência do resultado de referência produzido pelo software e $P_{\text{ruído}}$ é a potência da diferença entre os resultados gerados por um módulo de hardware a sua referência em software.

O termo potência é usado aqui para indicar que o valor da amplitude é elevado ao quadrado, como é feito para calcular potência de sinais elétricos.

A Tabela 6.1 mostra os resultados relativos a precisão numérica dos resultados quando o cálculo de *ZNCC* é executado sobre um conjunto de vídeos do *benchmark* (WU; LIM; YANG, 2013).

Para cada Vídeo analisado destacamos a relação sinal ruído mínima e máxima e as medidas estatísticas de média e desvio padrão.

Esses resultados mostram que, no pior caso analisado, a taxa SNR é maior que 151dB o que, de acordo com a Equação 6.1, significa que, mesmo no pior caso, a amplitude do sinal ainda é muito maior que a amplitude do ruído.

Tabela 6.1: Precisão numérica dos resultados por vídeo

Vídeo	SNR médio	Desvio padrão SNR	SNR mínimo	SNR máximo
Jogging	263,7573	13,5191	185,9035	318,6646
Basketball	284,8956	14,3957	200,0000	319,0908
BlurBody	247,5114	12,9246	151,3806	311,7282
BlurFace	256,2436	10,7929	201,2090	311,8979
Bolt2	270,3936	16,3726	151,4707	312,0357
Boy	268,6749	14,2540	168,9043	311,8479
CarDark	278,0822	13,5107	192,9669	312,0522
Crowds	287,8422	18,2189	184,9447	312,0896
Deer	265,9956	14,9130	176,8533	311,9644
Gym	286,4469	14,8780	202,0466	312,0744
Human2	272,8098	13,3531	210,1415	311,7508
Human8	279,7999	9,3439	205,8981	312,0559
Human9	283,7874	14,2997	189,0932	312,0777
Singer1	281,2374	14,9284	206,9845	311,9358
Skater2	276,1292	15,0146	196,7276	312,0321
Subway	279,6314	14,0191	193,4933	312,0756
Woman	281,7562	11,2130	196,9517	312,0582

6.2 Avaliação de desempenho

A avaliação de desempenho foi feita usando a média da taxa *frames* por segundo (fps) para a execução de uma aplicação com uma sequencia de 1000 *frames*. Foi feita uma implementação puramente em software executando em um processador (CPU) e uma implementação acelerada por uma GPU para comparar com o desempenho do módulo proposto, prototipado em FPGA.

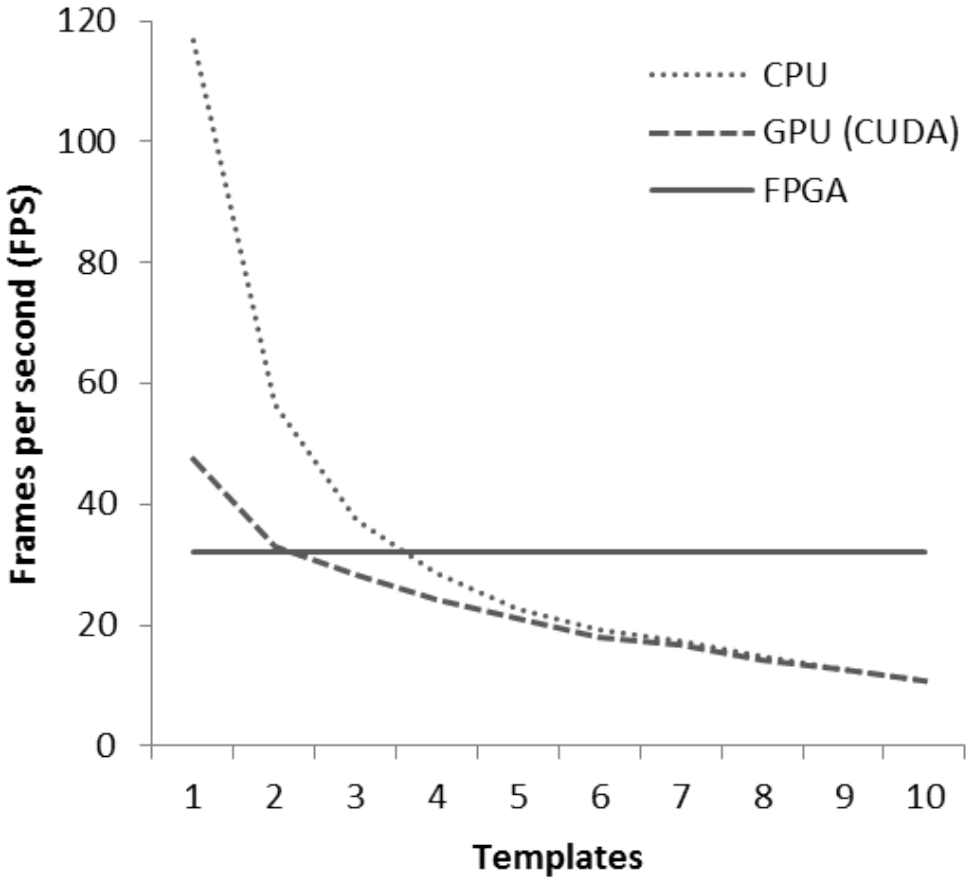
Os testes com CPU foram feitos usando OpenCV executando em um processador core i7, com frequência 3.3GHz e 16GB de memória RAM.

A implementação acelerada pela GPU foi feita usando CUDA, executando em uma GTX760 com frequência de 1GHz e 2GB de memória de vídeo dedicada GDDR5.

O FPGA utilizado foi uma stratix IV, executando a 200MHz de frequência.

O gráfico da Figura 6.1 mostra medidas de desempenho dos três sistemas em FPS fazendo a quantidade de *templates* variar de 1 a 10.

Figura 6.1: Comparação de performance em FPS variando a quantidade de templates de 1 a 10 para as implementações CPU, GPU e FPGA.



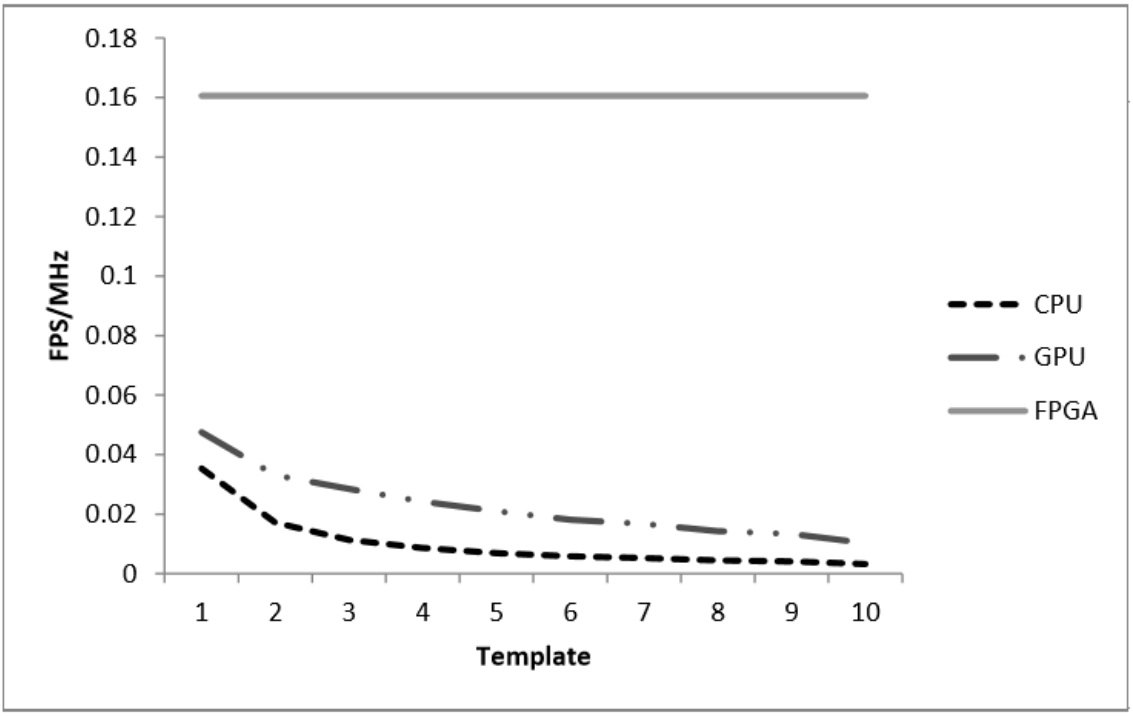
O gráfico mostra que na escala de 1 a 10 *templates* a implementacão em FPGA mantém uma taxa de processamento constante de 32,12 FPS. Os *templates* são processados em paralelo na arquitetura, por isso não há acréscimo no tempo de processamento. Além disso, o tempo de comunicação também não muda porque e a largura de banda do barramento PCIe não está sendo completamente utilizada.

Pode ser observado que para uma quantidade pequena de *templates*, a CPU tenha uma performance superior a do FPGA, considerando que a esta CPU opera a uma frequência 16,5 vezes maior que a frequência de operação do FPGA. A partir de quatro *templates* a performance do FPGA é superior a da CPU chegando a um speedup de 3x quando se usa 10 templates.

O gráfico da Figura 6.2 mostra um outro tipo de medida de desempenho de sistemas, essa medida é obtida quando se considera a frequência de operação de cada sistema, fazendo a divisão entre a taxa de processamento em fps pela frequência de operação em MHz (fps/MHZ). Essa métrica evidencia que arquiteturas com mais níveis de paralelismo podem impactar tanto no desempenho do sistema, quanto na possibilidade de reduzir a frequência de operação deste. A

redução da frequência de operação de um sistema é desejável já que ela tem impacto na potência dinâmica dissipada (KLUMPERINK et al., 2000).

Figura 6.2: Comparação de performances em FPS por MHz variando a quantidade de *templates* de 1 a 10 para as implementações CPU, GPU e FPGA.



6.3 Estudo de caso

A validação do módulo ZNCC proposto no estudo de caso de rastreamento tem como objetivo validar os resultados de um sistema que utiliza o módulo proposto, em uma aplicação real. O estudo de caso deste trabalho foi o algoritmo de rastreamento de pedestres apresentado na Seção 2.4.

A quantidade de *templates* usada no algoritmo de rastreamento de pedestres deve ser escolhida para maximizar os acertos e minimizar os erros do rastreamento. Assim, para demonstrar a viabilidade de um sistema de rastreamento utilizando o módulo proposto para o cálculo da ZNCC *multitemplate*, o vídeo *Jogging* do *benchmark* (WU; LIM; YANG, 2013) foi processado pela aplicação em 10 versões de configuração, cada configuração com uma quantidade de *templates* (1 a 10).

As métricas de qualidade do rastreamento usadas foram precisão e sobreposição, sugeridas no artigo WU; LIM; YANG (2013). Elas foram medidas usando o *ground truth* marcado a mão disponibilizado na página do *benchmark* (WU; LIM; YANG, 2017).

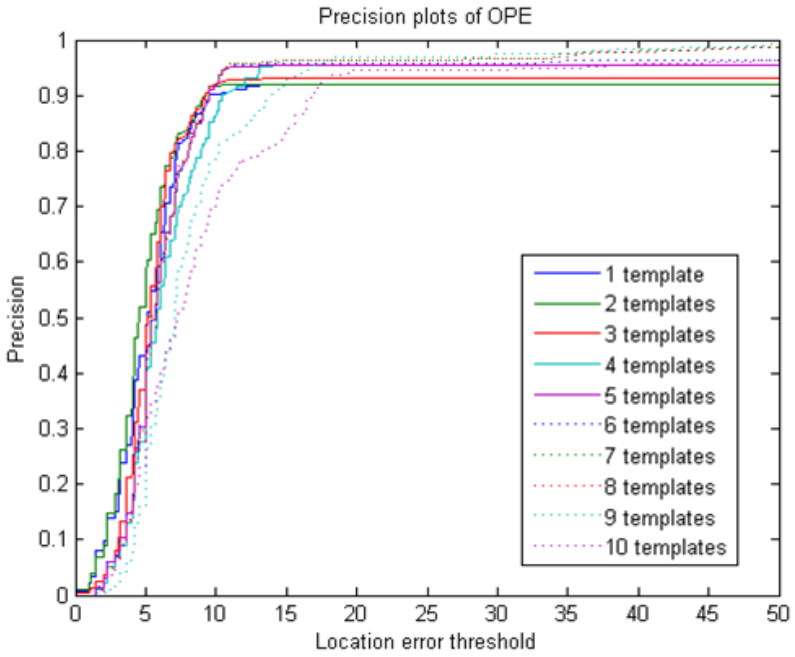
6.3.1 PRECISÃO

A precisão é uma taxa de acertos calculada com base na distância euclidiana entre a posição do centro de massa do objeto na imagem referência, marcada manualmente, e a posição do centro de massa do objeto marcado pelo algoritmo de rastreamento usando o módulo proposto.

A determinação se houve um acerto ou um erro em determinado *frame* é feita com base em um valor limiar. Se a distância calculada for menor que esse limiar, é computado um acerto, se não, computa-se um erro.

O limiar de distância para contagem de acertos do algoritmo é variado entre 0 e 50 pixels, para cada limiar tem-se um percentual de acerto do algoritmo, esse é o valor da precisão. O gráfico da Figura 6.3 mostra 10 curvas de medidas de precisão do rastreador usando o módulo proposto. Cada curva corresponde a uma quantidade de *templates* (1 a 10).

Figura 6.3: Medida de precisão do algoritmo de rastreamento multitemplate.



Estudos da literatura (WU; LIM; YANG, 2013) recomendam considerar o limiar de 20 pixels para ordenar a qualidade dos rastreadores usando a métrica de precisão. Para esse limiar a quantidade de *templates* que maximiza a precisão do algoritmo de rastreamento nesse vídeo é 9 *templates* com taxa de precisão acima de 95%.

6.3.2 SOBREPOSIÇÃO

A métrica de sobreposição considera a taxa de sobreposição entre o retângulo que contém o objeto, marcado manualmente (*ground truth*) e o que foi marcado pelo algoritmo de rastreamento implementado com o módulo proposto. Seja o retângulo de referencia r_t e o retângulo resposta do algoritmo r_a, a sobreposição (S) pode ser calculada usando a Equação 6.2

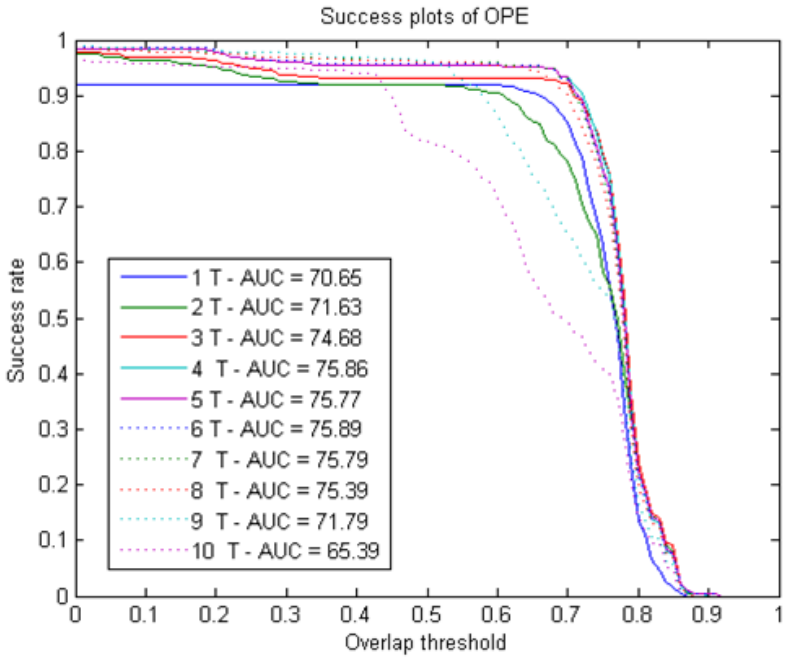
$$S = \frac{r_t \cap r_a}{r_t \cup r_a} \tag{6.2}$$

O resultado da Equação 6.2 é um valor correspondente ao percentual de sobreposição entre os retângulos, sendo 0, quando o retângulo dado pelo algoritmo não tiver nenhum ponto em comum com o retângulo marcado no *ground truth* e 1, quando eles tiverem todos os pixels sobrepostos.

A partir da sobreposição a taxa de sucesso pode ser calculada, comparando a sobreposição obtida a um limiar de sobreposição (t_0), esse limiar é variado entre 0 e 1 para construir o gráfico da figura 6.4.

No gráfico de taxa de sucesso baseada em sobreposição o a literatura (WU; LIM; YANG, 2013) recomenda o uso da área abaixo da curva (AUC) para avaliar os algoritmos. Usando essa métrica, a melhor curva é a que corresponde ao uso de 6 *templates*, com uma AUC que cobre 75,89% da área total do gráfico.

Figura 6.4: Medida de sucesso de sobreposição do algoritmo de rastreamento usando de 1 a 10 templates.



Os resultados de precisão e sobreposição medidos nesta seção apontam a boa qualidade do rastreamento executado pelo sistema, quando comparado aos resultados disponíveis em WU; LIM; YANG (2017), para melhor avaliar o algoritmo de rastreamento seriam necessários mais testes com outras métricas e outros vídeos, como o foco do trabalho é o desenvolvimento do módulo ZNCC *multitemplate*, nos limitamos a esse estudo de caso.

6.4 Conclusões

Neste capítulo foram apresentados Experimentos e resultados que comprovam a uma boa relação sinal ruído entre os resultados obtidos com o módulo proposto e os resultados de referência de ZNCC.

Também foi avaliada a eficiência temporal da arquitetura desenvolvida neste trabalho, atestando a adequação do desempenho do módulo proposto a aplicações em tempo real (30FPS).

O estudo de caso desse projeto foi utilizar o módulo proposto em uma aplicação de rastreamento. Os resultados de precisão e sobreposição mostrados nesse capítulo atestam a viabilidade do uso do módulo proposto nesse tipo de aplicação, demonstrando que a relação sinal ruído medida na seção 6.1 resulta em bons resultados de precisão e sobreposição.

7

CONCLUSÕES E TRABALHOS FUTUROS

Neste capítulo são feitas as considerações finais do trabalho, as conclusões que podem ser obtidas do que foi exposto nessa dissertação, as principais contribuições do trabalho e algumas ideias de trabalhos futuros.

Esta dissertação apresentou o desenvolvimento de uma arquitetura configurável que explora conceitos de paralelismo e pipeline para resolver, de forma eficiente, o casamento de padrões com múltiplos padrões e por análise de similaridade dada pela métrica ZNCC.

Como mostrado no primeiro capítulo deste trabalho, algoritmos para solucionar o problema de *template matching* são usados em uma vasta gama de aplicações que vão desde buscas em imagens estáticas a rastreamento em vídeo. Muitas vezes essas aplicações são executadas em tempo real.

Template matching é o problema de encontrar um objeto em uma cena. Quando se trata da busca por múltiplos objetos independentes ou de uma coleção de representações do mesmo objeto, faz-se necessária a busca por múltiplos *templates*, o que aumenta o custo computacional da busca por análise de similaridade.

No segundo capítulo deste trabalho, foram analisadas algumas das métricas de similaridade ou dissimilaridade mais utilizadas na abordagem de análise de similaridade por janela deslizante para solucionar o casamento de padrões. A escolha da métrica ZNCC foi justificada por entregar resultados normalizados e com média zero e invariantes a mudanças lineares de iluminação e contraste.

Para contornar o alto custo computacional do cálculo da ZNCC *multitemplate* foi proposta uma implementação em hardware.

A arquitetura proposta nesse projeto executa a fórmula da equação 4.7, que é equivalente a fórmula original da métrica ZNCC apresentada na equação 2.6, sendo mais adequada a uma lógica de cálculo em hardware. Essa arquitetura explora conceitos de paralelismo e pipeline para realizar o cálculo da métrica ZNCC entre um *frame* e múltiplos *templates* de forma eficiente.

O módulo desenvolvido foi implementado em FPGA (stratix IV) e permite a execução do cálculo da ZNCC *multitemplate* em tempo real com 32,12 FPS, utilizando uma frequência de operação de 200MHz.

Existiam algumas soluções semelhantes na literatura, como mostra o o capítulo 3, no entanto esse trabalho preencheu a lacuna de uma arquitetura que processa múltiplos *templates* como a que foi proposta por *SANG; LIAO; YUAN (2011)*, com o diferencial de manter a precisão dos dados de entrada em 8bits independente da quantidade de *templates* utilizada, e utilizando a métrica ZNCC que é robusta a variações de iluminação e contraste.

O trabalho de *CHEN et al. (2012)* realizou modificações na fórmula original da ZNCC para deixá-la mais adequada a uma implementação em hardware, essas modificações tiraram a característica dos resultados de serem normalizados, com média zero e invariantes a variações de iluminação e contraste. Este trabalho também manipulou a fórmula original para deixá-la mais adequada a uma implementação em hardware, mas sabendo que algumas características dos resultados da ZNCC importantes para as aplicações que os utilizam, tomou-se o cuidado de manter a a fórmula implementada equivalente à fórmula original.

Para suportar múltiplos *templates* a arquitetura proposta explora técnicas de paralelismo no processamento de cada linha mas mantém parte do processamento das janelas de imagem

em sequência. Desta forma a implementação *multitemplate* foi possível, com desempenho de tempo real, sem reduzir as dimensões dos *templates*. Ao contrário da implementação proposta por*HASHIMOTO; ITO; NAKANO (2013)*, que limita as dimensões suportadas a valores muito pequenos.

A maior contribuição deste trabalho foi propor uma arquitetura em hardware capaz de calcular resultados ZNCC entre uma imagem e múltiplos *templates* mantendo um desempenho compatível com aplicações de tempo real (30fps). As técnicas de pipeline e paralelismo usadas nessa arquitetura podem ser vistas no capítulo 4 deste trabalho. Na sequência, o capítulo 5 mostra os detalhes da implementação dessa arquitetura em FPGA. Finalmente o capítulo 6 expõe alguns resultados obtidos a partir dessa implementação.

A precisão numérica dos resultados gerados pelo módulo foi medida em relação aos resultados de referência gerados por software utilizando vídeos do *benchmark (WU; LIM; YANG, 2013)* e foram obtidas as relações sinal ruído mostradas na tabela 6.1, essa tabela mostra que o valor SNR mínimo dentre todos os resultados dos vídeos analisados foi 151,38dB, significando que mesmo no pior caso o valor do sinal ainda é muito maior de o valor do ruído.

O gráfico da Figura 6.1 mostra que o desempenho da arquitetura proposta para as quantidades de *templates* de 1 a 10 se mantém em 32,12FPS, viabilizando sua execução em em aplicações com vídeos de 30FPS.

O módulo proposto também foi validado no contexto de uma aplicação de rastreamento de pedestres. Neste caso, o vídeo *Jogging* do conjunto disponibilizado por *WU; LIM; YANG (2013)* foi utilizado. As medidas de precisão e sobreposição foram analisadas para o algoritmo de rastreamento *multitemplate* apresentado na Seção 2.4 deste trabalho, com a quantidade de *templates* variando entre 1 e 10. Nesse estudo de caso foram obtidas as medidas de precisão a 20 pixels de 96% utilizando 9 *templates*; e de AUC no gráfico de sobreposição maior que 75% utilizando 6 *templates*. Um resultado satisfatório de acordo com *WU; LIM; YANG (2013)*.

7.1 Trabalhos Futuros

Trabalhos futuros podem explorar a integração do módulo proposto com um sistema HTTP com infraestrutura Ethernet para controlar a posição da câmera enquanto é feito o rastreamento.

A arquitetura proposta também pode ser integrada com outras arquiteturas de processamento de vídeo no mesmo FPGA, como a arquitetura de segmentação de vídeos proposta por *(BARBOSA et al., 2015)*. Uma plataforma como essa poderia executar algoritmos de rastreamento baseados em movimento como o proposto por *(NAYAK; PUJARI, 2015)*, de forma mais eficiente e precisa.

Aplicações com *Deep learning (LECUN; BENGIO; HINTON, 2015) (SCHMIDHUBER, 2015)*, também podem se beneficiar de execuções paralelas de múltiplas correlações, em cada camada. Um trabalho futuro interessante seria reutilizar os módulos da arquitetura que calculam

as correlações para acelerar algoritmos de *deep learning*.

REFERÊNCIAS

ALBUQUERQUE, E. S. et al. An FPGA-based accelerator for multiple real-time template matching. In: INTEGRATED CIRCUITS AND SYSTEMS DESIGN (SBCCI), 2016 29TH SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2016. p.1–6.

ALTERA. **Stratix IV Device Handbook Altera**. Accessed: 2015-12-21, `https://www.altera.com/content/dam/altera-www/global/en_US/pdfs/literature/hb/stratix-iv/stratix4_handbook.pd`.

ALTERA. **Industry Solutions**. Disponivel em: <https://www.altera.com/solutions/industry.html>. Acesso em: 09 de julho de 2016.

AWAD, M. FPGA supercomputing platforms: a survey. In: INTERNATIONAL CONFERENCE ON FIELD PROGRAMMABLE LOGIC AND APPLICATIONS, 2009. **Anais...** [S.l.: s.n.], 2009. p.564–568.

BARBOSA, J. P. et al. A high performance hardware accelerator for dynamic texture segmentation. **Journal of Systems Architecture**, [S.l.], v.61, n.10, p.639–645, 2015.

BENNAMOUN, M.; MAMIC, G. J. **Object recognition**: fundamentals and case studies. [S.l.]: Springer Science & Business Media, 2012.

BRENT, R. P.; KUNG, H. Systolic VLSI arrays for polynomial GCD computation. **IEEE Transactions on Computers**, [S.l.], v.33, n.8, p.731–736, 1984.

BRISTEAU, P.-J. et al. The navigation and control technology inside the ar. drone micro uav. **IFAC Proceedings Volumes**, [S.l.], v.44, n.1, p.1477–1484, 2011.

BRUNELLI, R.; POGGIO, T. Face recognition: features versus templates. **IEEE Transactions on Pattern Analysis & Machine Intelligence**, [S.l.], n.10, p.1042–1052, 1993.

CHEN, C.-h. **Computer vision in medical imaging**. [S.l.]: World scientific, 2014. v.2.

CHEN, J.-Y. et al. Real-time FPGA-based template matching module for visual inspection application. In: IEEE/ASME INTERNATIONAL CONFERENCE ON ADVANCED INTELLIGENT MECHATRONICS (AIM), 2012. **Anais...** [S.l.: s.n.], 2012. p.1072–1076.

CORMEN, T. H. **Introduction to algorithms**. [S.l.]: MIT press, 2009.

CROW, F. C. Summed-area tables for texture mapping. **ACM SIGGRAPH computer graphics**, [S.l.], v.18, n.3, p.207–212, 1984.

CUI, Y. et al. Multiple template-based fluoroscopic tracking of lung tumor mass without implanted fiducial markers. **Physics in medicine and biology**, [S.l.], v.52, n.20, p.6229, 2007.

DI STEFANO, L.; MATTOCCIA, S.; TOMBARI, F. ZNCC-based template matching using bounded partial correlation. **Pattern recognition letters**, [S.l.], v.26, n.14, p.2129–2134, 2005.

DIMOND, R.; RACANIERE, S.; PELL, O. Accelerating large-scale HPC Applications using FPGAs. In: COMPUTER ARITHMETIC (ARITH), 2011 20TH IEEE SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2011. p.191–192.

GHARAVI-ALKHANSARI, M. A fast globally optimal algorithm for template matching using low-resolution pruning. **IEEE Transactions on Image Processing**, [S.l.], v.10, n.4, p.526–533, 2001.

GIDEL. **PROCmegaFIFO**. Disponivel em: <http://www.maxxvision.com/fileadmin/content/Produkte/FPGA_Plattformen/Downloads/Gidel_FPGA_Plattformen_PROCMegaFIFOIP.pdf>. Acesso em: 29 de junho de 2016.

GIDEL. **ProcWizard**. Disponivel em: <http://http://www.gidel.com/procwizard.htm>. Acesso em: 29 de junho de 2016.

GROUT, I. **Digital systems design with FPGAs and CPLDs**. [S.l.]: Newnes, 2011.

GUPTA, N.; GUPTA, N. A VLSI architecture for image registration in real time. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, [S.l.], v.15, n.9, p.981–989, 2007.

HASHIMOTO, K.; ITO, Y.; NAKANO, K. Template Matching using DSP slices on the FPGA. In: COMPUTING AND NETWORKING (CANDAR), 2013 FIRST INTERNATIONAL SYMPOSIUM ON. **Anais...** [S.l.: s.n.], 2013. p.338–344.

HUMENBERGER, M. et al. A fast stereo matching algorithm suitable for embedded real-time systems. **Computer Vision and Image Understanding**, [S.l.], v.114, n.11, p.1180–1202, 2010.

JONES, G. A.; PARAGIOS, N.; REGAZZONI, C. S. **Video-based surveillance systems**: computer vision and distributed processing. [S.l.]: Springer Science & Business Media, 2012.

JUNG, J.-H. et al. A novel template matching scheme for fast full-search boosted by an integral image. **IEEE Signal Processing Letters**, [S.l.], v.17, n.1, p.107–110, 2010.

KLUMPERINK, E. A. et al. Reducing MOSFET 1/f noise and power consumption by switched biasing. **IEEE Journal of Solid-State Circuits**, [S.l.], v.35, n.7, p.994–1001, 2000.

KUON, I.; TESSIER, R.; ROSE, J. FPGA architecture: survey and challenges. **Foundations and Trends in Electronic Design Automation**, [S.l.], v.2, n.2, p.135–253, 2008.

KURUPPU, G. et al. Comparison of different template matching algorithms in high speed sports motion tracking. In: INDUSTRIAL AND INFORMATION SYSTEMS (ICIIS), 2013 8TH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.445–448.

LECUN, Y.; BENGIO, Y.; HINTON, G. Deep learning. **Nature**, [S.l.], v.521, n.7553, p.436–444, 2015.

LEWIS, J. Fast normalized cross-correlation. In: VISION INTERFACE. **Anais...** [S.l.: s.n.], 1995. v.10, n.1, p.120–123.

LINDOSO, A.; ENTRENA, L. High performance FPGA-based image correlation. **J. Real-Time Image Processing**, [S.l.], v.2, n.4, p.223–233, 2007.

MAHMOOD, A.; KHAN, S. Exploiting transitivity of correlation for fast template matching. **IEEE Transactions on Image Processing**, [S.l.], v.19, n.8, p.2190–2200, 2010.

MART, D. A computational investigation into the human representation and processing of visual information. **Free-man, San Francisco, CA**, [S.l.], 1982.

MISHRA, P. et al. Robust template matching based obstacle tracking for autonomous rovers. In: ELECTRONICS, COMPUTING AND COMMUNICATION TECHNOLOGIES (CONECCT), 2013 IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2013. p.1–5.

NARASIMHAN, S. G.; NAYAR, S. K. Contrast restoration of weather degraded images. **IEEE transactions on pattern analysis and machine intelligence**, [S.l.], v.25, n.6, p.713–724, 2003.

NAYAK, S.; PUJARI, S. S. Moving Object Tracking Application: fpga and model based implementation using image processing algorithms. In: COMPUTING COMMUNICATION CONTROL AND AUTOMATION (ICCUBEA), 2015 INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2015. p.932–936.

QAYYUMA, A. et al. Vegetation height estimation near power transmission poles via satellite stereo images using 3d depth estimation algorithms. **International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences**, [S.l.], 2015.

SANG, H.; LIAO, D.; YUAN, Y. VLSI implementation of multiple large template-based image matching for automatic target recognition. In: SEVENTH INTERNATIONAL SYMPOSIUM ON MULTISPECTRAL IMAGE PROCESSING AND PATTERN RECOGNITION (MIPPR2011). **Anais...** [S.l.: s.n.], 2011. p.80050A–80050A.

SCHMIDHUBER, J. Deep learning in neural networks: an overview. **Neural Networks**, [S.l.], v.61, p.85–117, 2015.

TATE, R.; NORTHERN III, J. Fast template matching system using VHDL. In: REGION 5 CONFERENCE, 2008 IEEE. **Anais...** [S.l.: s.n.], 2008. p.1–5.

TSAI, D.-M.; LIN, C.-T. Fast normalized cross correlation for defect detection. **Pattern Recognition Letters**, [S.l.], v.24, n.15, p.2625–2631, 2003.

UCHIDA, A.; ITO, Y.; NAKANO, K. Fast and accurate template matching using pixel rearrangement on the GPU. In: NETWORKING AND COMPUTING (ICNC), 2011 SECOND INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2011. p.153–159.

WU, Y.; LIM, J.; YANG, M.-H. Online Object Tracking: a benchmark. In: IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR). **Anais...** [S.l.: s.n.], 2013.

WU, Y.; LIM, J.; YANG, M.-H. **Visual Tracker Benchmark**. Disponivel em: <http://www.visual-tracking.net>. Acesso em: 02 de junho de 2017.

XILINX. **Aplications**. Disponivel em: <http://www.xilinx.com/applications.html>. Acesso em: 09 de julho de 2016.

YANG, C.; DURAISWAMI, R.; DAVIS, L. Fast multiple object tracking via a hierarchical particle filter. In: COMPUTER VISION, 2005. ICCV 2005. TENTH IEEE INTERNATIONAL CONFERENCE ON. **Anais...** [S.l.: s.n.], 2005. v.1, p.212–219.

YILMAZ, A.; JAVED, O.; SHAH, M. Object tracking: a survey. **Acm computing surveys (CSUR)**, [S.l.], v.38, n.4, p.13, 2006.