DISSERTAÇÃO DE MESTRADO

IMPLEMANTAÇÃO DO ALGORITMO DE SUBTRAÇÃO DE FUNDO PARA DETECÇÃO DE OBJETOS EM MOVIMENTO, USANDO SISTEMAS RECONFIGURÁVEIS

CAMILO SÁNCHEZ FERREIRA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA
DEPARTAMENTO DE ENGENHARIA MECÂNICA

IMPLEMANTAÇÃO DO ALGORITMO DE SUBTRAÇÃO DE FUNDO PARA DETECÇÃO DE OBJETOS EM MOVIMENTO, USANDO SISTEMAS RECONFIGURÁVEIS

CAMILO SÁNCHEZ FERREIRA

Orientador: Prof. Dr. Carlos H. Llanos Quintero

DISSERTAÇÃO DE MESTRADO

Publicação: ENM.DM-48A/12

Brasília, 9 de Março de 2012

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

DISSERTAÇÃO DE MESTRADO

IMPLEMANTAÇÃO DO ALGORITMO DE SUBTRAÇÃO DE FUNDO PARA DETECÇÃO DE OBJETOS EM MOVIMENTO, USANDO SISTEMAS RECONFIGURÁVEIS

CAMILO SÁNCHEZ FERREIRA

Dissertação de Mestrado submetida ao Departamento de Engenharia Mecânica da faculdade de Tecnologia da Universidade de Brasília como requisito parcial para a obtenção do grau de Mestre em Sistemas Mecatrônicos

Banca Examinadora

Prof. Dr. Carlos H. Llanos Quintero, _____
ENM/UnB
Orientador

Prof. Dr. Ricardo Pezzuol Jacobi, _____
CIC/UnB
Examinador interno

Prof. Dr. Altamiro Amadeu Susin, _____
Dep. Engenharia Elétrica/UFRGS
Examinador externo

Brasília, 9 de Março de 2012

FICHA CATALOGRÁFICA

SÁNCHEZ FERREIRA., CAMILO
IMPLEMANTAÇÃO DO ALGORITMO DE SUBTRAÇÃO DE FUNDO PARA DETECÇÃO DE OBJETOS EM MOVIMENTO, USANDO SISTEMAS RECONFIGURÁVEIS [Distrito Federal] 2012.
xiv, 94p. 210 × 297 mm (ENM/FT/UnB, Mestre, Sistemas Mecatrônicos, 2012). Dissertação de Mestrado – Universidade de Brasília. Faculdade de Tecnologia.
Departamento de Engenharia Mecânica.

1. Detecção de Movimento
2. FPGAs
3. Processamento de Imagens
4. Sistemas Embarcados
I. ENM/FT/UnB
II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

SÁNCHEZ-FERREIRA, CAMILO. (2012). Implementação do Algoritmos de Subtração de Fundo para Detecção de Objetos em Movimento, Usando Sistemas Reconfiguráveis. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-48A/12, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 94p.

CESSÃO DE DIREITOS

AUTOR: Camilo Sánchez Ferreira.

TÍTULO: Implementação do Algoritmos de Subtração de Fundo para Detecção de Objetos em Movimento, Usando Sistemas Reconfiguráveis.

GRAU: Mestre **ANO**: 2012

Camilo Sánchez Ferreira

Dedicatória

Este trabalho está dedicado à minha família: meus pais María Gabriela e Alirio, meu irmão Santiago e meu sobrinho Sebastián pelo apoio incondicional na minha vida e nos meus estudos.

CAMILO SÁNCHEZ FERREIRA

Agradecimentos

Agradeço primeiramente a minha família, já que eles foram meu leme e minha inspiração para continuar sempre lutando e alcançar as minhas metas. Porque, sentindo muita saudade deles, sei que estão comigo de coração, me acompanhando em cada passo, em cada sorriso e em cada problema que possa aparecer. Porque devo para eles tudo o que eu sou hoje, tudo o que tenho alcançado e tudo o que alcançarei na vida.

Aos meus amigos: Rosita, Diana, Liliana, Guillermo, Diego, Jesús e Juan Pablo, que foram um apoio constante desde o começo até o fim, e que, em pouco tempo passaram a formar parte importante da minha vida, se convertendo na minha segunda família.

Aos meus amigos do laboratório: Jones, Daniel, Sergio, Janier e André, por ser minha mão direita e pela amizade, deles aprendi muito e espero continuar aprendendo.

Ao meu orientador, o Professor Carlos Humberto Llanos pela amizade, apoio, confiança, dedicação e paciência. Pelo seu tempo nas longas reuniões e por ter me permitido trabalhar do seu lado, aprendendo mais um pouco dele cada dia.

Aos meus amigos compatriotas e brasileiros, que são tantos que estou correndo o risco de esquecer alguns, e que tem me dado um grande apoio e amizade, aqui em Brasília e desde a distancia: Fabia, Lily, Claudia Patricia, Tatiana, Vivi, Lorena, Diana Jimena, Lizeth, Alexander, Milton, Rodrigo, Margarita, Andrés, Miguel Ordoñez, Ronald, Anita, Alvaro, Alejandro Perez, Lucy, Miguel, Esteban, Nathalie, Isabella, Istjak, Carlos, Alvaro Tobón, José Luis, Pablo Lopez, Leo, Paola, Vivian, Vanessa, Alexander, Diego Alejandro, Sulge, Sara, Johanna, Professora Sonia, Professora Claudia, Maria Andrea, Pili, Harry, Ana Rosa, Claudia Ochoa, Jorge. A galera da capoeira, Artur, Regina, Antônio, Anita, Piolho, Sirio, Felipe, Lola, Gabi, Renatha, entre muitos outros.

Ao CNPQ (Conselho Nacional de Desenvolvimento Científico e Tecnológico) pelo apoio financeiro deste trabalho.

Ao Grupo de automação e Controle (GRACO) e todos os meus professores pelo suporte e formação acadêmica.

CAMILO SÁNCHEZ FERREIRA

RESUMO

Atualmente, o mercado e a comunidade acadêmica têm requerido aplicações baseadas no processamento de imagens e vídeo com varias restrições de tempo real. Por outro lado, a detecção de objetos em movimento é uma etapa muito importante em aplicações de robótica móvel e segurança. Com o fim de encontrar um desenho alternativo que permita o rápido desenvolvimento de sistemas de detecção de movimento em tempo real, este trabalho propõe uma arquitetura *hardware* para a detecção de objetos em movimento baseada no algoritmo de subtração do fundo, sendo implementado em FPGAs (*Field Programmable Gate Arrays*). Para alcançar isto, foram executados os seguintes passos: (a) a imagem de fundo (em níveis de cinza) é armazenada em uma memoria SRAM externa, (b) é aplicada uma etapa de filtragem passa-baixa nas imagens de fundo e no quadro atual, (c) é realizada a operação de subtração entre as duas imagens, e (d) é aplicado um filtro morfológico sobre a imagem resultante. Posteriormente é calculado o centro de gravidade do objeto para ser enviado para um computador (via interface RS-232 desenvolvida no processador embarcado Nios II da Altera Corp.). Adicionalmente, o sistema foi implementado sobre um robô móvel para a calibração e validação de um sensor de distâncias baseado em um sistema de visão omnidirecional. Tanto os resultados práticos da detecção de movimento como os resultados de síntese têm demostrado a viabilidade dos FPGAs na implementação dos algoritmos propostos sobre uma plataforma de *hardware*. O sistema implementado fornece um *pixel (picture element)* processado por cada ciclo de relógio da FPGA depois de um período de latência, sendo 32 vezes mais rápido do que o mesmo algoritmo implementado em *software* (isto foi testado utilizando o sistema operacional de tempo real *xPC Target* da MathWorks).

ABSTRACT

Currently, both the market and the academic communities have required applications based on image and video processing with several real-time constraints. On the other hand, detection of moving objects is a very important stage in mobile robotics and surveillance applications. In order to achieve an alternative design that allows the rapid development of real time motion detection systems this work proposes a hardware architecture for motion detection based on the background subtraction algorithm, which is implemented on FPGAs (Field Programmable Gate Arrays). For achieving this, the following steps are executed: (a) a background image (in gray-level format) is stored in an external SRAM memory, (b) a low-pass filter is applied to both the stored and current images, (c) a subtraction operation between both images is obtained, and (d) a morphological filter is applied over the resulting image. Afterward, the gravity center of the object is calculated and sent to a PC (via RS-232 interface developed on Nios II embedded processor from Altera Corp.). Additionally, the system was implemented on a mobile robot for calibration and validation of a distance sensor based on a omnidirectional vision system. Both the practical results of the motion detection system and synthesis results have demonstrated the feasibility of FPGAs for implementing the proposed algorithms on a hardware platform. The implemented system provides one processed pixel per FPGA's clock cycle (after the latency time) and speed-ups the software implementation (using the real-time *xPC Target* OS from MathWorks) by a factor of 32.

RESUMEN

Actualmente, el mercado y la comunidad académica han requerido aplicaciones basadas en el procesamiento de imágenes y video con varias restricciones de tiempo real. Por otro lado, la detección de objetos en movimiento es una etapa muy importante en aplicaciones de robótica móvil y vigilancia. Con el fin de encontrar un diseño alternativo que permita el rápido desarrollo de sistemas de detección de movimiento en tiempo real, este trabajo propone una arquitectura *hardware* para la detección de objetos en movimiento basada en el algoritmo de sustracción de fondo, siendo este implementado en FPGAs (*Field Programmable Gate Arrays*). Para alcanzar esto, fueron ejecutados los siguientes pasos: (a) la imagen de fondo (en niveles de gris) es almacenada en una memoria SRAM externa, (b) es aplicada una etapa de filtrado pasa-bajas sobre las imágenes de fondo y el cuadro actual, (c) es realizada la operación de sustracción entre las dos imágenes, y (d) es aplicado un filtro morfológico sobre la imagen resultante. Después, es calculado el centro de gravedad del objeto y es enviado para un computador (vía interfaz RS-232, desarrollada en el procesador embarcado Nios II de Altera Corp.). Adicionalmente, el sistema fue implementado sobre un robot móvil para la calibración y validación de un sensor de distancia basado en un sistema de visión omnidireccional. Tanto los resultados prácticos de la detección de movimiento como los resultados de síntesis han demostrado la viabilidad de las FPGAs en la implementación de los algoritmos propuestos sobre una plataforma *hardware*. El sistema implementado proporciona un *pixel (picture element)* procesado por cada ciclo de reloj de la FPGA (después de un período de latencia) y es 32 veces mas rápido que el mismo algoritmo implementado en *software* (esto fue probado utilizando el sistema operativo de tiempo real *xPC Target* de MathWorks).

Sumário

Lista de Figuras

Lista de Tabelas

LISTA DE SÍMBOLOS

AHDL	Altera Hardware Description Language
ALU	Unidade Aritmética e Lógica
ASIC	Application Specific Integrated Circuits
CAD	Computer Aided Design
CPLD	Complex Programmable Logic Device
DCT	Discrete Cosine Transform
DLP	Data Loss Prevention
DSP	Digital Signal Processor
EDA	Electronic design automation
FFT	Fast Fourier Transform
FIR	Finite impulse response
FPGA	Field Programmable Gate Array
fps	Frames per second
GFLOP	Giga Floating-Point Operations per Second
GOP	Giga-operações por segundo
GPGPU	General Purpose Graphics Processing Unit
GPP	General Purpose Processor
GPU	Graphics Processing Unit
HDL	Hardware Description Language
ILP	Instruction Level Parallelism
LABs	Logic array blocks
LEs	Logic Elements
MIMD	Multiple Instruction Streams, Multiple Data
MIPS	Milhões de instruções por segundo
MISD	Multiple Instruction, Single Data
NIST	National Institute of Standard and Technology
PCA	Principal Component Analysis

PCI	Peripheral Component Interconect
rDPAs	Reconfigurable data-path arrays
RISC	Reduced instruction set computer
RTL	Register Transfer Level
SAD	Sum of absolute differences
SIMD	Single instruction and multiple data
SISD	Single instruction and single data
SoC	Sistem on Chip
TET	Task execution time
VHDL	VHSIC-HDL Very-High Scale of Integration Circuit - Hardware Description Language
VLSI	Very Large Scale of Integration

1 Introdução

1.1 Contextualização

A visão computacional tem uma grande importância, principalmente devido as suas possíveis aplicações em robótica móvel, multimídia, navegação autônoma, entre outras. A maioria dos algoritmos de visão computacional têm sido implementados em processadores convencionais, baseados em um modelo Von Neumann. Frequentemente, algumas arquiteturas baseadas no modelo de Von Neumann têm sido aplicadas à visão computacional, aumentado a rapidez na execução dos algoritmos pela realização de uma mesma operação sobre vários dados, gerando assim resultados independentes. A principal desvantagem destas arquiteturas é o baixo desempenho na sua implementação sobre sistemas embarcados de tempo real, devido ao elevado custo e consumo de potência. Hoje em dia, quase 90% dos algoritmos em *software* são implementados em sistemas embarcados, indicando assim um crescimento da área, e por sua vez, uma tendência para que a utilização dos FPGAs seja dominante em um futuro próximo [1].

Recentemente as GPUs (*Graphic Processor Units*) têm sido usadas para aplicações gráficas tomando vantagem das unidades paralelas de ponto flutuante e aumentado o desempenho dos algoritmos [2],[3]. Embora as aplicações desenvolvidas sobre GPUs conseguem aumentar notavelmente a velocidade dos algoritmos, existem também os seguintes aspectos: (a) as soluções baseadas em GPUs também apresentam gargalos quando todas as fontes de dados são acessadas desde uma memória global ou quando tem-se que direcionar acessos simultâneos desde diferentes segmentos de memoria, (b) as GPUs não são adaptadas para aplicações especificas e, geralmente, são difíceis de configurar para realizar só as operações que o algoritmo requer, e (c) estes circuitos integrados operam em frequências altas, o que leva a um grande consumo de potência. Estes aspectos representam uma grande desvantagem para aplicações em sistemas embarcados [4].

A visão computacional em tempo real é uma tarefa custosa do ponto de vista computacional, visto que mesmo imagens de tamanho pequeno são submetidas a diversas operações para extração de informações tais como: filtragens, transformações de cores, binarização, detecção de bordas, etc. Essas operações requerem uma grande velocidade de processamento, que a maioria dos sistemas comuns não consegue suprir, ocasionando falhas e atrasos nas aplicações [5].

O *hardware* de um sistema computacional prevê a ocorrência de atrasos oriundos da presença de gargalos no fluxo de dados e de instruções. Tais atrasos são agravados quando a quantidade de informações é muito elevada, como nos sistemas de processamento de imagens. As câmeras comuns (filmadoras e fotográficas) adquirem imagens estaticamente ou em vídeos, com uma taxa de 30 fps (*frames per second* - quadros por segundo). Porém, não é difícil encontrarem-se aplicações não-industriais com câmeras que adquirem com taxas de mais de 150 fps. Quando essas imagens podem ser armazenadas para análise e tratamento posteriores, sistemas mais simples equipados com *hardware* adicional para captura e gravação em alta velocidade suprem as necessidades da aplicação. Porém, em sistemas embarcados de tempo real (com as restrições de memória, tamanho, peso e velocidade) a presença dos gargalos pode levar ao colapso [5].

Adicionalmente, a detecção de movimento é um componente essencial para muitas aplicações em visão, como por exemplo, sistemas de segurança, aplicações militares, navegação de robôs móveis, compressão de vídeo, planejamento de trajetórias, entre outras. A maioria destas aplicações demandam um baixo consumo de potência, desenho compacto e leve, assim como uma plataforma de alta velocidade para o processamento de imagens em tempo real [6]. Atualmente, a literatura mostra três formas para detecção de movimento em sequencias de imagens: (a) subtração do fundo, (b) diferenciação temporal e (c) fluxo ótico. O algoritmo mais utilizado é a subtração do fundo, já que este é um algoritmo econômico computacionalmente e presenta um alto desempenho. Este algoritmo consiste em subtrair cada quadro da sequencia de vídeo da imagem de referência ou fundo (previamente armazenada em memória). Isto implica que, para uma imagem de $M \times N$, o sistema tem que realizar $M \times N$ operações, mantendo a taxa de processamento de 25 a 30 fps ou maior.

A execução direta de algoritmos mapeados em *hardware* em FPGA oferece aumentos de velocidade tipicamente entre 10 e 100 vezes, em comparação com o mesmo algoritmo em *software*. Isso tem atraído muitas pesquisas na área de *Processamento Digital de Sinais*. Os FPGAs oferecem ainda outras vantagens em relação aos microprocessadores genéricos e aos DSPs (*Digital Signal Processors*) como, por exemplo, sua flexibilidade em aplicações de alto desempenho e ainda, mais particularmente, em aplicações que possam explorar larguras de bits específicas e com alto grau de paralelismo de instruções [8].

Neste trabalho, é implementado um algoritmo de subtração do fundo para detecção de objetos em movimento em uma placa de desenvolvimento DE2 da Terasic com um FPGA Cyclone II. Para alcançar isto, a imagem de fundo em níveis de cinza é armazenada em uma memória SRAM externa. O sistema realiza um pre-processamento filtrando a imagem de entrada usando convolução espacial antes do armazenamento do fundo para realizar a subtração. Uma vez realizada a subtração das imagens, a imagem resultante é segmentada utilizando um limiar, para depois ser submetida a um ultimo processo de filtragem, desta vez morfológico, com o fim

de eliminar o ruido da ultima etapa. Finalmente, o centro de gravidade do objeto é calculado e enviado para um computador via interface RS-232 utilizando o processador embarcado Nios II da Altera. No processador também é implementado um sistema de transformação das coordenadas retangulares da posição do objeto para coordenadas polares, com fim de obter informação sobre a posição do objeto, para ser utilizada em diversas aplicações como seguimento de objetos, robótica, navegação, etc. Além disso, para validar o sistema, foram implementados diferentes testes cujos resultados são apresentados aqui. O sistema também foi implementado em um sistema de visão omnidirecional como sensor de distância. Este foi colocado sobre um robô móvel.

1.2 Definição do problema e motivações

Um dos grandes desafios tecnológicos de nossa época está no desenvolvimento de sistemas que consigam reproduzir, ou ao menos aproximar, as capacidades humanas de sentir e, principalmente, interpretar o mundo [9], [10]. A visão do homem pode não ser a melhor dentre os animais (na verdade estamos em um patamar inferior com relação a muitas outras espécies), mas a nossa capacidade de interpretação de informações visuais é bastante vasta. A velocidade com que nosso cérebro é capaz de processar imagens, detectando padrões e extraindo informações é algo extraordinário e extremamente difícil de alcançar com os modelos e tecnologias atuais. A análise de informações visuais é uma vasta área, já muito explorada, mas com muito a ser descoberto e desenvolvido. O estudo de técnicas para automação ou semi-automação do processo de análise dessas informações requer a utilização dos avançados recursos tecnológicos que se têm hoje à disposição.

Por outro lado, a etapa de detecção de movimento é um componente essencial nos processos de automação de sistemas que precisam fazer um seguimento dos objetos numa cena. Além disso, os algoritmos de detecção de movimento não são utilizados só para processos industriais, mas também em uma vasta variedade de tarefas, como os sistemas de entretenimento como os jogos eletrônicos ou em dispositivos utilizados dia a dia como telefones celulares e câmeras digitais. Neste tipo de dispositivos portáveis ou diferentes equipamentos com funcionamento *out-door*, nos quais o desempenho é um fator importante e determinante, representam na atualidade uma das maiores demandas no desenvolvimento de pesquisas para se desenvolver tecnologias capazes de melhores resultados, fazendo a diferença para os consumidores exigentes.

Por outro lado, no GRACO (Grupo de Automação e Controle) do programa de sistemas mecatrônicos da Universidade de Brasília são realizados muitos trabalhos em diferentes áreas, tais como robótica industrial, robótica móvel, automação de manufatura, onde os sistemas de visão têm importantes aplicações.

Pelos motivos expostos aqui, o desenvolvimento de uma arquitetura capaz de realizar a detecção de objetos em movimento, com um baixo custo computacional e empregando uma mínima quantidade de recursos em *hardware* (sem perdas no seu desempenho) é relevante tanto em aplicações de engenharia como em ciência da computação.

1.3 Objetivos

1.3.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma arquitetura de baixo custo computacional para detecção de objetos em movimento em tempo real, utilizando o algoritmo de subtração do fundo.

1.3.2 Objetivos Específicos

Os objetivos específicos deste trabalho são os seguintes:

1. Estudar os diferentes algoritmos existentes na literatura para a detecção de movimento e suas possibilidades de implementação em sistemas embarcados.
2. Desenvolver um sistema de captura da imagem de fundo, armazenando-a em uma memória SRAM externa, realizando a correta sincronização com o sistema de captura e visualização das imagens.
3. Implementar uma etapa de pre-processamento das imagens com o fim de obter melhores resultados na etapa de detecção.
4. Desenvolver uma arquitetura para a detecção de movimento, utilizando o algoritmo de subtração do fundo.
5. Implementar um algoritmo para o cálculo do centro do objeto, sendo esta uma informação importante para etapas posteriores.
6. Desenvolver um sistema de comunicação para o envio dos dados gerados pela arquitetura de detecção, com o fim de que forem utilizados para outro tipo de aplicações como seguimento de trajetórias, navegação, etc.
7. Implementar um algoritmo de transformação de coordenadas retangulares para coordenadas polares em *software*, com fim de obter informação importante sobre a posição do objeto, que possa ser utilizada para diversas aplicações.

8. Desenvolver e implementar um sensor de distância baseado em subtração do fundo para um sistema de visão omnidirecional, com o fim de validar os resultados do sistema de detecção de movimento em uma situação real.

1.4 Resultados Alcançados

Neste trabalho foi desenvolvido um sistema de detecção de movimento baseado no algoritmo de subtração do fundo. O sistema trabalha como uma arquitetura de tempo real em forma de *pipeline*, fornecendo um pixel por ciclo de relógio (23,03 MHz) depois de um período de latência, fazendo possível processar imagens de 800×480 com uma taxa de 60 fps (um pixel processado cada 43,43 ns). Adicionalmente, o mesmo algoritmo para detecção de movimento foi implementado em um computador, rodando a 2,2 GHz, 2,0 GB de memória RAM usando o sistema operacional de tempo real *xPC Target* da MathWorks. O tempo médio para o processamento de uma imagem de 10×10 foi de aproximadamente 138,1 μs, (valor do TET - *Task execution time*). Assim, um pixel de saída é processado em 1,381 μs. Portanto, a arquitetura em *hardware* proposta neste trabalho alcança uma velocidade de processamento 32 vezes maior em comparação com a mesma implementação feita em *software*.

Por outro lado, o algoritmo de detecção foi implementado (sobre um sistema de visão omnidirecional) para a calibração e validação de um sensor de distância visando aplicações reais em robótica móvel. Este sistema apresentou um bom desempenho, além de mostrar que é uma técnica adequada para este tipo de aplicações. Os resultados deste sistema são apresentados com maior detalhe na seção 5.7.

1.5 Estrutura do Texto

Esta dissertação está organizada da seguinte forma:

Nos capítulos 2 e 3, é feita uma revisão bibliográfica sobre os diferentes algoritmos de processamento de imagens e detecção de movimento, assim como também serão revisadas as plataformas geralmente utilizadas focando o nosso estudo nos FPGAs.

Em seguida, o capítulo **??** é explicada a plataforma de *hardware* utilizada na implementação do sistema proposto. Algumas características de interesse para a definição das arquiteturas são realçadas e detalhes específicos do FPGA utilizado são discutidos rapidamente.

No capitulo 4 as implementações das arquiteturas propostas em uma linguagem de descrição de *hardware* de alto nível são mostradas. Detalhes de configuração e explicações do código-fonte são feitas, mostrando os passos de cada implementação. Os testes realizados para a validação

do sistema e resultados experimentais são discutidos no capítulo 5, seguido das conclusões no capítulo 6.

2 Algoritmos de Processamento de Imagens e Detecção de Movimento

Neste capítulo se faz uma pequena introdução à área de processamento de imagens e vídeos, mostrando os algoritmos mais comuns de tratamento de imagens e detecção de movimento, destacando alguns dos principais problemas de implementação. Finalmente, é feita uma reflexão sobre os sistemas reconfiguráveis e suas vantagens e desvantagens com relação às demais tecnologias.

A percepção é a relação entre estímulos visuais e alguns modelos previamente existentes no mundo [10]. Há uma grande lacuna entre a imagem e os modelos propostos (por exemplo, ideias, conceitos, etc.) que explicam, descrevem ou resumem as informações contidas na imagem. Para cobrir este espaço, os sistemas de visão computacional têm vários elementos capazes de "ligar" a informação visual com uma saída, que pode ser uma descrição final, uma interpretação ou uma tomada de decisões [11]. Portanto, a visão computacional é responsável de desenhar ferramentas e aplicações de algoritmos capazes de adquirir, digitalizar e processar as informações de uma imagem para extrair da mesma determinadas propriedades, equivalente à inspeção humana.

2.1 Aplicações do Processamento de Imagens

A utilização de técnicas de processamento de imagens na resolução de problemas, por meio da análise de informações visuais, já envolve inúmeras áreas de aplicação, principalmente devido ao grande avanço tecnológico na área de computação ao longo das últimas três décadas. Dentre as áreas de aplicação mais comuns, podemos citar: a área militar, automação industrial, medicina, sensoriamento remoto, segurança e multimídia [9].

Na área militar, são encontradas aplicações como rastreamento de alvos móveis, auxílio à navegação de veículos não-tripulados e identificação de cenas e alvos em imagens aéreas.

Na área de automação industrial são muito comuns equipamentos de inspeção visual de produtos, rótulos e embalagens, com o intuito de efetuar o descarte de produtos com não-conformidades. Outra utilização é na localização de objetos para manipulação por robôs e na monitoração de processos de fabricação em tempo real, para realimentação de controladores.

Na medicina, os sistemas de auxílio a diagnósticos permitem aos médicos uma maior segurança quando da análise de imagens oriundas de tomógrafos, aparelhos de raios X, ultrasonografia e ressonância magnética. Atualmente são implementadas outras técnicas além das mais simples (como ajustes de contraste e brilho) visando a construção de equipamentos mais sofisticados que oferecem, com base na imagem e em um banco de dados, listagens de possíveis doenças do paciente.

No campo de *ciência e engenharia de materiais*, imagens de microscópios são muito utilizadas em análises metalográficas e cristalográficas, permitindo assim a identificação de estruturas e padrões de formação em materiais, auxiliando na melhoria de processos de fabricação e na busca de novas aplicações e novos materiais.

O sensoriamento remoto permite a análise e interpretação não só de imagens no espectro visual, mas também imagens oriundas de diversos tipos de sensores, como infravermelhos e magnéticos. Desta forma permite-se a identificação de zonas de mineração, correntes marítimas, focos de incêndios, desmatamento e crescimento urbano. Isto permite um melhor planejamento por parte do governo e empresas, no sentido de aproveitar os recursos naturais e planejar estradas, rodovias e zoneamentos territoriais.

Sistemas de controle de acesso utilizam técnicas de leitura de digitais assim como análise de padrões da iris, retina ou ainda o reconhecimento de faces. O reconhecimento óptico de caracteres pode identificar automaticamente uma assinatura, permitindo a autenticação de cheques ou mesmo a conversão de imagens em textos.

Já no campo do entretenimento, novas aplicações surgem a todo momento. Videogames extremamente realistas utilizam tecnologias muito avançadas de computação gráfica, bem como algoritmos otimizados, principalmente de renderização de imagens. Câmeras fotográficas já são capazes de efetuar um disparo automático pela detecção de um sorriso. Por outro lado, as altas resoluções dos sensores comuns em aparelhos celulares exigem que existam algoritmos de compressão e descompressão de imagens e vídeos embarcados em dispositivos cada vez menores. Houve ainda, no início do século XXI, a consolidação dos sistemas de televisão digital, com recursos avançados, como interatividade e alta definição de imagens.

A Internet já é utilizada como meio de transmissão de vídeos até mesmo em tempo real. Por isto precisa-se de aquisição e processamento velozes, bem como de algoritmos de compressão e descompressão de dados extremamente sofisticados para permitir maiores resoluções em canais

de transmissão com largura de banda limitada.

2.2 Imagens Digitais

Todo sistema de visão computacional tem por objetivo principal processar informações de uma imagem. Isso exige que aquela informação possa ser compreendida e analisada pelo computador. Devido a isto, é muito importante entender como está conformada uma imagem digital.

O termo "*imagem*" refere-se a uma função bidimensional de intensidade de luz $f(x, y)$, onde x e y representam as coordenadas espaciais e o valor de f (em ponto qualquer) é proporcional ao brilho (ou nível de cinza) da imagem neste ponto [12]. O nível de cinza em um ponto da imagem vai depender da quantidade de luz que incide sobre a cena (Equação 2.1). Os componentes $i(x, y)$ e $r(x, y)$ são chamados de *iluminação* e *reflexão* respectivamente. O produto das duas funções vai produzir a função $f(x, y)$.

$$f(x, y) = i(x, y)r(x, y) \tag{2.1}$$

A natureza da iluminação vai ser determinada pela fonte de luz; porém, a reflexão dependerá das características dos objetos na cena [13].

Uma *imagem digital* é uma imagem $f(x, y)$ que tem sido discretizada tanto nas coordenadas espaciais como no valor do nível de cinza. Uma imagem digital pode-se considerar como uma matriz na qual seus índices de fila e coluna identificam um ponto da imagem, sendo que o valor correspondente ao elemento da matriz indica o nível de cinza nesse ponto (Figura 2.1). Os elementos de uma distribuição digital deste tipo denominam-se elementos da imagem ou mais usualmente *pixels* do inglês *picture element* (*pixeis* no português) [12].

2.3 Tipos de Operações em uma Imagem

É comum a estratificação das operações de processamento de imagens e vídeos em três níveis: *baixo*, *médio* e *alto*, sendo que a diferença entre os níveis baseia-se na relação produzida entre os dados de entrada e de saída. As operações de *baixo nível* recebem uma imagem em sua entrada e produzem uma imagem em sua saída. Operações de *nível médio* recebem uma imagem na entrada e produzem atributos de imagem na saída. Já as operações de *alto nível* recebem os atributos em sua entrada e os interpretam, em geral, para perfazer alguma ação ou tomar alguma decisão [12]. A seguir será descrito melhor cada nível, mostrando os tipos de paralelismo mais presentes em cada um.

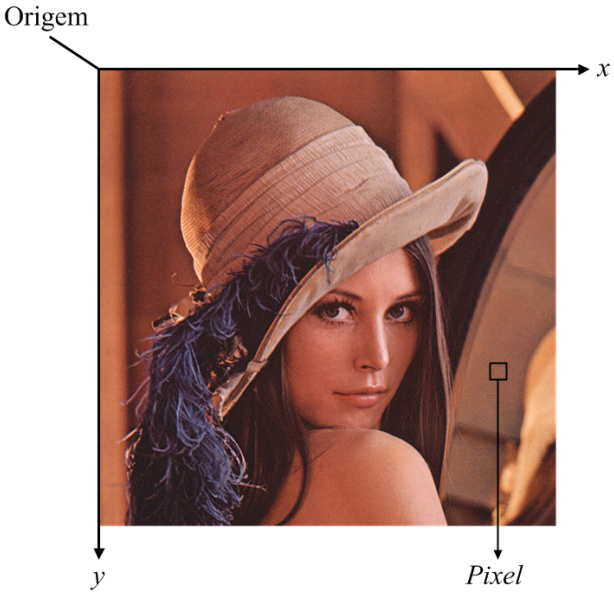

Figura 2.1. Convenção de eixos utilizada para a representação de imagens digitais.

2.3.1 Operações de Baixo Nível

As operações de baixo nível transformam imagens em imagens, ou seja, os operadores lidam diretamente com a matriz de dados em nível de pixeis. Tais operações incluem as transformações de cores, ajustes de contraste, filtragem, redução de ruídos, realce de bordas, assim como algumas transformações no domínio da frequência, etc. Um dos principais objetivos dessas operações é o realce de características da imagem, preparando-a para operações subsequentes no estágio de nível intermediário, com a simples visualização ou mesmo uma extração de características [10].

Há três tipos de operações neste nível: (a) pontuais, (b) de vizinhança e (c) globais. As mais simples são as pontuais, que transformam um pixel de entrada em um pixel de saída, sendo que o resultado dessas operações não depende dos valores vizinhos ao pixel (Figura 2.2).

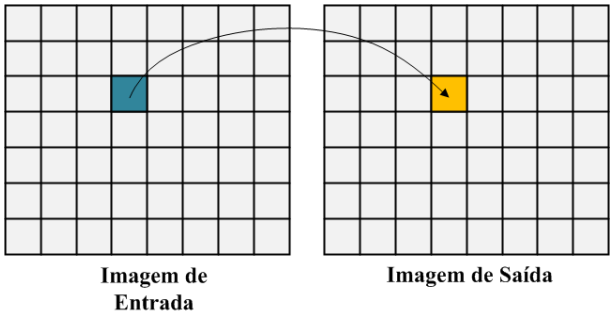

Figura 2.2. Operação pontual: Um pixel de entrada gera como resultado um pixel de saída [10].

As operações de vizinhança (vide Figura 2.3) são mais complexas do que as operações pon-

tuais, utilizando, para cada pixel de saída, os dados dos pixeis vizinhos. A complexidade do algoritmo vai depender do tamanho da vizinhança utilizada. Algumas operações de este conjunto são a convolução e correlação espacial, a filtragem, a suavização, o realce de contornos, entre outras [10].

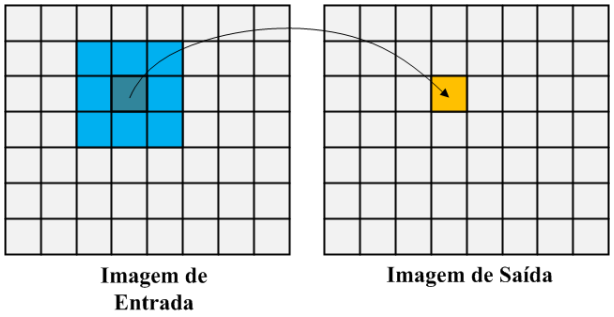

Figura 2.3. Operação de vizinhança: um conjunto de pixels de entrada gera um pixel de saída [10].

No caso das operações globais, para a geração de um único pixel de saída, tem-se que trabalhar com todos os pixeis de entrada. Um bom exemplo é a *transformada rápida de Fourier*, onde as operações lidam com quantidades de dados enormes de cada vez (Figura 2.4) [10].

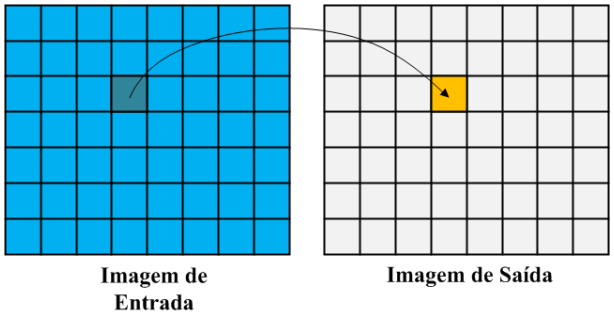

Figura 2.4. Operação global: todos os pixels da imagem de entrada geram um pixel de saída [10].

2.3.2 Operações de Nível Intermediário

Nas operações de nível intermediário, imagens são traduzidas em formas mais abstratas de informação, retratando certos atributos ou características de interesse da imagem. As operações aqui continuam lidando com os dados no nível de pixel, mas suas saídas apresentam uma redução significativa na quantidade de dados em relação às entradas. A segmentação da imagem em regiões (vide Figura 2.5), a extração de bordas, linhas e contornos, bem como outros atributos de interesse (como dados estatísticos) fazem parte do rol de operações desta classe. A redução na quantidade de dados (ou a conversão de uma imagem em um conjunto de características que a representa) é o objetivo desta etapa do processamento. As estruturas de cálculo aqui também

são bastante regulares, necessitando de muitas repetições de operações independentes sobre a imagem de entrada [10].

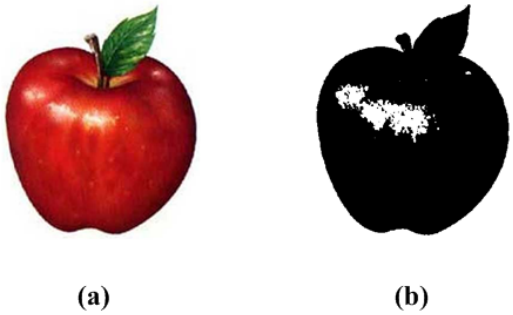

Figura 2.5. Exemplo de segmentação (operações de nível intermediário): (a) imagem original. (b) imagem segmentada

2.3.3 Operações de Alto Nível

A função das operações de alto nível é a interpretação do conjunto abstrato de dados que vêm dos níveis intermediários e a execução de uma análise do conteúdo de uma cena com base em algum conhecimento. Tais operações incluem o reconhecimento e classificação de objetos, ou ainda a tomada de decisões de controle. As estruturas de processamento nesta etapa são mais irregulares, utilizando operações inerentemente sequenciais (sobre uma massa menor de dados) em relação às outras etapas. Essas características aliam-se aos baixos requisitos de largura de banda de acesso à memória [10].

2.3.4 Cadeia Geral de um Sistema de Visão Computacional

Na maioria sistemas de visão computacional busca-se reduzir a quantidade de dados a ser processados. Geralmente, em uma cadeia de processamento de imagens e vídeos, a grande variedade de operações vai das operações mais regulares e com grande fluxo de dados, até as irregulares e com poucos dados no fim da cadeia. Uma típica cadeia de processamento combina os três tipos de operações encima mencionadas (operações de nível baixo, intermediário e alto) como mostrado na figura 2.6. Nesta figura está-se representando a redução na quantidade de dados em cada etapa para uma imagem de $N \times N$ com P bits de precisão por cada pixel (o que seria $N^2 \times P$ bits por quadro) em um típico exemplo de reconhecimento facial. Na etapa de *pré-processamento* há uma filtragem para eliminação de ruídos, mantendo assim a mesma quantidade de dados da imagem original ($N^2 \times P$ bits). Na etapa de *segmentação* há uma redução da quantidade de dados, pela transformação da imagem filtrada em uma imagem binária, de apenas 1 bit por pixel (redução para N^2 bits). Na etapa de *análise* há uma redução maior na quantidade

de dados, com a produção de um vetor de características representativas do contorno da imagem binarizada, resultando em K bits de dados. Por último, na etapa de *interpretação*, é gerada a informação sobre o reconhecimento ou não da face, com o fornecimento na saída de apenas 1 bit.

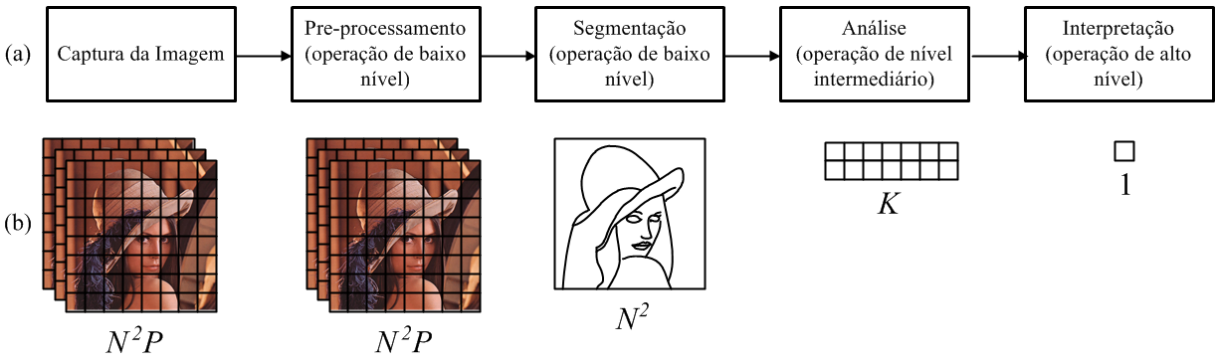

Figura 2.6. (a) Cadeia de processamento e (b) redução na quantidade de dados (adaptada de [10])

2.4 Algoritmos para o Processamento de Imagens

Devido à grande quantidade de algoritmos para o processamento de imagens, é inviável mencioná-los em sua totalidade neste trabalho. Por este motivo, nesta parte do trabalho serão consideradas as ferramentas mais comuns e com maior número de características para serem implementadas em sistemas reconfiguráveis.

O processamento de imagens procura modificar e preparar os pixeis de uma imagem digital para obter uma forma mais adequada, visando operações posteriores. Há dois grandes tipos de pré-processamento de imagens: (a) o melhoramento e (b) a restauração de imagens.

O melhoramento de imagens tenta aperfeiçoar a qualidade da imagem ou realçar aspectos particulares dentro da mesma. Este objetivo apresenta, geralmente, um pouco de subjetividade acerca do que é a qualidade, sendo que o resultado vai depender da operação e da aplicação. Os resultados podem produzir uma imagem bem diferente da original e alguns aspectos da imagem podem ser sacrificados com o fim de melhorar outros. Por outro lado, o objetivo da restauração de imagens é recuperar a imagem original que foi degradada por fatores conhecidos como uma distorção geométrica dentro do sistema da câmera [12].

Os dois tipos de operações tomam a imagem adquirida como uma entrada e produzem uma imagem modificada como saída. Muitas das operações mais comuns estão relacionadas com a aplicação de filtros lineares sobre a imagem original. Um exemplo óbvio é a eliminação de ruído na imagem [14]. Tanto para fins de melhoramento como de restauração de imagens, os processos mais utilizados são os de convolução e/ou correlação, que serão discutidos a seguir.

2.4.1 Filtragem Espacial por Convolução/Correlação

2.4.1.1 Fundamentos Matemáticos

Qualquer sistema de aquisição e transmissão de dados (uma câmera adquire e transmite dados visuais) está sujeito ao ambiente em que se encontram os dados e os equipamentos utilizados. Fatores como temperatura, pressão, luminosidade, campos elétricos e magnéticos, etc., podem influenciar sobremaneira na qualidade de uma determinada imagem. Com a influência desses fatores externos, uma imagem pode sofrer degradação ou perda de qualidade, em virtude de introdução de ruídos, perda de contraste, borramento e distorções [9].

A eliminação de ruídos em sinais é um campo de estudo bem consolidado, sendo essencial para isso a utilização de filtros. Um filtro consiste em um operador de transformação da imagem original em uma imagem com características de interesse realçadas [9]. Existem duas categorias básicas de técnicas para filtragem de imagens: (a) filtragem no domínio da frequência e (b) filtragem no domínio espacial. A primeira categoria envolve a atuação sobre o espectro de frequências da imagem, utilizando principalmente a transformada de Fourier, além de outras transformadas úteis como a transformada *wavelet*. Já a segunda categoria consiste na manipulação direta dos pixeis da imagem [10].

Neste contexto, existem basicamente três tipos de filtros: (a) passa-baixa, (b) passa-faixa e (c) passa-alta, sendo cada um responsável por realçar/atenuar determinadas características da imagem. Um filtro passa-baixa atenua detalhes da imagem, causando assim um efeito de borramento, suavizando mudanças abruptas de intensidade na imagem (mudanças abruptas representam altas frequências espaciais). Já um filtro passa-alta realiza um aguçamento dos detalhes mais finos da imagem, realçando as altas frequências espaciais. A maioria das aplicações de filtros espaciais é baseada em uma operação chamada de *convolução* descrita a seguir (Figura 2.7). Este tipo de operações são classificadas como operações de vizinhança, já que, como foi mostrado na Figura 2.7, um conjunto de pixeis de entrada gera um pixel de saída [5].

O primeiro passo consiste em sobrepor a máscara do filtro a uma região da imagem, centrada em um pixel. A escolha da máscara definirá o tipo de filtragem efetuado. O segundo passo consiste em multiplicar cada valor da máscara pelo valor do pixel superposto. O terceiro passo corresponde à soma dos produtos do passo anterior e a atribuição do valor da soma ao pixel correspondente ao pixel central na imagem de saída. O último passo consiste em deslocar máscara centralizando-a em um pixel vizinho ao pixel anterior na imagem original, e repetirem-se os passos anteriores. Essa sequência deve ser repetida até todos os pixeis da imagem original terem sido contemplados [10].

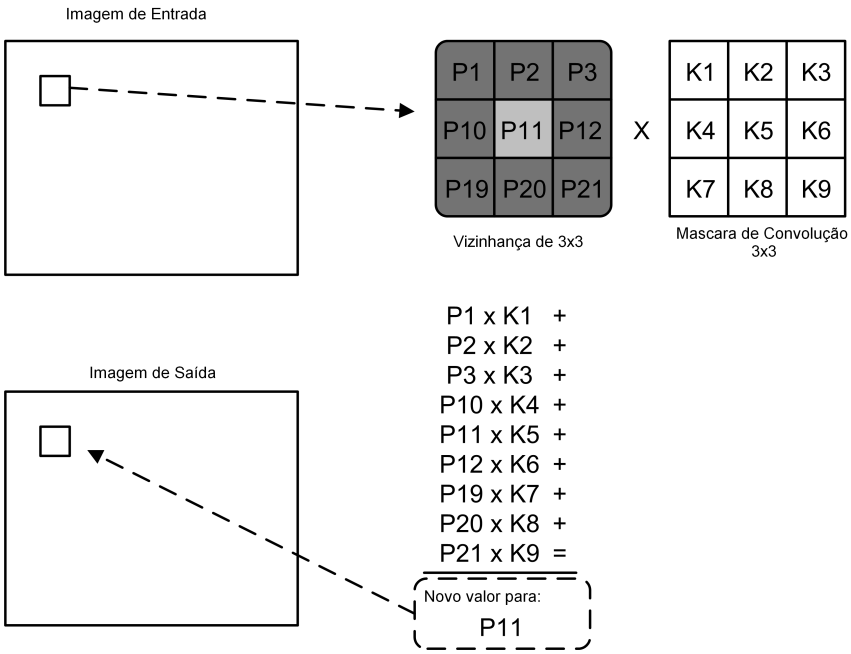

Figura 2.7. Operação de convolução [5]

Uma imagem digital é representada por um espaço finito de pixeis, pelo que as operações de janelamento apresentam uma distorção na sua aplicação sobre as bordas da imagem, já que os pixeis de borda não possuem todos os seus vizinhos. Um método bastante comum de se evitar esse efeito de borda consiste em aplicar o filtro apenas sobre pixeis que possuem todos os vizinhos (eliminando os outros). Desse modo, ocorre uma redução no tamanho da imagem final em relação à original [10].

A operação de janelamento é muito semelhante às operações de convolução e correlação discretas bidimensionais, cujas definições matemáticas são dadas pelas Equações 2.2 e 2.3, respectivamente [12].

$$k(x,y) * i(x,y) = \sum_{l=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} k(l,j)i(x-l, y-j), \tag{2.2}$$

$$k(x,y) \circ i(x,y) = \sum_{l=-m/2}^{m/2} \sum_{j=-n/2}^{n/2} k(l,j)i(x+l, y+j). \tag{2.3}$$

Os filtros, máscaras ou *kernels*, $k(x,y)$, podem ter tamanhos e formas variados, possuindo valores determinados arbitrariamente de acordo com a função. No tópico a seguir, serão mostrados alguns exemplos de aplicação da convolução e da correlação.

2.4.1.2 Complexidade Computacional

As operações baseadas em janelamento são de alto custo computacional e constituem uma carga computacional grande para a maioria dos processadores sequenciais. A complexidade das operações baseadas em janelamento pode ser expressada em termos das operações matemáticas elementais necessárias para processar uma imagem. Particularmente, a complexidade computacional pode ser calculada analisando a convolução espacial em imagens. O número total de operações necessário para desenvolver a operação de convolução espacial (considerando uma imagem de tamanho $M \times N$ e uma máscara de $w \times w$) é de $w^2 \times M \times N$ operações. Por exemplo, para desenvolver uma convolução numa imagem de 512×512 com uma máscara de 7×7 são necessárias mais de 50 milhões de operações. Com o fim de alcançar um desempenho de tempo real (pelo menos 30 fps) é necessária uma potência computacional de muitas giga-operações por segundo ($GOPs$). Assim, para uma máscara maior, a convolução torna-se ainda mais custosa computacionalmente devido ao crescimento quadrático de w no número de operações. Em termos gerais, uma operação baseada em janelamento tem uma complexidade computacional de $O(w^2 \times M \times N)$ para uma imagem de tamanho $M \times N$ e uma máscara de $w \times w$ [15].

2.4.1.3 Exemplos de Aplicação

- **Filtragem Passa-Baixas**: a forma da resposta à função impulso necessária para implementar um filtro espacial passa-baixas indica que o filtro tem todos os seus coeficientes positivos. Para um filtro espacial de 3×3, a construção mais simples consistiria em uma máscara na qual todos os seus coeficientes são iguais a um (Figura 2.8(a)) [12].

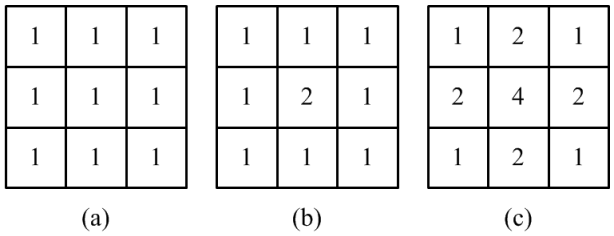

Figura 2.8. Exemplos de filtros passa-baixas [12].

  Porém, este filtro pressupõe que a influência de todos os pixeis é igual. Outra consideração é que, quanto mais longe esteja o pixel do central, seu valor será menor, para isso tem-se a máscara da Figura 2.8(b). No caso de precisar que o pixel central tenha ainda mais importância, pode-se utilizar o filtro da Figura2.8(c). Na Figura 2.9 mostra-se o resultado da filtragem para uma imagem com a máscara mostrada na Figura 2.8(a).

- **Filtragem Passa-Altas**: O perfil de resposta ao impulso necessária para implementar

(a) (b)

Figura 2.9. Filtragem espacial passa-baixa: (a) imagem original. (b) Imagem filtrada (máscara da Figura 2.8(a))

um filtro passa-altas indica que o filtro deve ter coeficientes positivos perto do centro e negativos na periferia. Para uma máscara de 3×3 esta condição vai-se cumprir escolhendo um valor positivo no centro e tomando coeficientes negativos nos demais valores da máscara (Figuras 2.10 (a), (b) e (c))

-1	-1	-1
-1	8	-1
-1	-1	-1

(a)

-1	0	1
-1	0	1
-1	0	1

(b)

-1	0	1
-2	0	2
-1	0	1

(c)

Figura 2.10. Exemplos de filtros passa-altas. Estas máscaras detectam bordas na imagem.

A figura 2.11 mostra a aplicação mais clássica do filtro de realce 3×3 mostrado na Figura 2.10 (a). Observe-se que a soma dos coeficientes é igual a zero, assim, quando a máscara está sobre uma área com nível de cinza constante ou pouco variável, a saída proporcionada pela máscara é zero ou um valor muito pequeno. Assim, quando a máscara encontra-se sobre uma região onde os níveis de cinza têm grandes mudanças, o resultado da convolução fornece valores muito altos, realçando estas regiões na imagem de saída (geralmente, estas regiões pertencem às bordas dos objetos presentes na imagem).

- **Correlação**: a *correlação* (Equação 2.3) pode ser aplicada para filtragem do mesmo modo que a convolução, bastando para isso calcular a transposta da máscara usada na convolução. Por exemplo, caso desejássemos filtrar uma imagem por correlação utilizando a máscara do filtro *Prewitt Vertical* (Figura 2.12 (a)), bastaria executar a operação de convolução com a máscara do filtro *Prewitt Horizontal* (Figura 2.12 (b)).

  Porém, há outra aplicação muito simples da correlação na identificação de padrões de objetos [10]. A Figura 2.13 ilustra esse conceito. Nessa figura, a primeira imagem é a

Figura 2.11. Filtro passa-altas aplicado sobre a imagem da Figura 2.9 (a)

-1	0	1
-1	0	1
-1	0	1

(a)

-1	-1	-1
0	0	0
1	1	1

(b)

Figura 2.12. Exemplos de filtros passa-altas. (a) Prewitt Vertical. (b) Prewitt Horizontal

original, composta de alguns símbolos; a segunda imagem é o padrão a ser buscado, e a última imagem é o resultado da operação de correlação entre a imagem original e o padrão. O ponto com nível de cinza mais intenso corresponde à posição de melhor correspondência entre o padrão e a imagem.

2.4.2 Limiarização

Muitas vezes não é de interesse saber a intensidade luminosa de um determinado objeto, mas simplesmente obter sua forma. Para tanto, uma imagem pode ter reduzida a quantidade de informação presente, passando por uma operação de *limiarização*. A limiarização de uma imagem consiste na transformação de seus valores de intensidade ("escala de cinza") em uma representação com apenas dois valores, ou seja, uma imagem *preto-e-branco* no sentido literal.

A limiarização (ou binarização), na sua forma mais comum, pode ser uma operação do tipo pontual (tópico 2.3.1), sendo caracterizada pela comparação do valor de cada pixel com um valor limiar global pré-determinado. Caso o pixel apresente um valor maior que o limiar, sua saída correspondente será um pixel de valor '1'; Caso contrário, a saída terá valor '0' [10].

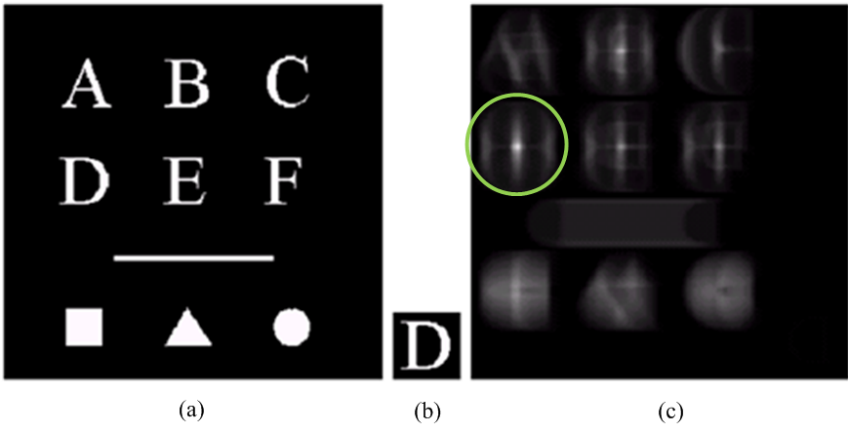

Figura 2.13. Casamento por correlação: (a) imagem original, (b) padrão a ser identificado, (c) imagem resultante da operação de correlação da imagem original com o padrão (ressaltado na imagem com um círculo verde).

A Figura 2.14 mostra três operações de limiarização para a mesma imagem (ver Figura 2.14(a)) de oito bits (valores de 0 a 255), com valores de limiar iguais a 128, 64 e 32 obtendo os resultados mostrados nas Figuras 2.14(b), 2.14(c) e 2.14(d) respectivamente.

Nota-se que houve uma perda gradativa da quantidade de informações, principalmente do fundo da imagem, de acordo com o limiar.

2.4.3 Morfologia Binária

A *morfologia binária* é uma ferramenta utilizada geralmente para extrair componentes de uma imagem que sejam úteis na representação e descrição de uma região, tais como contornos, esqueletos, etc. As técnicas morfológicas também são de interesse para processos como *filtragem morfológica*, *redução* e *recortado* [16].

As transformações morfológicas são aquelas que modificam a estrutura e forma dos objetos presentes em uma imagem. Estas ferramentas, além de ser úteis para a extração de características, permitem a eliminação do ruído produzido em todo processo de segmentação (limiarização) [13].

Na morfologia binária existem dois operadores básicos, chamados de *erosão* e *dilatação*. A erosão atua no sentido de remover pixeis das bordas externas dos objetos, enquanto que a dilatação acrescenta pixeis a essas bordas.

O processo de cálculo de erosão ou dilatação possui estrutura semelhante à convolução e correlação. Ambas são operações de vizinhança, utilizando máscaras para o processo. No caso da morfologia, o processo baseia-se em sobrepor uma máscara à vizinhança, para, posteriormente, efetuar operações lógicas entre os elementos sobrepostos. A máscara neste caso é chamada

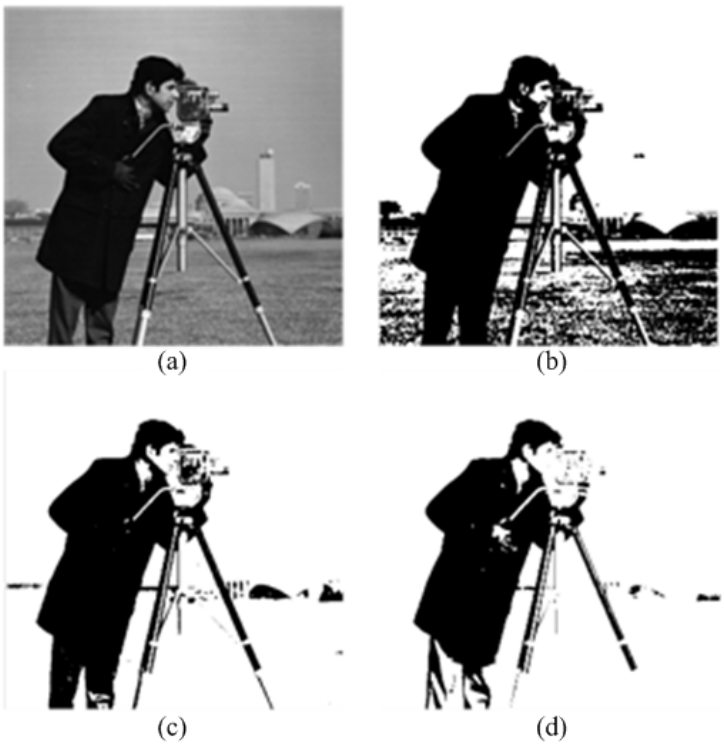

Figura 2.14. Exemplo de limiarização com limiares diversos [10]. (a) Imagem original. (b) 128. (c) 64. (d) 32.

elemento estruturante, pois está relacionado à estrutura, à forma [10].

O elemento estruturante é binário, assim como a imagem, possuindo apenas dois valores: branco e preto, verdadeiro e falso, zero e um. Um exemplo das operações de dilatação e erosão é dado na Figura 2.15. Nesse exemplo, um pixel branco é considerado como '0' e um pixel preto como '1', para facilitar a visualização.

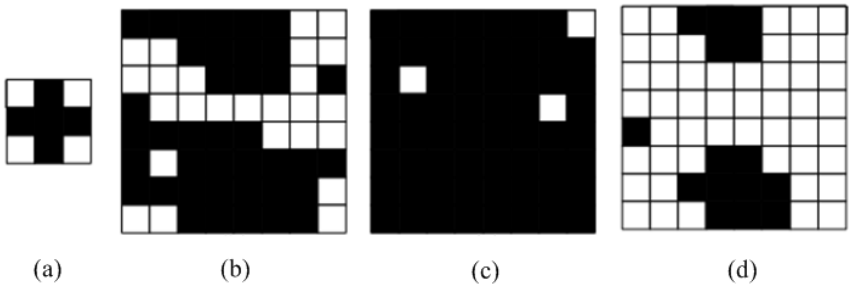

Figura 2.15. (a) Elemento estruturante, (b) imagem original, (c) dilatação, (d) erosão.

Em uma erosão, comparam-se os valores do elemento estruturante e os valores da vizinhança de sobreposição. Caso todos os pixeis '1' sejam sobrepostos a pixeis '1', o resultado também será '1'. Se ao menos um dos pixeis '1' do elemento estruturante sobrepor-se a um pixel '0', o resultado será '0'.

Por outro lado, em uma dilatação, caso ao menos um pixel '1' do elemento estruturante sobrepor-se a um pixel '1', o resultado será '1'. Se não houver nenhuma sobreposição de um pixel '1' por outro pixel '1', o resultado será '0'.

As operações de dilatação e erosão são classificadas como operações de vizinhança, pois necessitam de dados de uma vizinhança para seus cálculos. As arquiteturas propostas para essas operações são semelhantes às arquiteturas das operações de convolução/correlação. Os elementos de processamento para dilatação e erosão são baseados em operações lógicas simples, e seu projeto pode ser feito com base em tabelas-verdade. Considerando o elemento estruturante e a vizinhança de interesse da Figura 2.15, o cálculo da erosão e da dilatação pode ser dado como mostrado na Figura 2.16, onde k_i representam os valores da máscara e f_i os valores da imagem.

Erosão

k_i	f_i	e_i
0	0	1
0	1	1
1	0	0
1	1	1

Dilatação

k_i	f_i	e_i
0	0	0
0	1	0
1	0	0
1	1	1

$$e_i = \overline{k_i.f_i} + k_i.\overline{f_i} + k_i.f_i$$

$$d_i = k_i \,.\, f_i$$

$$Erosão = e_1.e_2.e_3.e_4.e_5.e_6.e_7.e_8.e_9$$

$$Dilatação = d_1 + d_2 + d_3 + d_4 + d_5 + d_6 + d_7 + d_8 + d_9$$

Figura 2.16. Operações de Erosão e Dilatação binárias [10].

Em aplicações práticas, combinações de dilatação e erosão são muito frequentes, e bastante poderosas na filtragem de imagens binárias. Geralmente, essas combinações são simplesmente a aplicação em sequência desses operadores. Por exemplo, seja a imagem com um texto digitalizado e já binarizada da Figura 2.17(a). Nota-se a presença de diversos pontos pretos espalhados pela imagem, caracterizando algum tipo de ruído. Realizando uma erosão da imagem, obtem-se como resultado a imagem da Figura 2.17(b), onde o ruído não está mais presente. Entretanto, pode-se perceber que as letras do texto ficaram um pouco degradadas. Efetuando uma operação de dilatação, obtêm-se a imagem da Figura 2.17(c), na qual o ruído não está presente, e o texto teve seu aspecto melhorado. Essa sequência erosão-dilatação é chamada *abertura*. Sua operação análoga é conhecida como *fechamento* (sequência dilatação-erosão) [10].

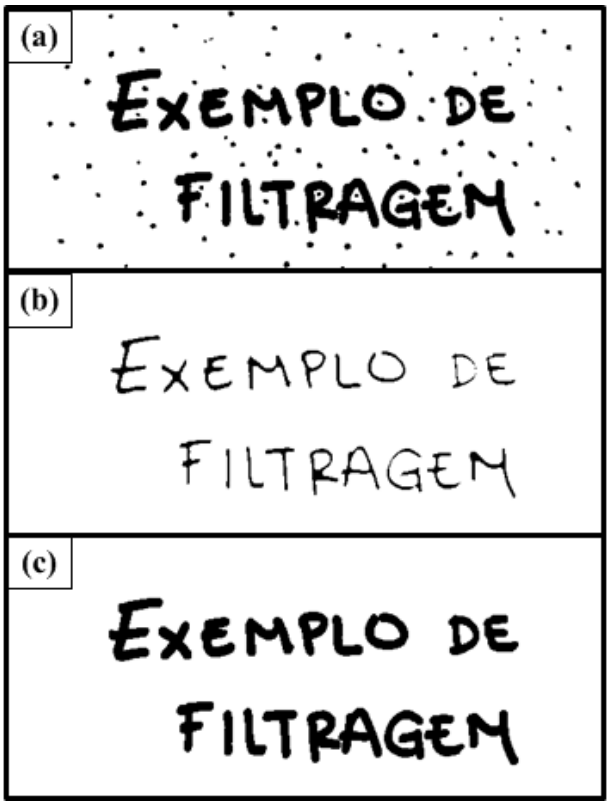

Figura 2.17. Exemplo de filtragem morfológica (*abertura*) utilizando combinações de erosão e dilatação. (a) Imagem original, (b) erosão da imagem original, (c) dilatação da imagem anterior.

2.5 Detecção de Movimento

A detecção de objetos em movimento em sequencias de imagens é uma tarefa muito importante para se ter sucesso em etapas futuras de um sistema de visão computacional, por exemplo como seguimento e reconhecimento de objetos, planejamento de trajetórias, entre outros. O objetivo principal do processo de detecção de movimento é segmentar os pixeis que pertencem aos objetos que estão se movimentando [17]. Para alcançar isto, existem varias abordagens para a tarefa de detecção de movimento: (a) subtração do fundo, (b) subtração de dois quadros consecutivos e (c) fluxo ótico [18].

2.5.1 Subtração do Fundo

A detecção de objetos em movimento por subtração do fundo pode ser descrita como ilustrado na Figura 2.18. Inicialmente, cada quadro da sequência é subtraído do fundo. Ao mesmo tempo, o quadro atual pode ser utilizado para atualização do fundo. Depois, a imagem resultante da subtração é segmentada com o fim de produzir uma imagem binária que ressalte as regiões em

movimento da imagem [16].

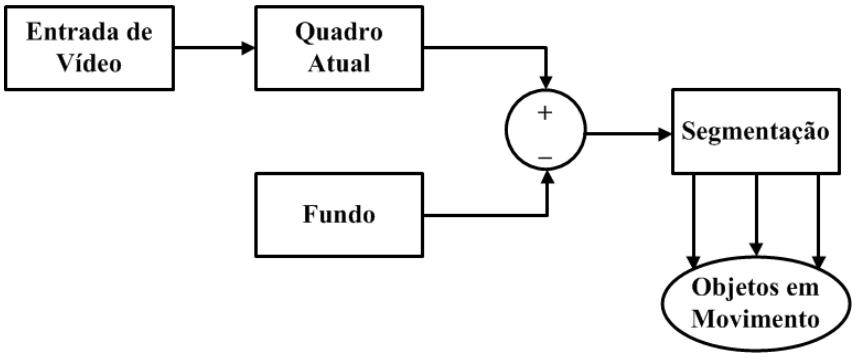

Figura 2.18. Algoritmo de subtração do fundo para detecção de movimento [19].

Cada nova imagem da sequencia (Figura 2.19(a)) é subtraída da imagem de fundo (Figura 2.19(b)). Ao mesmo tempo, a imagem atual é utilizada para atualizar a imagem de fundo. Depois, a imagem resultante da diferença é segmentada para produzir uma imagem binária (Figura 2.19(c)), que indica as regiões da imagem onde se apresenta o movimento. Finalmente, o objeto em movimento é detectado (Figura 2.19(d)).

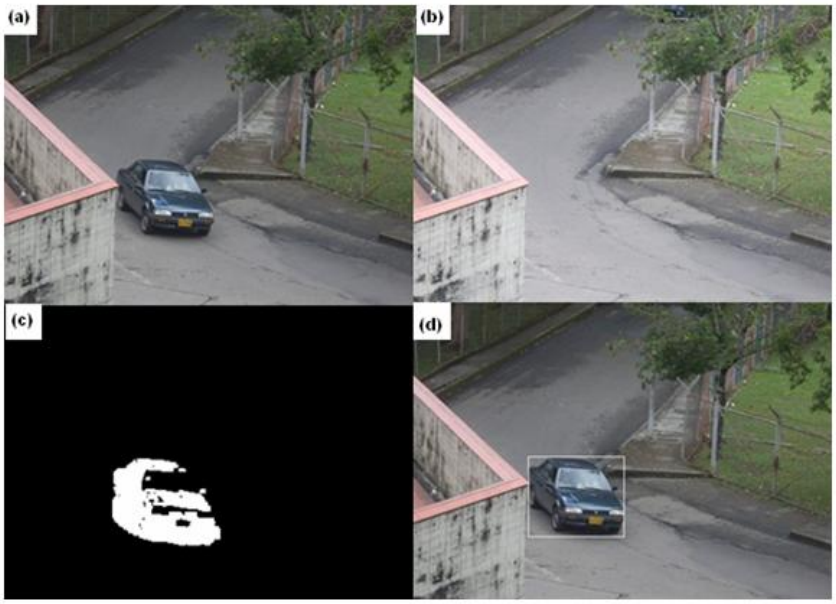

Figura 2.19. Detecção de movimento mediante subtração de fundo [16]. (a) Imagem atual. (b) Fundo. (c) Imagem subtraída e segmentada. (d) Objeto detectado.

Matematicamente, o algoritmo de subtração do fundo pode ser dado pela equação 2.4, onde T_d é um limiar predeterminado, $f(x, y, t)$ é uma imagem tomada no tempo t e $B(x, y)$ é a imagem de referência (ou fundo). Na análise dinâmica da imagem, todo pixel na imagem de movimento

$d(x, y, t)$ com valor "1" é considerado como pertencente a objetos em movimento na cena [17].

$$d(x,y,t) = \begin{cases} 1 & \text{se } |f(x,y,t) - B(x,y)| > T_d \\ 0 & \text{os pixeis restantes} \end{cases} \tag{2.4}$$

Este método de detecção de movimento vai depender das características do fundo, pelo que este pode ser calculado de varias formas. A maneira mais comum e mais simples consiste em calcular a média de um numero determinado de imagens iniciais, isto faz com que o sistema seja mais estável para mudanças pequenas de luz.

2.5.2 Subtração de Dois Quadros Consecutivos

Consiste na detecção de movimento achando a diferença entre dois ou três quadros consecutivos em uma sequência de vídeo. Este método adapta-se muito bem às mudanças dinâmicas na cena, mas apresenta erros geralmente na detecção de conjuntos de pixeis relevantes de alguns tipos de objetos em movimento. Matematicamente, o algoritmo de subtração de quadros consecutivos pode ser dado pela equação 2.5, onde T_d é um limiar predeterminado, $f(x, y, t)$ é uma imagem tomada no tempo t e $f(x, y, t-1)$ é uma outra imagem da sequência em um tempo anterior (geralmente de um a três quadros anteriores). $d(x, y, t)$ é a imagem de movimento, onde todo pixel com valor "1" é considerado como parte de um objeto em movimento.

$$d(x,y,t) = \begin{cases} 1 & \text{se } |f(x,y,t) - f(x,y,t-1)| > T_d \\ 0 & \text{os pixeis restantes.} \end{cases} \tag{2.5}$$

Este algoritmo pode ser considerado como uma variação do algoritmo de subtração do fundo, já que o método consiste em uma subtração de imagens, tendo em conta que, neste caso, a imagem de referência ou fundo está mudando com o tempo.

2.5.3 Fluxo Ótico

O fluxo ótico é definido como o movimento aparente do padrão em uma sequência de imagens. Pode ser considerado como uma aproximação do campo de movimento de cada pixel na imagem. Geralmente, a estimação de movimento por fluxo ótico se realiza usando algoritmos de gradientes espaço-temporais. Estes algoritmos trabalham determinando as mudanças de brilho espaciais e temporais (gradientes) do padrão de níveis de cinza da imagem, estimando o fluxo ótico desde estes parâmetros. Este método é baseado no fato de que objetos em movimento produzem mudanças no brilho [20]. A maior vantagem é que o método gera um campo vetorial completo, com um vetor de movimento por cada pixel, o que é muito útil em várias aplicações. Além disso,

a complexidade do algoritmo é invariante com o tipo de dados na entrada; onde este somente depende do tamanho da imagem [20].

Se em um tempo t, as coordenadas de um pixel na imagem com seu valor de intensidade (nível de cinza) é $I(x, y, t)$, e em um tempo $(t + \Delta t)$, o pixel tem-se movimentado para uma nova posição, sua localização na imagem passa a ser $(x + \Delta x, y + \Delta y)$, e seu nível de cinza $I(x + \Delta x, y + \Delta y, t + \Delta t)$. Baseados na hipótese de que a intensidade do pixel é constante, cumpre-se a equação embaixo (2.6) [21].

$$dI(x, y, t)/dt = 0, \tag{2.6}$$

ou seja que,

$$dI(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2.7}$$

Supondo que u e v são duas componentes do fluxo ótico ao longo das coordenadas x e y. Estas são definidas como:

$$u = dx/dt, v = dy/dt. \tag{2.8}$$

Fazendo a expansão em séries de Taylor de (2.7), obtém-se que:

$$\frac{\partial I}{\partial x}\frac{dx}{dt} + \frac{\partial I}{\partial y}\frac{dy}{dt} + \frac{\partial I}{\partial t} = 0, \tag{2.9}$$

de (2.8),

$$I_x u + I_y v + I_t = 0. \tag{2.10}$$

Esta é a equação básica do fluxo ótico [21]. Onde I_x denota a derivada parcial de $I(x, y, t)$ em x, I_y denota a derivada parcial de $I(x, y, t)$ em y, e I_t denota a derivada parcial de $I(x, y, t)$ no tempo. A equação (2.10) é também expressada como mostrado na equação 2.11.

$$\nabla \mathbf{I} \bullet \mathbf{U} + I_t = 0, \tag{2.11}$$

onde, $\nabla \mathbf{I} = (I_x, I_y)$ denota a direção do gradiente, e $\mathbf{U} = (u, v)^T$ denota o fluxo ótico.

Geralmente, as técnicas de fluxo ótico analisam a imagem como um todo, calculando o fluxo por cada pixel, independentemente do movimento. Isto resulta em uma grande quantidade de processamento desnecessário [22].

Os algoritmos de fluxo ótico podem detectar movimento numa sequência de imagens, mesmo que não sejam usadas câmeras fixas. Porém, a maioria destes métodos são muito custosos computacionalmente e não podem ser implementados em sistemas de tempo real sem um *hardware* especializado [19].

2.6 Conclusões do Capítulo

Neste capítulo foram explicados os conceitos básicos sobre imagens e o seu processamento, assim como os algoritmos que serão implementados neste trabalho para o tratamento das mesmas e para a detecção de objetos em movimento em sequências de vídeo.

Todos os algoritmos são de alto custo computacional devido à grande quantidade de informação que possuem as imagens, mas isto pode ser parcialmente resolvido com a utilização de arquiteturas paralelas e a sua implementação em *hardware*, como será visto no próximo capítulo.

Os três algoritmos de detecção de movimento têm alto desempenho, porém, o fluxo ótico é um algoritmo muito complexo (é necessário armazenar mais de uma imagem), precisando de uma grande quantidade de recursos de memória, o que é o seu principal inconveniente para a implementação em sistemas embarcados. Por outro lado, a subtração do fundo e de quadros consecutivos são algoritmos de baixo custo, porém, a subtração de quadros consecutivos tem problemas para detectar a forma do objeto,, geralmente, fazendo mais complexa uma posterior etapa de reconhecimento.

3 Plataformas para o Processamento de Imagens e Vídeo

Neste capitulo apresenta-se um breve histórico sobre as plataformas utilizadas para o processamento de imagens. Também é feita uma reflexão sobre os sistemas reconfiguráveis e suas vantagens e desvantagens com relação às demais tecnologias, tendo em conta aplicações similares às abordadas neste trabalho. Por último, é feita uma descrição de alguns trabalhos anteriores realizados na área de processamento de imagens e detecção de movimento.

Com o desenvolvimento dos *chips* de alto rendimento e assim como as tecnologias de processamento digital de sinais, o processamento de sinais em tempo real tem sido usado em muitas áreas da engenharia. Como é bem conhecido, diferentes aplicações têm diversas demandas. Por exemplo, um *chip* com uma velocidade de 50 milhões de instruções por segundo (MIPS) pode alcançar os requerimentos do processamento de sinais de áudio [23].

Por outro lado, perto do 70% da informação obtida pelas pessoas vem da visão e, portanto, adquirir e processar esta informação é muito importante. Muitas áreas, como são a multimídia, as comunicações, televisão de alta definição, processamento de imagens, reconhecimento de padrões e visão computacional, entre outras, têm colocado requisitos de desempenho muito altos para a aquisição e processamento de imagens de vídeo. Neste conxtexto, tanto o armazenamento de imagens como o processamento das mesmas têm sido desenvolvidos rapidamente [24].

3.1 Histórico das Plataformas de *Hardware* para o Processamento de Imagens e Vídeos

No fim da década de 1950 ocorreu nos Estados Unidos, mais precisamente em um laboratório do NIST (*National Institute of Standard and Technology*), um dos primeiros registros do pro-

cessamento de imagem digital; onde havia sido construído um equipamento digitalizador que armazenava as imagens na memória de um computador que as processava. Nesses primeiros testes os experimentos envolveram o uso de filtros de realce de bordas.

As áreas de inspeção industrial e imageamento médico foram uns dos que mais demandaram o rápido desenvolvimento de aplicações de processamento de imagens. Já naquela época, devido ao paralelismo facilmente identificado nas operações de níveis baixo e médio, as arquiteturas específicas para processamento de imagens eram construídas objetivando o paralelismo massivo. Os primeiros computadores utilizados para processamento digital de imagens eram grandes *main-frames* paralelos, porém, a busca pela miniaturização e os avanços nas tecnologias VLSI levaram ao desenvolvimento de soluções de processamento de alto desempenho, pequenas e com baixo consumo de energia, oferecendo a possibilidade de utilização de dispositivos de tamanho reduzido, mas com grande poder de processamento [10].

No contexto supracitado era comum, quando as aplicações possuíam requisitos de processamento em tempo real, a utilização de múltiplas placas com diversos processadores trabalhando em paralelo, especialmente em aplicações médicas e militares, onde o custo geralmente não é um fator determinante. Entretanto, quando do surgimento dos primeiros processadores programáveis de sinais digitais, esse pensamento começou a mudar. No fim da década de 1980 surgiram os primeiros DSPs comerciais, desenvolvidos especialmente para acelerar os algoritmos de processamento de sinais, o que forneceu as bases para a era dos *sistemas computacionais embarcados*.

Ainda nos anos 1980, houve o surgimento dos *dispositivos lógicos programáveis*, como os *CPLDs* e *FPGAs*, cujo objetivo era unir a flexibilidade do *software* com a velocidade do *hardware* dedicado dos *ASICs*. Nos anos 1990, o desempenho tanto dos *DSPs* quanto dos FPGAs cresceu bastante, de modo a atender à demanda dos dispositivos multimídia, surgindo o conceito de *SoC - System on Chip*, cuja premissa básica é o embutimento de todo o poder de processamento necessário em uma única pastilha [10].

Um esforço mais recente da comunidade científica tem sido na adaptação do grande poder de processamento presente nas unidades de processamento gráfico (*GPUs - Graphics Processing Units*) encontradas nos *PCs* e *laptops* modernos para o processamento de imagens e vídeos. Unidades de co-processamento são mais utilizadas atualmente em sistemas não embarcados, mas já estão sendo desenvolvidas muitas pesquisas na área [10].

3.2 Plataformas de *Hardware* para Processamento de Imagens e Vídeos

Devido aos grandes avanços tecnológicos nas últimas décadas, tem-se hoje à disposição diversos tipos de plataformas de *hardware*, com vantagens e desvantagens (umas em relação às outras). Neste contexto, é importante analisar as plataformas disponíveis para implementação, de modo a efetuar a melhor escolha da arquitetura de *hardware* a ser utilizada. Em [25] pode ser encontrado um bom resumo de alguns sistemas de processamento de imagens e vídeos. Os pontos de maior interesse para este trabalho são listados a seguir.

3.2.1 ASICs - *Application Specific Integrated Circuits*

Os *circuitos integrados de aplicação específica* oferecem o melhor desempenho dentre os processadores para algoritmos de imagens e vídeos, devido ao seu alto grau de especialização. Uma de suas principais desvantagens é o tempo de desenvolvimento do projeto, assim como a pouca flexibilidade de configurações pós-fabricação oferecida.

3.2.2 GPPs - *General Purpose Processors*

Os *processadores de propósito geral* englobam os processadores da linha Pentium, ARM e Spark, entre outros. São extremamente flexíveis na camada de *software*, porém sua utilização é geralmente feita em conjunto com um sistema operacional. Essa integração *hardware/software* já vem sendo desenvolvida há um tempo considerável, sendo considerada bastante otimizada. Os processadores encontrados nas residências e estações de trabalho possuem um alto desempenho, sendo arquiteturas altamente paralelizadas, envolvendo estruturas de pipeline com características que permitem explorar o paralelismo de instruções em aplicações de processamento de imagens e vídeos [25]. Tais processadores, porém, são grandes consumidores de energia, se tornando inadequados para sistemas embarcados.

Mais recentemente, surgiram as arquiteturas com mais de um núcleo de processamento (arquiteturas *multicore*), aumentando consideravelmente o desempenho desses processadores. Essas arquiteturas *multicore* surgiram, principalmente, devido às limitações de frequência de operação enfrentadas pelos processadores dos computadores pessoais atuais. Porém, já existem sistemas embarcados com arquiteturas *multicore*, que possui a arquitetura mostrada na Figura 3.1.

A idéia por trás da utilização de um sistema multiprocessado em um aparelho multimídia, como o IPhone, é a exploração de processamento paralelo, já que o mesmo dispositivo integra

funções de navegação na internet, telefonia móvel, câmera fotográfica e acessibilidade avançada (*display* LCD multi-toque e sensores internos de movimento, tais como acelerômetros e giroscópios).

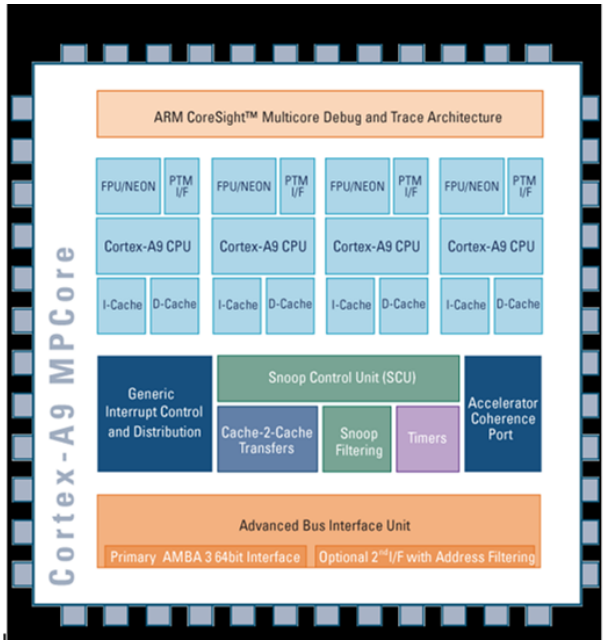

Figura 3.1. Arquitetura multicore presente no novo IPhone da Apple Corporation [10]

3.2.3 Processadores de Sinais Digitais (DSPs)

Os DSPs são processadores programáveis com características arquiteturais que os tornam vantajosos para o processamento de sinais. Muitos dos DSPs possuem arquitetura de acesso à memória do tipo Harvard, unidades MAC e registradores de deslocamento otimizados, além de instruções específicas de processamento de sinais em seu conjunto de instruções, permitindo assim uma compilação mais eficiente dos programas. Por permitirem programação através de linguagens de alto nível oferecem uma grande flexibilidade, facilitando o desenvolvimento das aplicações. Além disso, possuem tamanho reduzido e consumo relativamente baixo de energia, o que os coloca como os melhores candidatos quando o assunto é o desenvolvimento de um sistema embarcado. Os DSPs podem trabalhar tanto com aritmética de ponto fixo quanto de ponto flutuante, dependendo da precisão necessária à aplicação. Outra característica muito forte dos DSPs é a previsibilidade de seu processamento, garantindo um bom grau de segurança nas aplicações. Com todos esses pontos positivos, os DSPs são uma opção bastante viável em aplicações de processamento de imagens e vídeos. Mais recentemente têm sido incluídos como co-processadores em sistemas embarcados comuns, como telefones celulares, câmeras digitais, etc [10].

3.2.4 Processadores de Imagens Digitais

Assim como os DSPs possuem características arquiteturais que os tornam mais apropriados para o processamento de sinais, em desenvolvimentos recentes surgiram processadores programáveis com unidades de *hardware* dedicadas a operações de processamento de imagens. A maioria desses blocos de processamento específico executa operações de baixo nível, como convolução e morfologia [26].

Alguns DSPs integram conversores A/D a cada foto-sensor da matriz de aquisição de imagem e operam em todos os pixeis de forma paralela. A dificuldade de fabricação desses sistemas é enorme, de modo que ainda não existem comercialmente, sendo mais factíveis para resoluções baixas, como 64×64 pixeis [26].

Outros processadores de imagens possuem *buffers* de linha que permitem o armazenamento temporário de partes da imagem para um processamento mais rápido, porém essa estrutura demanda um espaço grande no *chip*, de modo que os sistemas implementados com esses *buffers* apresentam restrições no tamanho das imagens [27].

3.2.5 GPUs - *Graphics Processing Units*

Nos anos 2000 chegou ao mercado um novo tipo de processador, dedicado a atender às demandas de interpretação (renderização) de gráficos tridimensionais em tempo real dos modernos vídeo-jogos. Um processador desse tipo atualmente ultrapassa a quantidade de 120 *GFLOPs (Giga Floating-Point Operations per Second - Bilhões de operações de ponto-flutuante por segundo)* de poder de processamento, enquanto um processador Intel Pentium IV de 3.0GHz executa apenas 12 *GFLOPs* [25].

Grande parte do poder de processamento das GPUs está em sua arquitetura que explora paralelismo e estruturas de *pipeline* de modo avançado. O rol de operações que esses processadores executam é restrito, de modo que pôde ser extremamente otimizado. Uma GPU tipicamente possui diversos processadores dedicados trabalhando em paralelo, cada um com até 128 bits de profundidade de cores (4 vezes mais que os usuais 32 bits) por pixel [10]. Isso permite um grau de precisão na interpretação muito alto, alcançando um alto grau de realismo.

As unidades de processamento gráfico (*GPUs*) eram de certa maneira similares aos *ASICs*, com poucos recursos de configuração. Porém com o tempo foi sendo incorporada maior flexibilidade e possibilidades de programação, atraindo a atenção da comunidade de computação de alto desempenho. Atualmente diversas empresas e centros de pesquisa de todo o mundo já possuem trabalhos em desenvolvimento com o intuito de utilizar as GPUs para outras aplicações

diferentes à renderização 3D, um conceito conhecido como *GPGPU (General-Purpose processing on a Graphics Processing Unit)*. As *GPUs* já são utilizadas em problemas de processamento de imagens e vídeos em tempo real, em aplicações complexas como a reconstrução de imagens médicas de ressonância magnética e ultra-som. Um dos problemas enfrentados pelas primeiras gerações de GPUs eram os barramentos utilizados para conexão aos PCs, já que o barramento utilizado, *PCI (Peripheral Component Interconect)*, não era suficientemente rápido para atender às aplicações. O advento do barramento PCI Express está, ao menos temporariamente, suprindo às necessidades de conexão desses dispositivos. Outra questão importante é que as GPUs atuais foram desenvolvidas para apresentarem um alto desempenho, a despeito do consumo de energia, consumindo ainda mais que um GPP comum. Essa característica deixaria as GPUs de fora do mercado de aplicações embarcadas, porém já há um esforço de desenvolvimento de GPUs para sistemas embarcados.

3.2.6 Sistemas de Lógica Reconfigurável

Os sistemas de lógica reconfigurável são geralmente arranjos de componentes lógicos, interligados por uma rede programável. Seu nascimento se deu com o intuito de unirem-se a flexibilidade dos processadores e o desempenho dos ASICs em um único dispositivo. Os principais dispositivos dessa família são os FPGAs e os CPLDs. Suas características construtivas e de programação as tornam extremamente flexíveis e adaptáveis às necessidades específicas de cada aplicação, permitindo a implementação dos algoritmos diretamente em *hardware*. Assim como os DSPs, estes dispositivos oferecem uma grande previsibilidade de execução, característica essencial em sistemas com requisitos de tempo real. Por serem tão flexíveis, os FPGAs podem ser programados para explorar de modo eficiente as diferentes formas de paralelismo presente nos algoritmos de processamento de imagens e vídeos. Desta maneira, pode-se ter a implementação de diferentes algoritmos de baixo, médio e alto níveis, permitindo assim que sistemas completos sejam embarcados em um único *chip*.

Outra característica importante é a grande largura de banda de acesso à memória, e a flexibilidade de configuração desse acesso, permitindo um grau de especialização muito grande, adequando cada tipo de operação às memórias disponíveis de modo otimizado. Sua principal desvantagem é o grande consumo de energia, se comparado aos DSPs, e o tempo de treinamento de projetistas da área. Porém, com o constante avanço tecnológico, tal desvantagem vem diminuindo cada vez mais, fazendo com que os FPGAs já despontem como fortes candidatos a quebrar a hegemonia dos DSPs em aplicações embarcadas, principalmente para processamento de imagens e vídeos [8], [28]. Diversas ferramentas, como geradores de código, bibliotecas em linguagens de alto nível como C, *software* de simulação, ambientes gráficos de desenvolvimento e etc., vêm sendo

desenvolvidas com o intuito de elevar-se o nível de abstração do desenvolvimento de sistemas com FPGAs, de modo a facilitar o treinamento de equipes de projetistas [10].

3.3 Paralelismo em Operações de Processamento de Imagens e Vídeos

Essencialmente, imagens e vídeos digitais são sinais multidimensionais com grande quantidade de dados, exigindo muitos recursos de processamento e de memória [25]. Por exemplo, seja uma típica imagem digital, com $M \times N$ pixeis e P bits de precisão. Tal imagem possui $M \times N \times P$ bits de dados, sendo que cada pixel pode ser suficientemente representado por 1 byte, ou 8 bits. No caso de aplicações médicas e científicas, nas quais a utilização de 12 ou mais bits por pixel é necessária para um aumento da precisão dos resultados. Para o caso de imagens coloridas apresenta-se a triplicação da massa de dados, uma vez que o padrão mais comum é o *RGB* (com três canais de cores de mesma precisão). Caso o interesse da aplicação seja no processamento de vídeos, a taxa de aquisição de imagens deve ser levada em consideração, pois nesse caso, a velocidade de processamento deve ser tal que todas as imagens sejam processadas, sem perdas de quadro. O processamento de uma imagem única é feito sobre as duas dimensões espaciais do plano da imagem. Já no caso de um vídeo, muitas vezes é necessário atuar-se sobre os dados de imagens em sequência, levando em consideração também a dimensão temporal.

Quando se trabalha com o processamento de imagens e vídeos em tempo real, o problema de lidar-se com a grande quantidade de dados e cálculos torna-se uma grande preocupação. Uma simples câmera de vídeo digital, capturando um vídeo colorido de resolução padrão VGA (640×480) a 30 quadros por segundo, requer um processamento a uma taxa de 27 milhões de pixeis por segundo. Casos mais críticos, como câmeras de alta definição (*HD-High Definition*), onde cerca de 83 milhões de pixeis por segundo devem ser processados para uma resolução de 1280×720 pixeis por quadro, já são comuns. E esse é um problema que só tende a crescer, pois o mercado clama por taxas de aquisição e resoluções cada vez maiores, exigindo assim que uma quantidade cada vez maior de dados seja processada em intervalos de tempo cada vez menores.

O conceito de processamento paralelo (algo familiar para os projetistas da área de arquitetura de computadores) é uma das chaves para lidar-se com problemas envolvendo quantidades massivas de dados [29]. Muito do que se pode fazer na implementação de sistemas eficientes de processamento está centrado em quão bem a implementação (de *hardware* e *software*) explora as diferentes formas de paralelismo em um algoritmo: paralelismo em nível de dados (*DLP - Data Level Parallelism*) e paralelismo em nível de instrução (*ILP - Instruction Level Parallelism*). O paralelismo em nível de dados é encontrado quando se pode aplicar a mesma operação a dife-

rentes conjuntos de dados. Já o paralelismo em nível de instrução encontra-se na aplicação de distintas operações independentes de forma simultânea [29].

Para ter-se uma melhor visualização dos conceitos de paralelismo e como se aplicam aos problemas analisados, é necessário tentar enxergar mais de perto os tipos de operações envolvidas no processamento de imagens e vídeos.

3.3.1 Classificação de Arquiteturas Paralelas

Para se alcançarem melhores resultados no desenvolvimento e na implementação de arquiteturas específicas de processamento de imagens e vídeos, deve-se tomar proveito das características de cada algoritmo, identificando-se os principais problemas de desempenho e o que pode ser feito para melhorar esses pontos mais críticos.

A classificação de arquiteturas de *hardware* pode ser feita segundo diversos critérios, sendo a mais comum a taxonomia de Flynn, em que os computadores são divididos em classes e o processo de computação resulta da interação entre um fluxo de instruções e um fluxo de dados [10], [30]. Seguindo a proposta de Flynn, existem quatro classes de arquiteturas:

- **SISD** - *Single Instruction, Single Data*: possui um fluxo único de instruções e um fluxo único de dados. Um Elemento de Processamento (EP) executa uma seqüência de instruções sobre um conjunto de dados, sendo executada uma operação por vez. Esse modelo corresponde à arquitetura clássica de Von Neumann e engloba os microprocessadores comuns das estações de trabalho.

- **MISD** - *Multiple Instruction, Single Data*: trata-se de um modelo puramente teórico, não tendo sido registrada nenhuma implementação desta categoria. Consistiria na aplicação de várias instruções ao mesmo tempo sobre um único dado.

- **SIMD** - *Single Instruction Multiple Data*: consiste na aplicação de um fluxo único de instruções seqüenciais aplicado sobre fluxos de dados distintos, de modo paralelo. Pode ser vista como várias máquinas SISD executando a mesma instrução sobre conjuntos de dados independentes, de forma paralela.

- **MIMD** - *Multiple Instruction, Multiple Data*: corresponde a várias instruções sendo aplicadas simultaneamente a múltiplos conjuntos de dados. A maioria dos computadores ditos paralelos se encaixa nessa categoria.

3.4 *Hardware* Programável

A Computação Reconfigurável (*RC - Reconfigurable Computing*) é o uso de lógicas programáveis para acelerar a computação [8]. Esse conceito surgiu nos anos oitenta, com o lançamento no mercado dos FPGAs. A principal novidade desse conceito era que os dispositivos podiam ser reprogramados quantas vezes fosse necessário e que diferentes algoritmos eram implementáveis diretamente em *hardware*, de modo semelhante às implementações de diferentes *softwares* com um processador convencional.

Como já comentado, a execução de algoritmos diretamente em *hardware* oferece algumas vantagens, como a aceleração na velocidade de processamento (geralmente entre 10 e 100 vezes), o alto grau de especialização que uma implementação pode atingir, e as otimizações que podem ser feitas devido à grande flexibilidade desses dispositivos [10].

Os FPGAs representam estruturas em formas de arranjos de elementos lógicos *LEs* (portas/memórias). A arquitetura de um FPGA consiste em um conjunto de arranjos de blocos lógicos configuráveis, blocos I/O configuráveis assim como interconexões programáveis. Adicionalmente, os recursos lógicos podem incluir ALUs, elementos de memória e decodificadores. Os três diferentes tipos de elementos de programação para um FPGA são (a) RAM estática, (b) anti-fusível, e (c) flash EPROM [31].

A arquitetura genérica de um FPGA é mostrada na figura 3.2, onde observa-se que segmentos interconectados de metal podem ser ligados de forma arbitrária por chaves programáveis para formar as redes de conexões entre as células. FPGAs podem ser usadas em praticamente qualquer sistema de lógica digital e proporcionam os benefícios de níveis elevados de integração, sem os riscos do custo de desenvolvimento personalizado dos ASICs.

Quando da escolha de um determinado FPGA, três aspectos determinam o desempenho de um circuito implementado e também a densidade (grau de integração) do FPGA. Devem ser considerados: (a) a tecnologia de programação utilizada, (b) a arquitetura interna dos blocos lógicos e (c) a arquitetura de roteamento [10].

3.4.1 Tecnologia de Programação

A programação de um FPGA implica em ligar e desligar inúmeras chaves lógicas e as propriedades elétricas dessas chaves influenciam sobremaneira nas características de desempenho de cada dispositivo. A Tabela 3.1 mostra as três principais tecnologias de programação de um FPGA e algumas características de cada uma [33].

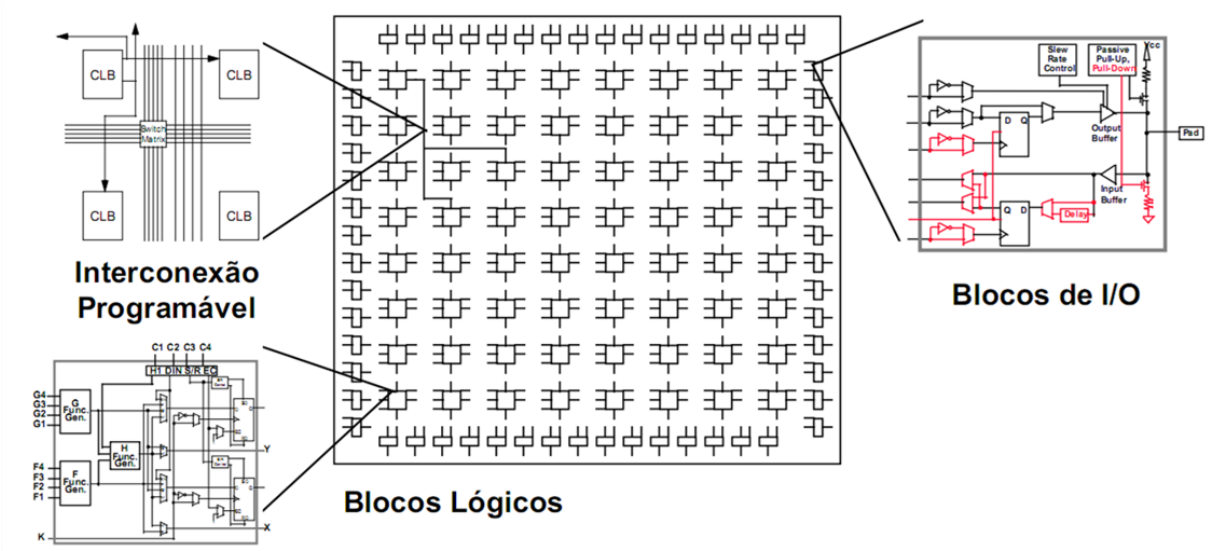

Figura 3.2. Estrutura de um FPGA [32]

Tabela 3.1. Resumo das tecnologias de programação de FPGAs [10]

Tecnologia / Característica	SRAM	Anti-Fuse	Gate-Flutuante
Tamanho	Pequeno	Grande	Médio
Velocidade de Programação	Alta	Baixa	Média
Volatilidade	Sim	Não	Não
Reprogramabilidade	Sim	Não	Sim

3.4.2 Blocos Lógicos

Os blocos lógicos dos FPGAs comerciais são capazes de implementar uma grande variedade de funções lógicas, tanto combinacionais quanto sequenciais, podendo ser simples como um transistor ou tão complexo quanto um microprocessador. Geralmente pode-se dividi-los em dois grupos básicos: (a) de granularidade fina e (b) de granularidade grossa.

Os blocos lógicos de granularidade fina (*fine-grain*) executam operações de apenas um bit de largura, definindo circuitos no nível de portas lógicas. Desse modo, tem-se uma grande flexibilidade de projeto, porém o trabalho de reconfiguração do dispositivo é bastante demorado, e como possuem a largura de apenas um bit, geralmente são utilizados aos milhares, mesmo para aplicações simples, necessitando de grande quantidade de trilhas de conexão.

Já os blocos lógicos de granularidade grossa (*coarse-grain*), representados principalmente pelas rDPAs (*reconfigurable data-path arrays* - arranjos de via de dados reconfiguráveis) já operam em nível de transferência entre registradores (RTL - *Register Transfer Level*), tendo sido muito

utilizados para a implementação de aceleradores de *hardware* para diversas aplicações [34].

Ambos os tipos de blocos lógicos possuem adicionalmente algum elemento de lógica sequencial, geralmente um flip-flop tipo D.

3.4.3 Arquitetura de Roteamento

A arquitetura de roteamento define a interconexão entre os blocos lógicos, sendo de extrema importância na síntese de um circuito, pois pode determinar a frequência máxima de operação e qual o espaço ocupado pela implementação. É importante saber se uma dada arquitetura permite o roteamento completo, ou seja, se um determinado circuito pode ser implementado naquele FPGA específico [10].

Assim como qualquer dos outros elementos da estrutura de um FPGA, os blocos de roteamento e interconexão ocupam espaço, sendo um ponto importante o balanceamento entre a flexibilidade da arquitetura de roteamento e a área a ser ocupada por ela.

3.4.4 Ciclo de desenvolvimento com FPGAs

O ciclo de desenvolvimento de um projeto com FPGAs exige o uso de ferramentas EDA (*Eletronic Design Automation*) apropriadas, devido à grande complexidade envolvida nesses dispositivos. A Figura 3.3 mostra as diferentes etapas desse processo. Na maior parte das vezes o projetista é responsável pela etapa de especificação e entrada do projeto, seja por meio de HDLs (*Hardware Description Languages*), seja por meio de desenhos esquemáticos. As outras etapas geralmente são automatizadas e otimizadas nas ferramentas de EDA, permitindo que o projetista trabalhe em um nível mais alto de abstração [10].

A etapa de entrada do projeto é geralmente feita de duas maneiras: (a) utilizando uma linguagem de descrição de *hardware*, ou (b) utilizando uma ferramenta de projeto esquemático. As linguagens de descrição de *hardware* mais comuns são a *Verilog* e a *VHDL*, sendo esta última um padrão IEEE para descrição de *hardware*. Ambas são linguagens de alto nível, permitindo uma maior abstração ao projetista [10].

Devido ao grau de abstração do projeto de alto nível, em que os projetistas se preocupam mais com o funcionamento do sistema que com seu desempenho, essa fase de entrada do projeto não é otimizada, sendo necessária a utilização de técnicas complexas para otimizar os circuitos gerados, minimizando a utilização de recursos, por meio de redução das equações lógicas. Essa fase é chamada *mapeamento*. A fase seguinte é a de *posicionamento*, em que um aplicativo específico seleciona o local em que cada bloco lógico da implementação será colocado, de acordo

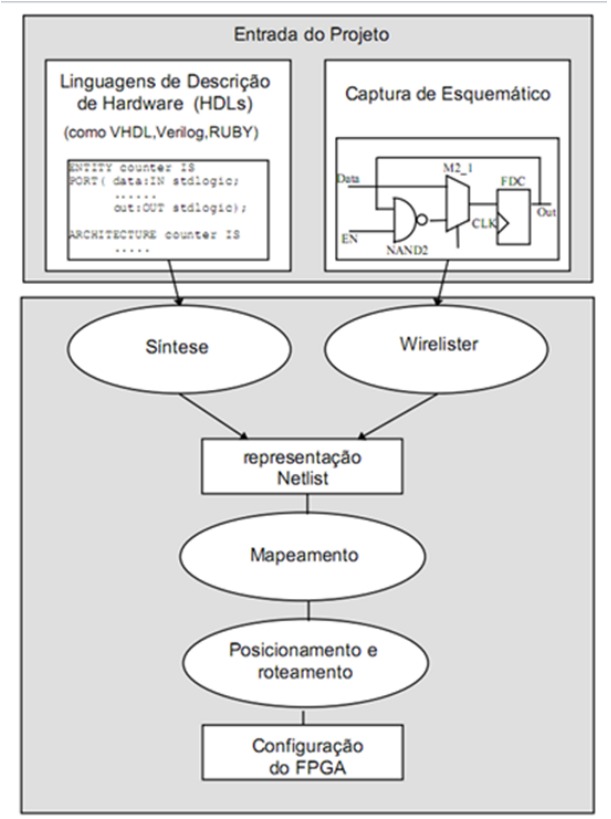

Figura 3.3. Etapas de um projeto com FPGAs [32]

com o modelo e arquitetura do FPGA. Após o posicionamento, começa a fase de *roteamento*, em que um algoritmo utiliza a arquitetura de interconexão para ligar os blocos lógicos, sendo necessário assegurar que todas as conexões desejadas sejam feitas, e ao mesmo tempo tentar minimizar os atrasos do circuito, maximizando sua velocidade de processamento. A última etapa é a geração do arquivo de configuração do FPGA, ou seja, os dados necessários para configurar o circuito desejado dentro do dispositivo reconfigurável [10].

Este trabalho foi desenvolvido, na maior parte, utilizando a linguagem Verilog em conjunto com uma ferramenta de desenho esquemático integrada à ferramenta EDA Quartus II 10.0, da Altera Corporation. O Quartus II permite o desenvolvimento de projetos por meio da descrição em linguagens de *hardware* e por meio de desenhos esquemáticos, automatizando os processos de síntese, mapeamento e roteamento.

3.5 Trabalhos Correlatos

Muitos trabalhos têm sido desenvolvidos usando FPGAs para acelerar tarefas de processamento de imagens, principalmente para aplicações em sistemas embarcados com requerimentos de tempo real. Em [35] é apresentada uma arquitetura biológico-inspirada para detecção de

movimento utilizando fluxo ótico. A abordagem mencionada pode ser implementada tanto em dispositivos FPGA como em ASIC e alcança uma taxa de processamento de 177 fps (198×96 pixeis). Também em [36], é apresentada uma implementação em FPGA de um sensor de estimação de movimento que utiliza um algoritmo de fluxo ótico alcançando 15 fps para imagens de 640×480 pixeis.

Um sistema de visão artificial aplicado ao controle de um sistema mecânico é apresentado em [37]. Este utiliza um filtro de correlação visando determinar o centro de massa de um objeto cada 4,51 ms para imagens pequenas (256×256). Em [38] é apresentado um sistema para filtragem e estimação de movimento em sequências de imagens usando SAD (*sum of absolute differences*). Este utiliza uma arquitetura sistólica para estimar movimento cada 5 ms em imagens de 640×480.

A implementação de um sistema robusto de controle visual em tempo real com um co-processador de imagens baseado em FPGAs para um pêndulo invertido é apresentado em [39]. Aqui, a posição do pêndulo é medida com um sistema de visão cujos algoritmos de processamento de imagens são implementados em *pipeline* assim como implementado em um dispositivo FPGA para cumprir requisitos de tempo real. Além disso, os autores utilizam o algoritmo de realce de bordas para determinar o centro de massa do objeto detectado e pode alcançar uma saída de 580 fps com imagens de 128×101 pixeis.

Em [40] e [41] pode-se encontrar um sistema de processamento de imagens baseado em FPGAs para sistemas de vigilância. [41] apresenta uma implementação em FPGA para subtração do fundo em tempo real. A arquitetura implementada alcançou uma taxa de 32,8 fps com imagens de 1024×1024. Em [40] é implementada uma arquitetura *pipeline* para segmentação de objetos em movimento baseada em brilho, cores e análise de textura. A saída do sistema está em torno aos 25 fps, para resoluções de imagem de 720×576 pixeis.

Em [42] é apresentada uma implementação para acelerar processos de processamento de imagens usando dispositivos reconfiguráveis. Alguns dos algoritmos mais comuns de pre-processamento foram implementado com uma taxa de processamento alta. [43] usa reconfiguração dinâmica para reconhecimento de cor e cálculo de fluxo ótico em FPGA. Este alcança uma taxa de 30 fps para imagens de 160×120 pixeis.

Uma abordagem que utiliza *look-up tables* para evitar cálculos complexos foi proposta em [44], na qual foi implementada sobre um FPGA uma arquitetura para o processamento em tempo real de imagens catadióptricas. O trabalho apresenta uma uma boa taxa de saída, porém requer uma grande quantidade de blocos de memória para armazenar todas as distancias, as quais são calculadas *off-line*.

Na Tabela 3.2 se apresenta um resumo de alguns dos trabalhos citados nesta seção. Aqui se faz uma comparação entre eles, indicando a técnica utilizada assim como o desempenho em termos

Tabela 3.2. Comparação de desempenho dos trabalhos citados encima.

Ref.	Algoritmo principal	Tamanho das imagens	Quadros por segundo	Mega-pixeis por segundo	Ambiente de desenvolvimento
[38]	SAD	640×480	200	61,44	Virtex-E XCV2000E 66 MHz
[41]	Subtração do fundo	1024×1024	32	33,55	Spartan-3A XC3SD3400 66,5 MHz
[37]	Correlação	256×256	221	14,48	Vertex Pro 6000 66 MHz
[40]	Subtração do fundo	720×576	25	10,37	Virtex-4 LX 200
[39]	Realce de bordas	128×101	580	7,50	Cyclone EP1C12Q240C8 (66 MHz) DSP TMS320F240 (20MHz/16 bits)
[36]	Fluxo ótico	640×480	15	4,61	Virtex-4 FX 100 MHz
[35]	Fluxo ótico	128×96	177	2,17	Virtex 2000E BG560
[42]	Fluxo ótico	160×120	30	0,58	Virtex-E XCV2000E-8 70 MHz

da velocidade de processamento. Pode-se notar que o desempenho alcançado pelos sistemas que utilizam fluxo ótico como técnica base têm uma velocidade de processamento menor do que os demais sistemas aqui mostrados. Isto acontece porque os algoritmos de fluxo ótico são representam um grande consumo de recursos e são de alto custo computacional. Por outro lado, os algoritmos de subtração do fundo e SAD têm alcançado altas taxas de processamento, sendo esta uma das razões de se ter preferencia pelos algoritmos de subtração do fundo neste trabalho.

3.6 Conclusões do Capítulo

Neste capítulo foram descritas as plataformas utilizadas ao longo do tempo para o processamento de sinais e imagens. A necessidade de realizar estes processos com alto grau de de-

sempenho tem levado ao desenvolvimento de algoritmos e plataforma mais adequadas, a maioria delas características que permitem implementações em tempo real.

A implementação de algoritmos em paralelo (direitamente em *hardware*) faz com que seja cada vez mais possível acelerar processo para a filtragem, análise e processamento de imagens. Este fato coloca os FPGAs em uma posição privilegiada com relação a outras plataformas que somente permitem implementar algoritmos de forma sequencial.

Como mostrado na Tabela 3.2, os algoritmos de subtração do fundo têm melhor desempenho do que os algoritmos de fluxo ótico, já que estes últimos são de maior complexidade computacional. Portanto, para implementar um sistema de detecção de movimento de baixa complexidade são mais adequados os algoritmos de subtração do fundo.

4 Implementação do Sistema de Detecção de Movimento e Comunicação

Neste capítulo será descrita a metodologia utilizada para implementar o sistema de detecção de movimento, baseado no algoritmos de subtração do fundo, incluindo todas as suas etapas. Todos as arquiteturas foram desenvolvidas tendo em conta o ambiente de desenvolvimento descrito no anexo A

A arquitetura geral do sistema é mostrada na Figura 4.1. O sistema de captura foi fornecido pelo fabricante do kit DE2, o qual inclui: (a) o controlador e a configuração do sensor CMOS, (b) a conversão do sistema RAW (saída fornecida pela câmera descrita na secção A.3) para RGB e (c) a sincronização e o controlador do *display* LCD (utilizando o controlador da memória SDRAM). Neste trabalho foram desenvolvidas as arquiteturas para os blocos ressaltados na Figura 4.1, correspondentes à redução de cores, o armazenamento da imagem de referência e a arquitetura de detecção de movimento, assim como o módulo de comunicação RS-232 utilizando o processador embarcado Nios II da Altera.

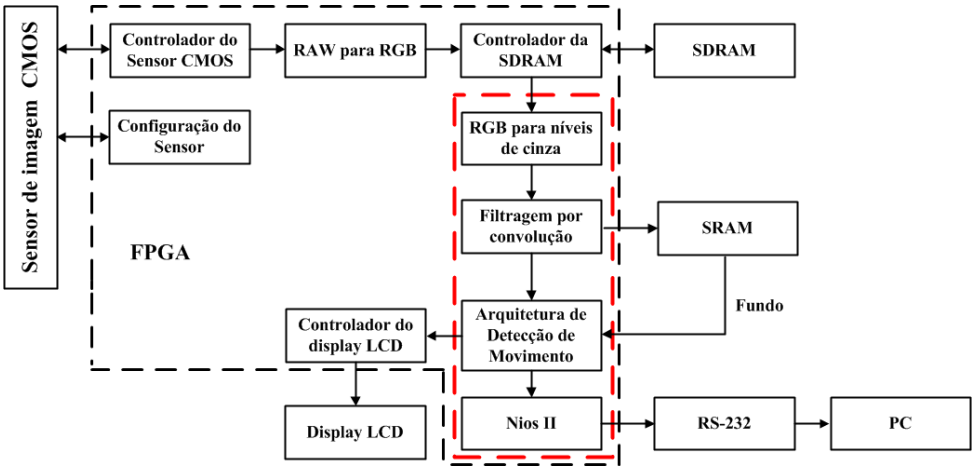

Figura 4.1. Arquitetura geral do sistema

4.1 Redução de Cores

A câmera fornece três canais de cores de 12 bits, porém o sistema de controle de captura implementado pelo fabricante prevê a utilização de 8 bits apenas. Desse modo, todo o processamento foi implementado com base na premissa de ter-se 8 bits por canal de cor. Devido às limitações de memória do kit de desenvolvimento, a imagem de fundo tem que ser convertida a uma imagem em níveis de cinza (8-bits por pixel), para depois ser armazenada na memória SRAM. Isto é devido que a memória SDRAM do kit DE2 está sendo utilizada para realizar a sincronização do sinal fornecido pela câmera com o sinal recebido pelo *display*.

Exitem muitos métodos para determinar o valor apropriado de nível de cinza, tratando-se da conversão de imagens coloridas. Neste trabalho, o bloco de transformação de cor calcula a média dos canais de cor (R,G,B), fornecendo assim para os blocos subsequentes um único pixel (Figura 4.2).

Neste trabalho, os cálculos para a redução de cor são feitos em um ciclo de relógio (Figura 4.2). Para atingir isto, a média aritmética é calculada multiplicando a soma dos três canais (R,G,B) por uma constante na borda de subida, e dividindo este produto entre uma potência de dois na borda de descida. Isto reduz o problema da divisão à utilização de um simples *shift register* (evitando assim a utilização de um circuito divisor). A soma dos três canais foi multiplicada por 170 e dividida por 512.

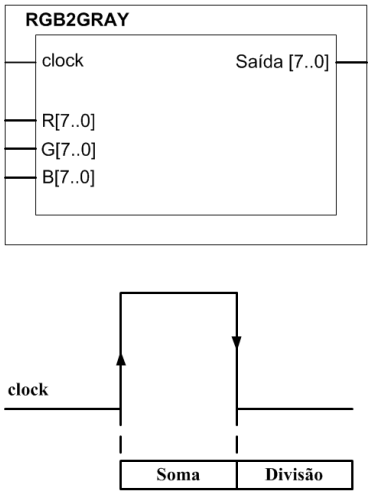

Figura 4.2. Bloco redutor de cores RGB para cinza.

4.2 Filtragem por Convolução

Depois da redução de cores a imagem tem que ser filtrada para eliminar o ruído gerado pelo sistema de captura. Como já mencionado na secção 2.4.1, na convolução, para o processamento de uma vizinhança necessitamos ter todos os dados da vizinhança disponíveis. Neste trabalho, o tamanho de vizinhança considerado foi de 3×3 pixeis. Devido que a câmera envia os dados da imagem de acordo com a sua captura, os pixeis formam um fluxo que tem como origem o canto superior esquerdo da imagem e termina no canto inferior direito.

4.2.1 Carregamento da Vizinhança

A arquitetura básica para a operação de janelamento é mostrada na Figura 4.3, a qual representa um arranjo sistólico. Neste caso, cada pixel é representado por um quadro. O processo descrito na Figura 4.3 é similar ao processo da Figura 2.7, porém, em lugar do deslocamento da máscara sobre a imagem, é a imagem que é deslocada através de um *pipeline*. A estrutura mostrada na parte inferior da Figura 4.3 é um *shift register*, onde por cada ciclo de relógio, os pixeis são deslocados na mesma direção, descartando o mais antigo, e carregando o mais novo. Cada linha do *shift register* é equivalente exatamente a uma linha da imagem, e os elementos em amarelo são memórias locais para armazenamento temporario.

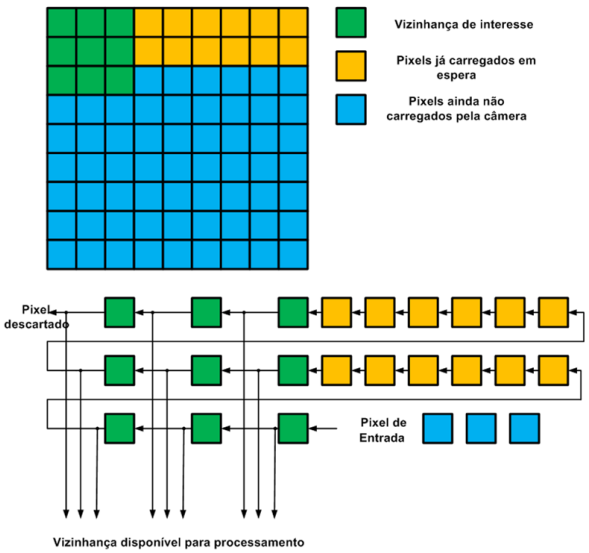

Figura 4.3. Arquitetura para operação de janelamento [5], [10].

Tendo em conta que o *pipeline* deve estar completamente cheio para desenvolver a operação de convolução, este processo tem uma latência inicial, dada pela equação 4.1 [10], para uma

imagem de $M \times N$ e uma máscara de $L \times P$.

$$Latência = (P-1)M + L. \tag{4.1}$$

Neste trabalho, imagens de 800×480 estão sendo processadas utilizando uma máscara de 3×3. Portanto, a latência inicial do processo de convolução é de 1603 ciclos de relógio. Este tempo é necessário para carregar a primeira vizinhança, depois disto, o sistema fornece um pixel por cada ciclo de relógio.

O *display* utilizado neste projeto possui largura visível de 800 pixeis, então, conforme a equação 4.1, o comprimento do registrador de deslocamento, para uma máscara 3×3, seria de 1603 elementos. Porém, tal registrador não permitiria o acesso aos pixeis desejados da vizinhança, então a arquitetura foi dividida em três partes (Figura 4.4), dois registradores de deslocamento (na cor amarela) e um bloco com a função de controlar o carregamento e disponibilização dos pixeis (na cor verde).

Outro fato importante é sobre os parâmetros de configuração do registrador. A princípio, seu comprimento seria de 797 pixeis (800 do total subtraindo 3 da máscara), porém, devido às restrições na ferramenta de desenvolvimento (Quartus II da Altera Corp.), a implementação necessita ser feita utilizando uma combinação de blocos RAM e elementos lógicos. Assim, para obter os 800 pixeis de uma linha da imagem, cada linha de blocos verdes toma 32 pixeis (elementos lógicos) e cada registrador (representados pelos blocos amarelos) toma 768 pixeis (Blocos RAM) como mostrado na Figura 4.4.

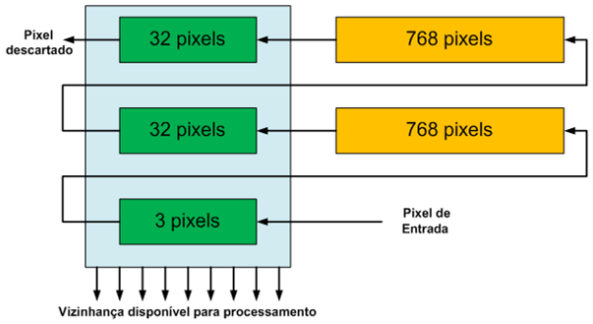

Figura 4.4. Divisão da arquitetura em três blocos [5], [10].

4.2.2 Convolução

A implementação da arquitetura para convolução consiste na descrição em VHDL do processo mostrado na Figura 4.5. A disponibilização da vizinhança, que foi descrita na secção 4.2.1, é representada pela parte (a) da Figura 4.5, faltando a implementação dos multiplicadores e o

somador, representados pelas partes (b) e (c) respectivamente.

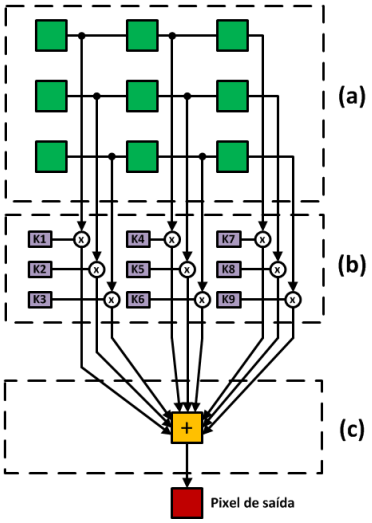

Figura 4.5. Arquitetura de convolução [5], [10].

As nove multiplicações (parte (b) da Figura 4.5) são realizadas na borda de subida do relógio, e a soma (parte (c) da Figura 4.5) é sensível à borda de descida. Um bloco contendo apenas constantes foi disponibilizado como máscara de convolução; neste caso foi utilizado um filtro de média, representado na Figura 2.8 (a). A figura 4.6 mostra o diagrama de blocos implementado no Quartus II, ressaltando as partes (a), (b) e (c) da Figura 4.5.

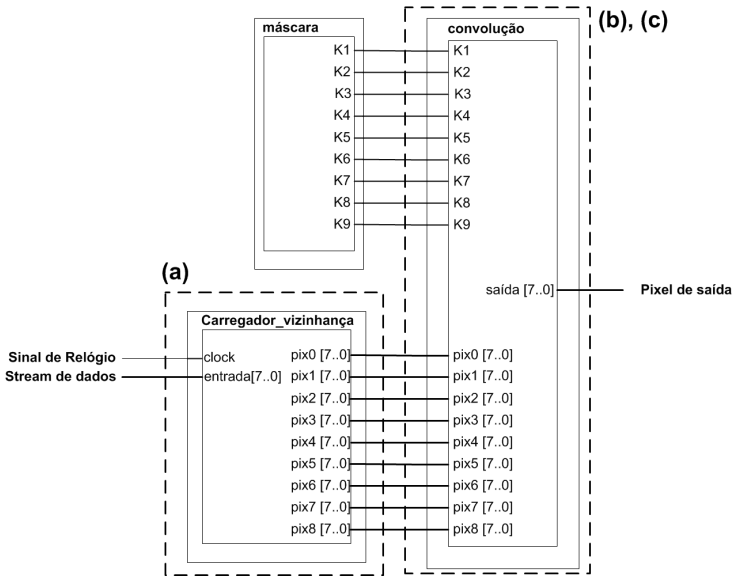

Figura 4.6. Diagrama de blocos implementado no Quartus II para a arquitetura de convolução.

4.3 Armazenamento do Fundo

As imagens capturadas pela câmera são de tamanho 800×480 com uma profundidade de 8 bits por pixel, fornecendo um pixel por ciclo de relógio. A imagem de referência é armazenada em uma memória SRAM externa do *kit* de desenvolvimento DE2. É uma memória *ISSI* IS61LV25616AL de alta velocidade com 512 KBytes organizados em 262.144 palavras de 16 bits. Devido a esta organização, os dados têm que ser armazenados colocando dois pixeis em cada posição de memória. Assim, o sistema armazena um pixel por ciclo de relógio, porém a posição de memória é atualizada cada dois ciclos de relógio como mostrado na Figura 4.7.

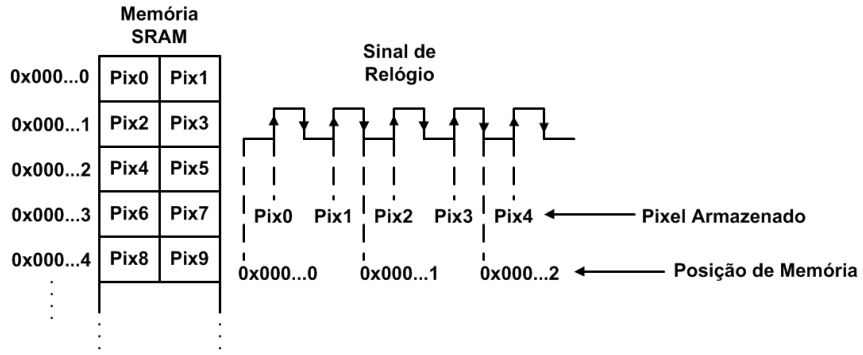

Figura 4.7. Sistema de armazenamento do fundo

O processo de armazenamento começa por meio de uma ordem de um teclado ligado ao *kit* de desenvolvimento DE2 e termina no momento em que tem sido armazenado o quadro completo, começando assim a leitura dos dados armazenados na memória. A leitura é feita atualizando a posição de memória cada dois ciclos de relógio, no primeiro ciclo são lidos os primeiros oito bits da posição atual, no segundo ciclo são lidos os 8 bits restantes, e assim sucessivamente.

4.4 Arquitetura de Detecção de Movimento

Nesta arquitetura (vide Figura 4.8), o algoritmo de subtração do fundo é implementado conforme é feita a leitura dos dados da memória, lembrando que tanto o quadro atual quanto a imagem de referência têm sido filtradas antes do processo de armazenamento como descrito na secção 4.2.2. Enquanto é feita a leitura dos dados da memória, estes são subtraídos, pixel a pixel, do quadro atual, sendo feita a subtração em um ciclo de relógio. Depois da subtração, o valor absoluto do resultado é calculado para realizar o processo de segmentação com o fim de obter uma imagem binária, onde os pixeis marcados com "1" correspondem a pixeis em movimento e os pixeis marcados com "0" aos pixeis que pertencem ao fundo. Com o fim de eliminar o ruído gerado pelo processo de segmentação é realizado um processo de filtragem morfológica, neste

caso, utilizando a operação de erosão descrita na secção 2.4.3. Por último, é realizado o cálculo do centro de gravidade do objeto, sendo este marcado na imagem e enviado para um computador via interface RS-232 controlado pelo processador Nios II da Altera (secção 4.5).

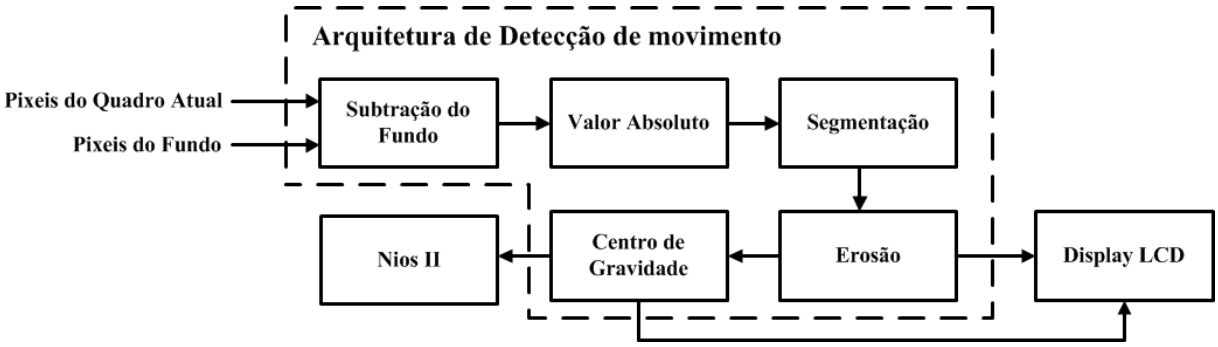

Figura 4.8. Arquitetura de detecção de movimento

Na continuação serão descritos cada um dos processos desta arquitetura.

4.4.1 Subtração do Fundo e Segmentação

Uma vez são lidos os dados da memória, estes entram no bloco de subtração junto com os dados do quadro atual, pixel a pixel, um por cada ciclo do relógio. A operação de subtração é realizada também em um ciclo de relógio. Depois, o valor absoluto é calculado para cada pixel na saída da subtração, utilizando o *MegaWizard Plug-In Manager* da Altera Figura 4.9. A saída do bloco do valor absoluto é recebida pelo bloco de segmentação, que realiza a operação de binarização. Devido que neste trabalho têm-se considerado objetos que apresentam alto contraste com relação ao fundo foi escolhido o algoritmo de limiarização para a segmentação da imagem, por ser de simples implementação e baixo custo computacional. A descrição do algoritmo para a limiarização em Verilog é mostrado na Listagem 1.

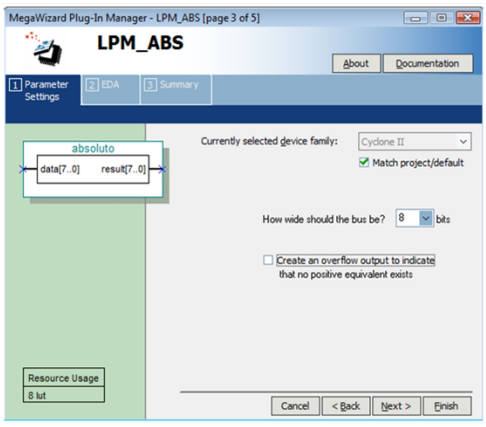

Figura 4.9. Tela do *MegaWizard Plug-In Manager* para configuração do bloco de valor absoluto

Listagem 1 Descrição do algoritmo de limiarização em verilog.

```
input CLK;

input [7:0] Motion;
input [7:0] Lim;

input [9:0] Pos_x;
input [9:0] Pos_y;

output [7:0] MotionSeg;
reg [7:0] MotionLim;

assign MotionSeg = MotionLim;

always @(posedge CLK)
begin
  if(Motion >= 8'd64 && (Pos_x >= 0 && Pos_x <= 790)&&(Pos_y >= 0 && Pos_y <= 479))
    MotionLim <= 255;
  else
    MotionLim <= 0;
end
endmodule
```

A arquitetura para a limiarização tem cinco entradas e uma saída, nas entradas encontram-se o sinal de relógio (`CLK`), as posições x e y do pixel na imagem (`Pos_x` e `Pos_y` respectivamente), o limiar (`Lim`) e a imagem de entrada (`Motion`). A saída é a imagem binarizada (`MotionSeg`). As entradas de posição (`Pos_x` e `Pos_y`) são utilizadas como referência para realizar a operação de binarização somente no quadro ativo da imagem.

4.4.2 Processo de Erosão

As operações morfológicas básicas, erosão e dilatação, também são implementadas utilizando como base o bloco de disponibilização de vizinhança. A Figura 2.16 mostra como são calculados os valores de erosão. O bloco de erosão recebe os nove pixeis da vizinhança, f_i na figura, e o elemento estruturante, k_i. O valor de e_i é calculado em um primeiro passo, sendo utilizado no segundo passo, que calcula finalmente o resultado da erosão, segundo as equações da Figura 2.16. A operação é implementada em apenas um ciclo de relógio, sendo o primeiro passo calculado na borda de subida e o segundo na borda de descida. A Figura 4.10 mostra a arquitetura completa para uma operação de erosão.

4.4.3 Cálculo do Centro de Gravidade

Neste trabalho foi desenvolvida a detecção de movimento somente para cenas com um objeto. Assim, o centro de gravidade foi calculado utilizando as equações 4.2, 4.3 e 4.4, para uma imagem

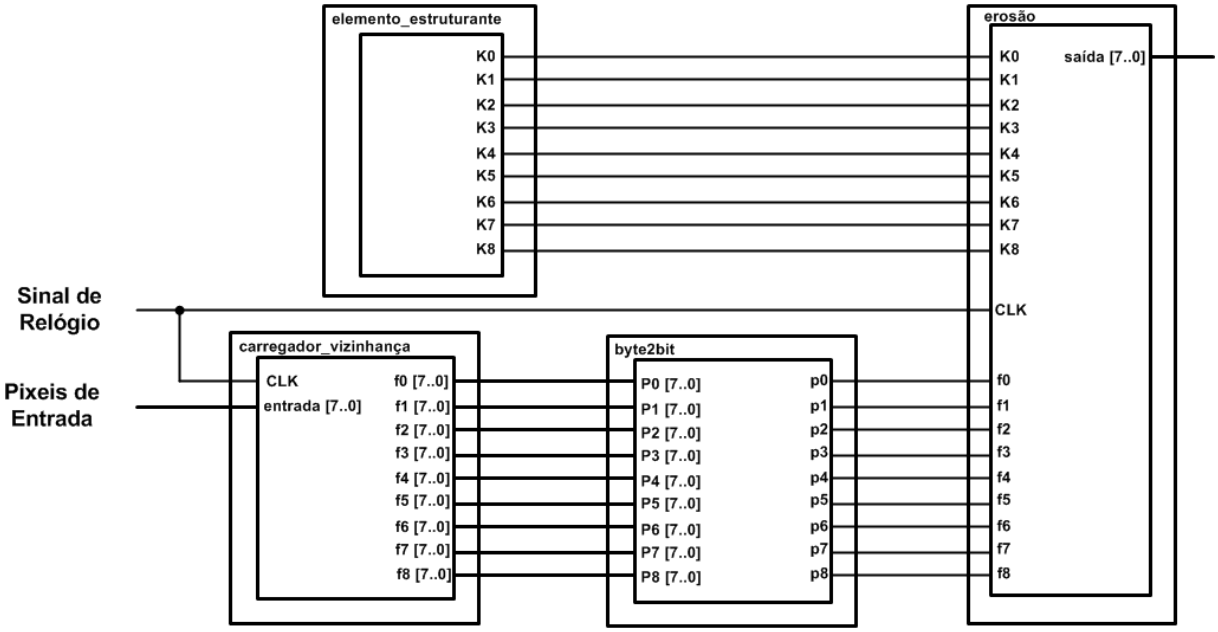

Figura 4.10. Arquitetura para erosão

de $M \times N$.

$$A = \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} B(i,j), \tag{4.2}$$

$$C_x = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} i.B(i,j)}{A}, \tag{4.3}$$

$$C_y = \frac{\sum_{i=0}^{M-1} \sum_{j=0}^{N-1} j.B(i,j)}{A}. \tag{4.4}$$

onde $B(i,j)$ corresponde à imagem de movimento binarizada, A corresponde à área do objeto, e C_x e C_y são os valores do centro de gravidade em x e y respectivamente. Neste caso está-se considerando uma imagem binária que representa o objeto em movimento, pelo que o valor $B(i,j)$ vai ser "1" se na posição (i,j) tiver um pixel em movimento e "0" se o pixel nessa posição for parte do fundo.

A imagem de movimento $B(i,j)$ está entrando pixel a pixel, a imagem tem que ser percorrida completamente para realizar o cálculo do centro de gravidade. Assim, os valores de C_x e C_y estarão disponíveis somente no final de cada quadro. Na Listagem 2 mostra-se a descrição em Verilog da arquitetura para o cálculo do centro de gravidade.

A saída deste bloco é enviada para a arquitetura de comunicação descrita na secção 4.5.

Listagem 2 Descrição do algoritmo em verilog para o cálculo do centro de gravidade.

```
1: always @(posedge CLK)
2: begin
3:   if (Pos_x == 0 && Pos_y == 0)
4:   begin
5:     Area <= 0;
6:     Sum_x <= 0;
7:     Sum_y <= 0;
8:   end
9:   else if (Motion == 255)
10:  begin
11:    Area <= Area + 1;
12:    Sum_x <= Sum_x + Pos_x;
13:    Sum_y <= Sum_y + Pos_y;
14:  end
15:  else if(Pos_x == 799 && Pos_y == 479)
16:  begin
17:    Center_x = Sum_x / Area;
18:    Center_y = Sum_y / Area;
19:  end
20: end
```

4.5 Comunicação RS-232

O envio dos dados do centro para o computador foi feito utilizando uma interface RS-232. Este tipo de comunicação foi escolhida já que a quantidade de dados que têm que ser enviados para o computador não é tão grande como para comprometer o desempenho do sistema, além disso, tendo em conta que os dados do centro estarão disponíveis somente no final de cada quadro.

O algoritmo para a comunicação dos dados foi implementado no processador Nios II da Altera, descrito na secção A.6. Para o desenvolvimento deste sistema, têm que ser seguidos os seguintes processos:

- Criação do processador na ferramenta *SoPC Builder* do Quartus II
- Instanciar o processador na arquitetura de detecção
- Programação do controlador de comunicação RS-232 em *software*.

4.5.1 Criação do Processador (*SoPC Builder*)

O *SoPC Builder* (Figura 4.11) é uma ferramenta integrada no Quartus II da Altera desenvolvida para la criação de processadores embarcados. Aqui todos los elementos e periféricos que vão ser utilizados pelo processador Nios II têm que ser descritos. Neste trabalho, o processador

está dedicado à comunicação dos dados via interface RS-232, recebendo os dados da arquitetura de detecção.

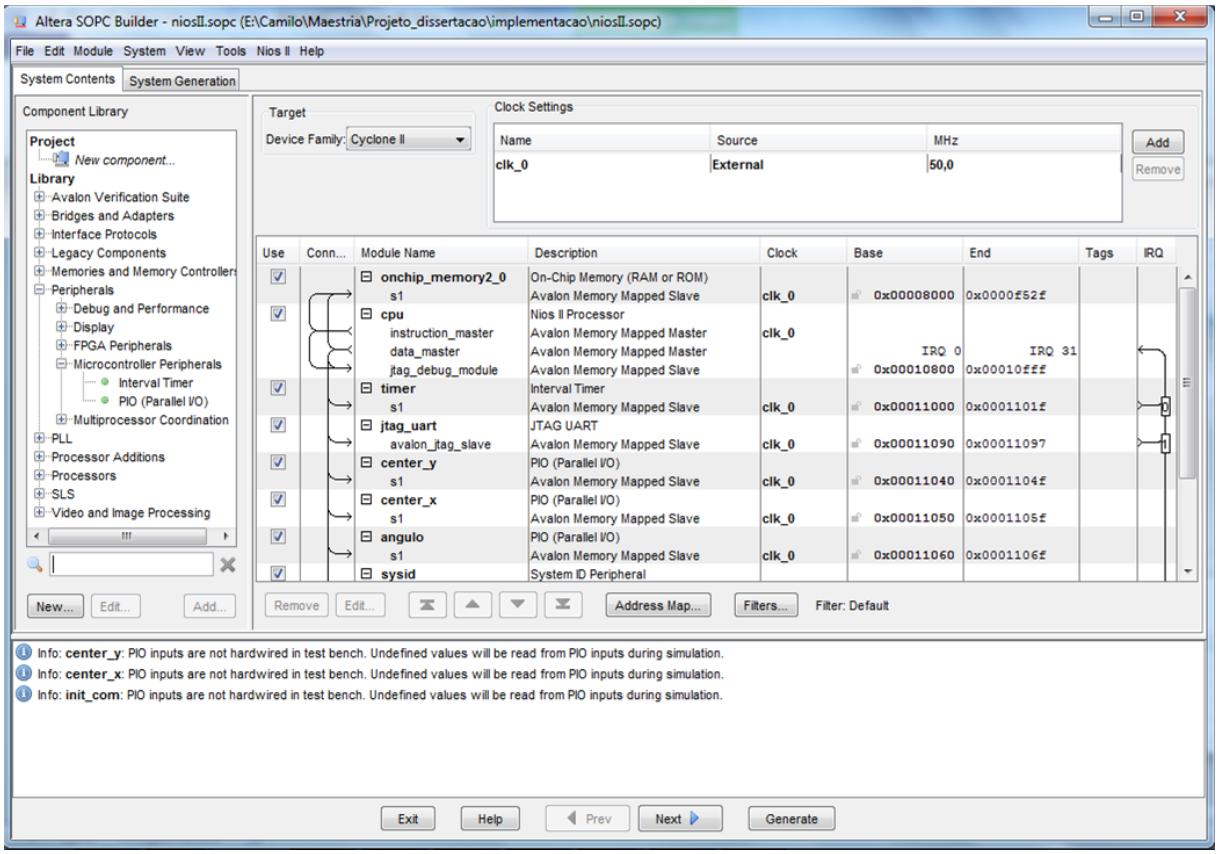

Figura 4.11. Ferramenta *SoPC Builder* da Altera Corp.

4.5.2 Instância do Processador na Arquitetura *Hardware*

Uma vez que o processador é criado, o mesmo tem que ser instanciado na arquitetura. Este processo pode ser realizado de duas formas: (a) instanciando mediante código (Verilog ou VHDL como mostrado na Listagem 3) e (b) no diagrama de blocos do programa Quartus II (Figura 4.12).

Listagem 3 Instancia do processador NiosII em verilog.

```
1: niosII_system c0 (
2:             .iCLK(CLOCK_50),
3:             .center_x(centrox),
4:             .center_y(centroy),
5:             .angulo(LEDR),
6:             .uart_rx(UART_RXD),
7:             .uart_tx(UART_TXD)
8:             );
```

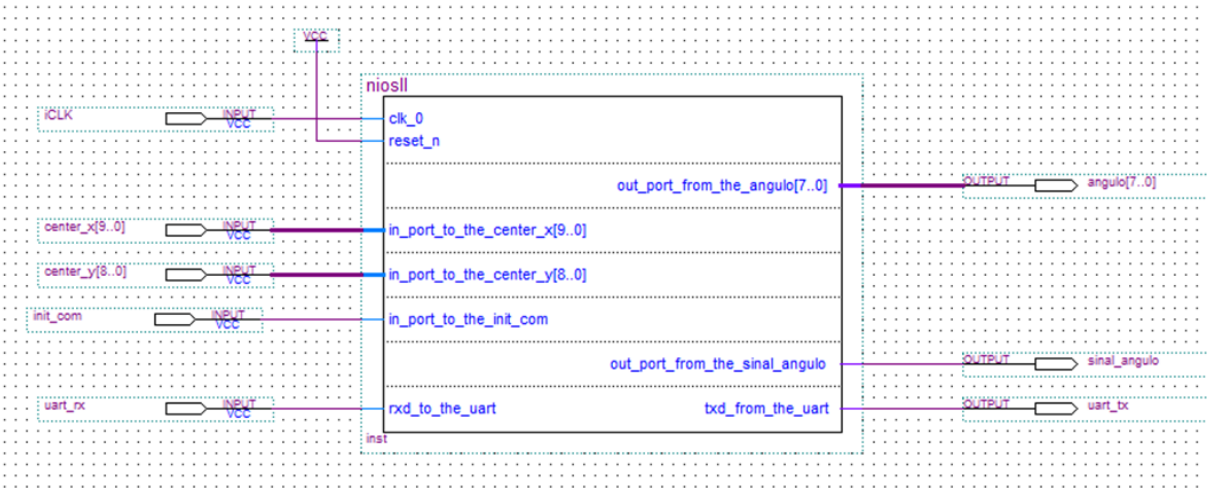

Figura 4.12. Instancia do processador Nios II no Quartus II da Altera

4.5.3 Programação do Controlador RS-232

Para a programação do controlador RS-232 foi utilizada ferramenta *Nios II SBT for eclipse*. A ferramenta compila o programa feito em linguagem C/C++, que é abaixado ao processador que se encontra programado no FPGA. Na Figura 4.13 mostra-se o ambiente de desenvolvimento da ferramenta.

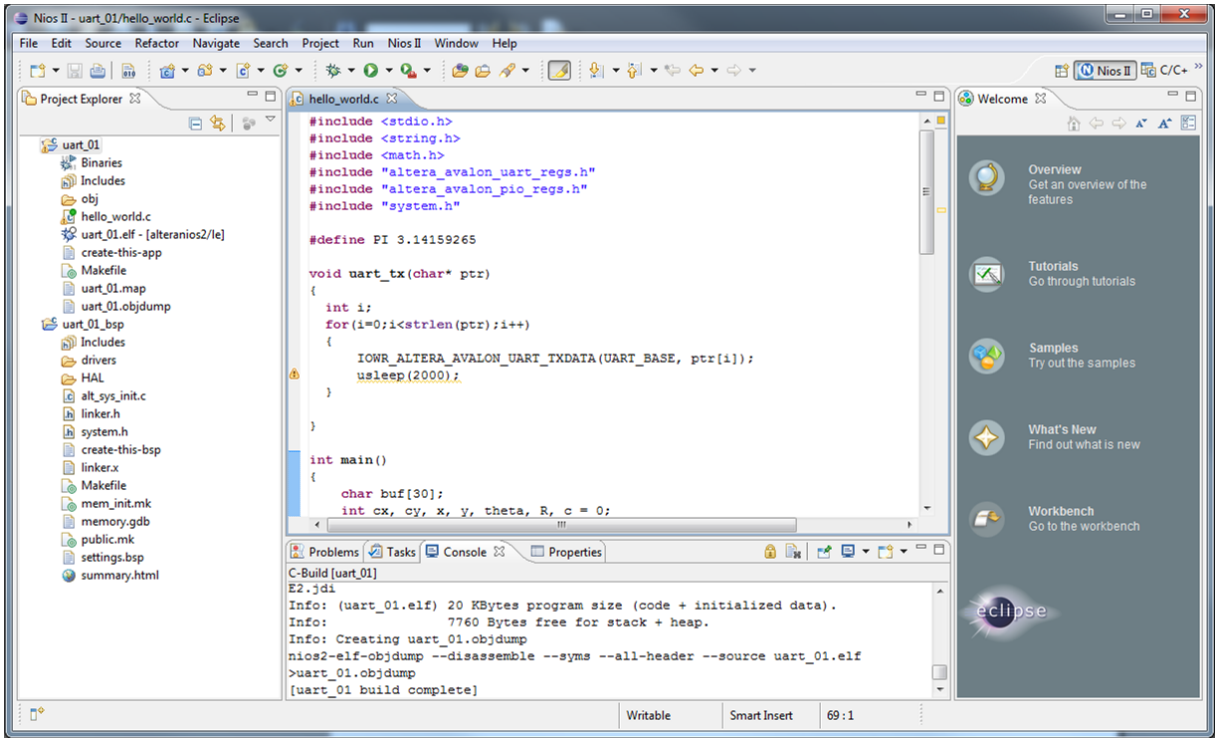

Figura 4.13. Ambiente de desenvolvimento *Nios II SBT for eclipse*

A ferramenta gera bibliotecas para o controle dos periféricos descritos no *SoPC Builder*,

neste caso, foram utilizadas as funções destas bibliotecas para o controle da interface RS-232. O processador recebe os dados do centro, C_x e C_y, mediante as funções citadas embaixo. Estas funções pertencem à biblioteca `altera_avalon_pio_regs.h`

```
cx = IORD_ALTERA_AVALON_PIO_DATA(CENTER_X_BASE)
```

```
cy = IORD_ALTERA_AVALON_PIO_DATA(CENTER_Y_BASE)
```

Depois da leitura dos dados da arquitetura, os dados são enviados via porta serial com a função da biblioteca `altera_avalon_uart_regs.h` citada embaixo.

```
IOWR_ALTERA_AVALON_UART_TXDATA
```

Os dados são recebidos por um hyperterminal ou pelo Matlab, e podem ser armazenados, com fim de ser utilizados para aplicações como o controle de robôs ou outros processos.

Além da comunicação, neste trabalho, o processador embarcado Nios II realiza operações de ponto flutuante, neste caso para realizar uma transformação de coordenadas retangulares para coordenadas polares, disponibilizando para o sistema externo (PC, robô, etc.) informação do ângulo e distancia nos quais encontra-se o objeto detectado, tendo em conta o sistema de referência mostrado na Figura 4.14.

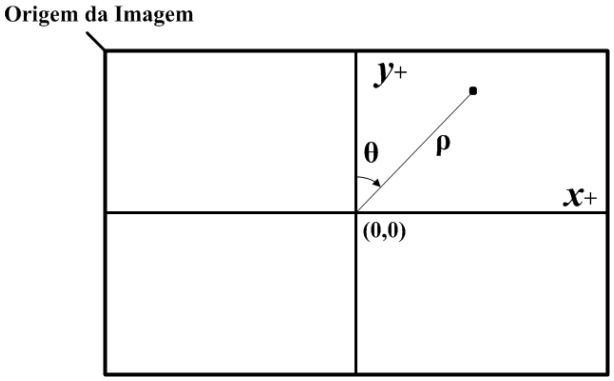

Figura 4.14. Sistema de referência utilizado para a transformação de coordenadas

Estas operações foram desenhadas para um sistema de visão omnidirecional (Figura 4.15) considerando a câmera no centro de coordenadas $(0, 0)$, permitindo assim conhecer o ângulo e a distância desde o robô até o objeto.

4.6 Conclusões do Capítulo

Em geral, o *kit* de desenvolvimento tem recursos limitados, já que a maior das memórias (SDRAM) está sendo utilizada para realizar a sincronização entre a câmera e o *display*. Este foi o primeiro problema na implementação da arquitetura de detecção de movimento, já que é

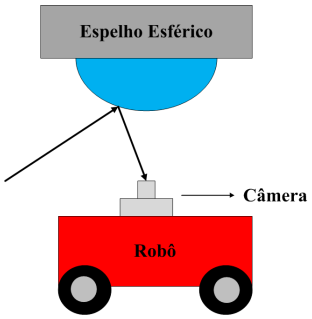

Figura 4.15. Sistema de visão omnidirecional para o qual foi projetado o sistema.

necessário armazenar a imagem de fundo ou referência para o correto funcionamento do algoritmo. Para a resolução deste problema foi implementado o sistema de armazenamento do fundo na memória externa SRAM (512 KBytes) do *kit* DE2. A imagem teve que ser armazenada em níveis de cinza (8 bits por pixel). Aqui, cada posição de memória tem 16 bits, pelo que tiveram que ser armazenados dois pixeis da imagem de fundo por cada posição de memória. Da mesma forma têm que ser lidos (dois pixeis por posição de memória) sem perder a sincronização entre a imagem da câmera (quadro atual), a imagem de referência (lida desde a memória), o sistema de comunicação e o *display*, sendo este um dos maiores desafios da implementação.

Neste capítulo mostrou-se como foram implementadas as arquiteturas de processamento de imagens e detecção de movimento em uma plataforma *hardware*, neste caso, um FPGA da família Cyclone II, aproveitando que estes dispositivos permitem realizar operações em paralelo, aumentando o desempenho do sistema, como mostrado no caso da convolução, onde são feitas nove multiplicações em paralelo.

O processador embarcado Nios II permite implementar facilmente processos de comunicação entre a placa DE2 e um dispositivo externo, assim como também permite que sejam implementadas algumas operações de ponto flutuante, cuja implementação seria mais complexa em *hardware*.

5 Testes de Implementação e Resultados

Neste capítulo são apresentados os testes realizados para cada uma das arquiteturas, os resultados de síntese assim como os resultados gerais do sistema de detecção. Adicionalmente, é realizada uma comparação entre o algoritmo de detecção de movimento por subtração do fundo implementado em software e a arquitetura apresentada neste trabalho, mostrando que a arquitetura implementada em hardware é 32 vezes mais rápida do que o algoritmo em software

.

5.1 Captura das Imagens

O sistema implementado neste projeto utilizou alguns blocos funcionais já prontos, e outros desenvolvidos totalmente durante o trabalho. A Terasic Inc. disponibiliza no seu site os blocos com as funcionalidades de controle da câmera e do *display* utilizados, onde o código-fonte também é fornecido.

A Figura 5.1 mostra o projeto do sistema completo utilizado, com destaque para os blocos de convolução, detecção de movimento e o processador Nios II, denominados *convolution*, *Motion_Detect* e *niosII_system*, respectivamente (estes nomes foram os utilizados no código-fonte). Além disso, o bloco *touch_tcon* (responsável pela sincronização do *display*) foi modificado, nele são feitos os processos de redução de cores e armazenamento do fundo, além de continuar sendo responsável pela sua tarefa original. Neste trabalho foram desenvolvidas as arquiteturas de filtragem, detecção de movimento, o processador embarcado, um bloco para interface externa via teclado e foram feitas modificações no bloco *touch_tcon*. Todos os outros blocos foram disponibilizados pelo fabricante.

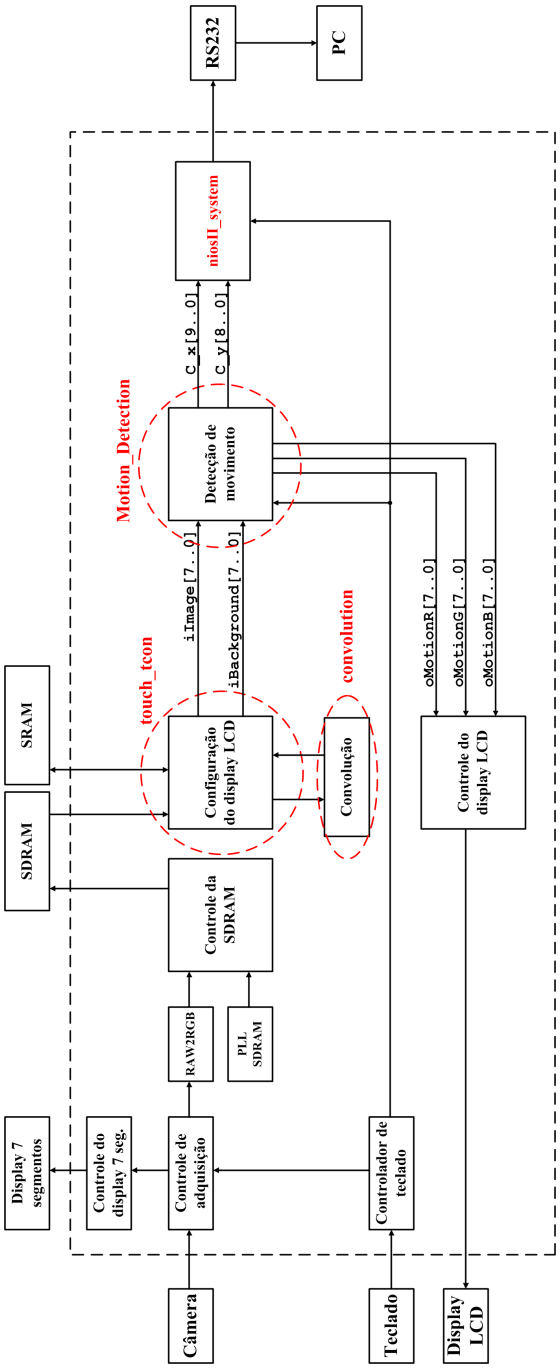

Figura 5.1. Diagrama de blocos do sistema implementado.

Os blocos de controle realizam operações como a configuração dos registradores da câmera e do *display* responsáveis por fixar os parâmetros de tamanho da imagem, taxa de aquisição e tempo de exposição. Há também um bloco responsável por transformar os dados do sensor da câmera em valores RGB.

O bloco de *detecção de movimento* foi inserido no final da arquitetura, antes de enviar os dados para o *display*, recebendo como entrada o sinal de relógio, a posição do pixel que está entrando na arquitetura, o limiar para a segmentação e as duas imagens necessárias para a detecção (`Background[7..0]` e `Image[7..0]` no diagrama RTL da Figura 5.1), que correspondem as imagens de fundo e do quadro atual respectivamente, fornecendo em sua saída a imagem de movimento, representada pelos canais `oMotionR[7..0]`, `oMotionG[7..0]` e `oMotionB[7..0]`.

A imagem de movimento é uma imagem binaria, pelo que seria necessário um canal só, porém, para uma melhor visualização no *display*, foram enviados os três canais para o *display*, já que a região em movimento vai ser representada em vermelho e o centro de gravidade em amarelo. O bloco de detecção de movimento também disponibiliza na saída os dados do centro de gravidade do objeto, que são recebidos pelo processador Nios II, conforme pode ser visto na Figura 5.2. A entrada `init_com` do processador Nios II controla o inicio da comunicação com o computador, esta ordem é enviada por um teclado externo à placa.

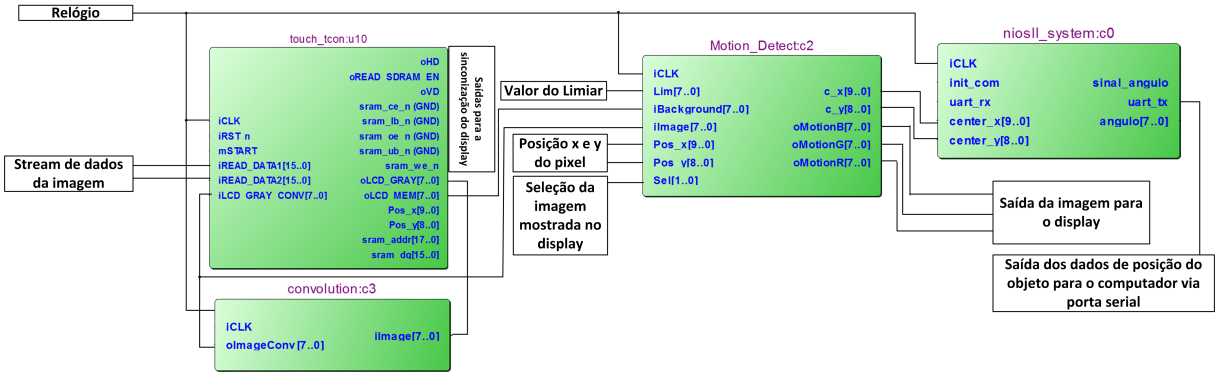

Figura 5.2. Blocos do sistema que foram criados neste projeto com as suas entradas e saídas.

A arquitetura de captura fornecida pelo fabricante, sem incluir o que foi feito neste trabalho, apresentou os resultados de síntese mostrados na Tabela 5.1. Considerando os blocos implementados e as modificações feitas neste trabalho, obtêm-se os resultados de síntese mostrados na Tabela 5.2, sendo a convolução a causa principal do aumento nos multiplicadores. Por outro lado, pode-se perceber um aumento na quantidade de bits de memória. Este é devido, principalmente, ao processador Nios II, já que este utiliza a memória *on-chip* como memória de programa. Outro fato muito importante para ressaltar é que a frequência máxima de operação é diminuída consideravelmente pelos acessos à memória.

Tabela 5.1. Resultados de síntese do sistema de captura

Elementos Lógicos	1.755 (5%)
Bits de Memória	57.400 (12%)
Elementos multiplicadores	0
PLLs	1 (25%)
Frequência de operação máxima	50,00 MHz (período = 20,00 ns)

Tabela 5.2. Resultados de síntese do sistema completo de detecção de movimento

Elementos Lógicos	9.221 (28%)
Bits de Memória	375.608 (78%)
Elementos multiplicadores	4 (6%)
PLLs	1 (25%)
Frequência de operação máxima (incluindo os acessos à memória)	23,03 MHz (período = 43,43 ns)

5.2 Redução de Cores

Na secção 4.1 foi descrita a arquitetura implementada para realizar a transformação de cores da imagem antes de ser filtrada. O bloco de redução de cores foi implementado dentro da arquitetura *touch_tcon* mostrada na Figura 5.1. Os resultados de síntese obtidos pela arquitetura foram os mostrados na Tabela 5.3

Este bloco é de baixo consumo de recursos, tendo em conta que, para isto foi desenvolvida a operação apresentada na secção 4.1. Isto reduz o problema da divisão à utilização de um simples *shift register* (evitando assim a utilização de um circuito divisor).

Tabela 5.3. Resultados de síntese do sistema de redução de cores

Elementos Lógicos	498 (1%)
Bits de Memória	384 (1%)
Elementos multiplicadores	0

Tabela 5.4. Resultados de síntese do sistema de filtragem por convolução

Elementos Lógicos	1.853 (5%)
Bits de Memória	16.256 (3%)
Elementos multiplicadores	4 (6%)

5.3 Convolução

Como já mencionado, neste trabalho, a filtragem por convolução é implementada a fim de diminuir o ruido que é gerado no processo de captura. A arquitetura implementada foi descrita na secção 4.2.2. A figura 5.3 mostra o resultado do processo de convolução obtido com a arquitetura implementada.

Figura 5.3. Resultado do processo de convolução. (a) Imagem original. (b) Imagem filtrada com um filtro de média.

Pode-se observar que a imagem filtrada tornou-se um tanto borrada, pela atenuação das características de alta frequência, eliminando ruido que pode ser gerado no processo de captura. O filtro de convolução apresentado neste trabalho utiliza uma máscara de 3×3 correspondente a um filtro de média. Os resultados de síntese são apresentados na tabela 5.4.

Segundo a equação 4.1, a latência do sistema de convolução é de 1603 ciclos de relógio para uma imagem de 800×480 com uma máscara de convolução de 3×3. Tendo em conta a frequência máxima de trabalho do sistema que, segundo os resultados de síntese mostrados na tabela 5.2, é de 23,03 MHz (um período de 43,43 ns), a latência do processo de convolução é de 69,62 μs. Após este tempo, a arquitetura de convolução fornece um pixel por cada ciclo de relógio.

5.4 Arquitetura de Detecção de Movimento

A arquitetura implementada para a detecção de movimento foi descrita na secção 4.4. Lembrando que a arquitetura se divide em diferentes etapas: (a) subtração entre a imagem atual e o fundo ou imagem de referência, (b) a segmentação da imagem resultante, (c) o processo de

filtragem morfológica utilizando erosão e por último o cálculo do centro de gravidade.

Na Figura 5.4 mostram-se os resultados de cada uma das etapas, sendo 5.4(a) o fundo ou imagem de referência, 5.4(b) o quadro atual, 5.4(c) subtração entre a imagem de fundo e o quadro atual, 5.4(d) imagem de movimento segmentada, 5.4(e) imagem de movimento filtrada e 5.4(f) imagem de movimento ressaltando o centro do objeto em cor amarela.

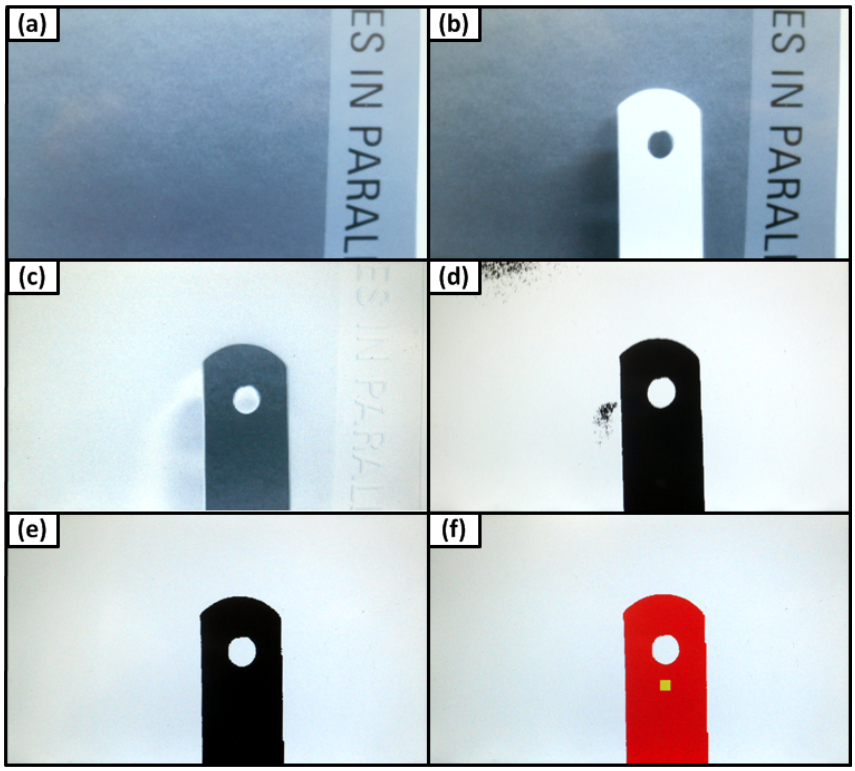

Figura 5.4. Resultado do processo de convolução. (a) Imagem de fundo. (b) Quadro atual. (c) resultado da subtração (incluindo o valor absoluto). (d) segmentação do movimento. (e) Imagem filtrada. (f) Representação final do objeto em movimento em vermelho, ressaltando o centro de gravidade em amarelo.

As imagens que correspondem as figuras 5.4(a) e 5.4(b), já têm sido filtradas antes de entrar na arquitetura de detecção de movimento. Depois da subtração das duas imagens e do cálculo do valor absoluto, obtém-se uma imagem em níveis de cinza mostrada na figura 5.4(c), que é convertida em uma imagem binária (Figura 5.4(d)) pelo processo de limiarização descrito na secção 4.4.1. Pode-se observar nesta imagem que o resultado da segmentação gera ruido, o que é resolvido aplicando a erosão e o resultado é apresentado na Figura 5.4 (e). Por último é calculado o centro de gravidade do objeto detectado (5.4(f)), representando o objeto na cor vermelha e o centro na cor amarela. Os resultados de síntese para a arquitetura de detecção de movimento são apresentado na Tabela 5.5.

Os multiplexadores são utilizados só para escolher a imagem a ser mostrada no *display*, sendo

Tabela 5.5. Resultados de síntese da arquitetura de detecção de movimento

Bloco	**Elementos Lógicos**	**Bits de Memória**	**Multiplicadores**
Subtração do fundo	16 (0,04%)	0	0
Valor absoluto	9 (0,02%)	0	0
Limiarização	6 (0,01%)	0	0
Erosão	722 (1,95%)	16.256 (3%)	0
Cálculo do centro	2.399 (6,47%)	0	0
Multiplexadores (x3)	48 (0,13%)	0	0
Bloco completo de detecção de movimento	**3.200 (8,63%)**	**16.256 (3%)**	**0**

Tabela 5.6. Latências dos blocos da arquitetura de detecção de movimento.

Bloco	**Numero de ciclos de relógio**	**Latência**
Subtração do fundo	1	43,43 ns
Valor absoluto	1	43,43 ns
Limiarização	1	43,43 ns
Erosão	1.603	69,62 μs
Cálculo do centro	384.000	16,68 ms
Bloco completo de detecção de movimento	**385.603**	**16,75 ms**

o seletor um sinal enviado pelo teclado. Pode-se observar que o bloco tem um baixo consumo dos recursos do FPGA, permitindo assim a implementação em FPGAs de baixo custo e obtendo um excelente desempenho. O bloco de *detecção de movimento* tem, assim como o bloco de convolução, uma latência inicial, na sua maior parte devido ao processo de erosão. A latência da arquitetura de detecção está dada pelo tempo que precisa o sistema para fornecer o primeiro pixel de saída, ou seja, o tempo necessário para realizar as operações de (a) subtração, (b) valor absoluto, (c) limiarização, (d) erosão e (e) cálculo do centro. Na tabela 5.6, mostra-se a latência de cada um dos processos, apresentando no final a latência total do bloco de detecção de movimento, lembrando que a frequência de operação do sistema é de 23,03 MHz (um período de 43,43 ns).

Como já mencionado, o sistema foi projetado para para cenas com alto contraste entre o objeto e a imagem de fundo. Na figura 5.5 mostra-se as limitações do sistema de detecção quando não se tem alto contraste entre o objeto e o fundo. Pode-se observar que nas áreas onde as cores do fundo e do objeto são iguais, o objeto não é detectado. Para corrigir estes erros é

Tabela 5.7. Resultados de síntese do bloco correspondente ao processador embarcado Nios II da Altera.

Elementos Lógicos	3.614 (9,75%)
Bits de Memória	285.696 (52,72%)

necessária a utilização de outras técnicas de segmentação de movimento, como o fluxo ótico ou segmentação por cores.

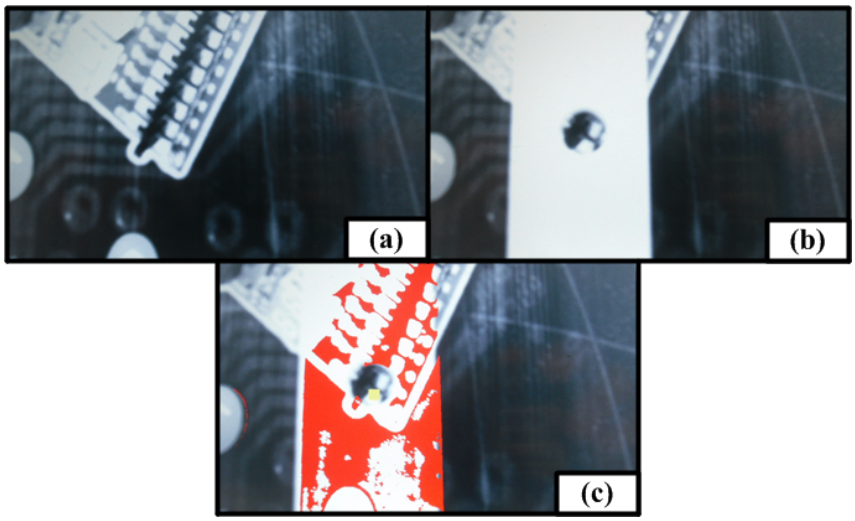

Figura 5.5. Problemas na identificação do objeto em movimento quando não existe alto contraste entre o objeto e a imagem de fundo. (a) Fundo. (b) Quadro atual. (c) Detecção de movimento.

5.5 Processador Nios II

Como já mencionado, o modulo de comunicação serial RS-232, assim como também a transformação de coordenadas retangulares para polares, foram desenvolvidos utilizando o processador embarcado Nios II da Altera Corp. Os dados são recebidos por um hyperterminal no computador, para serem utilizados depois em qualquer aplicação. Os resultados de síntese do bloco correspondente ao processador Nios II são mostrados na tabela 5.7.

A implementação do processador Nios II requer de grandes recursos de memória, porém, seu desempenho é alto e é muito versátil para a utilização de periféricos.

5.6 Testes de Validação

Para realizar a validação da arquitetura foram feitos dois testes. No primeiro teste, a câmera foi colocada na parte superior e o objeto em movimento se encontra sobre um plataforma horizontal, como mostrado na Figura 5.6.

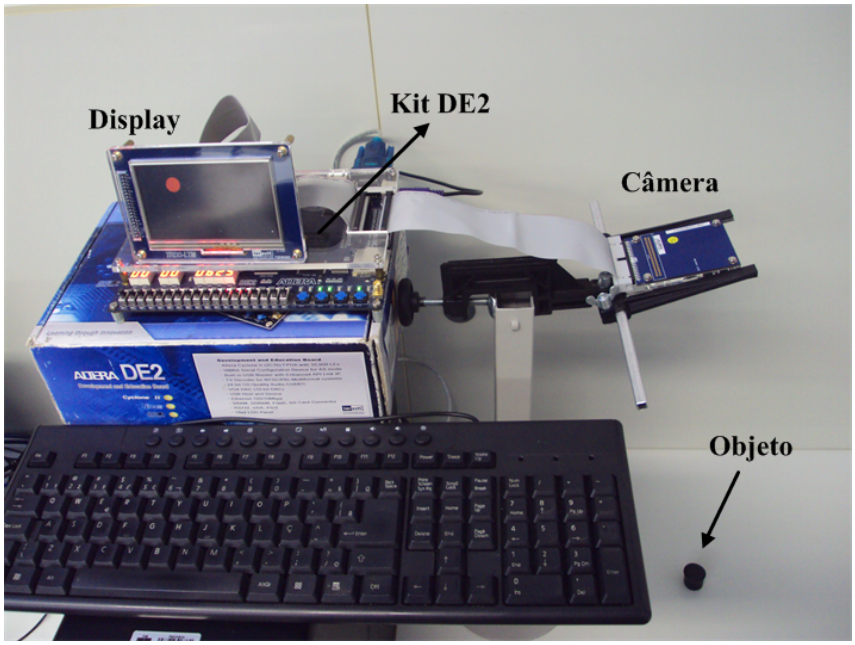

Figura 5.6. Primeiro teste. Objeto em movimento sobre uma plataforma horizontal.

Neste teste, o objeto foi colocado para realizar diferentes trajetórias, enviando os dados para o computador para serem plotados, como mostrado na Figura 5.7. Este gráfico mostra a trajetória do centro de gravidade do objeto se movimentado na superfície horizontal.

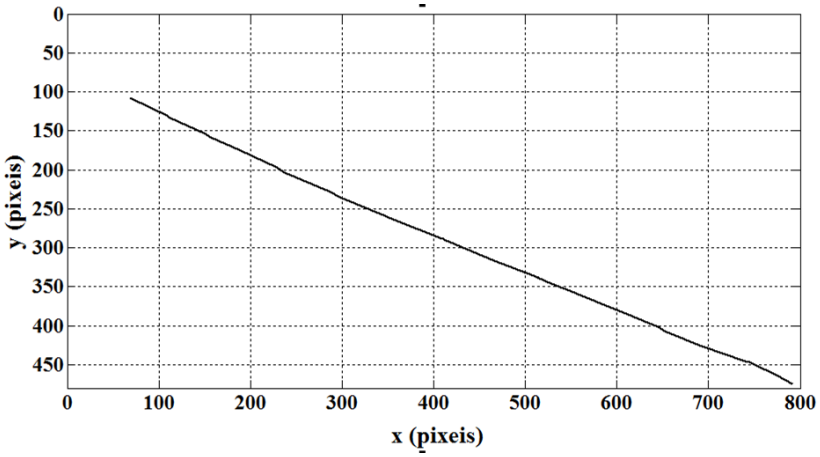

Figura 5.7. Primeiro teste. Trajetória de um objeto em movimento sobre uma plataforma horizontal.

No segundo teste, o sistema de detecção de movimento foi utilizado para monitorar o movimento de um pêndulo simples, como mostrado na Figura 5.8.

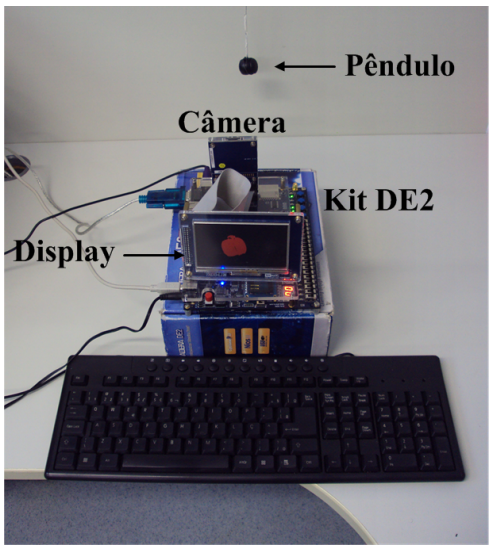

Figura 5.8. Segundo teste. Monitoramento do movimento de um pêndulo simples

Neste teste, o sistema faz a detecção do pêndulo oscilando e envia os dados do centro de gravidade para o computador, estes dados foram plotados e são mostrados nas Figuras 5.9 e 5.10. Na Figura 5.9 é mostrada a trajetória seguida pelo pêndulo, mostrando o comportamento esperado. Por outro lado, na Figura 5.10, mostra-se o gráfico o comportamento do centro na coordenada x com respeito ao tempo, neste caso, medido em número de quadros, apresentando uma tendencia sinusoidal.

Adicionalmente, o mesmo algoritmo para detecção de movimento foi implementado em um computador, rodando a 2,2 GHz, 2,0 GB de memória RAM usando o sistema operacional de tempo real *xPC Target* da MathWorks. O tempo médio para o processamento de uma imagem de 10×10 foi de aproximadamente 138,1 μs, (valor do TET - *Task execution time*). Assim, um pixel de saída é processado em 1,381 μs. Portanto, a arquitetura em *hardware* proposta neste trabalho alcança uma velocidade de processamento 32 vezes maior em comparação com a mesma implementação feita em *software*.

5.7 Implementação do Algoritmo em um Sistema de Visão Omnidirecional para o Cálculo de Distâncias.

Por outro lado, o sistema de detecção de movimento foi implementado em um sistema de visão omnidirecional, descrito na Figura 4.15. Este sistema objetiva a medição de distâncias para aplicações em robótica móvel. A Figura 5.11 ilustra o sistema construído, assim como seus

principais componentes: o espelho convexo e a câmera CMOS. A distância entre o centro do espelho e a lente da câmera é de aproximadamente 17 cm, e o centro do espelho está em linha vertical com o centro da lente da câmera.

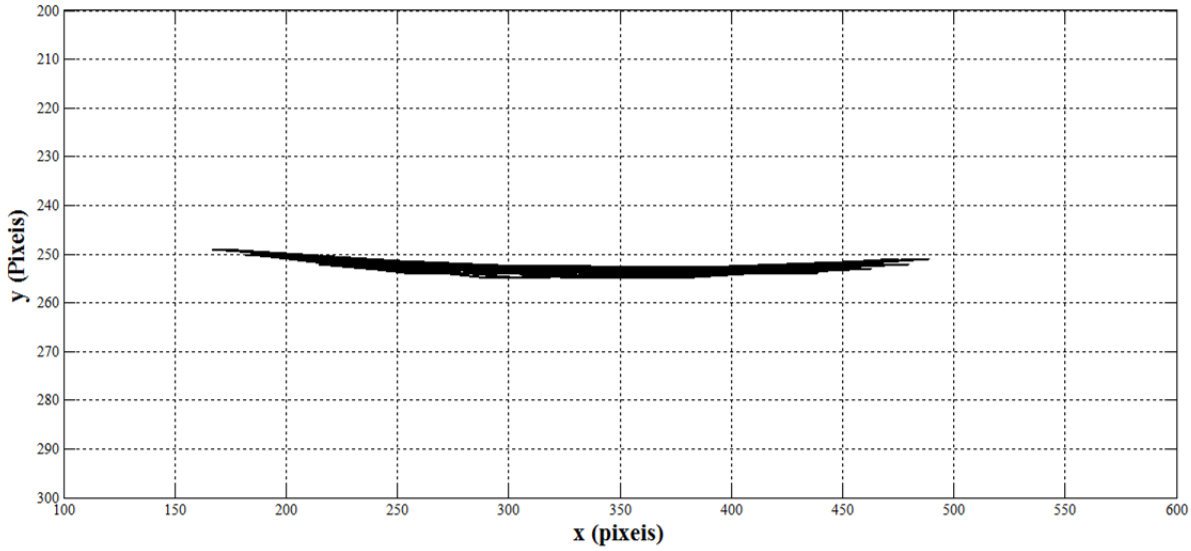

Figura 5.9. Segundo teste. Trajetória do pêndulo

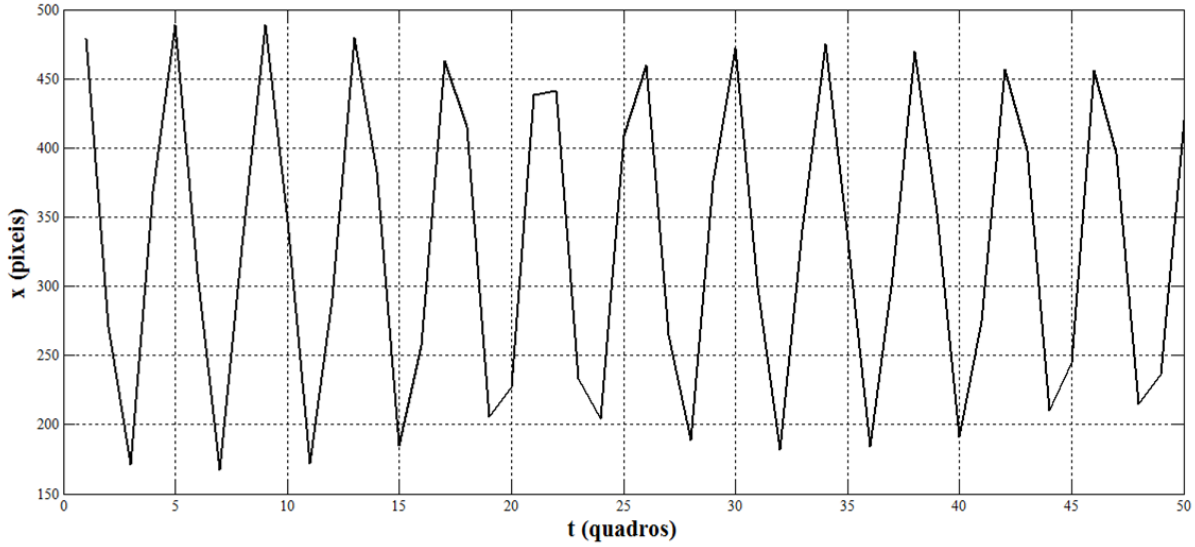

Figura 5.10. Segundo teste. Comportamento do centro na coordenada x com respeito ao tempo.

Para aplicações em robótica móvel, o sistema de visão omnidirecional provê uma imagem panorâmica na qual o robô está no cetro da mesma. A Figura 5.12 ilustra a imagem omnidirecional capturada assim como a cena real respectiva.

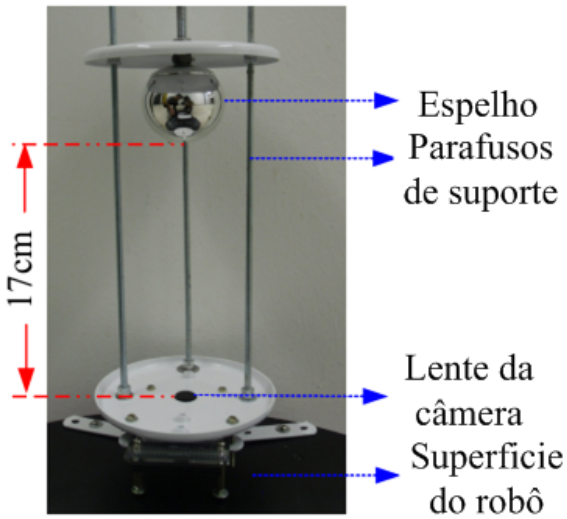

Figura 5.11. Componentes principais do sistema de visão omnidirecional proposto.

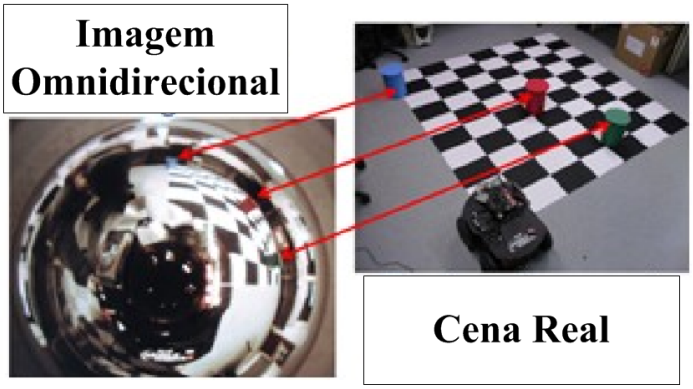

Figura 5.12. Sistema de visão omnidirecional e o robô na cena real rodeado por diferentes objetos.

O robô está disposto na cena com três objetos em torno dele e as setas indicam a correspondência entre objetos na imagem.

O sistema de visão omnidirecional proposto provê uma imagem panorâmica que pode ser processada com o objetivo de encontrar a distância (em pixeis) entre o centro do espelho e o objeto identificado na imagem. Para tanto, é implementado o algoritmo de detecção de movimento, obtendo primeiro a imagem de referência (sendo esta a cena sem objetos presentes nela), para, posteriormente, realizar a subtração entre o fundo e o quadro atual (como o objeto na cena). Uma vez o sistema tem sido calibrado, a distância do objeto (em centímetros) é calculada usando

um modelo matemático (descrito na secção 5.7.1).

Adicionalmente, usando um sistema de visão omnidirecional pode é possível estimar a direção dos objetos presentes em torno ao robô, proporcionando para este uma representação polar da cena.

5.7.1 Processo de Calibração.

A qualidade dos dados obtidos por um sistema de visão omnidirecional dependem de muito fatores como a geometria, a curvatura e a qualidade da superfície do espelho. Para aplicações de alta precisão é necessário realizar o processo de calibração usando equações da superfície do espelho e logo depois modelando os parâmetros geométricos de reflexão da luz. Porém, neste trabalho é desconhecida a geometria do espelho, pelo que torna-se impossível utilizar as equações de reflexão de luz.

O processo de calibração proposto permite que o sistema de visão seja caracterizado mediante uma função de interpolação que relaciona as distâncias em pixeis com as distâncias no mundo real. O processo de calibração foi desenvolvido associando objetos colocados a distâncias conhecidas com as distâncias estimadas na imagem (em pixeis), obtendo uma interpolação polinomial associada a cada seção do espelho convexo.

No trabalho descrito em [45] a imagem foi dividida em quatro seções polares realizando o mesmo processo de calibração para cada uma delas. Neste trabalho, o espelho foi dividido em seções de 40 graus (nove seções percorrem o espelho completo). Uma rede de pontos, que representa distâncias radiais entre 20 cm e 230 cm, foi disposta na cena. A Figura 5.13 ilustra as posições de calibração para uma seção particular na cena. Um objeto preto é colocado sobre cada um dos pontos vermelhos. O ambiente real de calibração é mostrado na Figura 5.14. É importante ressaltar que a cena não sofre variações de luz natural, é utilizada somente luz artificial assim como uma arena branca para obter alto contraste com o objeto preto.

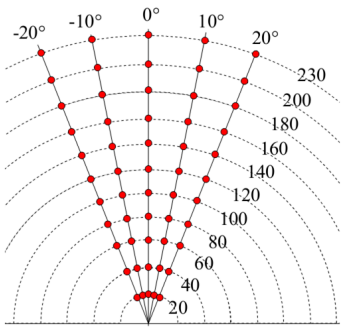

Figura 5.13. Posições de calibração para uma seção particular na cena.

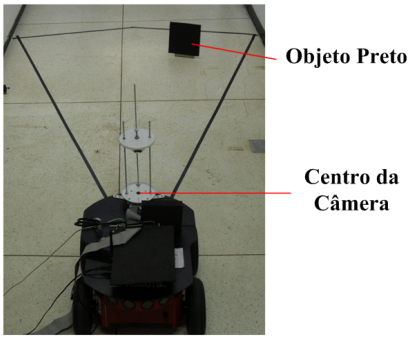

Figura 5.14. Sistema completo no ambiente de calibração.

A Tabela 5.8 apresenta uma análise estatístico do dados experimentais. Cada um dos valores de distância foi medido para cinco orientações diferentes, assim, foram usados um total de 55 pontos para a calibração de cada região.

Podem-se identificar duas regiões quanto ao comportamento dos valores de distância. A primeira região inclui as distâncias entre 20 e 120 cm, na qual uma pequena mudança no valor da distância gera uma grande mudança em pixeis. A segunda região é formada pelas distâncias entre 120 e 230 cm, na qual, grandes mudanças na distância, geram pequenas mudanças nos pixeis. Pode-se notar que, na primeira região (entre 20 e 120 cm), o erro tende a ser grande (exceto para 60 cm). Isto é causado por uma distorção dos objetos na imagem devida à geometria do espelho. A Figura 5.15 ilustra o comportamento da transformação pixeis/centímetros utilizando uma interpolação polinomial para cada uma das regiões.

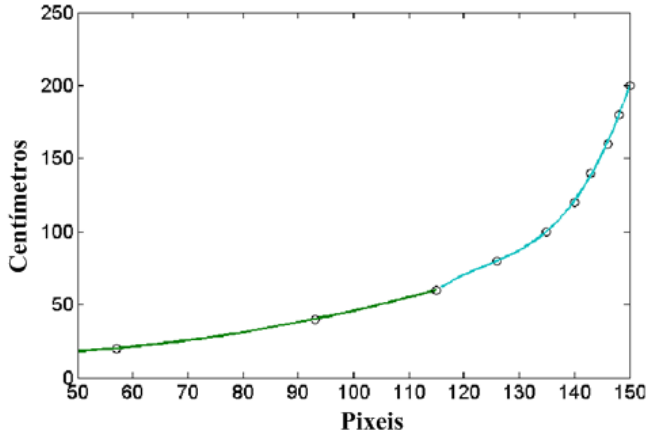

Figura 5.15. Funções da interpolação polinomial.

Tabela 5.8. Análise dos dados de calibração.

Distância (cm)	Distância média (pixeis)	σ (pixeis)
20	56,6	3,3
40	93,4	7,3
60	114,4	1,3
80	126,2	7,3
100	134,6	7,3
120	139,8	1,7
140	143,0	1,0
160	146,2	2,2
180	147,8	1,7
200	149,6	1,3
230	152,2	1,4

Angulo (graus)	Angulo médio (graus)	σ (graus)
-20	-24,8	4,8
-10	-14,4	11,6
0	-1,3	1,1
10	9,3	4,4
20	21,1	4,5

5.7.2 Estimação da Distância e Orientação

Em [45] as distâncias, tanto em pixeis como em centímetros, foram calculadas utilizando as bibliotecas de ponto flutuante descritas em [46]. Porém, é observado um incremento no consumo dos recursos *hardware*, especialmente para aplicações embarcadas. Neste trabalho foi utilizado o Nios II como processador encarregado de realizar as operações necessárias para o cálculo das distâncias, gerando uma redução no consumo de recursos.

Neste caso, o Nios II recebe as coordenadas do centro de massa geradas pela arquitetura em *hardware* e calcula a distância euclidiana assim como a orientação θ usando a função $atan()$. Finalmente, a interpolação polinomial é usada para a distância atual R, enviando as coordenadas (R, θ) para o PC.

5.7.3 Validação dos Resultados

A Figura 5.16 mostra um exemplo da cadeia de processamento. Na Figura 5.16(a) está ilustrando o fundo ou imagem de referência armazenada na memória. A Figura 5.16(b) apresenta o quadro atual, no qual aparece um objeto em torno ao robô. Posteriormente, na Figura 5.16(c) ilustra-se a subtração do quadro atual como o fundo, assim como o centro de gravidade do objeto detectado. Finalmente, a imagem da Figura 5.16(d) sobrepõe o quadro atual com a imagem da subtração objetivando uma melhor visualização dos resultados.

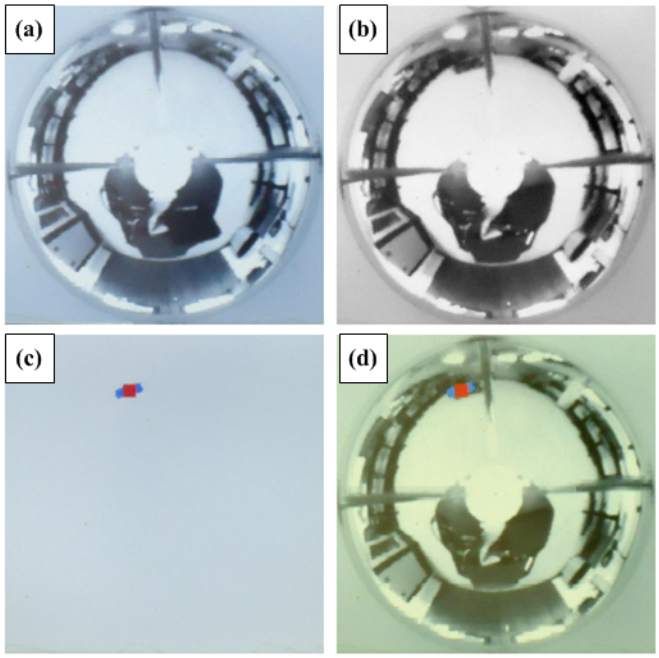

Figura 5.16. Resultados da cadeia de processamento.

Tabela 5.9. Resultados de validação . Distância em centímetros.

Distância Real (cm)	$P_1 = 70$	$P_2 = 74$	$P_3 = 115$	$P_4 = 135$	$P_5 = 170$	$P_6 = 200$
Objeto A	70	70	-	-	-	200
e_A	0%	5%	-	-	-	0%
Objeto B	80	-	-	132	170	248
e_B	14%	4%	-	-	0%	24%
Objeto C	83	83	107	132	170	235
e_C	19%	12%	7%	4%	0%	17%

Com o fim de testar o desempenho do sistema implementado, diversos experimentos foram implementados. Três diferentes objetos A, B e C, que correspondem a um robô cilíndrico, um robô móvel *pioneer* e o objeto de calibração, respectivamente, foram colocados em diferentes lugares em torno ao robô. A Figura 5.17 ilustra os objetos usados para testar o sistema completo. As distâncias e a orientação entre o centro da câmera e os objetos foram previamente medidas na arena. Adicionalmente, tanto a distância como a orientação são enviadas para o PC (via interface RS-232) e comparados com os valores reais.

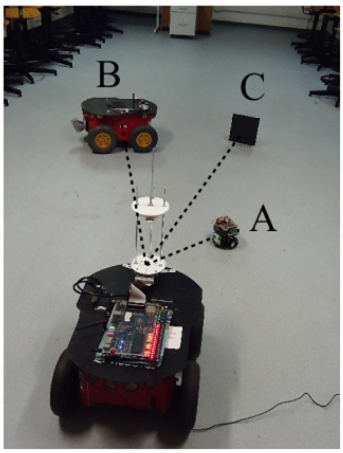

Figura 5.17. Objetos utilizados para testar o sistema completo.

A Tabela 5.9 apresenta os resultados de localização assim como os seus respectivos erros. Pode-se observar que a arquitetura proposta presenta melhor desempenho detectando o objeto A. Isto se dá, basicamente, pelo tamanho e pela forma cilíndrica do robô, já que produz uma sombra menor. Pode-se observar também que os erros são maiores para objetos mais distantes. Por exemplo, os resultados de localização para os objetos B e C apresentam grandes erros, em torno ao 24% e 17% respectivamente. Este fato pode-se explicar tendo em conta a geometria esférica do espelho, já que este apresenta distorções da imagem nas bordas dele (quanto mais longe estiver o objeto, menor será a sua projeção no espelho).

Tabela 5.10. Resultados de validação. Orientação em graus.

Angulo Real	A	B	C
$P_1 = 20$	20	19	20
$P_2 = -1$	9	-	9
$P_3 = -18$	-	-	-11
$P_4 = -8$	-	-4	-3
$P_5 = -14$	-	-8	-7
$P_6 = 11$	16	15	16

A Tabela 5.10 apresenta a estimação da orientação para cada objeto. A Figura 5.18 ilustra uma malha de localização em uma forma de gráfica polar para resumir os resultados alcançados para la estimação da distância e orientação. As regiões em cinza representam a posição estimada do objeto. Pode-se comprovar que o sistema produz grandes erros para distâncias maiores (vide ponto 6 para os objetos B e C).

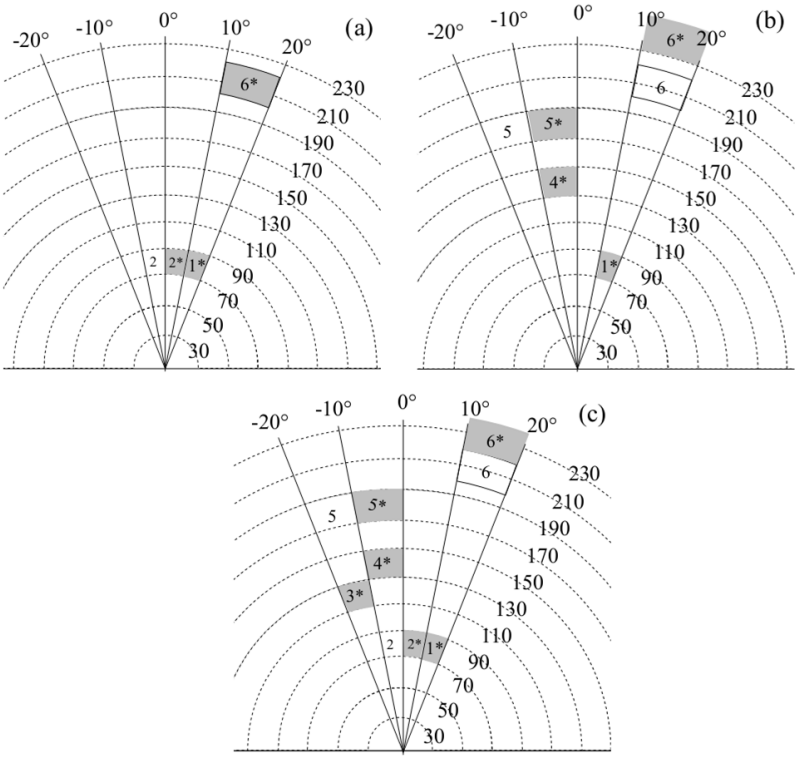

Figura 5.18. Estimação da distância e orientação para (a) objeto A, (b) objeto B e (c) objeto C. As regiões brancas representam as posições reais e as regiões cinza representam as posições estimadas pelo sistema.

5.8 Conclusões do Capítulo

Neste capítulo foram mostrados alguns exemplos práticos do funcionamento das arquiteturas propostas ao longo do trabalho. Os exemplos mostraram a viabilidade das implementações feitas, bem como a versatilidade e escalabilidade das arquiteturas. A flexibilidade do sistema permite a exploração de técnicas de processamento de imagens e visão artificial, que em *software* geralmente não seriam utilizadas para aplicações em tempo real, tendo em conta que a arquitetura proposta neste trabalho alcança uma velocidade de processamento 32 vezes maior do que o mesmo sistema implementado em uma plataforma *software*.

A plataforma de *hardware* utilizada mostra ser bastante útil para testes de aquisição, processamento e visualização de imagens em tempo real.

O sistema de visão omnidirecional tem melhor desempenho para a estimação de distâncias do que orientação dos objetos. Isto pode ser observado nos dados de calibração apresentados na Tabela 5.8, que mostra grandes valores de erro para os dados de orientação. Por outro lado, este sistema necessita de um alto contraste entre o fundo e os objetos na cena. Assim, quando o sistema funciona em condições de baixo contraste, são gerados erros na detecção dos objetos.

6 Conclusões e Propostas de Trabalhos Futuros

6.1 Comentários Finais e Conclusões

Neste trabalho foi desenvolvido um sistema de detecção de movimento baseado no algoritmo de subtração do fundo. O sistema trabalha como uma arquitetura de tempo real em forma de *pipeline*, fornecendo um pixel por ciclo de relógio (23,03 MHz) depois de um período de latência, tornando possível processar imagens de 800×480 com uma taxa de 60 fps (um pixel processado cada 43,43 ns). Este fato tem demostrado um incremento na velocidade de processamento de 32 vezes maior em comparação com a implementação da mesma arquitetura rodando em em uma plataforma *software*.

Adicionalmente, o sistema consegue realizar a detecção de um objeto, extraindo a sua forma e calculando o seu centro de gravidade. A posição do objeto é enviada para um computador ou alguma outra plataforma via interface RS-232.

Por outro lado, os resultados de síntese mostram que o consumo de área é baixo, utilizando só 28% dos elementos lógicos do FPGA para o sistema completo (que inclui o sistema de captura, a filtragem por convolução, redução de cores, o sistema de detecção de movimento, o cálculo do centro de gravidade e o processador Nios II). Porém, a arquitetura de detecção de movimento, que é o foco deste trabalho, corresponde a 9% do consumo dos elementos lógicos do FPGA, permitindo a implementação deste sistema em FPGAs de baixo custo.

A implementação do algoritmo de detecção de movimento foi implementado sobre um sistema de visão omnidirecional como ferramenta para a calibração e validação de um sensor de distância. Está técnica apresentou um bom desempenho, demostrando que a aplicação deste tipo de técnicas tem um grande potencial em áreas como robótica móvel, automação e controle de processos.

Por outro lado, foram estudados e analisados os principais algoritmos de processamento de imagens e detecção de movimento, a fim de extrair as características que permitissem a sua implementação em uma plataforma *hardware* de tempo real como os FPGAs.

6.2 Propostas de Trabalhos Futuros

Neste trabalho não foram consideradas algumas situações para a implementação da arquitetura em *hardware*. O sistema foi desenhado para realizar a detecção de um objeto que apresente alto contraste com o fundo numa cena onde não sejam presentes grandes mudanças na iluminação do local.

Assim, as principais propostas de trabalhos futuros são as seguintes:

- Desenvolver um sistema de atualização do fundo a fim de serem eliminados alguns problemas presentados com as mudanças de iluminação.
- Considerar a implementação do sistema de subtração do fundo no modelo de cores HSV, reduzindo assim a influencia da iluminação na cena. Isto também permitiria a implementação de um algoritmo de segmentação por cores, obtendo melhores resultados na limiarização dos objetos.
- Pode-se implementar um algoritmo de *labeling* para realizar a detecção de mais de um objeto em uma cena, permitindo assim a realização de etapas posteriores como o seguimento dos objetos.
- Implementar outros algoritmos de detecção de movimento, como o fluxo ótico, visando corrigir os problemas do sistema quando o objeto não apresenta alto contraste com o fundo. Realizar uma análise sobre o rendimento do algoritmo de fluxo ótico em comparação com o implementado neste trabalho e sobre o consumo de recursos do FPGA.
- Integrar o sistema em um ambiente robótico multi-agente para desenvolver tarefas de localização, mapeamento, planejamento e controle.
- Analisar o consumo de potência da arquitetura proposta. Esta análise é importante objetivando validar a eficácia dos algoritmos implementados para o processamento de imagens em tempo real.

REFERÊNCIAS BIBLIOGRÁFICAS

[1]F., Ramming. “Visions from IT engine room (interview)”. IFIP TC 10, Computer System Technology URL: `http://www.ifip.or.at/secretariat/tc_visions/tc10_visions.htm`.

[2]Y. Zhou, Y. Tan. “Gpu-based parallel particle swarm optimization”, Proc. IEEE Int. Congress on Evolutionary Computation, pp. 1493-1500, 2009.

[3]L. Veronese, R. Krohling, “Swarms’s flight: Accelerating the particles using c-cuda”, Proc. IEEE Int. Congress on Evolutionary Computation, pp. 3264-3270, Norway, 2009.

[4]D. Muñoz, C. Llanos, L. Coelho, M. Ayala-Rincón, “Comparison between two FPGA implementation of the particle swarm optimization algorithm for high-performance embedded applications”, The IEEE Fifth International Conference on Bio-Inspired Computing: Theories and Applications (BIC-TA 2010), vol. 1, pp. 1637-1645, 2010.

[5]J. Y. Mori, C. Sánchez-Ferreira, D. Muñoz, C. Llanos, P. Berger, “An Unified Approach for Convolution-Based Image Filtering on Reconfigurable Systems”, Proc. IEEE VII Southern Conference on Programmable Logic (SPL), pp. 63-68, Cȯrdoba, Argentina, 2011.

[6]E. M. Saad, A. Hamdy, M. M. Abutaleb, “FPGA-Based Implementation of a Low Cost and Area Real-Time Motion Detection”, 15th International Conference of Mixed Design MIXDES, pp. 249-254, Poznán, Poland, 2008.

[7]C. Torres-Huitzil, M. Arias-Estrada, “Real-time image processing with a compact FPGA-based systolic architecture”, Real-Time Imaging 10, pp. 177-187, 2004.

[8]M. Gokhale, P. S. Graham, “Reconfigurable computing - accelerating computation with field-programmable gate arrays”, Real-Time Springer Editorial, 2005.

[9]H. Pedrini, W. R. Schwarz, “Análise de Imagens Digitais - Princípios, Algoritmos e Aplicações”, Editora Thomson, 2008.

[10]Mori, J.Y. “Implementação de Técnicas de Processamento de Imagens no Domínio Espacial em Sistemas Reconfiguráveis”. Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação

ENM.DM-31/10, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF, 127p.

[11]D. H. Ballard, C. M. Brown, "Computer Vision". New York, 1982, 523 p. University of Rochester. Department of Computer Science. ISBN 0-13-165316-4.

[12]R. González, R. Woods, "Tratamiento Digital de Imágenes". Wilmington, USA: Addison-Wesley, 1996. ISBN 0-201-62576-8.

[13]A. Escalera, "Visión por computador: Fundamentos y métodos". Madrid: Prentice Hall, 2001. 304 p. ISBN 84-205-3098-0.

[14]G.J. Awcock, R. Thomas, "Applied Image Processing". McGraw Hill: Londres, 1996. University of Brighton. Department of Electrical and Electronic Engineering. ISBN 0-07-001470-1.

[15]C. Torres-Huitzil, M. Arias-Estrada, "Real-time image processing with a compact FPGA-based systolic architecture". Real-Time Imaging 10, pp. 177−187, 2004.

[16]C. Sánchez-Ferreira, P. Córdoba-Estrada, "Sistema de Vision Artificial para la Vigilancia de Propiedades Mediante procesamiento Digital de Imágenes de Video y Reconocimiento de Patrones". Trabajo de grado como requisito parcial para optar al titulo de Ingeniero Físico. Universidad del Cauca, Popayán - Colombia, 2008.

[17]D. Tsai, W. Chiu, "Motion detection using Fourier image reconstruction", Pattern Recognition Letters 29, pp. 2145-2155, 2008.

[18]F. Yuan, "A fast accumulative motion orientation model based on integral image for video smoke detection", Pattern Recognition Letters 29, pp. 925-932, 2008.

[19]Y. Dedeoglu, "Moving Object Detection, Tracking And Classifcation for Smart Video Surveillance", Bilkent University, Department of Computer Engineering and the Institute of Engineering and Science, August, 2004.

[20]J. L. Martín, A. Zuloaga, C. Cuadrado, J. Lázaro, U. Bidarte, "Hardware implementation of optical flow constraint equation using FPGAs", Computer Vision and Image Understanding 98, pp. 462, 2005.

[21]S. Wei, Z. Chen, H. Dong, "Motion Detection Based on Temporal Difference Method and Optical Flow field", Proc. Second International Symposium on Electronic Commerce and Security ISECS 2009, Nanchang, China, 2009

[22]S. Denman, V. Chandran, S. Sridharan, "An adaptive optical flow technique for person tracking systems", Pattern Recognition Letters 28, pp. 1232, 2007.

[23]W. Jian-ying, Y. Yu-tang, W. Yun-feng, D. Jun-jie, "Multi-DSP Real Time Image Processing System Based on Rapid IO Protocol". IEEE International Conference on Advanced Computer Theory and Engineering, Thailand, 2008.

[24]J. Akita, "An image sensor with fast objects position extraction function", IEEE Transactions on electron Devices, Nova Iorque, 2003.

[25]N. Kehtarnavaz, M. Gamadia, "Real-Time Image and Video Processing: From Research to Reality". Editora Morgan and Claypool, 2006.

[26]S. Kagami, "A real-time visual processing system using a general-purpose vision chip". ICRA - IEEE International Conference on Robotics and Automation, 2002.

[27]Murata, "Image processing LSI 'ISP-IV' based on local parallel architecture and its applications". ICIP - International Conference on Image Processing, 1998.

[28]R. Porter, "Evolution on FPGAs for Feature Extraction". PhD thesis on Information Technology; Queensland University of Technology, 2001.

[29]C.A.F. De Rose, P.O.A. Navaux, "Arquiteturas Paralelas". Editora Bookman, 2008

[30]F.P. Júnior, "Seleção de variáveis e características como aplicação paralela para cluster MPI". Dissertação de Mestrado em Ciência da Computação, Universidade Estadual de Maringá, 2006.

[31]Arias-Garcia, J. "Implementação em FPGA de uma biblioteca parametrizável para inversão de matrizes baseada no algoritmo Gauss-Jordan, usando representação em ponto flutuante". Dissertação de Mestrado em Sistemas Mecatrônicos, Publicação ENM.DM-36A/10, Departamento de Engenharia Mecânica, Universidade de Brasília, Brasília, DF.

[32]A.C.O.S. Aragão, "Uma arquitetura sistólica para solução de sistemas lineares implementada com circuitos FPGAs", Dissertação de Mestrado em Ciências, ICMC - Instituto de Ciências Matemáticas e de Computação, USP, 1998.

[33]F. Vahid, "Sistemas Digitais - Projeto, Otimização e HDLs", Primeira Edição, Editora Bookman, 2007.

[34]J. Becker, R.W. Hartenstein, "Configware e morphware going mainstream" - Journal of Systems Architectures, 2003.

[35]G. Botella, M. Rodriguez, A. Garca, E. Ros, "Neuromorphic configurable architecture for robust motion estimation" - International Journal of Reconfigurable Computing, 2008.

[36]Z. Wei, D. Lee, N. Brent, J. Archibald, B. Edwards, "FPGA-based embedded motion estimation sensor" - Interna-tional Journal of Reconfigurable Computing, 2008.

[37]K. Shimizu, S. Hirai, “CMOS + FPGA vision system for visual feedback of mechanical systems” - in Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE In-ternational Conference on, may 2006, p. 2060.

[38]G. Saldaña González, M. Arias-Estrada, “FPGA based acceleration for image processing applications” - in Image Processing, 2009.

[39]Y. Tu, M. Ho, “Design and implementation of robust visual servoing control of an inverted pendulum with an FPGA-based image co-processor” - in Mechatronics, v.21, no.7, 2011 Oct, p.1170(13), vol. 21, no. 7. Elsevier B.V., October 2011, p. 1170.

[40]T. Kryjak, M. Gorgon, “Real-time implementation of moving object detection in video surveillance systems using FPGA” - Computer Science, vol. 12, p. 149, 2011.

[41]R. Rodriguez-Gomez, E. Fernandez-Sanchez, J. Diaz, E. Ros, “FPGA implementation for real-time background subtraction based on horprasert model” - Sensors, vol. 12, p. 585, 2012.

[42]M. A. Vega-Rodríguez, A. Gómez-Iglesias, J. A. Gómez-Pulido, and J. M. Sánchez-Pérez, “Reconfigurable computing system for image processing via the internet” - Microprocess. Microsyst., vol. 31, p. 498, December 2007. [Online]. Available: `http://dl.acm.org/citation.cfm?id=1296324.1296347`

[43]F. Nava, D. Sciuto, M. D. Santambrogio, S. Herbrechtsmeier, M. Porrmann, U. Witkowski, U. Rueckert, “Applying dynamic reconfiguration in the mobile robotics domain: A case study on computer vision algorithms” - ACM Trans. Reconfigurable Technol. Syst., vol. 4, August 2011. [Online]. Available: `http://doi.acm.org/10.1145/2000832.2000841`

[44]L. Chen, M. Zhang, B. Wang, Z. Xiong, G. Cheng, “Real-time FPGA-based panoramic unrolling of high-resolution catadioptric omnidirectional images,” - in Proc. Int. Conf. on Measuring Technology and Mechatronics Automation, 2009, p. 502, Hunan, China.

[45]J. Mori, D. Muñoz, J. Arias-Garcia, C. Llanos, and J. Motta, “FPGA-based image processing for omnidirectional vision on mobile robots” in Proc. International Symposium on Integrated Circuits and System Design, 2011, pp. 113 - 118, João Pessoa, Brazil.

[46]D. Muñoz, D. Sánchez, C. Llanos, and M. Ayala-Rincón, “Tradeoff of FPGA design of a floating-point library for arith-metic operators” Journal of Integrated Circuits and Systems, vol. 5, no. 1, pp. 42 - 52, 2010.

Anexos

A. AMBIENTE DE DESENVOLVIMENTO

A.1 Kit de Desenvolvimento DE2

Todas as implementações utilizaram como base um *kit* de desenvolvimento para FPGAs da Terasic Inc. Esse *kit* utiliza um FPGA CycloneII da Altera Corp., fornecendo uma grande variedade de interfaces de entrada e saída para diversos periféricos. As especificações do *kit* são as que seguem (Figura A.1):

- FPGA Altera CycloneII 2C35
- Dispositivo de configuração Altera EPCS16
- USB Blaster para programação e controle de dispositivos
- 4 *pushbuttons*
- 18 *switches*
- 18 leds vermelhos
- 9 leds verdes
- Osciladores de 50MHz e 27MHz
- Interface de entrada e saída de áudio
- Porta de saída VGA com conversor DA de 10 bits
- Decodificador NTSC/PAL, para entrada de vídeo
- Controlador Ethernet 10/100 Mbits/s
- Controlador USB Host/Slave
- Interface serial RS-232
- Interface PS2
- Interface infravermelho
- Duas portas de 40 pinos de entrada/saída de uso geral
- Memória SRAM de 512KBytes

•Memória SDRAM de 8MBytes

•Memória Flash de 4MBytes

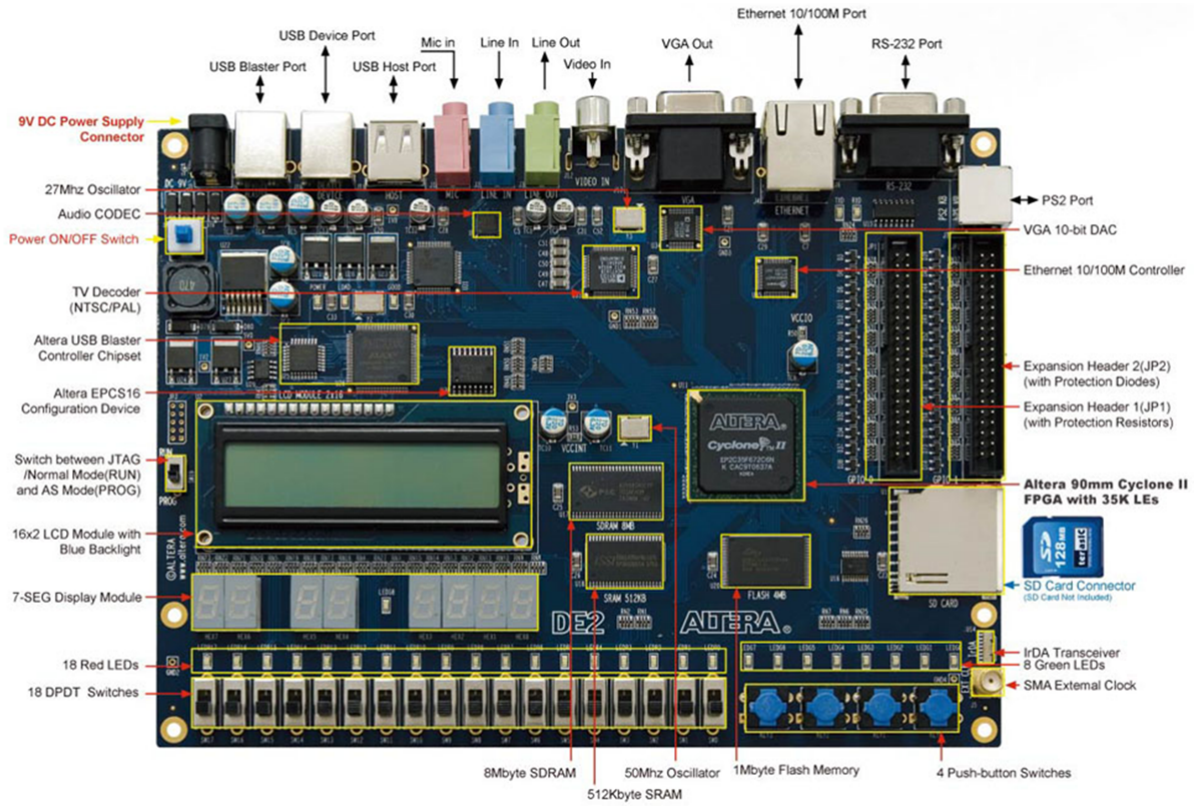

Figura A.1. Kit de desenvolvimento DE2, Terasic Inc.

A.2 FPGA Cyclone II

Os FPGAS da família Cyclone II da Altera possuem algumas características internas de interesse às implementações feitas neste projeto. O modelo utilizado no kit DE2 é o EP2C35, que tem as seguintes características:

•32.216 elementos lógicos (LEs)

•105 blocos de memória M4K

•483.840 bits de memória RAM

•35 multiplicadores embarcados

•4 PLLs

Os FPGAS desta família possuem arquiteturas baseadas em linhas e colunas para implementação das lógicas. Essas linhas e colunas são compostas de *logic array blocks* (LABs), *embedded memory blocks* e *embedded multipliers*. A Figura A.2 mostra a organização interna de um FPGA da família CycloneII.

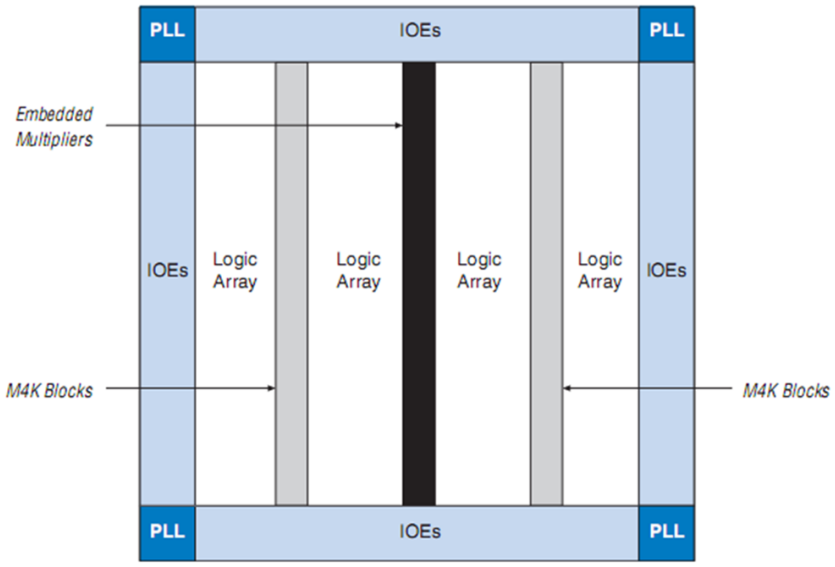

Figura A.2. Diagrama de organização interna de um FPGA da família Cyclone II.

Nos tópicos que seguem são apresentadas algumas características mais relevantes dos blocos internos da CycloneII.

A.2.1 *Logic Array Blocks/Logic Elements*

Cada LAB possui 16 elementos lógicos, que são pequenas unidades lógicas para implementação de funções lógicas de forma eficiente. Essas unidades possuem as seguintes características:

- *Look-up table*: é uma unidade de quatro entradas, capaz de implementar qualquer função lógica de 4 variáveis.
- Um registrador programável: cada registrado possui sinais de *clear*, *clock* e dados, podendo ser configurado para funcionar como um *flip-flop* tipo D, T, JK ou SR, permitindo uma grande gama de funcionalidades.

Com esses elementos lógicos são implementadas funções como contadores, somadores, subtratores, funções aritméticas, acumuladores e comparadores, além de ser possível também a implementação de pequenas memórias.

Tabela A.1. Características dos blocos M4K de memória

Característica	Descrição
Desempenho máximo	250 MHz
Total de bits de RAM por bloco	4608
Configurações possíveis	4K x 1, 2K x 2, 1K x 4, 512 x 8, 512 x 9, 256 x 16, 256 x 18, 128 x 32, 128 x 36
Bits de paridade	Um bit de paridade para cada byte. O bit de paridade, junto com alguma lógica configurada, pode implementar a checagem de paridade para detecção de erros e garantia da integridade dos dados.

A.2.2 Memória Embarcada

Os blocos de memória embarcada na CycloneII consistem de colunas de blocos de memória do tipo M4K. Esse tipo de memória inclui registradores de entrada que sincronizam a escrita, e registradores de saída que permitem trabalhar como em um *pipeline*, melhorando o desempenho do sistema.

Cada bloco M4K pode implementar memórias de diversos tipos, com ou sem bits de paridade, possuindo as características resumidas na Tabela A.1. Tais blocos ainda podem ser configurados para operarem de forma otimizada em diversos modos, descritos na Tabela A.2.

Cada multiplicador embarcado pode implementar duas multiplicações de 9×9 bits ou uma multiplicação de 18×18 bits, a frequências de até 250 MHz. Os multiplicadores são arranjados em colunas no FPGA.

A.2.3 Multiplicadores Embarcados

As FPGAs da família CycloneII possuem multiplicadores otimizados para funções de processamento digital de sinais que necessitem de muitas multiplicações, como filtros FIR, a FFT, DCT, entre outras. Cada bloco multiplicador pode ser utilizado de dois modos diferentes, dependendo da necessidade da aplicação, descritos na Tabela A.3.

A.3 Câmera

Para a captura das imagens foi utilizada uma câmera digital de 5 Mega Pixeis de resolução real, com conexão própria para o Kit DE2, modelo TRDB_D5M (Terasic Inc.), Figura A.3.

Tabela A.2. Modos de memória M4K

Modo de memória	**Descrição**
Single-port memory	Não permite leitura e escrita simultâneos.
Simple dual-port memory	Permite leitura e escrita simultâneos
Simple dual-port with mixed width	Permite diferentes comprimentos de dados de leitura e escrita.
True dual-port memory	Permite qualquer combinação de operações *dual-port*: duas leituras, duas escritas, uma escrita e uma leitura, com frequências distintas.
True dual-port with mixed width	É o modo *True dual-port*, com a possibilidade de diferentes comprimentos de dados de leitura e escrita.
Embedded shift register	Implementação de registradores de deslocamento. Os dados são escritos em cada endereço na borda de descida do *clock* e lidos de cada endereço na borda de subida.
ROM	A memória é pré-carregada utilizando-se um arquivo tipo .mif, durante a configuração do dispositivo.
FIFO *buffers*	Implementação de FIFO com *clock* único ou duplo (leitura e escrita).

Tabela A.3. Modos de operação dos multiplicadores embarcados

Modo de operação	**Descrição**
Multiplicador de 18 bits	Uma multiplicação de dois operandos de 18 bits.
Multiplicador de 9 bits	Duas multiplicações de dois operandos de 9 bits cada.

Algumas características dessa câmera são:

- Controle de tempo de exposição do sensor, configurável via registradores
- Sensor CMOS de 5 Mega Pixel, com resolução configurável (máximo de 2592×1944 pixeis ativos) e conversor AD de 12 bits por canal de cor.
- Barramento de dados de saída paralelo, com sinais separados R, G e B.
- Taxa de aquisição variável, de acordo com parâmetros de resolução e tempo de exposição do sensor, de acordo com a Tabela A.4.

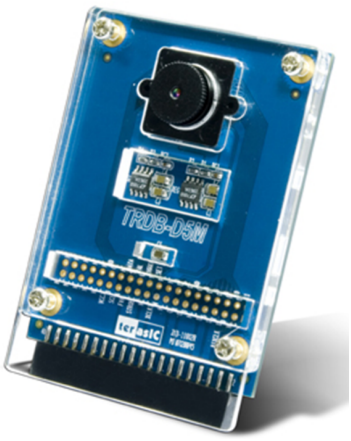

Figura A.3. Câmera modelo TRDB_D5M (Terasic Inc.).

A câmera TRDB_D5M consiste em uma matriz de pixeis de 2752 colunas por 2004 filas. A posição $(0, 0)$ representa o canto superior direito do arranjo, como mostrado na Figura A.4.

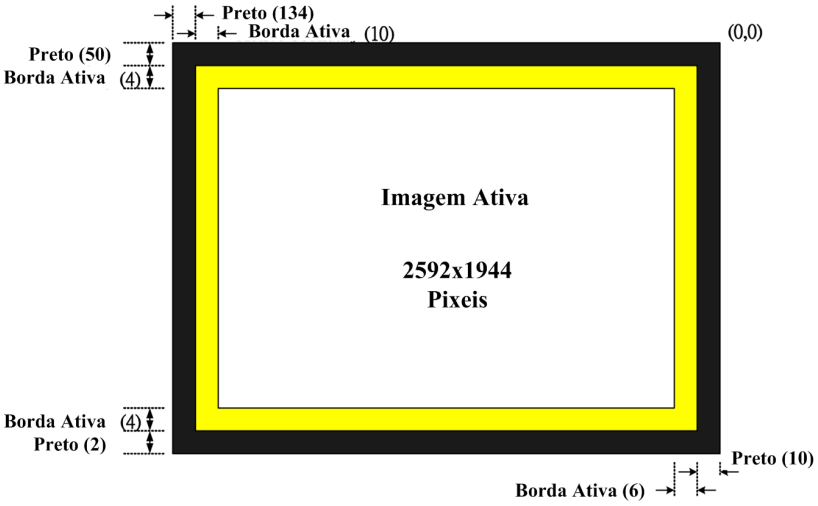

Figura A.4. Descrição do arranjo de pixeis

Tabela A.4. Resoluções padrão

Resolução	**Taxa de aquisição (fps)**
2592 × 1944 (resolução máxima)	15,15
2048 × 1536 QXGA	23
1600 × 1200 UXGA	35,2
1280 × 1024 SXGA	48
1024 × 768 XGA	73,4
800 × 600 SVGA	107,7
640 × 480 VGA	150
1920 × 1080 HDTV	34,1
1280 × 720 HDTV	67,6

Tabela A.5. Tipo de pixel por coluna

Coluna	Tipo de pixel
0 - 9	Preto (10)
10 - 15	Borda ativa (6)
16 - 2607	Imagem ativa (2592)
2608 - 2617	Borda ativa (10)
2618 - 2751	Preto (134)

Tabela A.6. Tipo de pixel por fila

Fila	Tipo de pixel
0 - 49	Preto (50)
50 - 53	Borda ativa (4)
54 - 1997	Imagem ativa (1944)
1998 - 2001	Borda ativa (4)
2002 - 2003	Preto (2)

O arranjo tem uma região ativa de 2592 colunas por 1944 filas no centro, representando a imagem de saía por defeito, rodeada por uma borda (também ativa), rodeada por uma borda de pixeis pretos (ver Tabelas A.5 e A.6).

A saída dos pixeis é no padrão *Bayer* que consiste em quatro cores (G_1 - verde, G_2 - verde, R - vermelho e B - azul), representando três filtros de cor. A saída da primeira fila alterna entre pixeis G_1 e R, e a segunda fila entre pixeis B e G_2. Os pixeis G_1 e G_2 têm o mesmo filtro de cor, porém eles são tratados por separado (ver Fig A.5).

A.4 *Display* LCD

A visualização das imagens, processadas ou não, foi feita utilizando um *display* LCD, modelo TRDB_LTM, da Terasic Inc. (Figura A.6). Esse *display* possui uma interface para conexão no Kit DE2, e as seguintes características:

- Interface de dados paralela RGB de 24 bits
- Correção de brilho e contraste configuráveis
- Resolução de $800 \times 480 \times RGB$ pixels

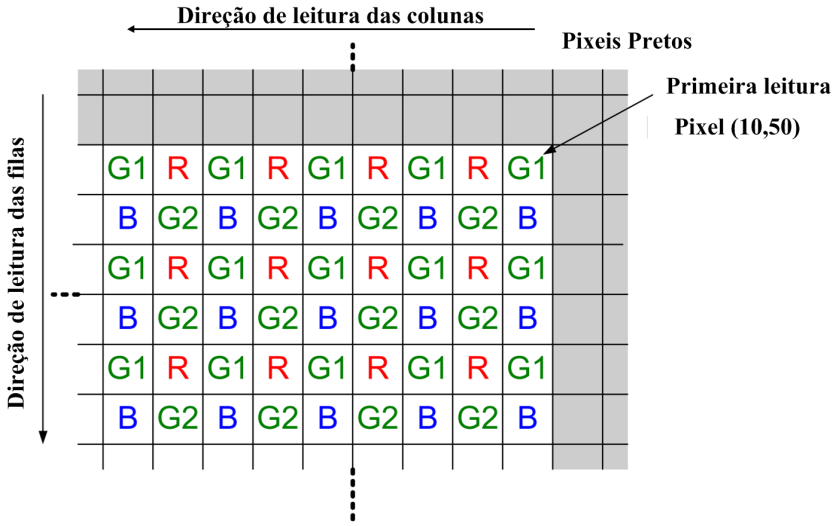

Figura A.5. Padrão de cores na saída da câmera (canto superior direito)

Figura A.6. *Display* LCD, modelo TRDB_LTM (Terasic Inc.).

O *display* está formado por três componentes principais: (a) um módulo para a tela táctil, (b) um conversor A/D e (c) um conector de expansão de 40 pinos. Todas as interfaces do *display* são ligadas com o FPGA utilizando o conector de expansão de 40 pinos. O módulo da tela táctil toma os sinais de controle geradas pelo FPGA como entrada e apresenta as imagens na tela. Finalmente, o conversor A/D vai converter as coordenadas do ponto (x, y) que foi tocado na tela.

A.5 *Software* Quartus II da Altera Corp.

Foi utilizada a versão 10.1 da ferramenta EDA Quartus II, da Altera Corp., a qual permite a criação e simulação de projetos, além da síntese e programação dos dispositivos FPGAs. As Figura A.7 mostra uma captura de do ambiente de programa.

O Quartus II suporta o desenvolvimento em três linguagens diferentes de descrição de *hard-*

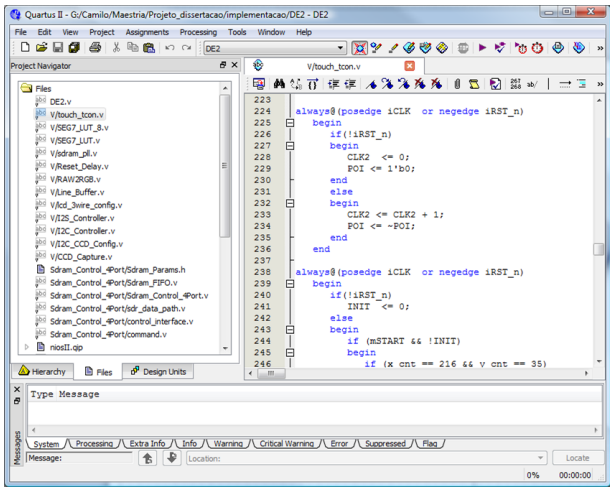

Figura A.7. Tela do Quartus II, mostrando um exemplo de código Verilog.

ware:

- VHDL - *Very High Speed Integrated Circuits Hardware Description Language*. É uma linguagem padrão ISO para a descrição de *hardware*.
- Verilog *Hardware Description Language*. Linguagem desenvolvida pela *Cadence Systems*, empresa especializada em ferramentas CAD para projetos de circuitos.
- AHDL - *Altera Hardware Description Language*. Linguagem desenvolvida pela própria Altera.

Existe ainda a possibilidade de se criar projetos utilizando diagramas de blocos representativos dos códigos descritos nas HDLs. A Figura A.8 mostra um exemplo no Quartus II.

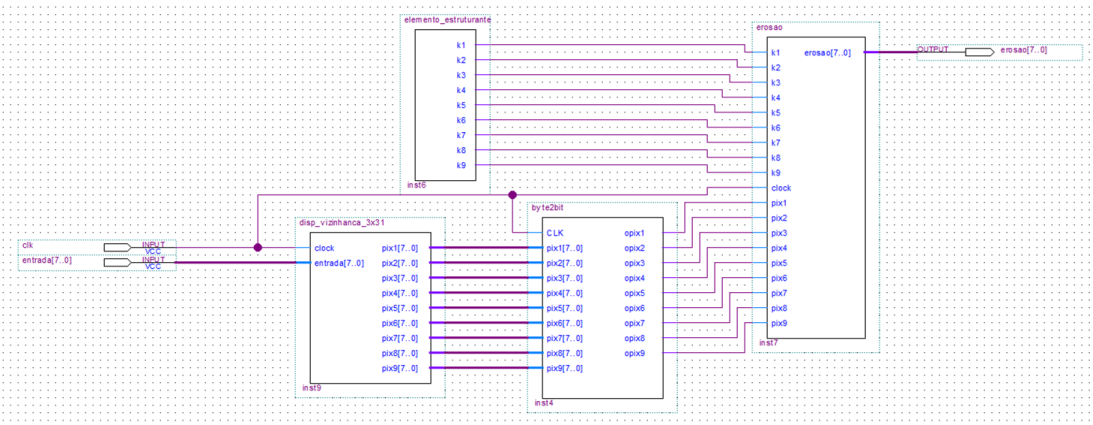

Figura A.8. Tela do Quartus II, exemplificando um projeto utilizando Diagramas de Blocos.

A.6 Processador Nios II da Altera Corp.

O processador Nios II é equivalente a um microcontrolador que inclui um processador e uma variedade de periféricos e memórias em um chip só. Na Figura A.9 mostra-se um exemplo de um processador Nios II. O Nios II é um processador RISC de proposito geral com as seguintes características:

- Conjunto de instruções completo de 32 bits
- 32 registradores de proposito geral
- 32 fontes de interrupção
- Multiplicadores e divisores 32×32 que produzem resultados de 32 bits.
- Instruções dedicadas para o cômputo de produtos de multiplicações de 64 e 128 bits
- Instruções de ponto flutuante para operações de precisão simples
- Acesso a uma grande variedade de periféricos no chip e interfaces para memórias e periféricos externos.
- Ambiente de desenvolvimento *software* baseado na ferramenta GNU C/C++ e o *Nios II Software Build Tools (SBT) for Eclipse* (Figura A.10).
- Desempenho acima de 250 DMIPS

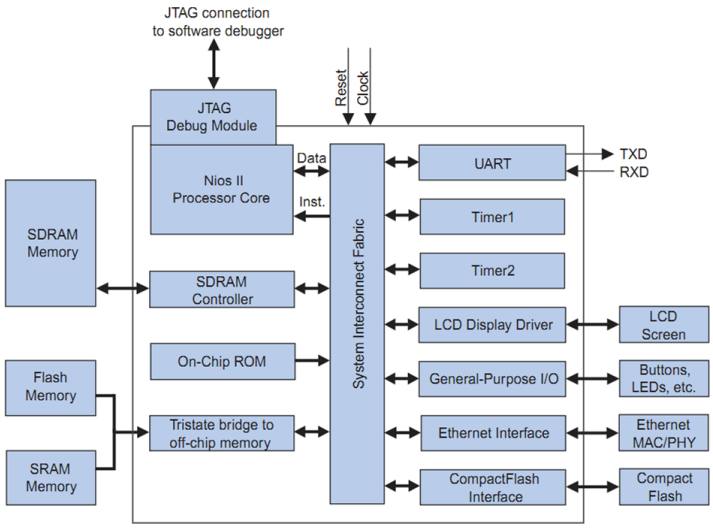

Figura A.9. Exemplo de um sistema em FPGA baseado no Nios II.

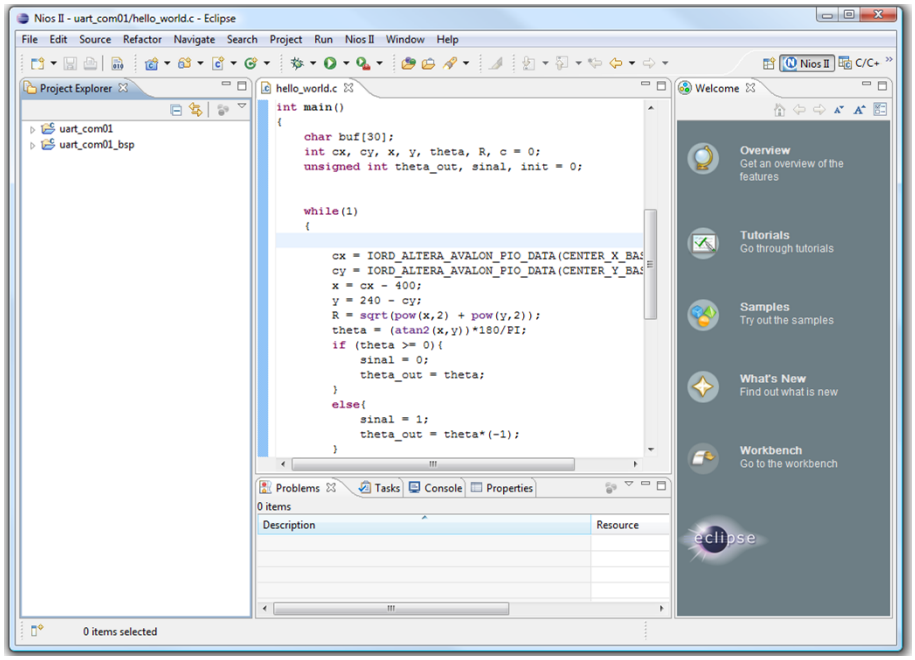

Figura A.10. Ambiente de desenvolvimento *software Nios II Software Build Tools (SBT) for Eclipse*