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

FÁBIO CELESTINO PEREIRA

Análise de Desempenho de Algoritmos para Auxílio ao Reconhecimento de Fissuras em Fachadas com Revestimento de Argamassa Visando Sua Embarcação em VANTs

Porto Alegre
2015

FÁBIO CELESTINO PEREIRA

Análise de Desempenho de Algoritmos para Auxílio ao Reconhecimento de Fissuras em Fachadas com Revestimento de Argamassa Visando Sua Embarcação em VANTs

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

Área de concentração: Controle e Automação

ORIENTADOR: Prof. Dr. Carlos Eduardo Pereira

Porto Alegre

2015

FÁBIO CELESTINO PEREIRA

Análise de Desempenho de Algoritmos para Auxílio ao Reconhecimento de Fissuras em Fachadas com Revestimento de Argamassa Visando Sua Embarcação em VANTs

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

Orientador: _____

Prof. Dr. Carlos Eduardo Pereira, UFRGS.

Doutor pela Universidade de Stuttgart – Stuttgart, Alemanha.

Banca Examinadora:

Prof. Dr. Altamiro Amadeu Susin, UFRGS

Doutor pela Institut National Polytechnique de Grenoble – Grenoble, França.

Prof. Dr. Luiz Carlos Pinto da Silva Filho, UFRGS.

Doutor pela Universidade de Leeds – Leeds, Reino Unido.

Prof. Dr. Renato Ventura Bayan Henriques, UFRGS.

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

Coordenador do PPGEE: _____

Prof. Dr. Alexandre Sanfelice Bazanella

Porto Alegre, março de 2015.

AGRADECIMENTOS

Agradeço a minha noiva, Natália Petry, pelo apoio, incentivo, amor e confiança durante todo estudo no mestrado.

Agradeço ao Grupo de Controle Automação e Robótica, que me acolheu em um de seus grupos de pesquisa, aos colegas de laboratório LASCAR pelo auxílio e principalmente ao Doutorando Dionísio Doering pelas conversas e orientações, sobretudo auxiliando nas dificuldades encontradas.

Agradeço a professora Ângela Borges Masuero, por me esclarecer sobre manifestações patológicas através de sua aula, conversas e fornecimento do material inicial para o estudo.

Agradeço ao professor Carlos Eduardo Pereira pela jornada, e o aceite em me orientar e prosseguir com o conhecimento ora adquirido neste trabalho, pelas suas ideias, apoio e incentivos e cobranças.

Agradeço ao professor Luiz Carlos Pinto da Silva Filho, por compreender e permitir o meu afastamento, possibilitando a continuidade deste trabalho até sua conclusão.

Agradeço ao aluno de mestrado Erico Nunes, por me auxiliar na execução dos algoritmos em um processador Soft.

Agradeço a todos da Universidade Federal do Rio Grande do Sul, incluindo os colegas de trabalho que encorajaram na realização deste trabalho, assim como os professores e o Programa de Pós-Graduação em Engenharia Elétrica, PPGEE, pelos conhecimentos transferidos e da oportunidade de realização do trabalho em minha área de pesquisa.

RESUMO

A utilização de Veículos Aéreos Não Tripulados (VANTs), também chamados UAV (Unmanned Aerial Vehicle) vem ganhando espaço nas mais diversas áreas, tais como inspeções em linhas de transmissão e em torres de fracionamento de refinarias, entre outros. Já na construção civil, estudos recentes estão sendo focados na utilização dos VANTs para inspeção de pontes, viadutos e estradas. O presente trabalho visa fornecer uma análise para o uso de VANT na área civil, na detecção de manifestações patológicas em revestimentos de argamassa, de forma a auxiliar na procura por fissuras em fachadas, sobretudo aqueles que a visualização esteja prejudicada, seja pela distância ou pela acessibilidade difícil ao local. Este trabalho analisa possíveis implementações de dois algoritmos de processamentos de imagens desenvolvidos a partir da ferramenta MATLAB para a indicação de presença de fissuras na alvenaria, obtendo o desempenho destes diferentes algoritmos quando executados em software em plataforma que possibilite a embarcação em VANTs. Utilizando a geração automática de código em C a partir do ambiente MATLAB, é realizada uma análise temporal em plataforma ARM e RISC dos algoritmos propostos, demonstrando a oportunidade de utilização de dispositivos na tarefa de processamento de imagem para a aplicação proposta. Esta análise possibilita a previsão do comportamento na utilização de um VANT, uma vez que isto pode impactar na velocidade durante a aplicação e consequentemente sua autonomia.

Palavras-chave: Veículo Aéreo Não Tripulado, VANT, Processamento de Imagens, Sistema Embarcado, Algoritmo de Detecção de Fissuras.

ABSTRACT

The use of Unmanned Aerial Vehicles (UAVs), has been gaining space in several areas, such as inspections of transmission lines, refineries fractionation towers among others. In the construction, recent studies have been focused on the use of UAVs for inspection of bridges, viaducts and roads. The present study aims to provide an analysis using UAVs in the civil construction area, in the detection of pathologic manifestations mortar coatings in order to aid in the search for cracks in the facades especially those that visualization is impaired, or may be the distance the accessibility difficult to spot. This paper provides an analysis of two algorithms of image processing developed from the Matlab tool for indicating the presence of cracks in masonry, getting the performance of these different algorithms when implemented in software platform that enables the vessel in UAV. Using the automatic generation of C/C++ code from the MATLAB environment is performed the temporal analysis on ARM and RISC plataform of the proposed algorithms demonstrates the opportunity to use devices in the image processing task for the proposed application. This analysis allows the prediction of the behavior of a UAV using one since it can impact velocity during application, and therefore their autonomy.

Keywords: Unmanned Aerial Vehicle, Image Analysis, Embedded System, Detection Cracks, Raspberry PI, FPGA.

SUMÁRIO

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS

ASIC: Application Specific Integrated Circuit

CLB: Configurable Logic Blocks

FPGA: Field Programmable Gate Array

GCAR: Grupo de Controle, Automação e Robótica

GCC: GNU Compiler Collection

GNU: Unix-like Operating System

GPS: Global Positioning System

GPU: Graphics Processing Unit ou Unidade de Processamento Gráfico

HDL: Hardware Description Language

IEEE: Institute of Electrical and Electronics Engineers

kB: kilobyte

kbps: Kilo bits por segundo

LDH: Linguagem de Descrição de Hardware

MB: Megabyte

MLP: Multilayer Perceptron

Mp: Megapixel

NORIE: Núcleo Orientado para a Inovação da Edificação

OMG: Object Management Group

PIXEL: Picture Element

PPGEE: Programa de Pós-Graduação em Engenharia Elétrica

RF: Radio Frequência

SoC: System-on-Chip

UFRGS: Universidade Federal do Rio Grande do Sul

VANT: Veículo Aéreo Não Tripulado

VHDL: VHSIC Hardware Description Language

VHSIC: Very High Speed Integrated Circuits

UAVs: unmanned aerial vehicles

UML: Unified Modeling Language

1 INTRODUÇÃO

O estudo das manifestações patológicas em edificações é de fundamental importância para avaliar a qualidade e desempenho das construções no meio da engenharia civil, uma vez que a competição empresarial e produtividade vêm buscando melhorias nas etapas da construção e manutenção de obras civis. A evolução dos estudos direcionados às inspeções e manutenções prediais tem sido de grande importância de modo a garantir o desempenho das edificações e prevenir acidentes devido ao processo de deterioração. Quanto mais precocemente forem detectadas estas deteriorações, menores serão os custos dos reparos, além de prolongar a vida útil de uma edificação.

Os métodos de manutenção e inspeção ainda são tarefas essencialmente caras, visto que muitas vezes requerem operações que podem colocar em risco a vida dos inspetores e, em função disto, exigem equipamentos de segurança. Além disto, métodos de inspeções tradicionais usam inspeção visual utilizando técnicas padronizadas e muitas vezes exigem que profissionais se desloquem para o local, a ser investigado, para determinar o seu nível de deterioração. A automatização deste processo pode resultar em grande economia monetária e pode levar a ciclos de inspeção mais frequentes [ABDEL et al. 2003].

É fundamental adotar um conjunto padrão de exames para verificar a capacidade funcional de uma edificação, de modo a uniformizar a análise, sendo que esta avaliação é usualmente complexa, visto envolver conceitos multidisciplinares e deste modo exige profissionais experientes. Estudos sobre manifestações patológicas em edificações são de grande importância, pois o surgimento destas manifestações pode ocorrer em diferentes etapas da vida útil do revestimento, e suas causas devem ser investigadas por um profissional experiente, de modo a identificar e adotar ações de reparo satisfatórias e evitando assim o surgimento de futuras manifestações. (DAL MOLIN, 1988; SILVA, 2007; PASQUALOTTO, 2012).

Uma das manifestações patológicas visíveis que podem proporcionar riscos futuros ao desempenho das edificações são as fissuras, as quais geram possíveis caminhos para a penetração de agentes agressivos externos, especialmente água, resultando em novas manifestações patológicas, como eflorescências, manchas de umidade, bolor ou mofo e descolamento de placas de revestimento. Por isso é necessário que as fissuras sejam detectadas o mais cedo possível. A Figura 1 ilustra alguns dos problemas causados pelo desplacamento do revestimento de fachadas de prédios que provavelmente se iniciaram em função de fissuras, levando à ocorrência de graves acidentes.

Fonte:http://g1.globo.com/rj/regiaoserrana/noticia/2014/03/mulher-e-atingida-por-reboco-que-caiu-do-10-andar-no-centro-historico.html

a)

Fonte:http://www.ofluminense.com.br/editorias/cidades/queda-de-reboco-assusta-e-danifica-carros-no-centro.

b)

Fonte:http://www.correiodopovo.com.br/Noticias/?Noticia=316357

c)

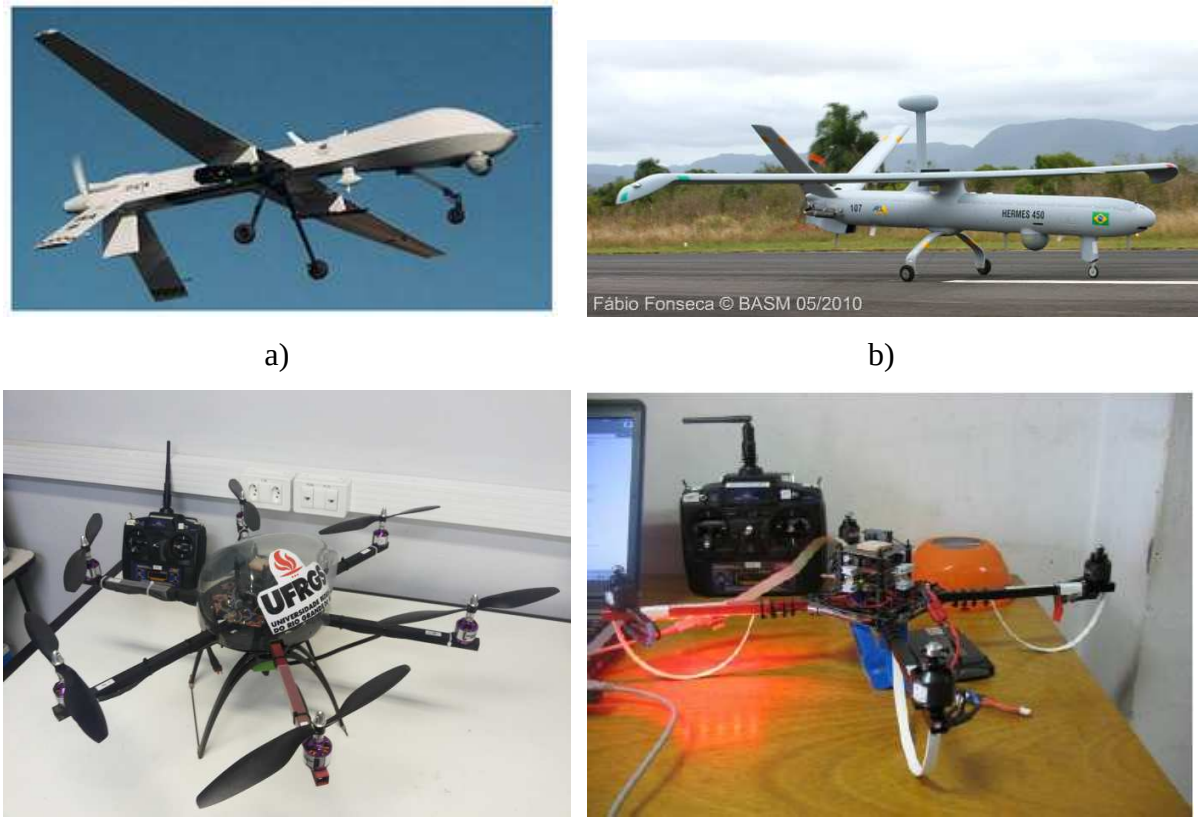
Figura 1 a) Queda de reboco na cidade de Petrópolis/RJ; b) Queda de reboco na cidade de Niterói/RJ; c) Queda de Reboco na cidade de Porto Alegre

Em regiões de alturas elevadas e ou de difícil acesso, o levantamento destas manifestações pode se tornar uma tarefa árdua e perigosa, sendo que muitas vezes os profissionais responsáveis pela inspeção necessitam utilizar andaimes ou plataformas para conseguir acessar estas regiões. Estes artifícios tornam a análise mais demorada e mais onerosa, tanto em função do tempo necessário para instalação dos equipamentos, além dos requisitos de segurança que devem ser levados em consideração. Neste contexto, o uso de dispositivos que facilitem o trabalho de inspeção, cresce em importância, tais como os veículos aéreos não tripulados (VANTs).

Os Veículos Aéreos Não Tripulados (VANTs), também chamados UAVs (Unmanned Aerial Vehicle), são aeronaves que podem voar sem tripulação, normalmente projetadas para operar em situações perigosas e repetitivas em regiões consideradas hostis ou de difícil acesso (FURTADO, et. al. 2008). A sua utilização vem ganhando espaço, nos estudos atuais e em

diversas aplicações, tais como em inspeções de linhas de transmissão, de torres de fracionamento de refinarias, entre outros (DONG et. al., 2012; DENG et. al., 2014). Na área da engenharia civil, os VANTs vem sendo utilizados para inspeções em pontes, viadutos e estradas (METNI et. al., 2006; RATHINAM et. al., 2006).

Estes equipamentos possuem versões de asas fixas ou rotativas, os quais apresentam características de autonomia de voo, distância a ser percorrida, altitude, peso máximo de carga, entre outras que devem ser analisadas para escolha do tipo de VANT é mais recomendado para uma determinada aplicação. Os VANTs de asas rotativas, também chamados de Rotorcraft Unmanned Aerial Vehicle (RUAV), vêm ganhando destaque em sua utilização, uma vez que podem operar em diferentes modos de voo que os de asa fixa são incapazes de praticar, como decolagem e pouso vertical, pairar em um ponto específico, ou efetuar voo lateral equidistante de uma superfície (ZHAO, JIANG, HAN, 2007). A Figura 2 apresenta alguns exemplos de VANTs, sendo os de asas fixas apresentados nas Figura 2 a) e b), e os de asas rotativas nas Figura 2 c) e d), estes últimos estão disponíveis para pesquisas no Laboratório do Grupo de Controle Automação e Robótica (GCAR) da UFRGS.



a) b)

c) d)

Figura 2 Veículas Aéreos Não tripulados. a) General Atomics MQ-1 Predator; b) Elbit Systems Hermes 450; c) VANT MikroKopter Hexacopter Disponível no Lab.GCAR; d) VANT MikroKopter Quadcopter Disponível no Lab.GCAR

Os VANTs são utilizados hoje tanto em aplicações militares, quanto em aplicações civis, principalmente pelos aspectos mencionados anteriormente. Na área de inspeções, por exemplo, sua utilização em áreas abertas é viabilizada pelo uso de sistemas de navegação baseada em tecnologia GPS como principal meio de posicionamento e controle de missão. Atualmente alguns estudos relacionados a outros meios para o posicionamento destes equipamentos vêm sendo realizados, como a utilização de imagens capturadas por uma câmera para o auxílio na sua navegação, aprimorando sua trajetória, pouso e decolagem (LANGE, S¨UNDERHAUF, PROTZEL, 2009; DRAPER, et. al., 2006; JENSENA, et. al., 2010).

De modo a obter imagens para as inspeções, um dos principais equipamentos integrados nos RUAVs são as câmeras para captura de imagem e vídeo, que auxiliam na análise posterior e também durante um voo controlado aonde não é possível ter visão do aparelho durante todo o voo. A proposta deste trabalho é analisar possibilidades de uso de VANTs para a detecção de fissuras de forma autônoma, sendo que, para isto, sistemas computacionais que executem algoritmos de detecção de fissuras devem ser embarcados nos VANTs, o que impõe restrições de desempenho e consumo energético destes sistemas. A dissertação avaliará alguns destes algoritmos e suas possíveis implementações. O sistema proposto fará uso de imagens capturadas por uma câmera no espectro visualque serão processados através de técnicas de processamento de imagem possibilitando a detecção da manifestação de forma automática, ou pelo menos adicionando informações importantes que auxiliam a interpretação humana.

Busca-se com este trabalho disponibilizar uma ferramenta que facilite a realização de inspeções e manutenções prediais, levando a uma percepção automática de detalhes importantes que se deseja saber quando das manifestações patológicas, especificamente as fissuras em argamassas.

Esta dissertação está estruturada da seguinte forma: o Capítulo 2 apresenta a fundamentação teórica, com seus principais conceitos, começando com uma breve abordagem sobre as manifestações patológicas e apresentando com mais detalhes o tipo de manifestação foco deste trabalho, as fissuras em revestimentos de argamassa, que podem ser encontradas em fachadas. Também no capítulo 2 são apresentados conceitos da teoria de processamento de imagens digitais, que foram utilizadas na elaboração dos algoritmos, assim como um classificador bayesiano, o filtro de partículas. No Capítulo 3 são apresentados trabalhos relacionados com o tema desta dissertação. No Capítulo 4 é abordada a proposta desta dissertação, contendo uma especificação do sistema baseado em engenharia orientada a objetos utilizando a linguagem UML, os algoritmos utilizados e a metodologia de implementação, descrevendo o ambiente de simulação utilizado e a etapa de geração de código para as

plataformas embarcadas. No Capítulo 5 são apresentados os resultados das implementações no ambiente de simulação assim como nas plataformas computacional, Raspberry PI e em FPGA (Field Programmable Gate Array), com analise dos cenários de utilização da aplicação. O Capítulo 6 apresenta as conclusões desta dissertação e menciona possíveis trabalhos futuros, que podem dar sequencia às pesquisas desenvolvidas.

2 FUNDAMENTAÇÃO TEÓRICA

A fim de auxiliar os leitores não totalmente familiarizados com os conceitos desenvolvidos nesta dissertação, este capítulo busca explicar de forma sucinta os conceitos, apresentando referências para que os leitores interessados possam encontrar um maior detalhamento sobre os conceitos utilizados. A primeira seção é relacionada ao conceito de manifestações patológicas, especificamente fissuras cuja identificação automática é o objeto foco e motivou a elaboração deste trabalho e nas seções subsequentes são apresentados conceitos de processamento de imagens digitais, que foram utilizadas na elaboração dos algoritmos, assim como de classificador bayesiano (filtro de partículas), empregado em um dos algoritmos, o qual foi utilizado para a detecção de possíveis fissuras localizadas nas fachadas.

2.1 MANIFESTAÇÕES PATOLÓGICAS EM EDIFICAÇÕES

São consideradas manifestações patológicas todas aquelas intercorrências que venham por ventura prejudicar o desempenho esperado de um edifício e/ou de suas partes, tais como seus subsistemas, elementos e componentes. Deste modo as manifestações patológicas podem ocorrer em toda a estrutura do edifício, desde a fundação até em revestimentos (VERÇOZA, 1991).

As edificações, de maneira geral, estão sujeitas à ação de diversos fatores, que vão desde os ambientais até sua relação com o entorno e com o próprio homem. Estas ações podem ocasionar transformações irreversíveis, caracterizadas como deterioração ou manifestação patológica (KIEL, 2005).

As manifestações patológicas são ocasionadas por diversos motivos, sendo que grande parte dos problemas que aparecem durante a vida útil da edificação tem sua origem nas fases de elaboração do projeto e na execução do serviço.

O estudo das manifestações patológicas em edificações é de fundamental importância para avaliar a qualidade e desempenho das construções, uma vez que devido a competição empresarial e produtividade procurasse uma melhoriaconstante nas etapas da construção e manutenção de obras civis.

Dentre as manifestações patológicas que ocorrem com maior frequência no estado do Rio Grande do Sul estão as fissuras, conforme estudo desenvolvido com base em um levantamento realizado pela CIENTEC/RS. Conforme apresentado na Figura 3 (DAL MOLIN, 1988).

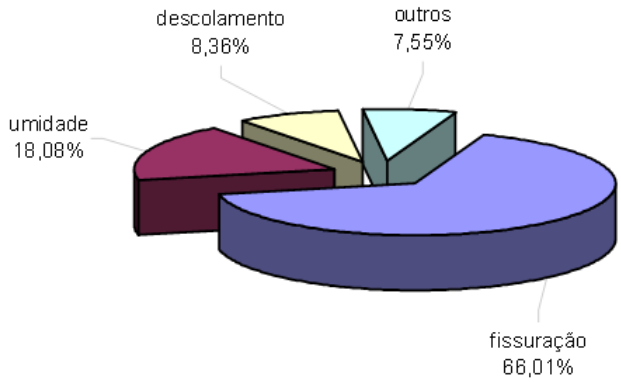


Figura 3 Incidência de manifestações patológicas atendidas pelo CIENTEC/RS.
Fonte: (DAL MOLIN, 1988)

No ano de 2013 entrou em vigor a norma NBR 15575 (ABNT, 2013) que tem por objetivo garantir o desempenho das edificações habitacionais. Desta maneira é possível observar a preocupação da construção civil em garantir que as construções apresentem desempenho e qualidades mínimas. Sendo assim, a detecção das fissuras é uma maneira de prevenir outros tipos de manifestações patológicas, deste modo garantindo o desempenho esperado das edificações.

As fissuras em revestimentos de argamassa estão ligadas a diversos fatores, entre eles pode-se destacar a retração, expansão e variação térmica. Estas se constituem como caminho para a penetração de agentes agressores externos, principalmente a água, que pode levar ao aparecimento de novas manifestações patológicas.

Para (BAUER, 1997), nas argamassas de revestimento, a incidência de fissuras geralmente está condicionada a fatores relativos à execução do revestimento de argamassa, solicitações hidrotérmicas e principalmente por retração hidráulica da argamassa.

A movimentação dos elementos construtivos pode ser causada por sobrecargas, variações de temperatura, retração, expansão por umidade, deformações elásticas, deformação lenta, recalques de fundação, recalques diferenciais, vibrações, ação do vento, ação do fogo, choques, explosões, terremotos, e outros, provocando assim, a fissuração (DUARTE, 1998; ELDRIDGE, 1982; THOMAZ, 1989).

Segundo (VIEIRA, 2006), “os agentes climáticos ou ambientais são fatores externos de deterioração (extrínsecos) que atuam sobre a edificação ao longo do tempo. O meio ambiente de um determinado local é o resultado de vários processos físicos agindo em diferentes escalas”.

A Figura 4 ilustra algumas das fissuras que podem ser encontradas em fachadas com revestimentos de argamassa. Conforme pode ser visto nesta figura, as fissuras encontradas podem ser horizontais, fissuras mapeadas, fissura por expansão, escalonada no encontro de paredes, concêntricas, cada uma representando um efeito causado por problemas verificadas durante a execução, efeitos climáticos, efeitos químicos, entre outros. Existem também fissuras localizadas próximas às esquadrias, estas com características bem determinadas, tendo suas causas apresentada em alguns trabalhos relacionados a fissuras em paredes.

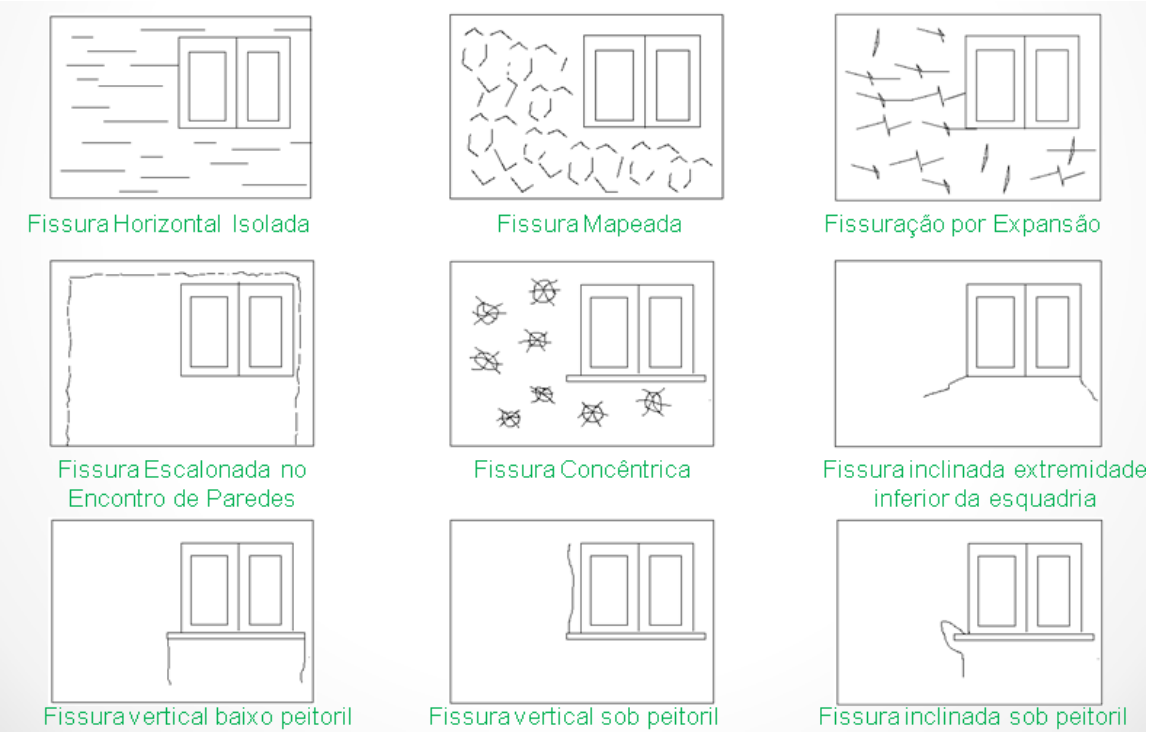


Figura 4 Algumas Fissuras que podem ser encontrada em fachada de argamassa (MASUERO, 2014).

Na Figura 5, é possível observar fotografias de fissuras em revestimentos de argamassas em fachadas de edificações.

MASUERO A.B. **Patologia dos Revestimentos**. 2014. 15p. Notas de aula.

Figura 5 – Fissuras em Fachadas com Revestimento Argamassado

2.2 TEORIA DE PROCESSAMENTO DE IMAGENS DIGITAIS

Segundo (PEDRINI, SCHWARTZ, 2008) “o desenvolvimento de sistemas autônomos que reproduzam as capacidades do sistema visual humano e que sejam capazes de reagir a estímulos visuais de forma adequada à área específica sob investigação ainda é um grande desafio. A capacidade humana para captar, processar e interpretar grandes volumes de dados de natureza visual estimula o desenvolvimento de técnicas e dispositivos cada vez mais sofisticados, com intuito da substituição do observador humano com uso de robôs, por exemplo.”

Para ser possível a identificação das fissuras em fachadas de revestimentos em argamassa, por meio de imagem, é necessária a utilização de alguns conceitos relacionados a imagens digitais, os quais serão apresentados a seguir. Estes conceitos auxiliam o entendimento da análise de imagens digitais, sendo também descritos alguns métodos utilizados para a identificação do objeto foco deste trabalho.

2.2.1 Imagem

Uma imagem pode ser definida como uma função de intensidade luminosa, denotada por f(x,y), cujo valor ou amplitude nas coordenadas espaciais (x,y) fornece a intensidade ou o brilho da imagem naquele ponto. A Figura 6 mostra uma imagem e a orientação do sistema de coordenadas utilizada. Por convenção, a origem da imagem está localizada no canto superior esquerdo da mesma (PEDRINI, SCHWARTZ, 2008).

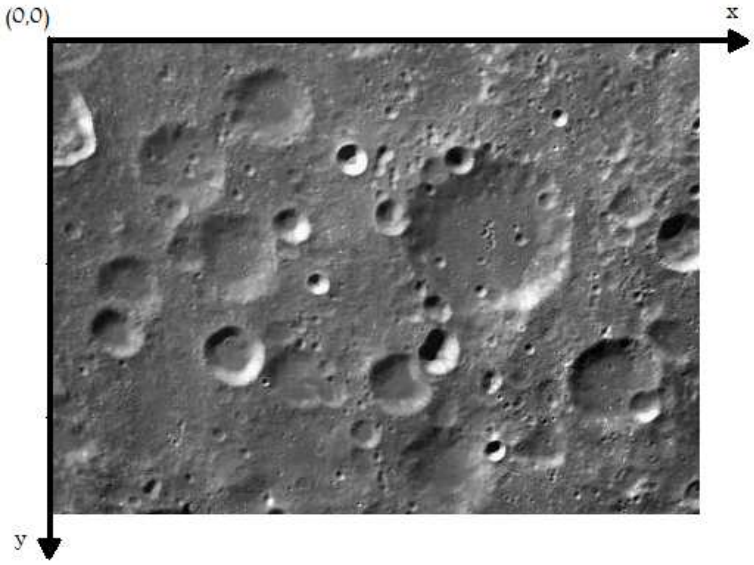


Figura 6 Convenção do sistema de coordenadas para representação de imagens digitais. (PEDRINI, SCHWARTZ, 2008).

Para o processamento computacional da imagem, a função f(x,y) precisa ser convertida para uma descrição discreta, assim a *imagem digital* é obtida por um processo de digitalização, envolvendo dois passos, a amostragem e a quantização.

A *amostragem* consiste em discretizar o domínio de definição da imagem nas direções x e y, gerando uma matriz de M x N amostras, respectivamente. A *quantização* incide em escolher o número inteiro L de níveis de cinza (no caso de imagens monocromáticas) permitidos para cada ponto da imagem (PEDRINI, SCHWARTZ, 2008).

Cada elemento f(x,y) dessa matriz de amostras é chamado pixel (acrônimo do inglês picture element), com $0 \leq x \leq M\text{-}1$ e $0 \leq y \leq N\text{-}1$. A imagem contínua f(x,y) é aproximada, portanto, por uma matriz de dimensão M pixels na horizontal (eixo x da Figura 6) por N pixels na vertical (eixo y da Figura 6), como mostrado na equação (1).

$$f(x,y) \approx \begin{bmatrix} f(0,0) & f(1,0) & \cdots & f(M-1,0) \\ f(0,1) & f(1,1) & \cdots & f(M-1,1) \\ M & M & O & M \\ f(0,N-1) & f(1,N-1) & \cdots & f(M-1,N-1) \end{bmatrix}_{[M \times N]} \quad (1)$$

A dimensão do pixel ao longo do eixo x ou ao longo do eixo y é a distância relacionada ao espaço físico entre as amostras digitalizadas, tendo assim um valor $L_{mín} \leq f(x,y) \leq L_{máx}$, tal que o intervalo $[L_{mín}, L_{máx}]$ é denominado escala de cinza. A intensidade f de uma *imagem monocromática* nas coordenadas (x,y) é chamada de nível de cinza da imagem naquele ponto. Uma convenção comum é atribuir a cor preta ao nível cinza mais escuro (por exemplo, valor 0) e atribuir a cor branca ao nível cinza mais claro (por exemplo, valor 255).

Assim, uma imagem digital pode ser representada por meio de uma matriz bidimensional, na qual cada elemento da matriz corresponde a um pixel. A Figura 7 mostra a representação matricial de uma imagem. Uma pequena região da imagem é destacada, formada por números inteiros correspondendo aos níveis de cinza dos pixels da imagem.

Em uma imagem digital monocromática, o valor do pixel é uma escalar entre $L_{mín}$ e $L_{máx}$, assim, para uma imagem que possui *multibandas* ou *multiespectrais* podem ser vistas como imagens nas quais cada pixel tem associado um valor vetorial $f(x,y)=L_1,L_2,...L_n$, onde $L_{mín} \leq L_i \leq L_{máx}$ e i=1,2,...,n.

a)

	590	591	592	593	594	595	596	597	598	599	x
650	128	135	137	135	145	145	151	170	169	161	
651	122	128	132	151	162	162	161	171	175	170	
652	116	122	133	154	163	165	163	170	177	172	
653	112	121	132	123	136	153	164	176	183	170	
654	113	119	129	119	121	137	157	169	180	170	
655	113	122	133	137	125	132	149	156	165	163	
656	112	127	141	133	119	132	151	150	151	148	
657	138	144	137	123	145	160	150	125	123	132	
658	132	136	130	124	149	156	136	112	126	133	
659	126	131	127	136	154	154	123	108	134	138	
660	119	119	116	140	145	138	111	103	129	135	
661	120	114	109	133	118	107	98	95	109	120	
662	137	134	131	130	104	97	105	102	103	113	

y

b)

Figura 7 Representação matricial. (a) imagem, (b) níveis de cinza correspondente à região da imagem em destaque.

Segundo (PEDRINI, SCHWARTZ, 2008) “uma imagem colorida é uma imagem multibanda ou multiespectral, onde a cor em cada ponto (x,y) é definida por meio de três grandezas: luminância, matiz e saturação. A luminância está associada com o brilho da luz, o matiz com o comprimento de onda dominante e a saturação com o grau de pureza da matiz. A maioria das cores visíveis ao olho humano pode ser representada por uma combinação de bandas das cores primárias, vermelha (R, red), verde (G, Green) e azul (B, blue). Uma representação comum para imagem colorida utiliza três bandas R, G e B com profundidade de 1 byte por pixel, para cada banda, ou seja, profundidade de 24 bits por pixel”. Modelos de representação de cores e imagem são discutidos no Anexo A.

Segundo (PEDRINI, SCHWARTZ, 2008) “a visão é uma das principais capacidades sensoriais dos seres humanos, que permite adequada percepção do ambiente que os cerca, envolvendo diversas funções complexas, tais como detecção, localização, reconhecimento e interpretação de objetos no ambiente. A área de visão computacional procura dotar as máquinas com capacidades visuais, como por exemplo, uma câmera fotográfica, onde o obturador da câmera possui função similar da pálpebra dos olhos, o diafragma da câmera controla a quantidade de luz que atravessa as lentes, similar à iris no olho humano, as lentes da câmera são análogas ao conjunto formado pelo cristalino e córnea, cujo objetivo comum é focalizar a

luz para tornar nítidas as imagens que serão formadas em uma superfície sensível, no olho humano a retina.”

Imagens reais frequentemente sofrem degradações durante seu processo de aquisição, transmissão ou processo. Essa degradação é normalmente chamada de *ruído*. Os tipos de ruído mais comumente modelados são o ruído *impulsivo*, *Gaussiano, uniforme*, *Erlang*, *exponencial*, *Rayleigh* e *Poissson* (PEDRINI, SCHWARTZ, 2008). Além deste, existem outros fatores que podem ocasionar degradações durante a aquisição como sobre ou sub exposição, contrate, luminosidade baixa, desfocagem, defeitos na lente entre outros.

2.2.2 Etapas de um Sistema de Processamento de Imagens

Um sistema de processamento digital de imagens é constituído de etapas ilustradas na Figura 8, capazes de produzir um resultado a partir do domínio do problema.

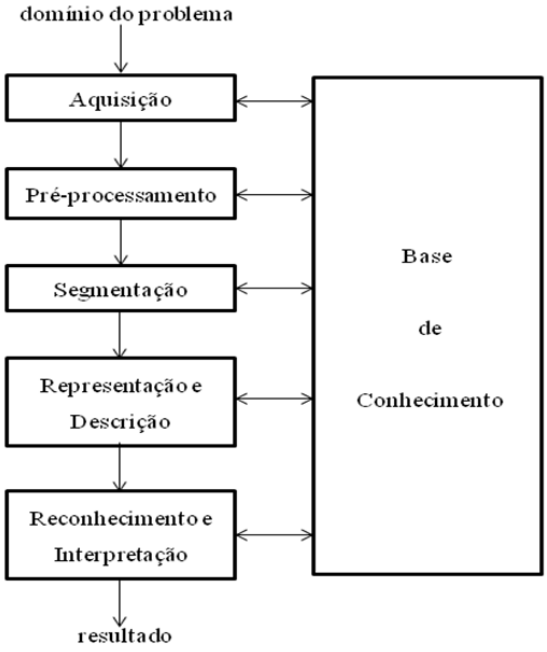


Figura 8 Etapas de um sistema de processamento de Imagens (PEDRINI, SCHWARTZ, 2008).

A etapa de *aquisição*, envolve a captura da imagem por meio de um dispositivo ou sensor, como por exemplo, uma câmera de vídeo, e a converte em uma representação adequada para o processamento digital subsequente. Estes dispositivos devem ser escolhidos levando em consideração o tipo de sensor, resolução e o número de níveis de cinza ou cores da imagem digitalizada.

A etapa de *pré-processamento* interfere na qualidade da imagem por meio de aplicações de técnicas para atenuação do ruído, correção de contraste ou brilho e suavização de determinadas propriedades da imagem.

A etapa de *segmentação* realiza a extração e identificação de áreas de interesse contidas na imagem, sendo baseada na detecção de descontinuidades, como bordas ou regiões da imagem por exemplo.

A etapa de *representação* e *descrição* tem como objetivo armazenar e manipular os objetos de interesse, extraídos da imagem e também na extração de características ou propriedades que possam ser utilizadas na discriminação entre classes de objetos.

A última etapa envolve o reconhecimento e interpretação dos componentes de uma imagem a partir do conhecimento do domínio do problema a ser abordado.

Todas as etapas requerem a utilização da base de conhecimento que é dependente da aplicação.

2.2.3 Realce de Imagens

O objetivo de realizar um realce na imagem é de destacar melhor as bordas e detalhes de uma dada imagem, que podem ter ocorrido degradações ocasionadas por ruídos, borramentos ou perda de contraste na aquisição.

Um dos algoritmos implementados utiliza um filtro para detectar as bordas, a fim de extrair informações das bordas da imagem, que são caracterizadas por mudanças abruptas de continuidade na imagem, calcula-se a transformação por derivada, que mede a variação da função, que é maior nas bordas e menor em áreas mais homogêneas.

A seguir será introduzido o conceito de máscaras empregadas no processamento de imagem. Tendo o processo de convolução como a base de operação matemáticas que combinam as duas matrizes, uma sendo a imagem e a outra uma máscara que determinará o efeito ou característica de operação desejada.

Para um melhor entendimento na análise de uma imagem, algumas técnicas envolvendo a caracterização da distribuição da intensidade dos pixels são importantes, através do histograma da imagem é possível obter esta informação.

2.2.4 Histograma

O *histograma* é uma representação através de um gráfico da distribuição dos níveis de cinza de uma imagem, indicando o número de pixels contida na matriz para cada nível de intensidade, demonstrando em um gráfico a relação de número de pixels versus intensidade do pixel ao longo dos eixos. A Figura 9 apresenta o histograma da imagem da Figura 10.

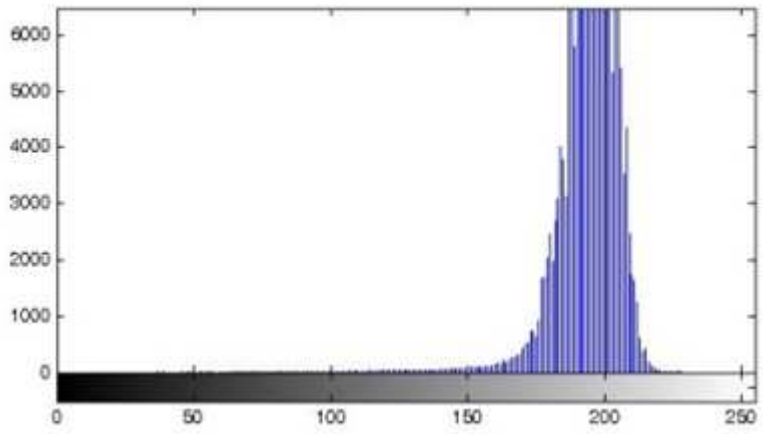


Figura 9 Histograma da Imagem da Distribuição de níveis de cinza..

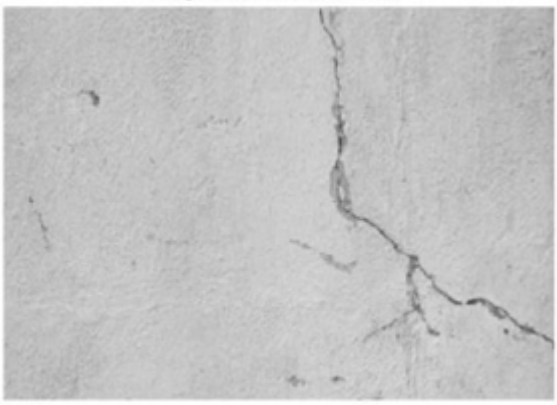


Figura 10 Imagem da representação da distribuição de níveis de cinza (PEDRINI, SCHWARTZ, 2008).

Considerando f(x,y) uma imagem representada por uma matriz bidimensional, com dimensões M x N pixels e contendo L níveis de cinza no intervalo [0, L_{max}], é possível obter o cálculo do histograma conforme apresentado no algoritmo 1, onde o histograma é representado por um vetor H com L elementos. O histograma pode ser visto como uma distribuição discreta de probabilidade, pois o número de pixels para um determinado nível de cinza pode ser utilizado para calcular a probabilidade de se encontrar um pixel com aquele valor de cinza na imagem.

Desta forma o histograma $p_k(f)$ pode ser expresso conforme a equação 2, onde $n_k = H(k)$ representa o número de ocorrências do nível de cinza k e $n = MN$ corresponde ao número total de pixels na imagem f. (PEDRINI, SCHWARTZ, 2008)

$$p_k(f) = \frac{n_k}{n} = \frac{H(k)}{MN} \quad (2)$$

Cálculo do histograma de uma imagem

1 // atribuir valor zero a todos os elementos do vetor

2 para i=0 até Lmax faça

3 $H[i] \leftarrow 0$

4 //calcular frequência de cada nível de cinza para todos pixeis da imagem

5 para $x = 0$ até $M - 1$ faça

6 para $y = 0$ até N -1 faça

7 $H[(f(x,y)] \leftarrow H[f(x,y)] + 1$

Algoritmo 1 – Cálculo do histograma de uma imagem (PEDRINI, SCHWARTZ, 2008).

2.2.5 Transformações de Imagens

As transformações podem ser lineares ou não-lineares, a transformação linear pode ser descrita pela equação 3, em que o parâmetro a é um valor fixo que controla a escala de níveis de cinza da imagem resultante e b ajusta seu brilho.

$$g = af + b \quad (3)$$

A transformação não linear pode ser descrita também pela equação 3, com o parâmetro a variável, podendo possuir operações baseadas em funções de logaritmos, raiz quadrada, exponencial e quadrado, por exemplo.

Uma função de transformação T de níveis de cinza é dada pela equação 4 em que f e g representam o nível de cinza dos pixels de uma imagem de entrada f e da imagem modificada g, respectivamente.

$$g = T(f) \quad (4)$$

Uma transformação não linear do modelo de cores RGB para a escala de cinza $C(x, y)$, pode ser realizada utilizando a transformação dada pela equação 4, compreendendo valores da intensidade do nível de pixel entre 0 e 255, sendo R representando o componente vermelho (Red), G o componente verde (Green) e B o componente azul (Blue).

$$C(x, y) = 0{,}2989 * R(x, y) + 0{,}5870 * G(x, y) + 0{,}1140 * B(x, y); \quad (5)$$

2.2.6 Filtragem de Imagem

Filtragens são operações realizadas no domínio do espaço ou no domínio da frequência. No *domínio espacial* as operações são realizadas diretamente no conjunto de pixels que compõe a imagem, na qual o nível de cinza de um ponto $f(x, y)$, depende do valor do nível de cinza original do ponto e dos outros pontos das vizinhanças de $f(x, y)$. A filtragem no domínio da frequência, é realizada através de transformação de Fourier, para posterior operação de filtragem pelo teorema da convolução.

Para o presente trabalho foram utilizadas técnicas de filtragem no domínio espacial.

No domínio espacial, o processo de filtragem normalmente é realizado por meio de matrizes denominadas *máscaras*, as quais são aplicadas sobre a imagem, cada posição da máscara está associada a um valor numérico, chamado de peso ou coeficiente. Sendo assim, a aplicação da máscara com centro na coordenada (x, y), sendo x a posição da coluna e y a posição de uma dada linha da imagem, consiste na substituição do valor do pixel na posição (x, y) por um novo valor, o qual depende dos valores dos pixels vizinhos e dos pesos da máscara, neste processo deve-se ter o cuidado de salvar estes valores cálculados gerando uma nova matriz, ficando o próximo cálculo realizado sempre com a matriz da imagem original para não acontecer de utilizar dos valores já operacionados. Os coeficientes do filtro são multiplicados pelos níveis de cinza dos pixels correspondentes e então somados, substituindo o nível de cinza do pixel central. A Figura 11 mostra uma máscara genérica de 3 x 3 pixels. Denotando os níveis de cinza da imagem sobre a máscara por $z_i = f(x, y), 1 \leq i \leq 9$, a resposta da máscara é dada pela equação 6, em que w_i representa os coeficientes da máscara (PEDRINI, SCHWARTZ, 2008). No caso de $w_1$ser negativo, deve-se ter cuidado para que o somatório não seja negativo, atribuindo um valor nulo para estes valores menores que zero.

$$R = w_1 z_1 + w_2 z_2 + \cdots + w_9 z_9 = \sum_{i=1}^{9} w_i z_i \quad (6)$$

w_1	w_2	w_3
w_4	w_5	w_6
w_7	w_8	w_9

Figura 11 Máscara de 3 x 3 pixels com coeficientes arbitrários.

Se o centro da máscara estiver em uma posição (x, y) na imagem, o nível de cinza do pixel posicionado em (x, y) será substituído por R, conforme equação 7. A máscara é então movida para a próxima posição de pixel na imagem e o processo se repete até que todas as posições de pixels tenham sido cobertas. A Figura 12 ilustra essa operação, em que a imagem e a máscara possuem dimensões $M \times N$ e $m \times n$ pixels, respectivamente (PEDRINI, SCHWARTZ, 2008).

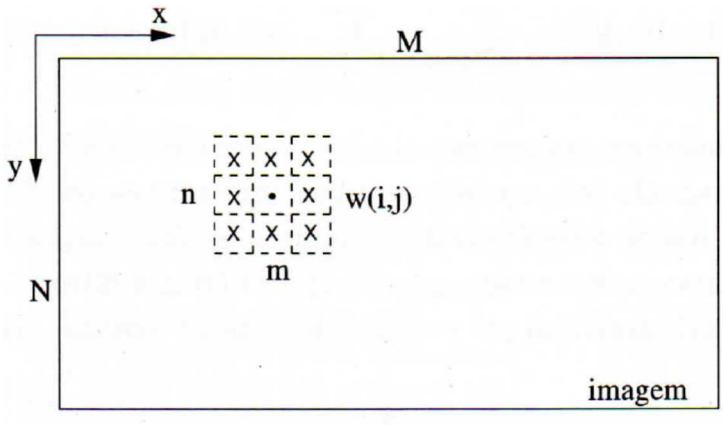


Figura 12 Filtragem no domínio espacial (PEDRINI, SCHWARTZ, 2008)

Para realizar as operações entre máscaras e imagem, é comum a utilização da operação por convolução, assim a equação 7 descreve uma convolução bidimensional na qual cada peso da máscara será refletido tanto na horizontal quanto na vertical. Sendo w a máscara ou filtro com dimensões $m \times n$ e f a imagem com dimensões $M \times N$ em pixels, e $g(x, y)$ a saída com dimensões $M \times N$ pixels.

$$g(x,y) = w(x,y) * f(x,y) = \sum_{i=-\frac{m}{2}}^{\frac{m}{2}} \sum_{i=-\frac{n}{2}}^{\frac{n}{2}} w(i,j) f(x-i, y-j) \quad (7)$$

O algoritmo 4.3 apresenta o processo de convolução de uma imagem.

Processo de convolução

1 $x1 \leftarrow [m/2]$

2 $y1 \leftarrow [n/2]$

3

4 para $x = 0$ até $M - 1$ faça

5 para $y = 0$ até $N - 1$ faça

6 soma $\leftarrow 0$

7 para $i = -x1$ até $x1$ faça

8 para $j = -y1$ até $y1$ faça

9 soma $\leftarrow$ soma $+ w(i,j) * f(x-i, y-j)$

10 $g(x,y) \leftarrow$ soma

Algoritmo 2 – Processo de convolução (PEDRINI, SCHWARTZ, 2008).

2.2.7 Segmentação de Imagens

A técnica de processamento de imagem ligada a segmentação tem como objetivo identificar corretamente a localização de determinada topologia ou forma de objetos de interesse. Trabalhos relacionados a segmentação de imagem buscam basicamente detectar descontinuidades ou similaridades na imagem, buscando mudanças abruptas nos níveis de cinza, caracterizadas nas bordas de regiões. A seguir serão descritas as técnicas empregadas no algoritmo de segmentação.

2.2.8 Detecção de Bordas

Uma borda é o limite ou a fronteira entre duas regiões com propriedades relativamente distintas de nível de cinza. São várias as técnicas de detecção de bordas, que estão associados aos cálculos de um operador local diferencial, utilizando a magnitude do gradiente no ponto ou pelo operador Laplaciano.

O gradiente pode ser utilizado para a diferenciação de imagem a fim de detectar mudanças significativas nos níveis de cinza da imagem. A equação 8 descreve o vetor gradiente $\nabla f(x,y)$ de uma imagem na posição (x,y) em que i e j são vetores unitários nas direções x e y, respectivamente.

$$\nabla f(x,y) = \frac{\partial f(x,y)}{\partial x} i + \frac{\partial f(x,y)}{\partial y} j \tag{8}$$

A identificação de pontos de borda baseada no conceito de gradiente é dada pelo algoritmo 3, onde deve ser aplicado um valor de limiar para indicar os valores da borda.

A forma matricial pode ser expressa pela equação 9.

$$\nabla f = \begin{bmatrix} G_x \\ G_y \end{bmatrix} = \begin{bmatrix} \dfrac{\partial f}{\partial x} \\ \dfrac{\partial f}{\partial y} \end{bmatrix} \tag{9}$$

A magnitude do vetor gradiente, utilizada para a detecção de borda, apresentada pela equação 10 equivale à maior taxa de variação de $f(x,y)$ por unidade de distância na direção de ∇f. Esta pode ser aproximada pela soma dos valores absolutos data pela equação 11, a fim de reduzir o custo computacional requerido pela equação 9, ou então pelo valor máximo entre os gradientes na direção x e y dado pela equação 12.

$$\nabla f = mag(\nabla f) = \sqrt{\mathrm{G_x^2} + \mathrm{G_y^2}} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{10}$$

$$\nabla f \approx |G_x| + |G_y| \tag{11}$$

$$\nabla f \approx \max\left(|G_x|, |G_y|\right) \tag{12}$$

Determinação de pontos de borda em uma imagem

1 para $x = 0$ até $M - 1$ faça

2 para $y = 0$ até $N - 1$ faça

3 //calcular a magnitude do gradiente $\nabla f(x, y)$

4 $\nabla f(x, y) \leftarrow \sqrt{\left(\frac{\partial f(x,y)}{\partial x}\right)^2 + \left(\frac{\partial f(x,y)}{\partial y}\right)^2}$

5 //efetuar a limiarização

6 se $\nabla f(x, y) > T$ então (x, y) é um ponto de borda

Algoritmo 3 – Determinação de pontos de borda em uma imagem (PEDRINI, SCHWARTZ, 2008).

$$\nabla f = mag(\nabla f) = \sqrt{\mathrm{G_x^2} + \mathrm{G_y^2}} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{10}$$

$$\nabla f \approx |G_x| + |G_y| \tag{11}$$

$$\nabla f \approx \max(|G_x|, |G_y|) \tag{12}$$

O ângulo da direção do vetor gradiente ∇f na posição (x, y) pode ser obtido pela formula da equação 13, obtendo um ângulo θ, que podendo ser útil para indicar a direção da linha de borda de uma imagem em um determinado ponto, que é perpendicular a θ

$$\theta(\mathrm{x}, \mathrm{y}) = \arctan\left(\frac{G_y}{G_x}\right) \tag{13}$$

Na Figura 13 estão alguns exemplos de detecção de bordas com diferentes operadores de gradiente, descritos na literatura de processamento de imagem (PEDRINI, SCHWARTZ, 2008; GONZALEZ, WOODS, 2002).

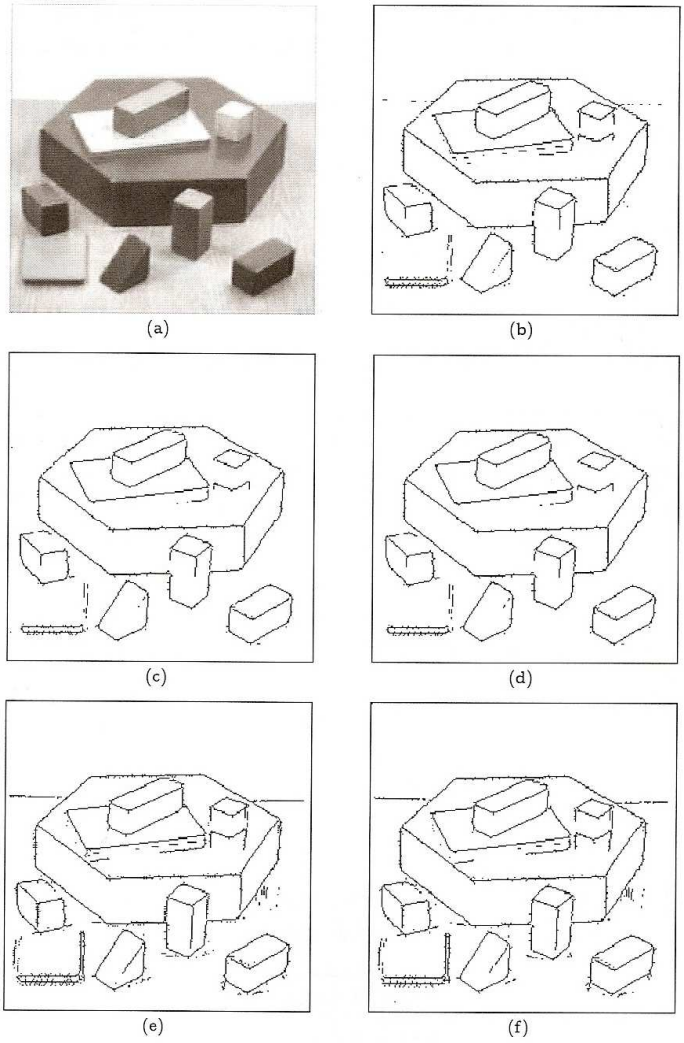


Figura 13 Detecção de bordas por meio de diferentes operadores de gradiente. (a) imagem original; (b) Roberts; (c) Prewitt; (d) Sobel, (e) Kirsch; (f) Frei-Chen (Pedrini, Schwartz, 2008).

No presente trabalho foi empregada a detecção de borda utilizando o operador (SOBEL I. 1990), pelas características observadas de realce nas linhas verticais e horizontais mais escuras que o fundo, sem realçar pontos isolados, tendo maior eficácia na detecção de fissuras do que os outros operadores analisados.

O operador (SOBEL, 1990), utiliza basicamente derivadas parciais calculadas separadamente na direção de duas coordenadas espaciais x e y e posterior combinação dos resultados. A equação 14 apresenta o operador que aproxima a magnitude do gradiente como a diferença de valores ponderados dos níveis de cinza da imagem nas duas direções.

$$G_x \approx [f(x+1,y-1) + 2f(x+1,y) + f(x+1,y+1)] - [f(x-1,y-1) + 2f(x-1,y) + f(x-1,y+1)]$$
$$G_y \approx [f(x-1,y+1) + 2f(x,y+1) + f(x+1,y+1)] - [f(x-1,y-1) + 2f(x,y-1) + f(x+1,y-1)] \quad (14)$$

As máscaras para o operador de Sobel podem ser implementadas como mostrado na Figura 14, onde os níveis de cinza dos pixels, de uma região, são sobrepostos pelas máscaras centradas no pixel (x, y) da imagem.

$G_x =$

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

$G_y =$

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

Figura 14 – Máscara de 3 x 3 pixels com os respectivos pesos do operador Sobel (PEDRINI, SCHWARTZ, 2008).

2.2.9 Limiarização

A limiarização é de grande importância para segmentação de imagens, pois utiliza um valor, chamado de limiar, que corresponde ao nível de cinza que melhor separa a escalas de cinza de uma imagem. O limiar pode ser utilizado, por exemplo, na detecção de borda, sendo um elemento de referência importante como observado no algoritmo 3.

Uma imagem com o processo de limiarização pode ser descrito pela equação 15, onde os pixels da imagem de saída $g(x, y)$ acima do limiar T receberão o valor lógico 1, enquanto abaixo do limiar receberão o valor lógico 0 que corresponderão ao fundo da imagem.(GONZALEZ, WOODS, 2002).

$$g(x,y) = \begin{cases} 1 \text{ se } f(x,y) > T \\ 0 \text{ se } f(x,y) < T \end{cases} \quad (15)$$

Podem ser utilizados os próprios valores dos níveis de cinza da imagem original, ao invés de valores lógicos, a partir do limiar, preservando as características da imagem original acima deste limiar. A equação 16 descreve está situação.

$$g(x,y) = \begin{cases} f(x,y) \text{ se } f(x,y) > T \\ 0 \quad\quad \text{se } f(x,y) < T \end{cases} \tag{16}$$

Um limiar T também pode ser atribuído por um valor fixo arbitrário, este tipo de parâmetro pode se comportar como esperado para determinados tipos de imagens que possuem características específicas na intensidade de pixels.

Um parâmetro fixo de limiar T independente da imagem não é uma boa solução, pois o processo de limiarização fica dependente da maneira que a imagem foi capturada. Uma pequena variação de luminosidade durante a captura faz com que haja o descarte de valores da escala de cinza que podem ser importantes no processo. Alguns métodos propostos, visão utilizar características da própria imagem para atribuir o valor da limiar T. Estes métodos procuram otimizar o processo de limiarização, como por exemplo, a *limiarização global, que* utiliza todas as intensidades de pixels da imagem ou uma *limiarização local,* que usa uma ou mais regiões da imagem para determinar limiares locais.

O método proposto por (RIDLER, CALVARD, 1978), que parte da utilização da limiarização global, seleciona de forma iterativa o valor de limiar de uma imagem, a qual se baseia na combinação de modelos de duas distribuições Gaussianas. O método inicia com uma estimativa inicial para o limiar, após refina o cálculo baseado na média das distribuições dos níveis de cinza do objeto (μ_1) e do fundo (μ_2), até que esses valores não sofram alterações em iterações sucessivas pela diferença entre limiares T, ficando $|T_{i+1} - T_i|$ suficientemente pequeno na iteração i e i+1. O algoritmo 4 descreve os passos deste método.

Limiarização Global Iterativa

1 Estimar um valor inicial $T = T_0$, para o limiar, por exemplo, igual à intensidade média da imagem.
2 Particionar a imagem em duas regiões, R_1 e R_2, com o limiar T, tal que todos os pixels com intensidade menor T sejam atribuídos à região R_1 e todos os outros sejam atribuídos à região R_2.
3 Na iteração i, calcular valores de intensidade média μ_1 e μ_2 associados às partições R_1 e R_2.
4 Calcula um novo limiar como $T = T_{i+1} = \frac{\mu_1^i + \mu_2^i}{2}$.
5 Repetir os passos (2) a (4) até que o limiar T não sofre alteração significativa em iterações sucessivas, ou seja, $T_{i+1} \approx T_i$.

Algoritmo 4 – Limiarização Global iterativa (PEDRINI, SCHWARTZ, 2008).

2.2.10 Divisão de Regiões

A divisão de regiões consiste na separação dos objetos presentes em uma imagem. Estes objetos são definidos por sua conectividade com pixels existentes em sua vizinhança. Esta conectividade pode ser obtida na analise de todos os 8 pixels em seu entorno, ou através de 4 conexões através de pixels na sua horizontal ou vertical.

Para ilustrar esse conceito, a imagem mostrada na Figura 15, é representada por uma matriz binária de 2^3 x 2^3, onde a Figura 15(a) apresenta o pixel verde que está conectado ao pixel vermelho em contectividade 4 e na Figura 15(b) corresponde a conectividades de 8 pixels na vizinhança do pixels representado na cor vermelha.

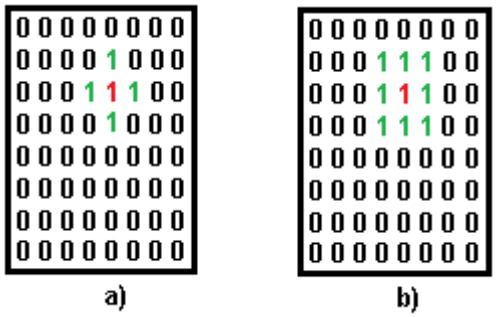


Figura 15 a) Um pixel binário representado em vermelho e 4 conectividades de pixels binários representado em verde. b) Um pixel binário representado em vermelho e 8 conectividades de pixels binários representado em verde.

No processo de separação dos objetos é necessária uma indicar sua conectividade, e inserir um valor de identificação representando então o objeto. Em (HARALICK, 1992) temos uma abordagem da separação e identificação dos objetos demonstrados a seguir: A primeira passagem atribui valores sequenciais temporários para cada pixel no plano da imagem com a verificação de suas vizinhanças.

Para ilustrar este processo, a Figura 16 está uma ilustração gráfica de uma matriz que representa uma imagem binária de 8 x 6 pixels.

Imagem binária

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

Figura 16 Representação de imagem binária 8 x 6 pixels.

A varredura pode iniciar tanto em linha como em coluna. Para este exemplo se inicia na primeira coluna, e sequencia-se para cada linha até o final da linha da primeira coluna, então se comuta para a próxima coluna, assim sucessivamente até a última coluna. Durante o processo de varredura na matriz, ao encontrar um pixel binário igual a 1(um), verifica-se a existência de pixel binário na vizinhança cuja varredura já tenha passado. A Figura 17 mostra os pixels destacados, onde se verifica a existência do pixel de vizinhança, caso não exista nenhum pixel é então atribuído um valor de referência sequencial.

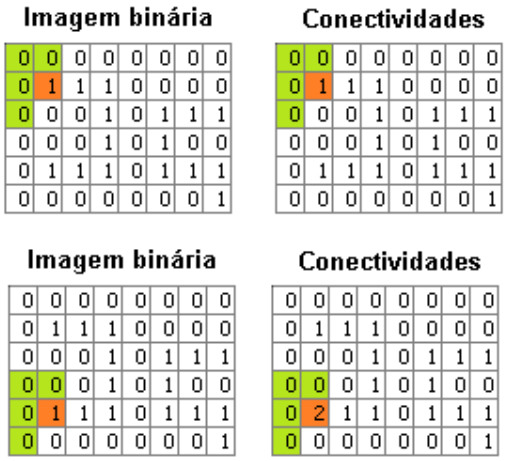


Figura 17 Varredura de pixels e verificação na vizinhança, caso não há pixel na vizinha é atribuído valor de identificação sequêncial.

Se na sequência de varredura ocorrer um pixel binário na vizinhança, o pixel de varredura receberá o mesmo valor de identificação do pixel de sua vizinhança. Na Figura 18, está a representação deste comportamento, no qual a varredura atribui o mesmo valor de referência do pixel de sua vizinhança que já tenho sido atribuído em uma varredura anterior.

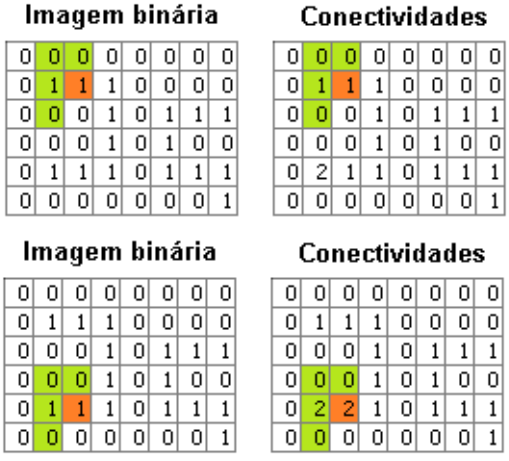


Figura 18 Atribuição de valor de identificação quando há pixels binários existente na vizinhança.

Agora, se durante a varredura de um determinado pixel, forem encontrados em sua vizinhança pixels cujos valores atribuídos nas varreduras anteriores forem diferentes entre si, se atribui ao pixel atual de varredura o menor valor de referência, e monta-se uma tabela indicando a conectividade de ambos os valores de referência neste caso. A Figura 19 apresenta este comportamento durante o processo de varredura, atribuindo ao pixel atual de varredura o menor valor de referência.

Figura 19 Atribuição de valor de identificação quando há pixels binários existente na vizinhança com valores distintos, atribui-se o menor valor ao pixel de varredura atual.

Neste processo inicial de varredura, a Figura 20 apresenta o resultado destas atribuições.

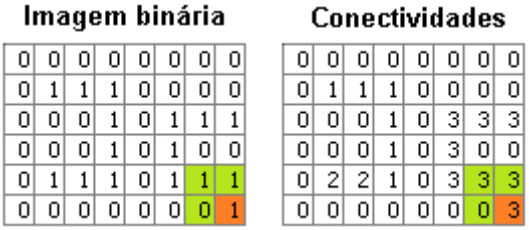


Figura 20 Resultado dos valores atribuídos após a primeira varredura.

O processo continua, utilizando a tabela de valores cuja vizinhança são semelhantes, neste exemplo os valores são “1” e “2”. Assim atribui-se os valores de referência iguais a “1” aqueles que possuem valores de referência igual a “2”.

Após o processo de atribuição a partir da tabela de valores e realizado um novo sequenciamento dos valores de referência obtendo assim o resultado final com os objetos separados identificados. A Figura 21 ilustra o resultado final para este exemplo.

Conectividades

0	0	0	0	0	0	0	0
0	1	1	1	0	0	0	0
0	0	0	1	0	2	2	2
0	0	0	1	0	2	0	0
0	1	1	1	0	2	2	2
0	0	0	0	0	0	0	2

Figura 21 Resultado dos valores atribuídos após sequenciamento final.

2.2.11 Morfologia Matemática

As operações de morfologia matemática utilizam-se da teoria de conjuntos para representar a forma dos objetos em uma imagem, como a união, a intersecção e a complementação. As operações básicas da morfologia digital são a dilatação e a erosão que podem ser realizadas para imagens binárias, em tons de cinza ou coloridas. (PEDRINI, SCHWARTZ, 2008)

Uma imagem binária pode ser representada por uma coleção de coordenadas discretas que correspondem aos pontos pertencentes ao objeto na imagem. Os objetos em uma imagem podem ser representados por pixels pretos (valor 1), enquanto o fundo por pixels brancos (valor 0), assim um conjunto $\{(x, y)|f(x, y) = 1\}$ expressa um objeto da imagem definido no espaço bidimensional dos números inteiros $\mathbb{Z}^2$, em que cada elemento do conjunto é um vetor bidimensional com coordenadas (x, y). A Figura 22 ilustra uma imagem binária 7 x 9 onde os pixels pertencentes aos objetos da imagem podem ser representados pelo conjunto $A = \{(3,2), (4,1), (4,2), (4,3), (5,1), (5,2), (5,3), (5,4), (6,2), (6,4), (6,5), (7,5)\}$

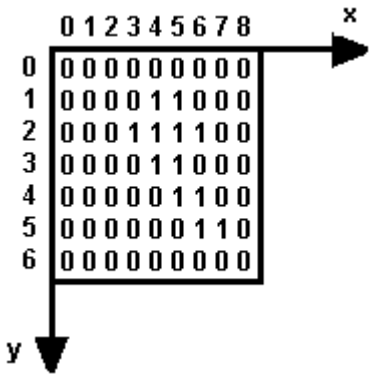


Figura 22 Representação de imagem pelo conjunto de pixels pertencentes ao objeto da imagem binária

Nas operações morfológicas, envolvendo o conjunto da imagem, comumente é utilizando um elemento estruturante que também é definido em $\mathbb{Z}^2$.A Figura 23 mostra alguns elementos estruturantes, onde a cruz (+) simboliza o ponto de origem b(0,0) e os pontos que pertencem ao objeto são marcados com círculos escuros, observando que a origem não é necessariamente um elemento de b (PEDRINI, SCHWARTZ, 2008).

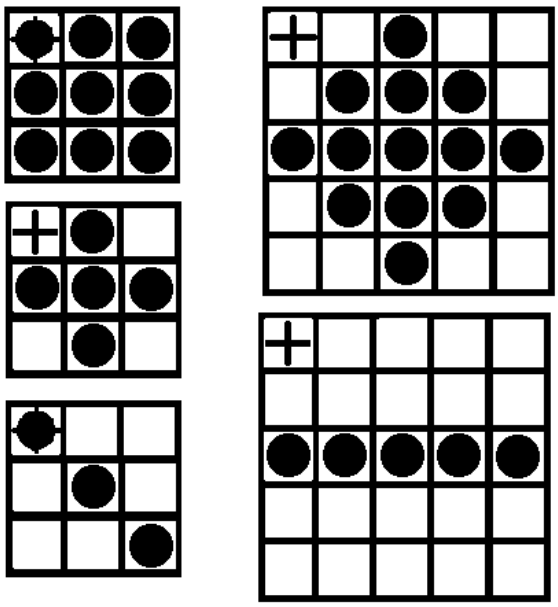


Figura 23 Exemplos de elementos estruturantes

Pela teoria dos conjuntos, seja o conjunto A no espaço $\mathbb{Z}^2$ com componentes definidos como $a = (a_1, a_2)$ representando um objeto da imagem, e B no espaço $\mathbb{Z}^2$ com componentes definidos como $b = (b_1, b_2)$. Assim seja um conjunto A que denota o conjunto de elementos da imagem, B é o conjunto de um elemento estruturante.

As operações morfológicas podem ser realizadas por meio da álgebra de Minkowski, (homenagem ao matemático russo Hermann Minkowski (1864-1909)). A definição de soma e subtração representada pelas equações 17 e 18, respectivamente. $\hat{B}$ é o conjunto dos elementos para todos os elementos $-b \in \hat{B}$, o mesmo que "reflexão" de B. (PEDRINI, SCHWARTZ, 2008)

$$\text{Adição} \qquad A \oplus B = \bigcup_{b \in B}(A + b) = \bigcup_{\substack{a \in A \\ b \in B}}(a + b) \tag{17}$$

$$\text{Subtração} \quad A \ominus B = \bigcap_{b \in B}(A - b) = \bigcap_{b \in \hat{B}}(A + b) \tag{18}$$

A adição de $A \oplus B$ é realizada pela translação de A com cada elemento de B, tomando-se a união de todas as translações resultantes. A subtração equivale a intersecção de todas as translações de A pelo elemento $-b \in \hat{B}$.

A seguir são apresentadas três operações morfológicas em imagem binária, utilizadas neste trabalho para o processamento de imagem: a dilatação, erosão e fechamento.

2.2.12 Dilatação de Imagem Binária

Para a operação de dilatação ($\mathcal{D}$) entre o conjunto A que representa o objeto imagem e o elemento estruturante B é realizada através da adição de Minkowski data pela equação 19. (PEDRINI, SCHWARTZ, 2008)

$$\mathcal{D}(A,B) = A \oplus B = \bigcup_{b \in B}(A + b) = \{\mathrm{p} \in \mathbb{Z}^2 | \mathrm{p} = \mathrm{a} + \mathrm{b}, \exists\ \mathrm{a} \in \mathrm{A}\ \mathrm{e}\ \exists\ \mathrm{b} \in \mathrm{B} \qquad (19)$$

Assim o processo de dilatação entre A e B, corresponde ao conjunto de todas as translações de B com os pontos da imagem em que há pelo menos um elemento não nulo (pixel com valor 1) em comum com o conjunto A. A Figura 24 representa um conjunto $A \subset \mathbb{R}^2$ com elemento estruturante B resultando na dilatação da imagem.

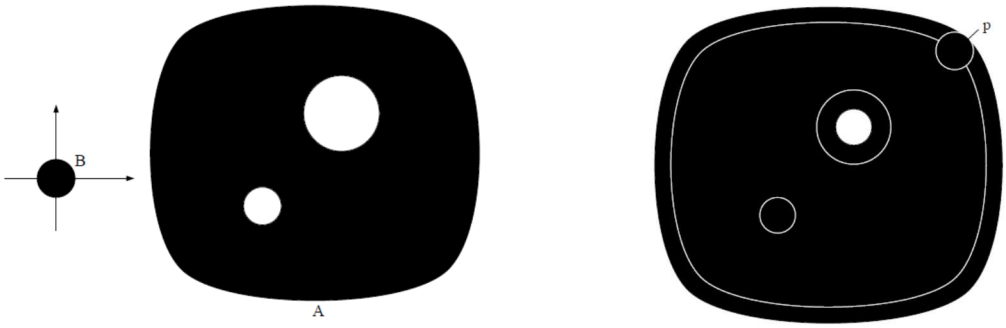


Figura 24 Representação de uma imagem $A \subset \mathbb{R}^2$ com elementro estruturante B e a Operação de Dilatação $A \oplus B$ (RITTER, WILSON, 2010)

Para melhor ilustrar o efeito da operação de dilatação para imagens binárias a Figura 25 apresenta o resultado da operação onde os cojuntos A representando o objeto imagem $A = \{(3,2), (4,2), (4,3), (5,2), (5,4), (6,1), (6,2), (7,1)\}$ e seja B representando o elemento estruturante $B = \{(0,0), (0,1), (1,0), (1,1)\}$, a operação resulta no conjunto de pixels da

imagem binária $\mathcal{D}(A,B)$=$\{(3,2),(3,3),(4,2),(4,3),(4,4),(5,2),(5,3),(5,4),(5,5),(6,1),(6,2),(6,3),(6,5),(7,1),(7,2),(7,3),(8,1),(8,2)\}$.

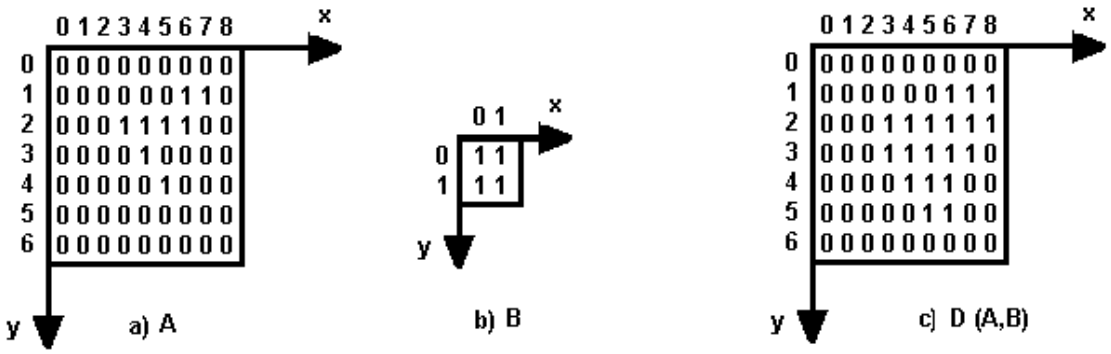


Figura 25 Operação de Dilatação $A \oplus B$

2.2.13 Erosão de Imagem Binária

Para o processo de *erosão* ($\mathcal{E}$) da imagem é utilizada a operação da subtração de Minkowski data pela equação 20. (PEDRINI, SCHWARTZ, 2008)

$$\mathcal{E}(A,B) = A \ominus B = \bigcap_{b\in B}(A-b) = \bigcap_{b\in \hat{B}}(A+b) \quad = \{\mathfrak{p}\epsilon\, \mathbb{Z}^2 | \mathrm{B}+\mathfrak{p} \subseteq \mathrm{A}\} \quad (20)$$

A equação 20 descreve que a erosão de A por B é o conjunto de todos os elementos de B transladados por $\mathfrak{p}$ que estão contidos em A. A Figura 26 está a representação da erosão.

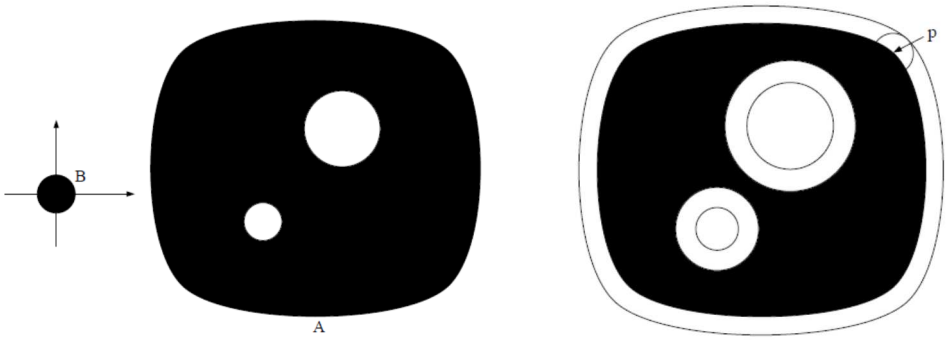


Figura 26 Representação de uma imagem $A \subset \mathbb{R}^2$ com elementro estruturante B e a Operação de Erosão $A \ominus B$ (RITTER, WILSON, 2010)

Para imagens binárias a operação de erosão está representada na Figura 27, onde o conjunto A representando o objeto imagem $A = \{(3,2), (4,2), (4,3), (5,2), (5,3), (5,4), (6,1),$

$(6,2), (6,3), (6,4), (7,1), (7,4)\}$ seja B representando o elemento estruturante $B = \{(0,0)(0,1)(1,0)(1,1)\}$, a operação resulta no conjunto de pixels da imagem binária $\mathcal{D}(A, B) = \{(4,2), (5,2), (5,3)\}$.

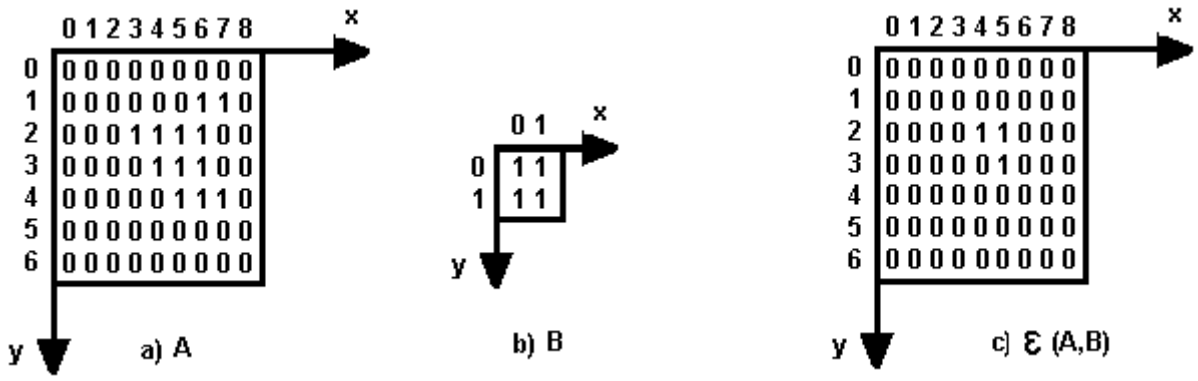


Figura 27 Operação de Erosão $A \ominus B$

É importante saber que o resultado da erosão de uma imagem pode não ser um subconjunto da imagem original, caso o elemento estruturante não contenha a origem. Desta forma os pixels que não corresponderem ao padrão definido pelo elemento estruturante não pertencerão ao resultado, sendo que a origem do elemento estruturante não precisa necessáriamente corresponder a um pixel de valor igual a 1 na imagem. A Figura 28 apresenta o resultado quando o elemento estruturante possui elemento nulo na origem.

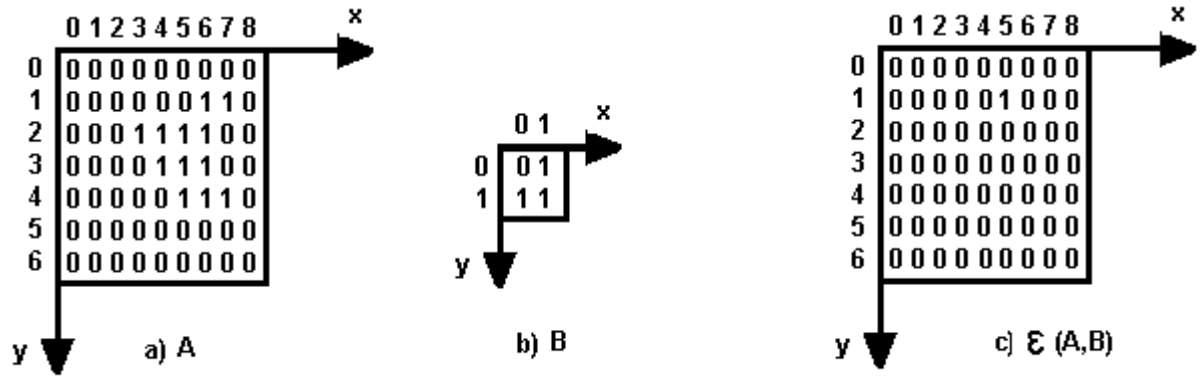


Figura 28 Operação de Erosão $A \ominus B$ para elemento estruturante com elemento nulo na origem.

2.2.14 Fechamento de Imagem Binária

O fechamento de imagem binária é realizado pela operação sequencial da dilatação entre A e B seguida de uma erosão do resultado por B, a equação 21 representa o processo de fechamento (PEDRINI, SCHWARTZ, 2008). Na Figura 29 está a representação do comportamento do fechamento da imagem na matriz binária, adicionando pixels na posição f(4,3) e f(5,3).

$$A \odot B = \mathcal{E}(\mathcal{D}(A,B),B) \tag{21}$$

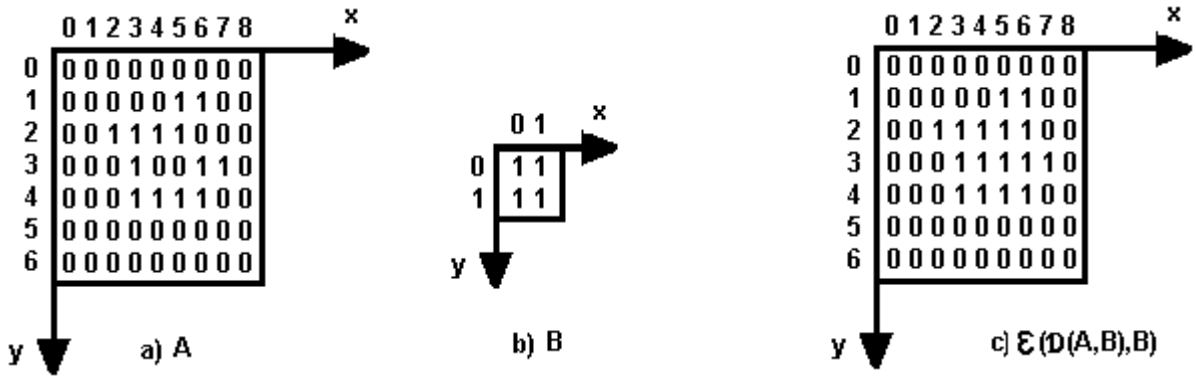


Figura 29 Operação de Preenchimento envolvendo operação de dilatação seguido de erosão $\mathcal{E}(\mathcal{D}(A,B),B)$.

2.2.15 Descrição de Imagem

Uma solução para análise de imagens é a seleção de um conjunto de características que podem ser extraídas de um objeto de interesse para sua classificação. Após a imagem ter sido segmentada em regiões ou objetos, os agrupamentos resultantes de pixels podem ser representados e descritos. Em um objeto a ser descrito, a visualização de sua forma contém informações importantes.As próximas seções apresentam um conjunto de técnicas para representação e descrição de objetos segmentados das imagens.

2.2.16 Diâmetro

O diâmetro é a maior distância entre dois pontos de um objeto, sendo invariante enquanto à sua rotação e translação. Assim seja uma borda B e os dois pontos p_i e p_j, pertencentes à borda então o diâmetro D apresentada pela equação 22 é uma medida de distância entre estes dois pontos.

$$\text{Diâmetro(B)}=\max_{i,j}[D(p_i,p_j)] \tag{22}$$

O diâmetro pode ser representado como um eixo maior de uma figura geométrica. Uma técnica para se obter este eixo maior é através dos pontos de uma aproximação poligonal do objeto. Com os pontos obtidos das extremidades do objeto, é realizado o cálculo das distâncias dos pontos dos extremos do objeto. O eixo maior será a maior distância entre eles. Assim sejam os pontos $p_1(x_1,y_1), p_2(x_2,y_2), p_3(x_3,y_3)$ *e* $p_4(x_4,y_4)$, representados na Figura 30, o cálculo da distância é apresentada na equação 23.

$$D = \sqrt{(x_{n1}-x_{n2})^2+(y_{n1}-y_{n2})^2} \tag{23}$$

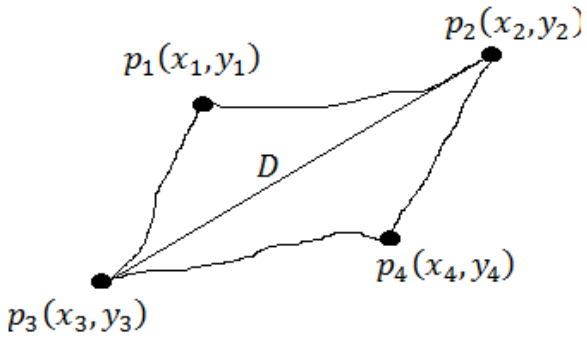


Figura 30 Aproximação Poligonal do Eixo Maior.

2.2.17 Área

A área pode ser expressa como o número de pixels que compreende o objeto ou região de interesse. Em uma imagem binária, a área de um objeto contido em um retângulo com dimensões de m x n pixels, em que os pixels f(x,y) do objeto são representados pelo valor 1, é definida pela equação 24

$$\text{Área} = \sum_{x=0}^{m-1}\sum_{y=0}^{n-1} f(x,y) \tag{24}$$

Embora a área possa ser utilizada como descritor de região, ela é geralmente aplicada a situações em que a escala dos objetos não varia. A área é invariante quanto a translação e rotação do objeto. (PEDRINI, SCHWARTZ, 2008).

2.3 CLASSIFICAÇÃO DE PADRÕES

A classificação de padrões visa determinar um mapeamento que relacione as propriedades extraídas de amostras com um conjunto de rótulos, apresentando a restrição de que amostras com características semelhantes devem ser mapeadas ao mesmo rótulo.

Quando se atribui um mesmo rótulo a amostras distintas, diz-se que tais elementos pertencem a uma mesma classe, esta é caracterizada por compreender elementos que compartilham propriedades em comum.

As abordagens para classificação de padrões dividem-se normalmente em duas categorias: abordagem sintática ou estrutural e abordagem estatística. Enquanto a abordagem sintática baseia-se na relação existente entre primitivas que compõe as amostras, a abordagem estatística considera que as amostras são obtidas de maneira independente a partir de uma distribuição de probabilidade fixa, porém, desconhecida. (PEDRINI, SCHWARTZ, 2008).

2.3.1 Estimação de Parâmetros

Estimação de parâmetros é um método de classificação supervisionado que visa obter características semelhantes de objetos através de amostras contidas no conjunto de treinamento que descrevem este objeto.

Com uma descrição dos objetos pode-se verificar a existência de margem de valores que podem compreender objetos que possuem formas semelhantes, descrevendoassim uma característica em comum.

2.3.2 Classificador Bayesiano

O Classificador Bayesiano é baseado na teoria de Thomas Bayes trata da probabilidade de acontecimento de eventos, alterando uma estimativa inicial com base em novas informações.

Seja um conjunto de amostras rotuladas como pertencentes a uma dentre m classes distintas, $\omega_1, \omega_2, \dots, \omega_m$, uma possível atribuição dos rótulos é dada de modo que cada amostra pertença à classe que maximize a probabilidade $P(\omega_i|x)$, para $i = 1,2, \dots, m$, ou seja, dado o vetor de características x, atribui-se a amostra à classe ω_i que apresenta a maior probabilidade condicional. (PEDRINI, SCHWARTZ, 2008).

Conforme a descrição sobre a atribuição dos rótulos, a classificação de uma amostra específica segue a regra de decisão mostrada na equação 25. Nesta regra, x é atribuído à classe ω_i caso $P(\omega_i|x)$, denominada probabilidade *a posteriori*, seja maior que qualquer outra $P(\omega_j|x)$. (PEDRINI, SCHWARTZ, 2008).

$$P(\omega_i|x) > P\left(\omega_j\middle|x\right) \quad j = 1, \dots m; i \neq j \tag{25}$$

Ambos os lados da regra de decisão da equação 25 podem ser avaliados por meio do teorema de Bayes, mostrado na equação 26, em que $P(x|\omega_i)$ denota a probabilidade de ocorrer x, dado que a amostra pertence à classe ω_i e $P(\omega_i)$ é denominada probabilidade *a priori*, ou seja, a probabilidade incondicional de ocorrência de uma determinada classe (PEDRINI, SCHWARTZ, 2008).

$$P(\omega_i|x) = \frac{P(x|\omega_i) * P(\omega_i)}{P(x)} \tag{26}$$

A substituição da equação 26 em 25 resulta na equação mostrada na equação 27, conhecida como *regra de Bayes para taxa mínima de erro*. Essa regra particiona o espaço de características nas regiões $R_1, R_2, \dots, R_m$, tal que, se $x \in R_i$ deve ser atribuído à classe ω_i. Dado que a probabilidade $P(x)$ do denominador da equação 26 é termo comum para todas as classes, esse é removido da equação 27 (PEDRINI, SCHWARTZ, 2008).

$$P(x|\omega_i) * P(\omega_i) > P\left(x|\omega_j\right) * P(\omega_{\mathrm{j}}) \quad j = 1, \dots m; i \neq j \tag{27}$$

Considerando a existência de apenas duas classes, pode-se transformar a equação 27 na razão mostrada na equação 28, denominada *razão de verossimilhança*. Utiliza-se essa equação como função discriminante entre as duas classes, ou seja, a atribuição de x passa a depender de sua localização espacial no espaço de características (PEDRINI, SCHWARTZ, 2008).

$$l_r(x) = \frac{P(x|\omega_1)}{P(x|\omega_2)} > \frac{P(\omega_2)}{P(\omega_1)} \quad implica\ que\ x\ \in\ \omega_1 \tag{28}$$

Com a regra de decisão particionada, o espaço de características nas regiões $R_1, R_2, \ldots, R_m$ denominadas regiões de decisão, dentro das quais uma determinada classe apresenta maior probabilidade de ocorrência, enquanto em suas fronteiras, denominadas *fronteira de decisão*, a ocorrência de classes distintas é equiprovável. Pode-se definir as fronteiras de decisão entre duas classes igualando-se as probabilidades *a posteriori*, conforme mostra a equação (29) (PEDRINI, SCHWARTZ, 2008).

$$P(\omega_1)P(x|\omega_1) = P(\omega_2)P(x|\omega_2) \tag{29}$$

A Figura 31 exemplifica um espaço de características unidimensional dividido em três regiões, $R_1, R_2\ e\ R_3$, representando as classes $\omega_1, \omega_2\ e\ \omega_3$, respectivamente. A fronteira de decisão ocorre em pontos nos quais duas distribuições *a posteriori* são equiprováveis. Dessa maneira, para efetuar a classificação de uma amostra x, deve-se determinar a qual região de decisão esta pertence. Por exemplo, uma amostra representada pelo vetor de características $x = [3]$ deve ser atribuída à classe ω_2, desde que x localize-se dentro da região de decisão R_2 (PEDRINI, SCHWARTZ, 2008).

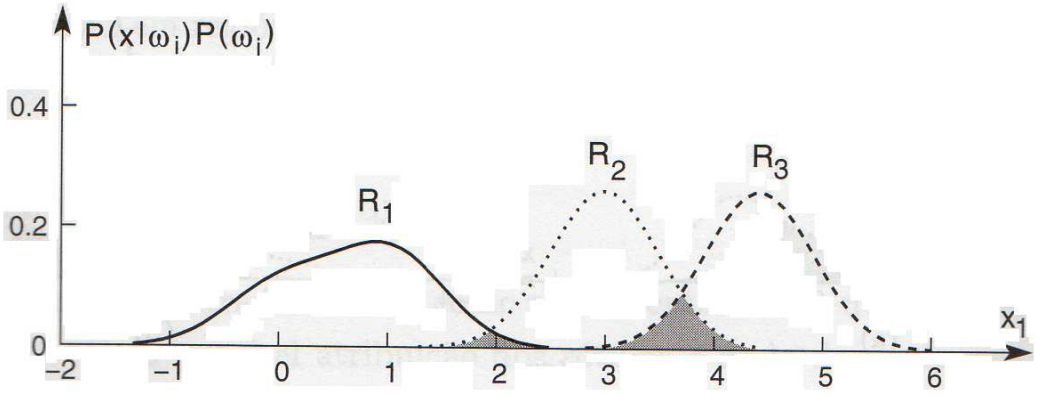


Figura 31 Regiões e fronteiras de decisão para aproximação das distribuições de probabilidade *a posteriori*. (PEDRINI, SCHWARTZ, 2008).

Alguns filtros de Bayes não paramétricos dependem de uma decomposição do espaço de estados, em que cada espaço os valores, corresponde à probabilidade cumulativa da densidade posterior em uma sub-região compacta do espaço de estado. Outros aproximam dos

espaços de estado por amostras aleatórias retiradas de uma distribuição anterior. Em todos os casos, o número de parâmetros utilizados para aproximar os valores posteriores pode ser variado. A qualidade da aproximação depende dos parâmetros numéricos utilizados para representar esta parte posterior. Como o número de parâmetros vai para o infinito, as técnicas não paramétricas tendem a convergir uniformemente para o valores posteriores corretos sob premissas de uniformidade específica. O filtro de partículas é uma implementação alternativa não paramétrica do filtro de Bayes pela qual os valores posteriores de uma amostra aproximam-se por um número finito de parâmetros. A ideia-chave do filtro de partículas é representar um estado posterior de amostras por um conjunto de estado com amostra aleatória extraídas deste estado posterior. Assim os filtros de partículas representam uma distribuição de um conjunto de amostras retiradas dessa distribuição. Em THRUN (2006) descreve-se os passos para implementação deste filtro.

2.4 LINGUAGEM DE MODELAGEM UNIFICADA

O Unified Modeling Language (UML) é uma linguagem de modelagem que tem sido utilizado pela sua potencialidade principalmente na programação orientada a objetos, contribuindo e beneficiando paradigmas do desenvolvimento de projetos desta natureza, especificando cenários através de modelagem de casos de uso, que aborda de forma funcional sem preocupação com sequências das ações de um sistema, de decisões ou as estruturas dos objetos. Os requisitos elaborados a partir desta linguagem auxiliam no entendimento entre uma equipe de um projeto, produzindo uma documentação que especifica e visualiza um sistema orientado a objeto, além de fornecer informações importantes, como por exemplo, informações de entrada e saídas envolvendo sistemas existentes entre outras. (BITTNER, 2003)

A organização OMG – Object Management Group é responsável por normatizar esta linguagem que tem sido adotada por boa parcela da indústria para descrever graficamente um sistema. A linguagem oferece diagramas conceituais que ajudam na especificação de um projeto. No escopo desta dissertação foram usados principalmente 3 diagramas UML: diagrama de casos de uso (¨use case diagrams¨), diagramas de classe (¨class diagrams¨) e diagramas de sequência (¨sequence diagrams¨), os quais serão detalhados na sequência.

O diagrama de casos de uso oferece uma visualização geral do sistema, sendo o mais utilizado no diálogo com o usuário, apresentando uma linguagem simples e de fácil compreensão, de modo a descobrir os principais elementos que vão estruturar o sistema,

validando assim sua estrutura inicial. O principal aspecto tratado neste diagrama são as funções do sistema. A Figura 32 mostra um exemplo de diagrama de caso de uso com alguns dos elementos descritos a seguir:

- Pacote: É o sistema proposto que representa os casos de uso. Os usuários e quaisquer outros sistemas que podem interagir com o sujeito são representados como atores. Os atores modelam entidades que estão fora do sistema. O comportamento exigido do pacote é especificado por um ou mais casos de uso, que são definidos de acordo com as necessidades dos atores.
- Ator: É o elemento que especifica um papel desempenhado por um utilizador ou qualquer outro sistema que interage com o sistema proposto, como por exemplo, através da troca de sinais e de dados mas que é externo ao sistema. Os atores podem representar papéis desempenhados pelos usuários humanos, hardware externo entre outros. Deve-se levar em consideração que um ator não representa necessariamente uma entidade física específica, mas apenas uma aspecto particular, ou seja um “papel” de alguma entidade que é relevante para a especificação de casos de uso e suas associações. Para o exemplo da Figura 32 os atores estão representados como cliente, administrador e o banco.
- Caso de Uso: É o elemento que representa a declaração de um comportamento oferecido. Cada caso de uso especifica algum comportamento, possivelmente incluindo variantes, que o sistema pode realizar em colaboração com um ou mais atores. Os casos de uso definem o comportamento oferecido do sistema sem referência à sua estrutura interna. Esses comportamentos, envolvem interações entre o ator e o sistema, podendo resultar em mudanças para o estado do sistema e as comunicações com o seu ambiente. Um caso de uso pode incluir possíveis variações de seu comportamento básico, incluindo o comportamento excepcional e tratamento de erros. Para o exemplo da Figura 32 os casos de uso estão representado por elipses sendo eles: sacar, transferir fundos, depositar dinheiro, registrar ATM, ler registro e cartão de identificação.
- Extend: É o elemento que demonstra a relação de um caso de uso que possui um prologamento do comportamento de um outro caso de uso (geralmente complementar). O caso de uso prolongado possui definição da extensão em pontos específicos mas de forma independente do caso de uso estendido. Por outro lado, no caso de uso que se prolonga normalmente define o comportamento que não podem necessariamente ter significado por si só. Em vez disso, o caso de uso de extensão define um conjunto de

incrementos com comportamento modulares que aumentam uma execução do caso de uso prolongado em condições específicas.

- Include: É o elemento que demonstra uma relação entre dois casos de uso, o que implica que o comportamento do caso de uso incluído (destino) é inserido no comportamento de um caso de uso (origem). O caso de uso destino possuí um nome de modo a apresentar um contexto do caso de uso proprietário. O caso de uso origem só pode depender do resultado (valor) do caso de uso incluído. Este valor é obtido como um resultado da execução do processo de uso incluído.

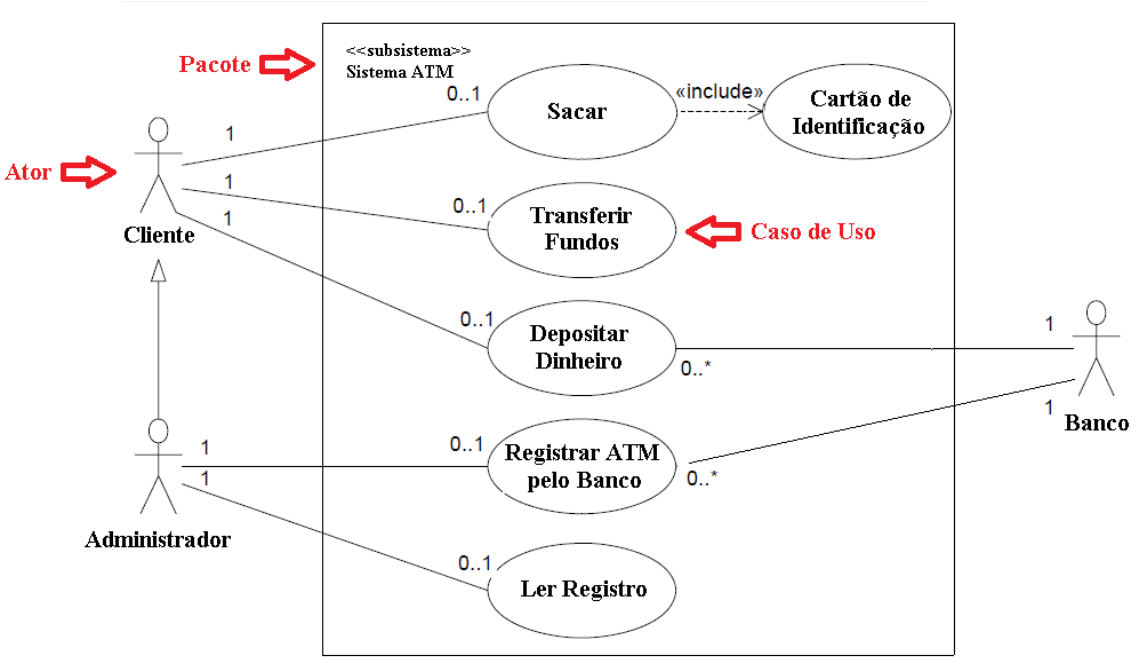


Figura 32 Exemplo de Diagrama de Caso de Uso. Adaptado de (OBJECT MANAGMENT GROUP, 2011).

Outro diagrama de importância em um projeto é o diagrama de classes que descreve as classes de um sistema e o relacionamento entre elas, fornecendo uma perspectiva ampla do projeto e as correlações entre suas classes e objetos que são compostos de funcionalidades e atributos.

O diagrama de classes é composto por elementos de associações, agregações, classes, composição, dependência, generalização, interface, interface de realização e realização. A utilização especifica de cada elemento abrange um conceito amplo, seus elementos gráficos podem ser consultados em (Object Management Group, 2011) e (HAMILTON, 2006). Neste diagrama a classe é a estrutura base para a criação de objetos que contém suas especificações, características (atributos) e métodos (ações / comportamentos). Um objeto pode ser agrupado

com outro objeto que possui característica comum, assim neste conceito também surge a definição de instâncias, que são objetos criados a partir de uma classe que possue a mesmas definições. A Figura 33 apresenta as diferentes formas de declaração de uma classe, podendo conter atributos e operações.

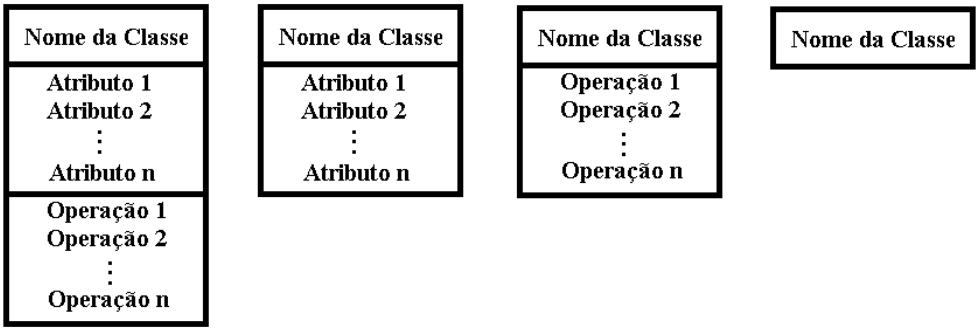


Figura 33 Quatro formas diferentes de apresentação de classes usando notação UML. Adaptado de (HAMILTON et al., 2006).

Em um diagrama de classe existem as relações, as quais podem ser de dependência, associação, agregação, composição e herança, podendo existir multiplicidade na relação, demonstrando quantos objetos podem referenciar a outro objeto. A Figura 34 mostra a multiplicidade entre duas classes representada com uma linha contínua interligando as classes indicando uma associação, a representação de quantidade é realizada por um numero e o asterisco (*) representa uma multiplicidade de vários sem um limite definido de quantidade. A Figura 35 mostra os elementos de relações utilizados entre as classes em um diagrama de classes.

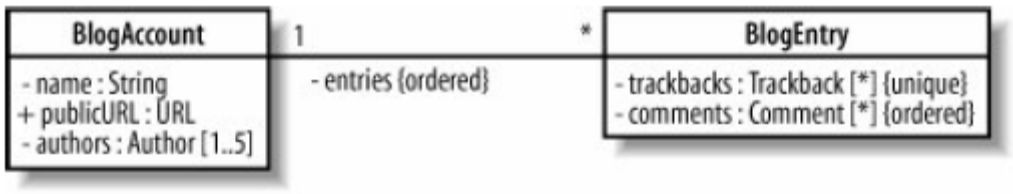


Figura 34 Exemplo de Multiplicidade entre classes (HAMILTON et al., 2006).

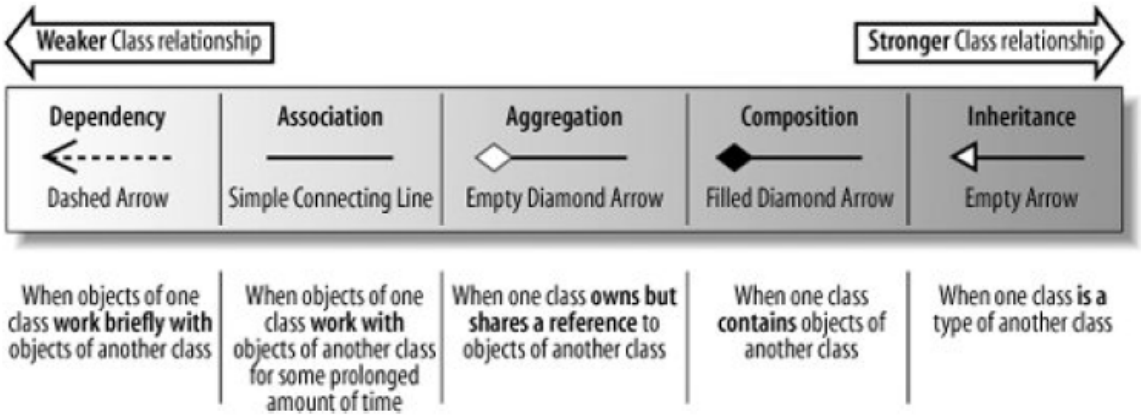


Figura 35 Elementos de relações entre classes (HAMILTON et al., 2006).

Outro diagrama importante a ser considerado são os de sequência, fundamental para verificar e assegurar os requisitos temporais de um sistema, apoiando-se no diagrama de classes para determinar os objetos das classes envolvidas em um processo, bem como o responsável por esse evento. [GUEDES, 2009].

Um projeto pode conter diversas classes, podendo ser difícil compreender e determinar a sequência global do comportamento. O diagrama de sequência representa essa informação de uma forma simples e lógica. A Figura 36 apresenta um diagrama de sequência com os elementos descritos em vermelho. Os objetos representados por um retângulo são as instâncias das classes representadas no processo. As linhas de vida compõem a dimensão vertical sendo composta de duas partes, a primeira representada por um retângulo com dois compartimentos, no compartimento superior a identificação do objeto é exibida e no compartimento inferior (cuja utilização é opcional), aparecem valores para os atributos definidos na classe do objeto, a segunda corresponde a circulo e uma linha vertical tracejada. As portas indicam um ponto em que a mensagem pode ser transmitida para dentro ou para fora. As mensagens podem ser síncronas ou assíncronas, e as especificações de execução podem conter restrições de duração.

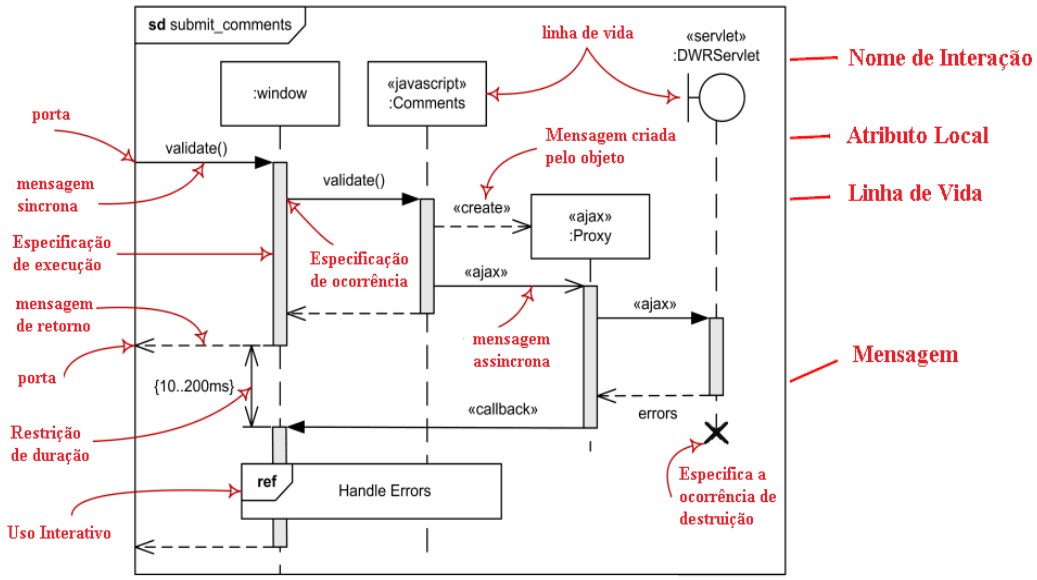


Figura 36 Exemplo de Diagrama de Sequencia. Adaptado de (Object Management Group, 2011).

Existem ainda outros diagramas de estruturas além do diagrama de classes, como os diagramas de Componentes e diagrama de objetos, que ajudam a identificar as especificações de componentes do sistema, assim como diagrama de perfil, de Estrutura Compostas, Implementação e de Pacotes. Outros diagramas comportamentais além do diagrama de casos de usos e o diagrama de sequencia são os diagramas de atividades, de iteração, de comunicação, de tempo e visão geral de interação que a linguagem UML oferece (HAMILTON et. al., 2006).

2.5 ARQUITETURAS DE SISTEMAS EMBARCADOS

Um sistema a ser embarcado em um VANT requer algumas considerações importantes, tais como o seu consumo de energia, o tempo de processamento para a realização de determinada tarefa, dimensões do sistema e robustez, entre outros aspectos, dependendo da aplicação. Aspectos como os requisitos temporais no processamento de determinada aplicação são de grande importância, visto que estes devem ocorrer dentro dos limites das condições de operações.

Para a aplicação que requer um processamento de imagens, o tempo de resposta das tarefas é um requisito importante, uma vez que a velocidade de locomoção do VANT estará diretamente ligada ao tempo do processamento da imagem a ser analisada, impactando também na duração e autonomia de voo.

Com o crescente avanço tecnológico, a cada dia são desenvolvidas e disponibilizadas plataformas de hardware e software, que permitem o seu uso em tarefas cada vez mais complexas.

2.5.1 Arquitetura ARM

Os processadores ARM (Advanced RISC Machine) são processadores com um pequeno conjunto de instruções com tamanho fixo e possuem tamanho reduzido com vantagem de consumirem pouca energia. O desenvolvimento destes processadores se originou em 1983, atingindo seu primeiro modelo comercial em 1985. Com a evolução tecnológica, a miniaturização de dispositivos eletrônicos vem ganhando cada vez mais espaço no nosso cotidiano. Notícias recentes, vem sendo veiculada sobre o desenvolvimento destes processadores pelos fabricantes mundiais de tecnologia como a Intel e AMD. Apesar de baixo poder de processamento, tornam-se atrativos pela sua portabilidade e baixo consumo de energia (FURBER, 2000).

Com o advento dos sistemas sobre chip (SoC - System-on-Chip), com a capacidade de integrar mais transistores em um único chip, foi possível formar todo um sistema eletrônico com dimensões cada vez mais reduzidos. Este conceito tem sido possível graças aos avanços de fabricação da nanotecnologia. Os chips estão atingindo a escala giga-gate trazendo a possibilidade de integrar todos os processadores e controladores de dispositivos periféricos em um chip.

O tradicional particionamento de hardware e software tem sido arranjados como funções críticas realizadas por hardware e funções específicas orientadas para o controle por software. O hardware específico é definido como um circuito integrado de aplicação específica (ASIC), que é um projeto especial dedicado para uma aplicação, e software significa a execução de um código em um processador de propósito geral. No século passado, a maioria dos sistemas embarcados precisava de hardware específico para processar aplicações multimídia, com a restrição de consumo de energia ou desempenho. Em geral, um hardware específico é melhor que um software para uma aplicação possuindomelhor desempenho e eficiência de energia. Mas hardware específico é menos flexível para se adaptar a novos recursos. Atualmente o tempo de permanência no mercado tornou-se tão importante quanto a produção de chips, uma mesma verão de chip SoC possibilita a flexibilidade da utilização de um chip para muitas aplicações. (CHEN, et al. 2009).

2.5.2 FPGAs

Os FPGAs (Field-Programmable Gate Array) possuem a característica de agregar a flexibilidade dos processadores de propósito geral com a customização de hardware em um baixo custo. Sua reconfiguração contribui para a economia de recursos, onde um bloco funcional pode realizar repetidas operações ao longo de uma tarefa em uma aplicação, fazendo com que o sistema tenha um tamanho menor, implicando na redução de preço, área e consumo de energia. Diferente dos dispositivos programáveis de menor densidade, os FPGAs apresentam uma grande densidade interna de blocos lógicos configuráveis (CLBs) que podem ser interconectados através de uma rede interna de roteamento de sinais (interconexões programáveis) com blocos de entradas e saídas (CARRO, 2001; ORDONEZ, et. al. 2003; ALLGAYER, 2009).

Existe uma grande variedade de modelos de FPGAs atualmente no mercado, entre os principais fabricantes podemos citar a Actel Corporation, Altera Corporation, Xilinx Inc., entre outros. As FPGAs comercializáveis possuem capacidades de programação distintas, tendo programação total ou parcial, com desempenho, consumo de energia variando de modelo para modelo. Os FPGAs podem ser utilizados para implementar praticamente qualquer projeto de hardware, visto que a utilização mais frequente deles é a de prototipação de circuitos integrados de aplicação específica (ASICs), mas no entanto, devido ao seu baixo custo presentes atualmente no mercado, eles vêm sendo mais utilizados diretamente no produto final (BOSA, 2009; Gonçalves, 2011).

A programação nestes dispositivos é realizada por linguagens de especificação de hardware chamadas de VHDL (Very High Speed Integrated Circuits Hardware Description Language) possuindo alta performance e flexívebilidade que são utilizados tanto na indústria como também para fins acadêmicos. Em VHDL a descrição de circuitos é realizada de forma estrutural e comportamental, sendo a forma estrutural indicando os diferentes componentes que integram o circuito e suas respectivas interconexões e a forma comportamental que descreve o circuito em seu comportamento e funcionamento sem determinar a sua estrutura. Na elaboração da descrição de um circuito utilizando VHDL, é comum possuir trechos implementados de maneira comportamental e estrutural, ficando a cargo do projetista a utilização coerente dos métodos (ORDONEZ, et. al., 2003; DUBEY, 2009).

2.5.3 Microprocessador Soft

Um microprocessador soft também chamado de softcore microprocessor ou a soft processor, é um núcleo microprocessado que pode ser totalmente implementado utilizando síntese lógica. Assim o processador combina os blocos de lógica convencional dentro de um circuito integrado, combinando software e hardware. Ele pode ser implementado através de dispositivos semicondutores diferentes contendo uma lógica programável como ASIC e FPGA. Uma vantagem no uso deste sistema a se considerar é a oportunidade de ser adicionados vários microprocessadores soft em uma única FPGA, estando somente limitados pelo tamanho da FPGA utilizada. Alguns trabalhos utilizaram esta opção para adicionar dezenas de microprocessadores soft em um único FPGA (VASSÁNYI, 1998).

Alguns algoritmos são difíceis de programar em HDL, podendo ser implementado através de processador embutido na FPGA (software). Assim alguns fornecedores de FPGA disponibilizam um conjunto de ferramentas que facilitam a implementação destes algoritmos.

Na Tabela 1 está a relação de alguns processadores existentes no mercado, tanto implementados em hardware como em software.

Tabela 1 – Processadores embutidos (DUBEY, 2009)

Nome do Processador	Tipo e Bits	Interface	Fabricante
MicroBlaze™	Soft/32	IBM Coreconnect	Xilinx
NIOS®	Soft/32	Avalon	Altera
LatticeMico32	Soft/32	Wishbone	Lattice
CoreMP7	Soft/32	APB	Actel
ARM Cortex-MI	Soft/32	AHB	Vendor independent
LatticeMico8	Soft/8	Input/output ports	Lattice
Core8051	Soft/8	NIL	Actel
Core8051s	Soft/8	APB	Actel
PicoBlaze	Soft/8	Input/Outp ports	Xilinx
PowerPC	Hard/32	IBM Coreconnect	Xilinx
AVR	Hard/8	Input/Outp ports	Atmel

3 TRABALHOS RELACIONADOS

Na pesquisa sobre trabalhos relacionados ao processamento de imagem aplicados na análise de fissuras e rachaduras, verificou-se que são quase inexistentes os trabalhos que abordam a detecção automática de fissuras em parede de alvenaria revestidas com argamassa localizada em fachadas. Conforme mencionado no capítulo anterior, fissuras em fachadas são um dos principais problemas em construções na região Sul do Brasil, sendo por isto escolhidas como objeto de estudo desta dissertação. Entretanto, foram encontrados vários trabalhos relacionados à detecção de fissuras em concreto e alguns em alvenaria, alguns deles com foco na implementação embarcada e outros somente explicitando uma metodologia. Alguns trabalhos que utilizaram de captura de imagens, para a detecção das fissuras, utilizaram de ambientes controlados, ou seja, proporcionando uma iluminação artificial de forma continua um fator determinante para uma boa aquisição da imagem.

A seguir estão relacionados alguns trabalhos sobre detecção de fissuras de forma automática ou semiautomática, alguns deles com abordagem na detecção somente outros empregando os algoritmos em equipamentos de apoio a serem empregados em campo.

No trabalho desenvolvido em (QADER, ADUBAYYEH, KELLY, 2003), foi realizada uma análise de técnicas de processamento de imagem para a detecção de fissuras com foco em inspeções de pontes. A análise na detecção de fissuras foi realizada comparando detecção de bordas, com Operador Sobel e Canny, além das transformadas de Fourier e Haar. O trabalho chegou a conclusão de que as detecções realizadas somente com o Sobel levaram a 68% de acuidade, com 25% de detecção de falsos positivos, levando em consideração 50 amostras contendo fissuras. Para Canny, apresentou uma acuidade melhor de 76%, a Transformada Rápida de Fourier uma acuidade de 64% e a transformada de Haar uma acuidade de 86%. Em resumo, a transformada de Haar obteve melhor detecção. Porém o trabalho enfatiza sobre o valor do limiar que podem interferir consideravelmente na detecção das fissuras, sendo necessário realizar um estudo da melhor forma de se obter este limiar.

O trabalho de (CHEN, et. al. 2006) utiliza um método de mensurar de forma automática e registrar o comprimento e largura de uma fissura a partir de imagens multitemporal. Durante a primeira aquisição da imagem que contém a fissura a ser mensurada, o usuário que irá operar o sistema deve marcar manualmente pontos na fissura da imagem digitalizada a fim de obter suas medidas. O algoritmo utilizado gera um modelo de esqueleto da fissura a partir dos pontos marcados, considerados como pontos sementes de forma a auxiliar nas medições posteriores. Os resultados experimentais demonstraram consistência entre a extração automática das

medidas e da medição manual na ordem de 0,05 mm. A extração da fissura utiliza como limiar a intensidade de pixels, obtida através de detecção de borda utilizando um filtro Gaussiano, separando a fissura do plano de fundo.

No trabalho desenvolvido por (METNI, HAMEL 2006) é proposto um mecanismo de auxilio na inspeção de pontes para detectar fissuras baseado em imagens obtidas com o emprego de um VANT. Um dos problemas no uso destes equipamentos em regiões como pontes e outras estruturas está justamente na orientação da câmera para obter imagens planas do objeto de inspeção. Assim, sistemas de controle de servo motores são utilizados para auxiliar na orientação da câmera para captura da imagem, sendo que a própria imagem adquirida é utilizada para controlar os servos motores, auxiliando na captura plana da imagem a ser analisada. Porém, neste trabalho a análise das fissuras não é feita de forma automática, sendo apenas armazenadas para posterior processamento em uma plataforma computacional.

Em (NIEMUELLER, 2006), foi proposto um sistema de detecção de rachaduras em tubulações de gasodutos, utilizando métodos de morfologia matemática e filtros lineares (filtro gaussiano), procurando aproximar-se ao modelo geométrico das fissuras para a detecção de padrões. Embora o método apresente boa acuidade com 7% para a probabilidade de falsos positivos e 2% para falsos negativos, foi comentada a questão computacional dispendiosa, que na abordagem aos métodos modernos podem ser vantajosos neste requisito.

No trabalho realizado por (OLIVEIRA, CORREIA, 2008), é proposto um sistema de detecção de rachaduras automáticas com base em imagens de estradas pavimentadas. O artigo confronta seis estratégias de classificação supervisionada, três paramétricas e três não paramétricas, lidando com as estratégias de classificação supervisionada, ou seja, após um especialista humano selecionar as imagens contendo rachaduras. As estratégias de classificação selecionadas no trabalho utilizam um espaço de características em duas dimensões, trabalhando com imagens de dimensões bastante reduzidas, na razão de 75 x 75 pixels. A Figura 38Figura 37 mostra a arquitetura do sistema proposto para o treinamento. Um tratamento de normalização é levado em consideração no trabalho, devido a variações na luminosidade que podem interferir durante a aquisição da imagem. A avaliação é baseada em um conjunto de métricas conhecidas, chegando à conclusão que um melhor desempenho pode ser conseguido utilizando estratégias paramétricas de classificação.

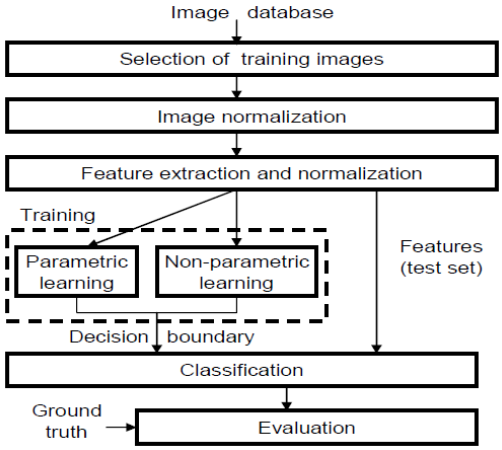


Figura 37 Arquitetura do Sistema proposto em (OLIVEIRA, CORREIA, 2008).

Em (LI, et. al. 2008) é apresentado um algoritmo de processamento de imagem para inspeções em linhas de transmissão com a utilização de VANTs. O método empregado usa o reconhecimento de padrões baseado em Rede de Pulso Acoplado Sequencial, com a utilização da transformada de Hough. O uso desta transformada, além de auxiliar na detecção da linha de transmissão, permite a determinação de sua direção, podendo auxiliar o VANT a seguir a linha através da imagem capturada. O trabalho reconhece o alto custo computacional exigido no cálculo da transformada de Hough, mas o mesmo se utiliza de uma referência que propõe uma melhoria para permitir o seu uso em tempo real.

Em (HOLMES, et. al. 2010) foi desenvolvido um sistema de detecção de rachaduras em pavimentações com a aplicação de selantes. Para a detecção da rachadura foi utilizada a descrição de objetos que caracteriza uma rachadura a partir de amostras de treinamento. Na descrição dos objetos foram empregadas técnicas para obter os maiores e menores comprimentos de eixo, área, excentricidade e orientação, tendo uma eficiência de 83% de resposta positiva durante analise no protótipo, sendo obtidas mais de 30.000 fotos. A Figura 38 apresenta o protótipo do sistema de detecção e reparo.

Figura 38 Prototipo do sistema de vedação de rachaduras de pavimento.

No trabalho realizado por (MOON, et. al. 2011) é apresentado um sistema de detecção de fissuras que pode analisar a superfície de concreto e visualizar as fendas de forma eficiente. O algoritmo é composto de duas partes; processamento de imagens e classificação de imagens. No primeiro passo, as fissuras são separadas da imagem de fundo usando filtragem e operação morfológica. Nesta etapa os dados particulares, tais como o número de pixels e a proporção do eixo maior para o eixo menor e da área de pixels são extraídos. No segundo passo, fissuras são identificadas utilizando redes neurais para automatizar a classificação das imagens. As redes neurais foram treinadas com 105 imagens de estrutura de concreto, e posteriormente testadas para 120 novas imagens. A taxa de reconhecimento de imagens com fissura neste trabalho foi de 90% e de imagem que não apresentavam fissuras de 92%.

Em (PRASANNA, et. al. 2012) foi proposto um sistema de detecção de fissuras em estruturas de concreto baseado na classificação por detecção de borda e analise e classificação através de seu histograma. A taxa de detecção do classificador foi de aproximadamente 76%, para proporcionar um sistema automatizado para inspeção visual em pontes. A Figura 39 apresenta o protótipo do sistema utilizado por um operador, fazendo parte do sistema, um computador portátil com câmera e fonte em uma estrutura móvel.

Figura 39 Protótipo do sistema de detecção de fissuras em concreto.

Em (FURTADO, 2012) foi desenvolvido um algoritmo de detecção de rachaduras para aplicação em ambientes subaquáticos. O algoritmo foi baseado no Filtro Bayesiano denominado Filtro de Partículas, sendo um filtro não paramétrico. No algortimo proposto é realizado o calculo probabilístico através dos dados da imagem em analise e de partículas criadas de uma variável aleatória base do conceito do Filtro de Partículas, após é realizada uma reamostragem que consistem em uma implementação probabilística da idéia darwinista da sobrevivência do mais forte propiciando desta forma uma detecção da rachadura. No algoritmo é necessário atribuir valores em alguns parâmetros que interferem na precisão da detecção destas rachaduras. Entretanto não foi embarcado em nenhum dispositivo, propondo esta realização para trabalhos futuros.

No estudo de (ESCHMANN et. al., 2012) foi proposta a utilização de VANT para inspeção em fachadas de edificações. O sistema utiliza captura de imagem através de uma câmera acoplada no VANT, e com a sequencia de imagens obtidas é realizado um processamento posterior, ou seja, o sistema não realiza nenhuma análise da imagem embarcada no VANT. O Processamento é realizado através de softwares comerciais existentes que une as imagens adquiridas para criar uma imagem panorâmica da fachada em alta qualidade. O trabalho propõe uma possibilidade da detecção de fissuras por filtro gaussiano ficando este requisito considerado para um trabalho futuro. Também é comentado no estudo uma melhor trajetória para a aquisição das imagens, uma vez que é utilizando alguns sistemas do próprio VANT a fim de auxiliar na captura das imagens durante o voo, sendo que parte do voo foi de

forma controlada pelo operador. A Figura 40 apresenta varias imagens capturadas durante o voo e posterior processamento com interpolação da imagem obtendo uma imagem com altíssima resolução.

Figura 40 Proposta do sistema de captura de imagem e posterior processamento. (ESCHMANN et al., 2012)

Em (SANTHI, et. al. 2012), foi proposta uma técnica automatizada para detectar as fendas horizontais, verticais e diagonais em edificações, calçadas, terrenos, estradas, objetos, entre outros, levando em consideração texturas complexas, má iluminação, e em fundos diferentes que podem influenciar na precisão. Em um primeiro momento, as imagens são processadas por processamento morfológico em escala cinza e, posteriormente, o resultado final é obtido através da filtragem das imagens utilizando filtro passa baixo, em seguida, aplicando os operadores de detecção de bordas. Sendo então analisado o comportamento através de inspeção visual dos resultados. Os resultados, segundo os autores, foram promissores, utilizando-se de uma técnica para uma limiarização adaptativa, para detectar todos os tipos de fissuras em todos ambientes que tornam inimitável em outros algoritmos.

Em (MARTINS, 2013), foi proposta a detecção da fissura em alvenaria empregando um aparelho celular que realiza o processamento de imagem com técnicas como normalização e limiarização da imagem, a fim de determinar a abertura da fissura, com o objetivo de automatizar a inspeção a ser realizada por um técnico, utilizando a estrutura apresentada na Figura 41. O método de monitoramento demonstrou ser eficaz na analise de corpos de provas utilizados na engenharia civil, comparado com os métodos convencionais de medição, tanto de alvenaria como em concreto. Fatores importantes considerados neste trabalho é a menor

subjetividade na identificação das alterações da região da fissura, sem depender de experiência do técnico em procedimentos de leitura e interpretação de instrumentos de metrologia, obtendo assim resultados mais exatos.

Figura 41 Estrutura proposta para detecção de Fissuras em concreto e alvenaria (MARTINS, 2013).

Em (CORNELIS, et. al. 2013) é proposto um método de detecção de fissura semi-supervisionado, de forma a ajudar na restauração e preservação de pinturas de arte com imagens adquiridas de grandes dimensões, construindo um classificador que é capaz de discernir a fissuras da imagem de fundo, empregando assim um classificador Bayesiano não paramétrico fruto de desenvolvimentos recentes. O método proposto tem a propriedade atraente de não ser paramétrica, desta forma possibilita a utilização de seleção de indicadores mais importantes, obtendo velocidade de processamento mais rápido do que outros métodos. O trabalho confirmou a obtenção de características de rachaduras satisfatórias, porém não levando ainda uma abordagem da margem de erros na caracterização das rachaduras.

Na Universidade Federal do Rio Grande do Sul, importantes contribuições são desenvolvidas no estudo de manifestações patológicas em edificações, contribuindo na melhoria das técnicas adotadas no processo construtivo, a fim de reduzir o surgimento de fissuras em fachadas de revestimento de argamassas. O Programa de Pós-Graduação em Engenharia Civil, Departamento de Engenharia Civil da Escola de Engenharia/UFGRS tem se

empenhado nesta área. Os pesquisadores do PPGEC vêm contribuindo, por exemplo, no desenvolvimento de sistemas especialistas como o Sistema Edificar, que é uma ferramenta que auxilia na decisão e no diagnóstico de recuperação de fissuras, através de módulos, oferecendo a possibilidade de o usuário interagir, respondendo a uma série de perguntas para a identificação das prováveis causas (SILVA, 1996). Outros trabalhos desenvolvidos por pesquisadores do programa correlacionados às manifestações patológicas encontram-se descritos em (CREMONINI, 1988; DAL MOLIN, 1988; SILVA, 1996; ANDRADE, et. al. 1997; MAGALHÃES, 2004; COSTA, 2005).

Conforme pode ser observado na Tabela 3, nenhum dos trabalhos analisados aborda todos os tópicos pesquisados na corrente dissertação: o uso de VANTs para auxiliar na análise de fissuras em fachadas com revestimento em argamassa, executando algoritmos de processamento de imagens em sistemas computacionais embarcados nos VANTS, visando um maior desempenho tanto em tempo de processamento, quanto na autonomia de voo e tempo de coleta de imagens.

Tabela 2 – Tabela dos trabalhos relacionados a monitoramento de fissuras

Trabalho	Tipo de Sensor	Material Analisado	Forma de Detecção	Técnicas utilizadas	Plataforma de processamento	Equipamento de apoio
CHEN, et. al. 2006	Imagem	Superfície de Concreto	Semi-Automática	Filtro Linear (Filtro Gaussiano)	Computador	Nenhum
ESCHMANN et. al., 2012	Imagem	Argamassa / Alvenaria	Semi-Automática	Filtro Gaussiano	Computador	VANT com Câmera para captura
SANTHI, et. al. 2012	Imagem	Superfície de concreto / argamassa	Automática	Filtro passa Baixo	Computador	Nenhum
MARTINS, 2013	Imagem	Superfície de concreto / argamassa	Automática	Normalização e Limiarização	Celular	Estrutura cilíndrica de apoio com Iluminação

Tabela 3 – Tabela dos trabalhos relacionados a detecção de fissuras

Trabalho	Tipo de Sensor	Material Analisado	Forma de Detecção	Técnicas utilizadas	Plataforma de processamento	Equipamento de apoio
QADER, ADUBAYYEH, KELLY, 2003	Imagem	Superfície Pavimentada	Automática	Sobel/Canny/Transformada Rápida de Fourier/ Transformada de Haar	Computador	Nenhum
METNI, HAMEL 2006	Imagem	Superfície de Concreto	Semi-Automática	Limiar	Computador	VANT com servo-mecanismo na camera
NIEMUELLER, 2006	Imagem	Tubulação Subterrânea	Automática	Filtro Linear e Morfologia Matemática	Computador	Nenhum
LI, et. al. 2008	Imagem	Linhas de Transmissão	Automática	Rede de Pulso Acoplado Sequencial, transformada de Hough	Computador	VANT
OLIVEIRA, CORREIA, 2008	Imagem	Superfície de Concreto	Automática	Normalização / Media / limite quadrático / Bayesian classifier /Parzen / k-Nearest Neighbor / Fisher's Linear classifiers.	Computador	Nenhum
HOLMES, et. al. 2010	Imagem	Superfície Pavimentada	Automática	Normalização / Limiarização / Filtros Lineares	Computador	Sistema embarcado em Reboque
MOON, et. al. 2011	Imagem	Superfície de Concreto	Automática	Filtro Linear / Passa Baixo / Morfologia Matemática / Redes Neurais	Computador	Nenhum
PRASANNA, et. al. 2012	Imagem	Superfície de Concreto	Automática	Limiar com Operador de Borda / Classificação por características Histograma	Computador	Carrinho com Câmera e computador Acoplados
FURTADO, 2012	Imagem	Superfície de rocha	Automática	Filtro de Partículas	Computador	Nenhum

4 PROPOSTA DA DISSERTAÇÃO

Conforme já discutido no Capítulo 1, o presente trabalho busca desenvolver um sistema que combine as seguintes características:

- Seja focado na determinação de fissuras em fachadas de argamassa, visto que, conforme apresentado na Seção 2.1, estas são as manifestações patológicas mais frequentes na região Sul do Brasil, em função das variações ambientais;
- Utilize processamento de imagens para detecção de fissuras, visto que estas imagens podem ser obtidas com câmeras amplamente disponíveis comercialmente a preços já bastante acessíveis;
- Utilize-se de um VANT para permitir o acesso a áreas remotas e de difícil acesso. Para isto pretende-se utilizar os RUAVs disponíveis no GCAR-UFRGS;
- Seja capaz de realizar a inspeção da forma mais automática possível, ou seja, pretende-se analisar as possibilidades de realizar a detecção de fissuras por algoritmos que executem em processadores embarcados nos VANTs.

Este capítulo foi estruturado da seguinte forma: a próxima subseção apresenta a especificação do sistema proposto, usando para isto diagramas UML; a subseção seguinte apresenta dois algoritmos de processamento de imagens para análise de fissuras, os quais foram selecionados para a implementação do sistema proposto. Na sequência, são apresentados os algoritmos de processamento de imagens desenvolvidos a partir da ferramenta MATLAB™, para a indicação de presença de fissuras na alvenaria. Após, é realizada uma abordagem na geração do código para as plataformas, que possibilite a embarcação no VANT.

A análise baseada nestas plataformas será importante, uma vez que o requisito temporal relacionado à performance de cada função desempenhada pelos algoritmos, ajudará na verificação de qual a melhor situação para o emprego no VANT. A relação de tempo x consumo na operação dos dispositivos é crucial durante um levantamento em campo.

4.1 ESPECIFICAÇÃO DO SISTEMA PROPOSTO UTILIZANDO UML

Para especificação do sistema proposto foi utilizada a linguagem Unified Modeling Language (UML), que possibilita a visualização do sistema, através da sua representação gráfica. A seguir estão os diagramas que apresentam as especificações lógicas do sistema.

4.1.1 Diagrama de Caso de Uso

Para o sistema proposto, é apresentado no diagrama de caso de uso, conforme Figura 42, a identificação dos seguintes atores:

- Operador do sistema: irá manipular o VANT, de forma a posicioná-lo no objeto em interesse e iniciar o sistema de processamento;
- Fachada: que é o objeto em análise
- Profissional: que irá analisar as imagens obtidas, bem como os resultados produzidos pelo sistema embarcado de detecção de fachadas.

O sistema que irá detectar a manifestação patológica terá a integração do VANT com uma entidade que irá processar a imagem. Este sistema realizará a captura da imagem de uma fachada e irá processá-la de acordo com a ação do operador, retornando as imagens que forem consideradas importantes com presença das fissuras de modo a possibilitar a análise do profissional.

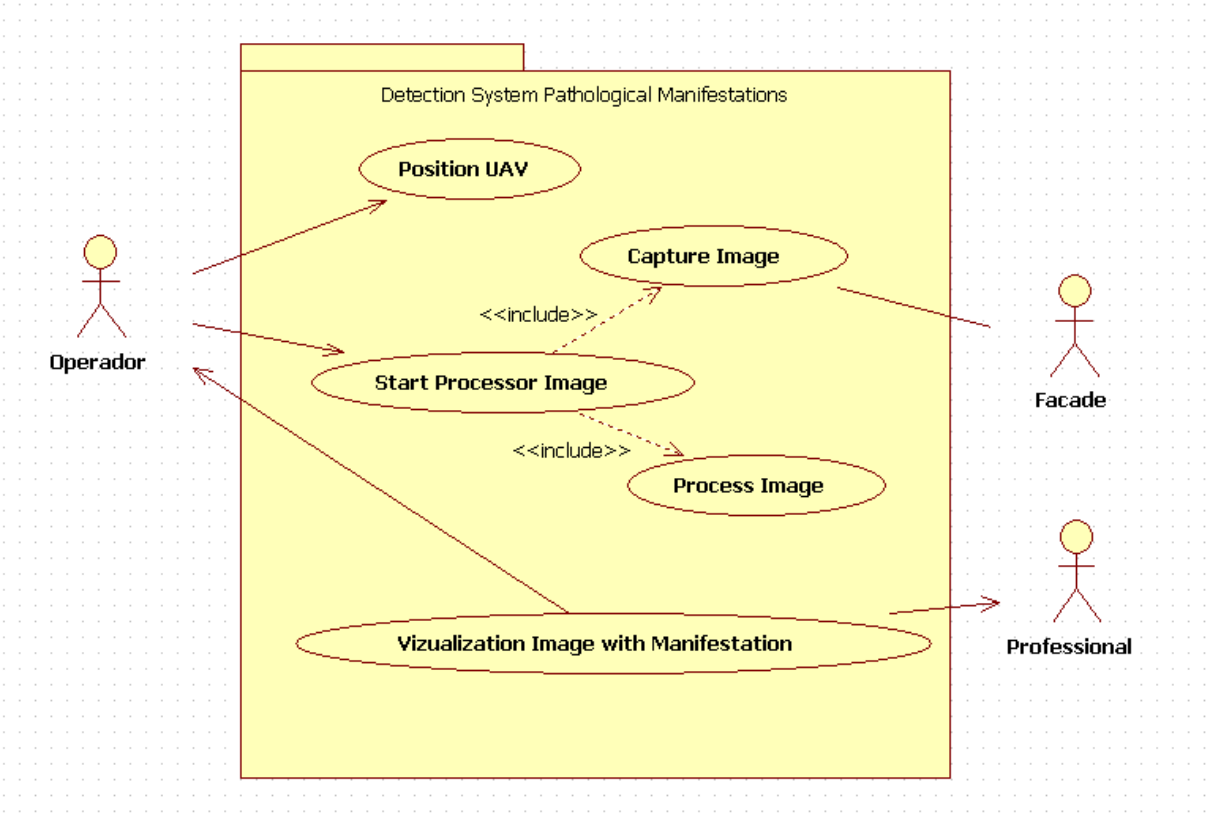


Figura 42 Diagrama de Caso de Uso para o Sistema proposto.

4.1.2 Diagrama de Classes

Através do Diagrama de classes pode-se analisar a estrutura das relações dos objetos do sistema, tanto aqueles já existentes e disponíveis quanto aqueles que serão desenvolvidos. No primeiro momento, a classe UAV representa o dispositivo que propiciará a captura da imagem na altura da fachada. Esta classe possui relações com as classes que integram os sistemas de navegação através do NavControl existente que possuí relação com as classes CompassModule e GPS, que são sensores que são utilizados na referência de posicionamento da Classe UAV, ambos possuem conexão com a classe FlightControl responsável pelo controle dos motores. A classe FlightControl possui relações com Altimeter que é um sensor que retorna valores de pressão atmosférica possibilitando uma medida de altura e também relação com a classe Remote Control responsável pela comunicação de controle com um operador que ira manipular o dispositivo UAV.

Na proposta do sistema de processamento de imagem integrado no UAV, a classe ImageProcessor, possuirá um relacionamento com a classe UAV e também com a classe Communicator, para um cenário que permite o operador obter informações do processamento da imagem como também de dados de localização do UAV através dos sensores.

A classe ImageProcessor, responsável pelo processamento da imagem, obterá os dados da imagem a partir da classe ImageCapture através da classe ImageSensor. A classe ImageProcessor deverá possuir um relacionamento com a classe sensor de distância, que servirá para verificar a distância correta para a captura da imagem da fachada, além de possibilitar que o sistema calcule a dimensão da fissura capturada. Este sensor pode também ter conectividade com a classe UAV, possibilitando um auxílio adicional no controle do UAV inclusive para evitar possíveis obstáculos.

O sistema que irá processar a imagem deverá realizar a detecção do objeto e armazenar dados de posição, sendo que estes podem ser obtidos pelas classes presentes no VANT, além de um sensor de distância, que terá um papel importante no auxílio da captura da imagem, e mensurar o objeto foco.

A Figura 43 apresenta o diagrama de classes, considerando o processamento embarcado.

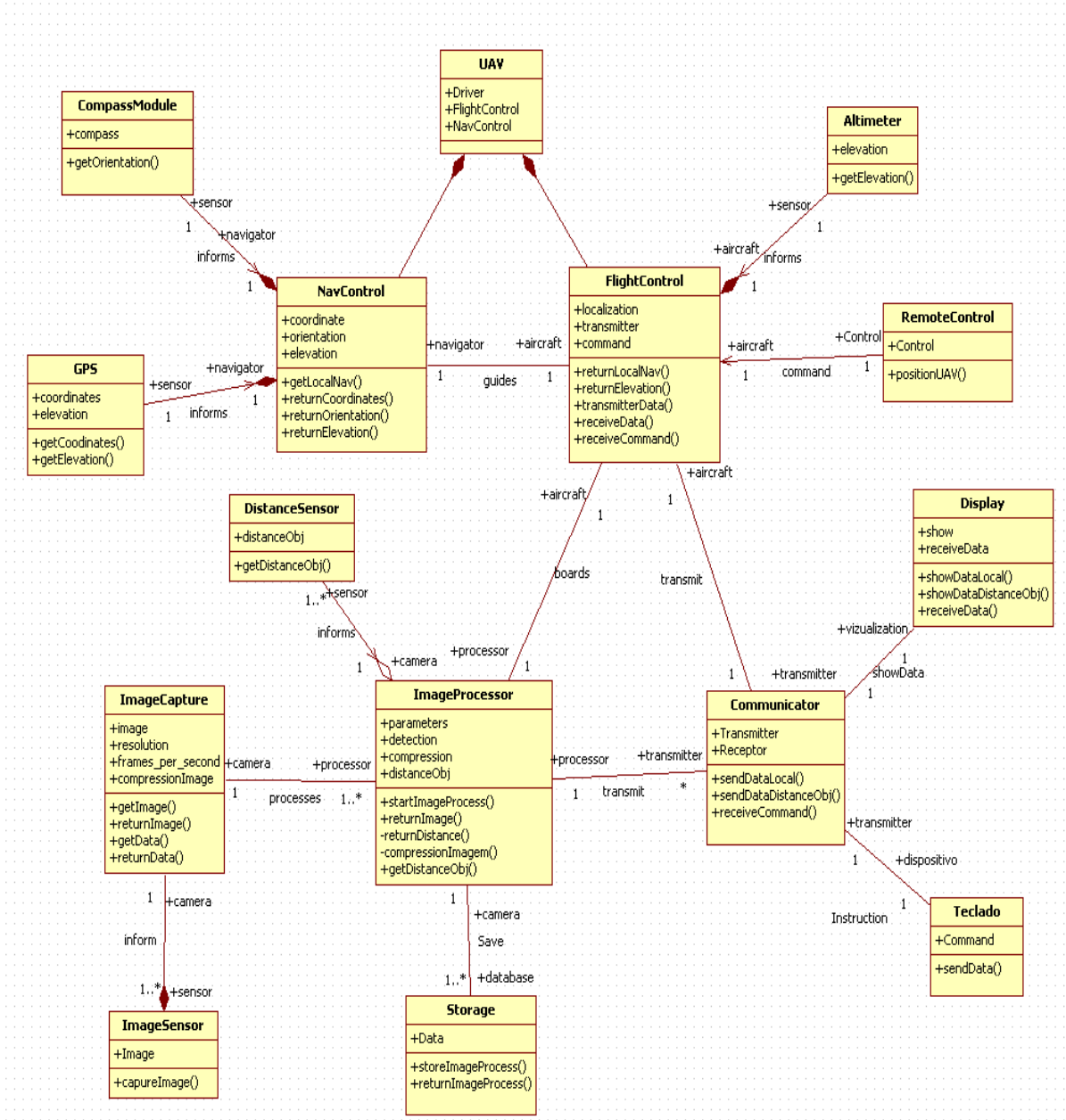


Figura 43 Diagrama de classes

4.1.3 Diagrama de Sequencia

O diagrama de sequência da Figura 44 apresenta o comportamento do sistema proposto com as interações entre as classes. Este contexto é essencial para a ordenação temporal das informações que serão trocadas entre as classes. Inicialmente o operador deverá posicionar o objeto UAV, de modo a possibilitar a captura da imagem. Na sequência, é executado o comando que verificará a distância do objeto em análise, no caso a fachada de uma edificação. A distância deverá estar dentro de uma margem de segurança para uma boa captura para ser iniciado o

processo de captura. Caso o retorno do valor da distância esteja dentro do valor recomendado, a imagem de retorno poderá ser já processada, caso não esteja, a imagem capturada pode ser descartada sem o processamento.

Quando o VANT estiver então dentro de uma margem de distância de captura da imagem, o processamento da imagem iniciará quando a imagemestiver disponível. Em meio da tarefa de aquisição e processamento, dados como orientação e posição do VANT serão importantes a fim de identificar em qual parte da fachada a fissura foi detectada. No armazenamento, ambos os dados deverão estar referenciando cada imagem processada.

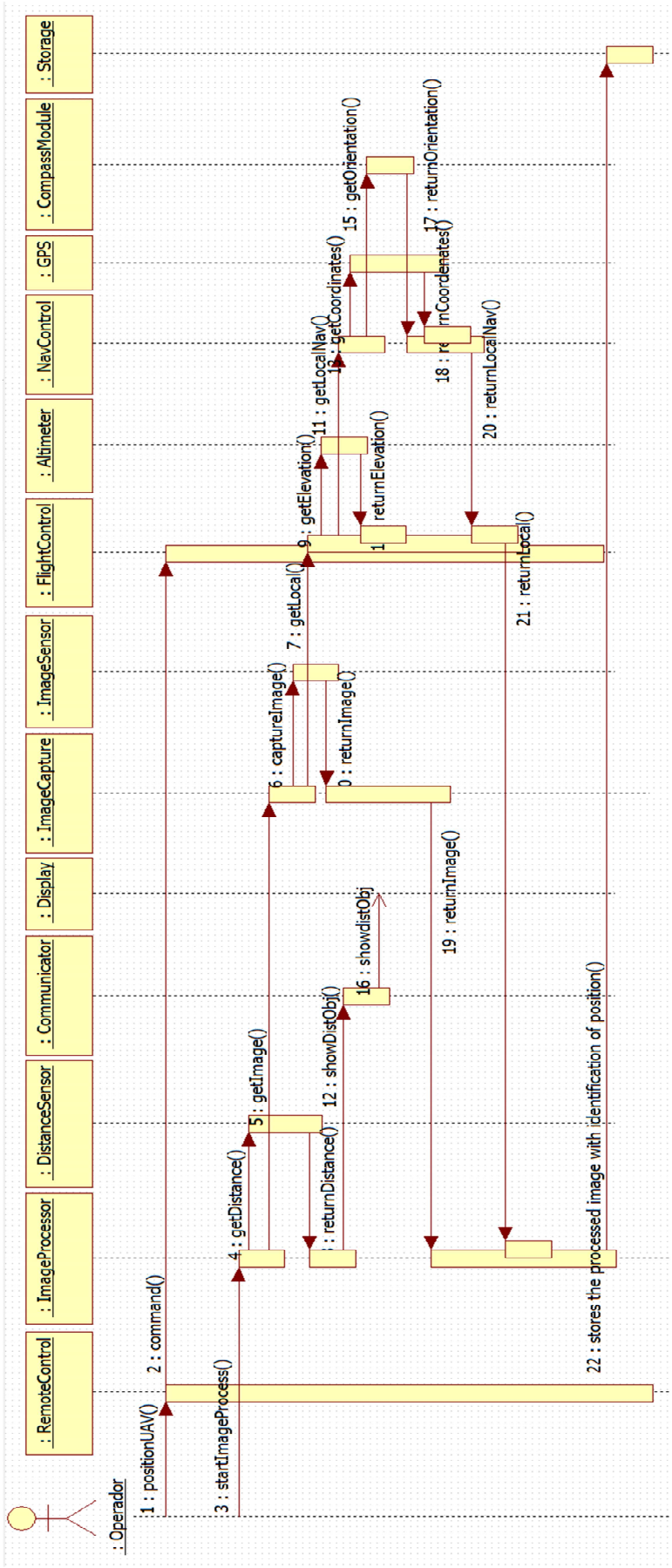


Figura 44 Diagrama de sequencia do sistema proposto

4.2 ALGORITMOS

Através do estudo de formas de detecção de fissuras, com o auxílio das metodologias adotadas em trabalhos relacionados, conforme apresentado no capítulo 3 desta dissertação, foram desenvolvidos e adaptados dois algoritmos. No primeiro algoritmo foi utilizado técnicas de processamento de imagem de maneira a tratar a imagem inicialmente para uma classificação posterior utilizando abordagem sintática. No segundo algoritmo foi empregado um classificador que utiliza abordagem estatística sem tratamento inicial da imagem. Durante a análise dos algoritmos os melhores resultados para a aplicação proposta basearam-se principalmente em (NIEMUELLER, 2006; HOLMES, 2010; FURTADO, 2012). Um destaque é no empregado do algoritmo Filtro de Particulas, fruto de trabalhos recentes, o método vem sendo adotado para acrescentar uma inteligência artificial na área de robótica, como por exemplo, o seu uso em carros autônomos.

4.2.1 Ambiente de Desenvolvimento e Simulação

A simulação dos algoritmos para análise da detecção proposta foi realizada na ferramenta MATLAB™ na versão 2012b, com um estudo prévio do comportamento das funções que melhor atende a detecção de fissuras em fachadas de revestimento em argamassa. As funções disponíveis no Image Processing Toolbox do MATLAB™ foram uteis na analise do comportamento dos algoritmos para as amostras, que foram disponibilizadas a partir dos trabalhos sobre manifestações patológicas, desenvolvidos na disciplina de Patologias dos Revestimentos, ministrada pela prof. Dra. Ângela Borges Masuero, realizados no NORIE, e obtidas em fotos de fachadas das edificações.

As amostras para analise foram adquiridas através de fotos capturadas de forma manual, de fachadas argamassadas com presença de fissuras, com distância aproximada entre 1 metro e 1,5 metros das paredes de forma paralela, com resoluções de 1 megapixels e 5 megapixels para uma analise do coportamento nos algoritmos. Foram adquiridas também imagens através de uma câmera acoplada em um VANT de modo a verificar a compatibilidade das imagens manuais. Através de várias amostras analisadas, constatou-se que a intensidade luminosa do ambiente pode influenciar numa boa detecção da fissura. As fotos que foram obtidas na fachada, em dias ensolarados sem a incidência do sol diretamente na fachada, foram as melhores imagens obtidas para a detecção utilizando os algoritmos propostos. Outra questão importante é que

tanto as fotos obtidas manualmente como as obitdas pelo VANT são semelheantes, mesmo devido ao movimento do VANT durante a captura da imagem. No anexo A.7 estão a relação destas imagens que foram utilizadas no aperfeiçoamento do algoritmo.

4.2.2 Algoritmo Baseado em Segmentação por Detecção de Bordas e Descrição

O primeiro algoritmo é baseado em Segmentação por Detecção de Bordas e Descrição, chamado de algoritmo A, utiliza a detecção de bordas com operador Sobel (SOBEL I. 1990). É basicamente composto por derivadas parciais calculadas separadamente na direção de duas coordenadas espaciais x e y e posterior combinação dos resultados.

Para o operador é necessário um limiar de referência. Foi proposto um valor compreendido entre o fator de 40% a 20% da magnitude do gradiente nos limites das bordas de uma fissura, sendo este fator relativo a características da predominância de intensidade de pixels maiores correspondendo o fundo da imagem e dos pixels de intensidade menores correspondendo uma fissura, está referência varia no entanto com a textura da fachada em análise.

A Figura 45, ilustra a sequência dos métodos utilizados no algoritmo A para a detecção da fissura.

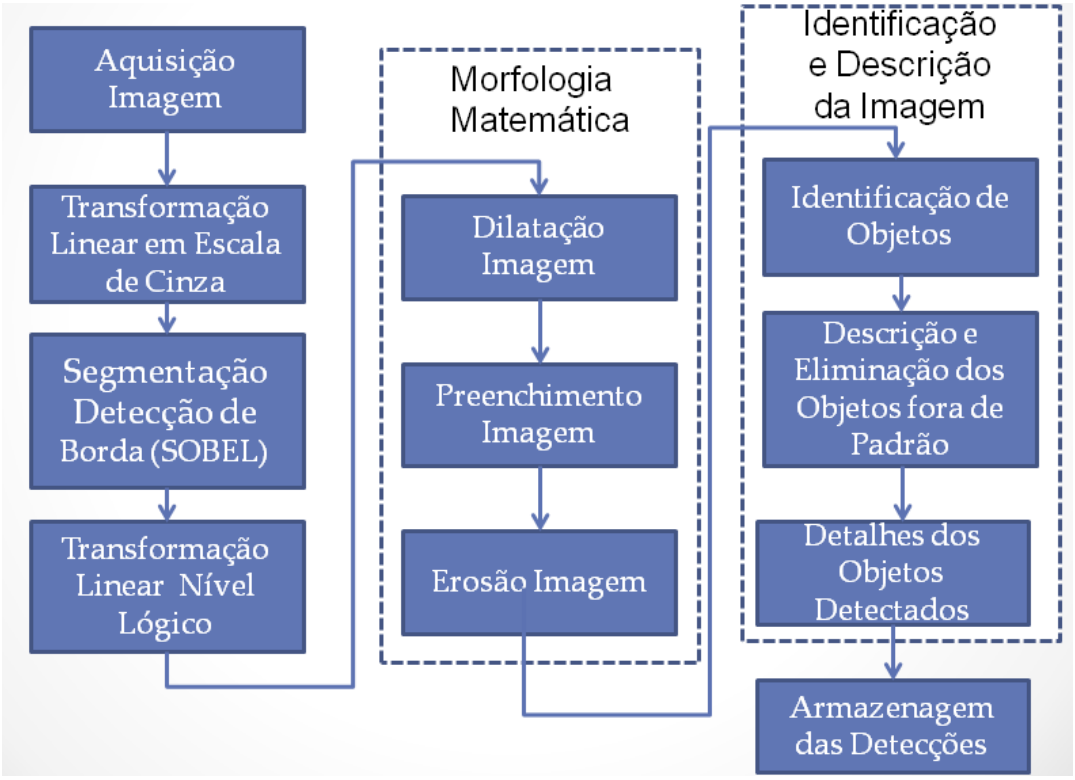


Figura 45 Sequência dos métodos de processamento de imagem utilizados no algoritmo A.

Os principais passos são descritos a seguir:

1º Passo – Aquisição da imagem, que poderá ser adquirida no modelo de cores **RGB** ou no modelo **YUV.** Detalhes de modelo de cores são apresentados no Anexo A.

2º Passo - Transformação em escala de cinza, contendo as intensidades da imagem dispostas em uma função f(x,y) que representa uma matriz com a quantidade de pixels de acordo com a resolução da aquisição. Para o modelo de cores RGB, este pode ser adquirido através da transformação descrita no Capítulo 2, item 2.2.5 dada pela equação 5. Para o modelo YUV, porém, não é necessária esta etapa, pois o componente **Y** já contém a informação de escala de cinza de uma imagem, conforme modelos de cores apresentados no Anexo.

3º Passo - Detecção de Borda: utiliza o operador Sobel com o limiar descrito anteriormente.

4º Passo - Transformação dos pixels para o nível lógico 0 e 1, aplicando um limiar de valores diferentes de zero para nível igual a 1, operação como descrito na equação 15.

5º Passo – Dilatação da imagem, para realizar conexão dos pixels das bordas da imagem a fim de não haver uma separação de um objeto, que caracterize única fissura, este realizado pelo elemento estruturante (7x7) apresentado na Figura 46.

6º Passo – Preenchimento da imagem, para cobrir os pixels vazios por dentro da borda da fissura, e também os pixels desconectados da borda, realizado pelo elemento estruturante (3x3) apresentado pela Figura 47.

7º Passo – Erosão da imagem - este processo efetua a eliminação dos pixels fora da borda da imagem, esta operação realizada pelo elemento estruturante diamante (3x3) representado na Figura 47.

8º Passo – Identificação dos objetos – É realizada a separação e identificação dos objetos com conectividade 8 dos pixels vizinhos, conforme demonstrado no capítulo 2, na seção 2.2.10

9º Passo – Descrição dos objetos da imagem processada - a obtenção de cada objeto obtido anteriormente e as descriçções da Área, do eixo maior, do eixo menor.

10º Passo – Eliminação dos segmentos fora do Padrão, sendo necessário atribuir características que descrevem uma fissura, como a quantidade de pixels limites para o eixo maior, para o eixo menor e para a interna da imagem segmentada. Foram obtidos os valores limites através de levantamento das dimensões que compreendem a fissura de forma a eliminar os objetos que estão fora do padrão. Estes valores limites são apresentados nas igualdades dadas na equação 30.

$$
\begin{aligned}
&\text{MaxPixels} = \text{NPixels} * 0.00015; \\
&\text{MinPixels} = \text{NPixels} * 0.009; \\
&\text{MinArea} = \text{NPixels} * 0.0015; \\
&\text{MaxArea} = \text{NPixels} * 0.09;
\end{aligned}
\tag{30}
$$

Assim, a fim de eliminar possíveis pontos que não representam segmentos característicos de uma fissura, o eixo maior não pode ser menor que a proporção de 0,015% dos pixels, para ser considerado como fissura, eliminando deste modo os segmentos menores que permanecem mesmo após os processos anteriores. Para o eixo menor, este não pode ter quantidade maior que 0,9% de pixels da imagem, o que pode configurar muito ruído ou algum objeto na imagem maior que não seja uma fissura. Para a área, considerando a imagem como um todo, deve ser maior que 0,15% e menor que 0,9% do percentual de todos os pixels da imagem, para se obter com maior exatidão segmentos nítidos que caracterizam a fissura. A relação para os tamanhos reais da fissura é dependente da distância e dimensões da imagem e consequentemente a quantidade de pixels na aquisição. Estes parâmetros deverão sofrer variação durante o emprego do algoritmo em campo, sendo importante adotar uma percentual que possua uma melhor desempenho num processamento da imagem.

11º Passo – Obtenção de detalhes dos objetos detectados, como orientação, centroide para indicação dos pontos da fissura e sua orientação ao longo do segmento são realizados para posterior indicação.

12º Passo – A armazenagem das detecções, com indicações e descrições.

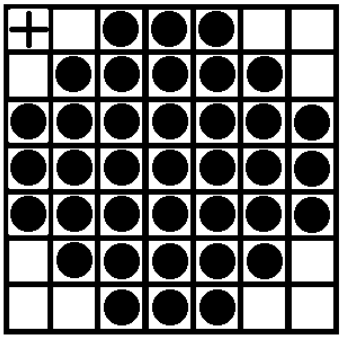


Figura 46 Elemento estruturante aplicado a operação de dilatação no algoritmo A.

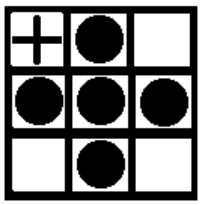


Figura 47 Elemento estruturante diamante aplicado a operação de erosão no algoritmo A.

No Apêndice A.1 está o algoritmo desenvolvido no ambiente MATLAB, sendo que para cada etapa dos passos descritos anteriormente foram geradas funções. Assim com o arquivo do programa principal chamado de main_matlab.m, que sequencia o processo de chamada de cada função, estão também os arquivos com respectivo nome das funções:

- fudFac.m – Função para automatizar o valor de referência fudgeFactor baseada nas intensidades de pixels máximas e mínimas da imagem, a fim de servir como referência no limiar do operador Sobel que será utilizado na função seguinte;
- sobelEdge.m – Função que emprega o Operador Sobel para a detecção de borda da imagem;
- dilateimage.m – Função quer realiza a operação matemática morfológica de dilatação da imagem;
- fillimage.m – Função quer realiza a operação matemática morfológica de preenchimento da imagem.
- erodeimagem.m - Função quer realiza a operação matemática morfológica de erosão da imagem.
- sepbwlabel.m – Função que realiza a identificação de objetos conectivos da imagem para a separação de cada objeto.
- defcrack.m – função que obtém dados de Área, Eixo Maior, Eixo Menor, Orientação de cada objeto para eliminação de objetos que não representam uma fissura.

4.2.3 Algoritmo Baseado em Filtro de Partículas

Para o segundo algoritmo utilizado, aqui considerado como algoritmo B, é empregado um filtro não paramétrico baseado no algoritmo de Bayes.

O filtro de partículas busca relacionar a probabilidade de um segmento de imagem ser caracterizado ou não por uma fissura, baseado na intensidade de pixel e a quantidade de pixels em sua vizinhança.

A Figura 48 representa os passos do algoritmo B.

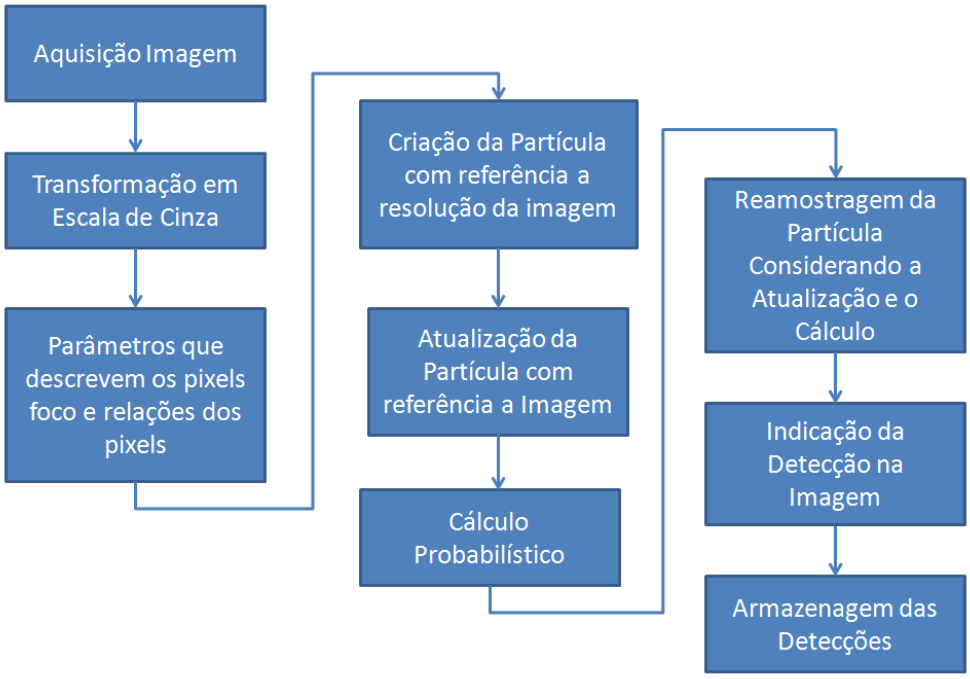


Figura 48 Sequência do método de detecção de fissuras baseada no Filtro de Partículas.

O algoritmo utilizado e simulado no MATLAB™ foi desenvolvido no trabalho realizado por (FURTADO, 2012), onde está disponível o código e a descrição do funcionamento do mesmo, com foco na detecção de rachaduras submarinas. No entanto, foram necessárias algumas modificações para a conversão do código em linguagem de programação C, sem mudar suas características fundamentais. Uma das modificações envolveu a utilização do termo “inf”, cuja informação transmite que um número foi suficientemente grande que não foi possível armazená-lo em uma operação. Por exemplo, a divisão por zero pode resultar nesta situação, o ambiente MATLAB trata desta questão com retorno de informação “inf”. Outra questão a considerar é na passagem de parâmetros de entrada para as funções. A conversão possibilita a passagem de uma matriz n x m, sendo m o numero de linha e n o numero de colunas, convertendo para a linguagem em C tratando estas matrizes como vetores sequenciados pelas linhas, espaçados cada elemento da matriz pelo número de linhas, isto se deve ao processo de conversão adotado na ferramenta. Estas questões são importantes no

momento da utilização das funções em linguagem C, uma vez que será necessária a passagem de parâmetros e das imagens na utilização destas funções geradas.

Os passos do algoritmo são:

1º Passo – Aquisição da imagem, que poderá ser adquirida no modelo de cores **RGB** ou no modelo **YUV** por exemplo.

2º Passo - Transformação da imagem em escala de cinza. Porém, no caso de aquisição pelo modelo YUV, basta processar somente o componente Y, não sendo necessário portanto a transformação linear em escala de cinza.

3º Passo – Atribuição dos parâmetros que descrevem os pixels foco e a relação com os pixels a serem transladados, para as amostras analisadas, sendo que os parâmetros que melhor descreveram as fissuras foram `Xstd_rgb = 10, Xstd_pos = 50, Xstd_vec = 50, Xrgb_trgt = 0.`

Onde Xstd_rgb define o intervalo de confiança para a intensidade do pixel alvo, Xstd_pos é o intervalo de confiança para a posição da partícula, Xstd_vec é o intervalo de confiança para o vetor de deslocamento da partícula e Xrgb_trgt é a matriz da cor alvo, ficando deste modo definida a estratégia para detectar uma fissura com a cor preta como foco na detecção com um intervalo de confiança Xstd_rgb = 10.

Para a criação de partícula segundo a teoria do filtro de partículas quanto maior o número melhor a probabilidade de acerto, assim para os valores pode-se tomar valores suficientemente grandes na ordem de milhares, sendo definido no parâmetro Npop_particles como 10.000, o de vetor de deslocamento que melhor satisfez a detecção nas amostras foi Xstd_vec = 50.

4º Passo – Criação da Partícula com referência aos valores atribuídos ao número de partículas Npop_particles e as dimensões da imagem.

5º Passo – Atualização das Partículas - é realizado o deslocamento dos pixels baseada na incerteza que envolve o valor do vetor de deslocamento e o intervalo de confiança da intensidade do pixel alvo. A Figura 49 representa o processo de atualização da partícula.

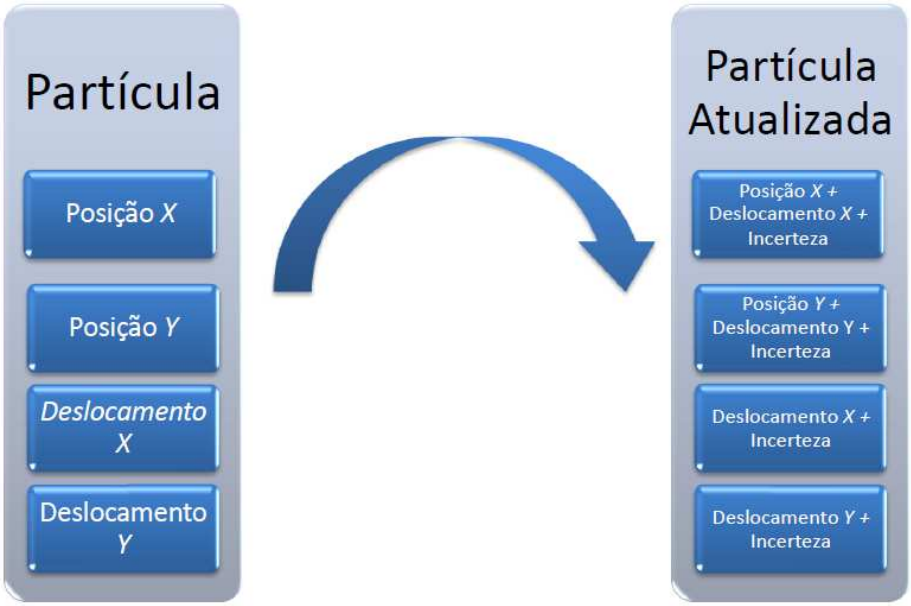


Figura 49 Informações Contidas na partícula e atualização da mesma (FURTADO, 2012)

6º Passo – Cálculo Probabilístico – Neste passo é calculada a probabilidade de uma dada partícula, existir ou não em certa posição, dependendo do valor atribuído a incerteza, dada em Xrgb_trgt, o valor da intensidade do pixel alvo dado em Xstd_rgb e a matriz com as informações de posição e deslocamento das partículas. Para que possa ser obtido um coeficiente que determine à probabilidade de existir de uma partícula foi feita a seguinte lógica: subtraiu-se do valor de referência em escala de cinza da partícula na dada posição o valor escola de cinza do alvo referente ao pixel alvo, com isso se obeve uma matriz de três linhas a qual será multiplicada por sua transposta, tendo como resposta um coeficiente em que sua magnitude é inversamente proporcional à probabilidade da partícula existir. No caso de ser um pixel de baixa intensidade será atribuída uma constante que irá dividir o valor do coeficiente calculado pela intensidade de pixel referência, sabendo que a magnitude deste coeficiente é inversamente proporcional a probabilidade de existir, dividi-lo significa aumentar a probabilidade de a partícula existir. Assim o resultado será uma matriz contendo a probabilidade de existir de cada partícula em uma dada posição.

7º Passo – Reamostragem – Após o calculo das probabilidades das partículas, é realizado a reamostragem pela qual somente as partículas com maior probabilidade sobrevivem, obtendo-se uma matriz com os valores de posições destas partículas com respectivos valores de probabilidade de existência.

8º Passo – Indicação e armazenagem – Com a indicação de existência de fissuras é armazenada a imagem.

No Apêndice A.2 está o algoritmo desenvolvido no ambiente MATLAB, onde estão separados em funções representando as etapas descritas anteriormente. Assim com o arquivo do programa principal chamado de filterparticlecrack.m, sequenciado pelo processo de chamada de cada função descrito a seguir:

- create_particle.m – Criação da Particula
- update_particle.m – Atualização da Particula
- calc_log_likelihood.m – Calculo da probabilidade baseada na intensidade do pixel alvo, na incerteza e na matriz de posição das particulas
- resample_particles.m – Reamostragem para eliminação de partículas com baixa probabilidade de existir.

4.2.4 Resultados obtidos do algoritmo com Detecção de Bordas e Descrição

Na Figura 50 está uma fissura encontrada em revestimento de alvenaria analisada com seus respectivos histogramas, já na Figura 51 é apresentada a sequencia de imagens após a utilização dos métodos para a detecção de fissura na imagem.

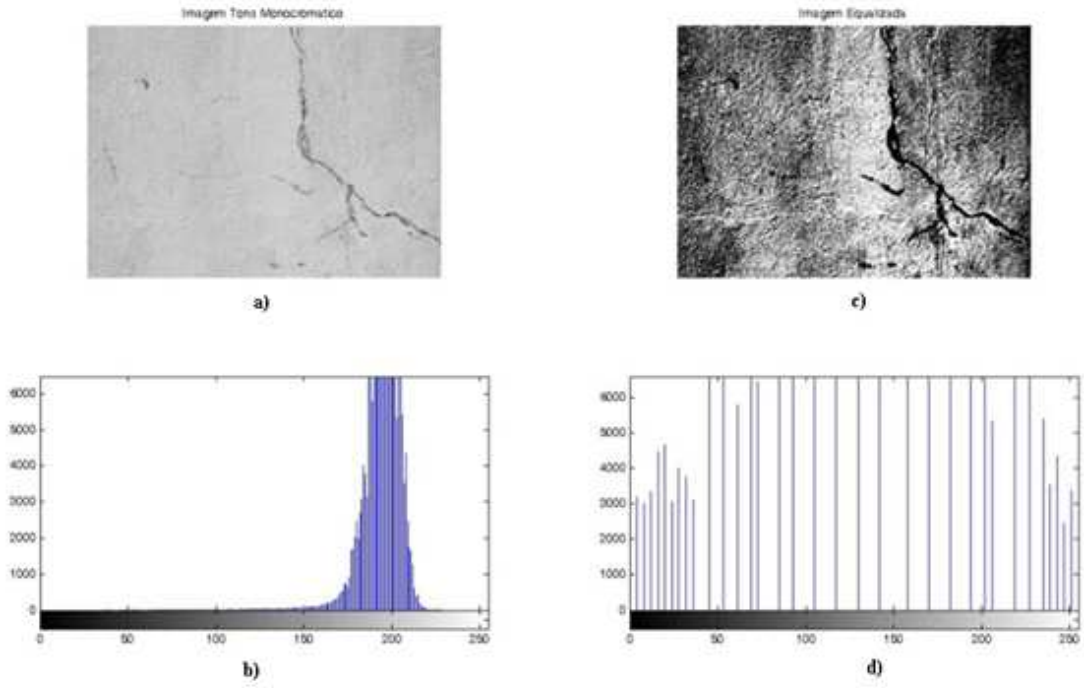


Figura 50 a) Imagem com Fissura em análise b) Histograma da Imagem, c) Imagem Equalizada, d) Histograma da imagem equalizada

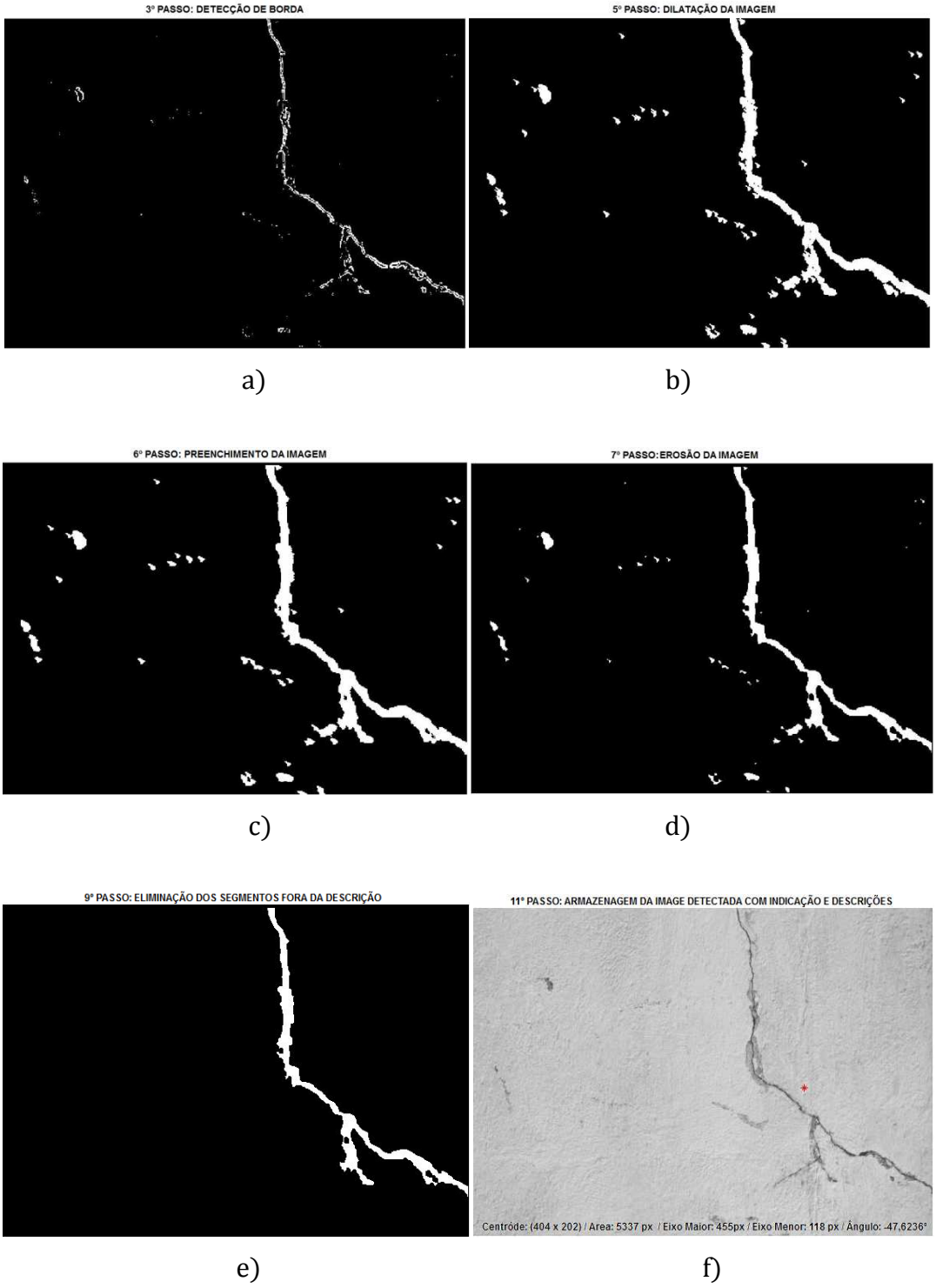


Figura 51 Sequência de imagens dos métodos utilizados para detecção de fissuras; a) Detecção de borda utilizando operador Sobel; b) Dilatação da Imagem; c) Preenchimento da Imagem; d) Erosão da Imagem; e) Eliminação dos objetos fora da descrição; f) indicação dos centroide das fissuras com os dados de Eixo Maior, Eixo Menor e Ângulo de inclinação da fissura.

Pelo histograma observa-se a concentração de pixels com intensidades maiores que representa o fundo da imagem, apresentando então poucos pixels com intensidade menores.

Para o algoritmo, o nível do limiar deverá está abaixo do nível de concentração da faixa destes pixels, de modo a enfatizar os de intensidade menores, sendo estas características de fissuras na imagem.

Devido as texturas comumente encontradas em fachadas revestidas com argamassa, pode-se perceber que existe a presença de pontos de intensidade de pixels baixa, mas que não representa uma fissura. Através da classificação, eliminando os objetos com descrições fora dos limites da equação 30, é possível descartar estes pontos, como mostrado na Figura 51.

A Figura 52 representa a imagem e os respectivos histogramas. Esta imagem apresenta dois tipos de fissura: uma que aparenta ser uma fissura com uma expansão e outra horizontal. A Figura 53, apresenta a sequência do processamento realizado a partir do algoritmo Sobel e na Figura 53 f) está a indicação das duas fissuras assim como as suas descrições. A descrição obtida pela quantidade de pixels do eixo menor e a quantidade de pixels do eixo maior, podem ser utilizadas para uma classificação futura das fissuras, uma vez que para uma fissura por expansão estas quantidades são aproximadas.

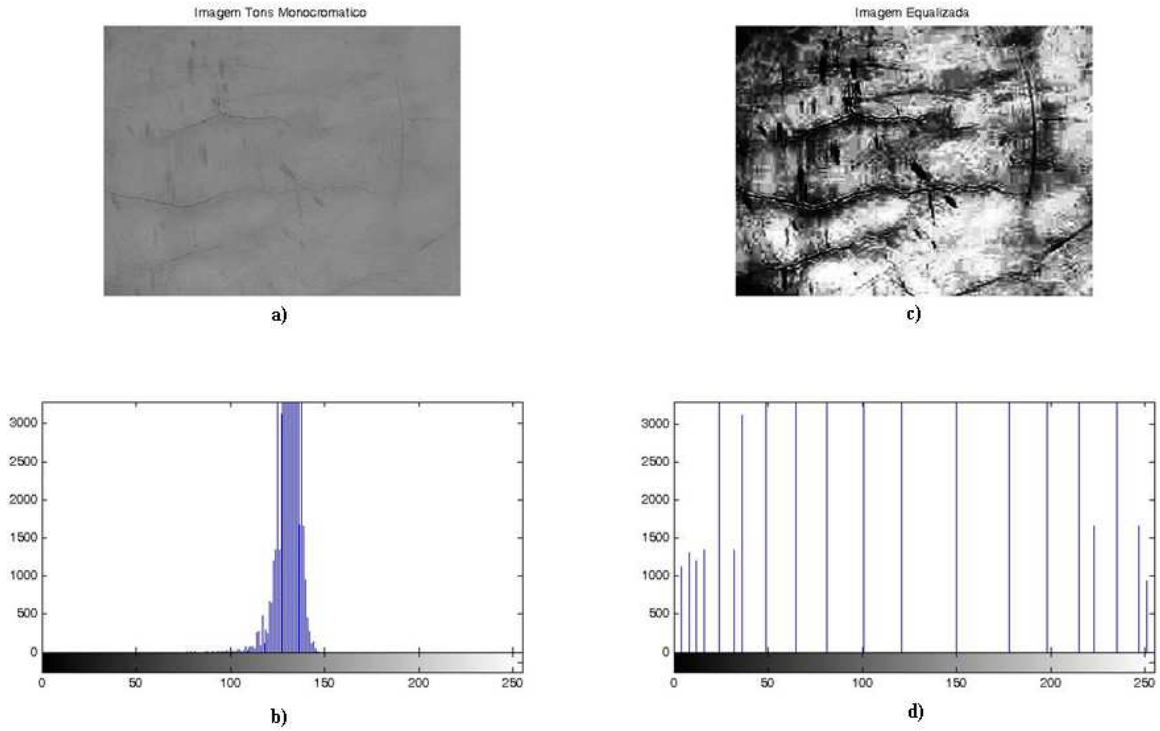


Figura 52 a) Imagem com Fissura em análise b) Histograma da Imagem, c) Imagem Equalizada, d) Histograma da imagem equalizada.

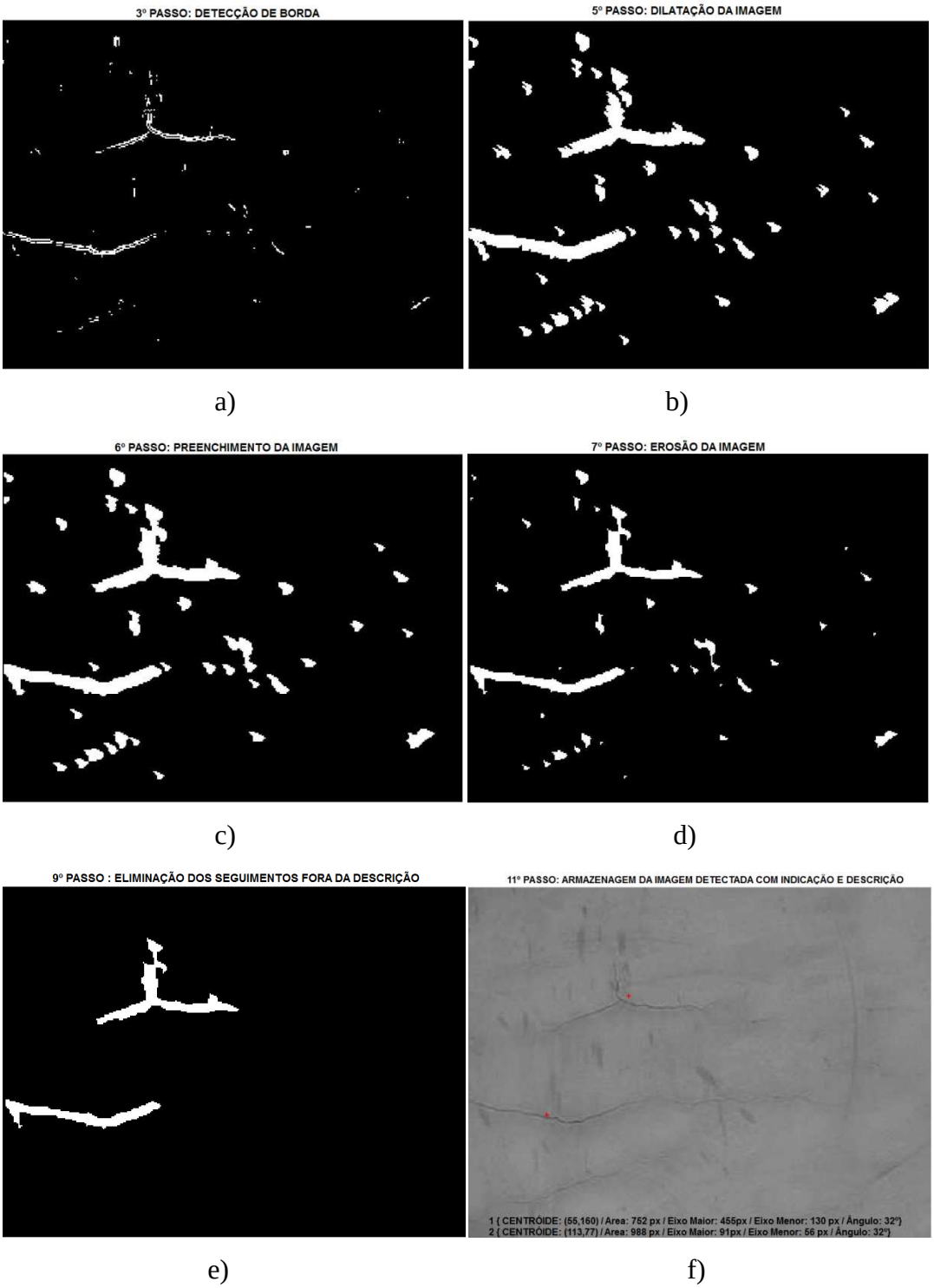


Figura 53– Sequência de imagens dos métodos utilizados para detecção de fissuras; a) Detecção de borda utilizando operador Sobel; b) Dilatação da Imagem; c) Preenchimento da Imagem; d) Erosão da Imagem; e) Eliminação dos objetos fora do padrões considerados; f) indicação dos centroide das fissuras com os dados de Eixo Maior, Eixo Menor e Ângulo de inclinação da fissura.

Na imagem da Figura 53, pode ser observada também a indicação de pontos após a detecção de borda devido à textura da imagem de fundo apresentar pixels com intensidade menores. Assim, nestes pontos apresenta uma magnitude de variação de gradiente que permanece na imagem, sendo eliminados estes pontos após a utilização da descrição do objeto de interesse.

4.2.5 Resultados obtidos do algoritmo Baseado em Filtro de Partículas

Nas Figura 54 e Figura 55 são apresentadas imagens de fissuras com as marcações dos pontos das partículas sobreviventes, indicando a presença de fissura nestes pontos. Na execução do algoritmo a matriz de retorno da reamostragem indica as coordenadas dos pontos dentro da resolução da imagem que possuem a maior probabilidade da partícula existir.

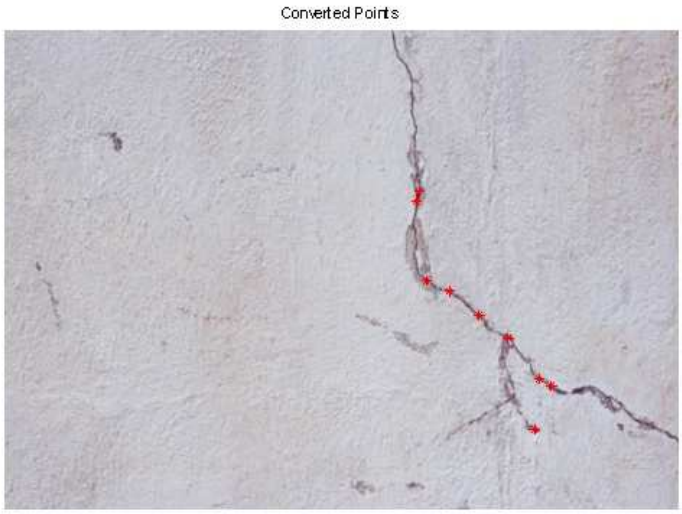


Figura 54 Detecção da fissura através do Filtro de Partículas

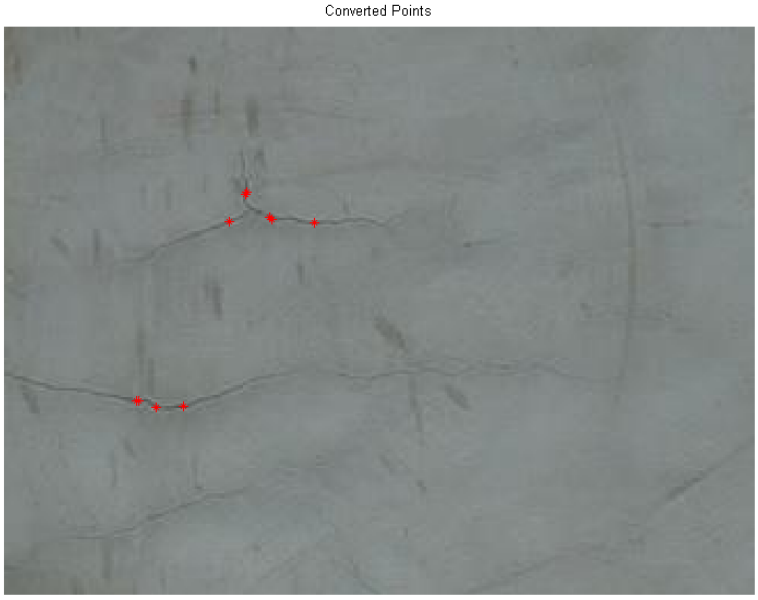


Figura 55 Detecção da fissura através do Filtro de Partículas

O método da detecção por filtro de partículas baseada em cores demonstra ser bem eficiente, embora o método possa retornar falsos positivos, caso ocorra concentração de pixels de baixa intensidade em uma determinada região. Um exemplo disto ocorre em imagens com presença de manchas em uma fachada, após o processamento estas regiões são marcadas como sendo fissura. Mas a sua indicação pode ser útil ainda, uma vez que estas manchas também são consideradas como tipos de manifestações patológicas que podem ser encontradas nas fachadas com revestimento de argamassa.

4.3 GERAÇÃO DO CÓDIGO PARA PLATAFORMAS

Para a geração de código foram usadas ferramentas que traduzem a programação utilizada no ambiente de simulação do MATLAB para linguagens como C, C++ ou para descritores de hardware HDL.

A ferramenta MATLAB Coder ™ gera código C e C + + a partir do código em MATLAB, permitindo também gerar funções MEX, que podem ser executados sem a necessidade deste ambiente, permitindo um processamento mais rápido. Na Figura 56 estão os casos de uso desta ferramenta.

Figura 56 Casos de uso do MATLAB Coder ™ (MATHWORKS, 2012)

A Figura 57 descreve as etapas a serem executadas para a geração de código.

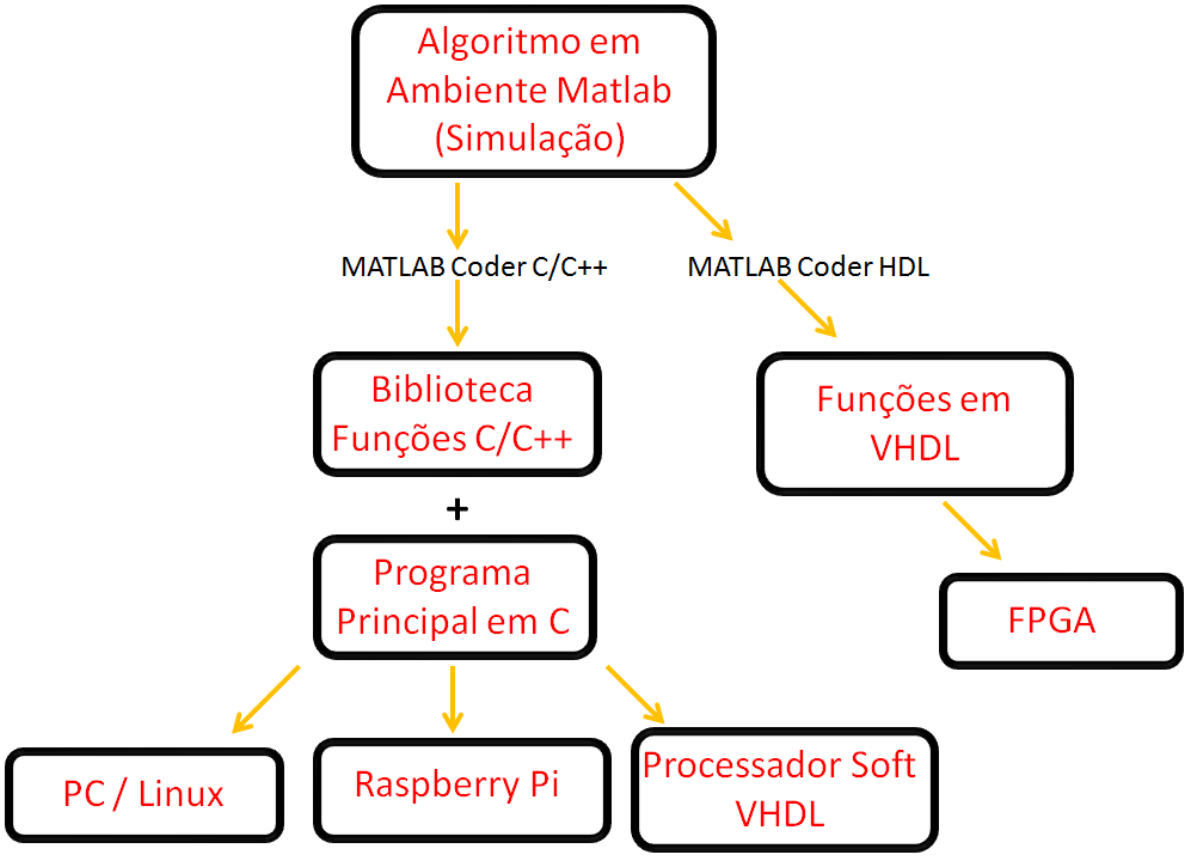


Figura 57 Esquema de Implementação nas Plataformas

Na geração do código em C no MATLAB Coder™ é gerado um relatório que identifica todos os motivos que devem ser sanados no algoritmo MATLAB, compatibilizando-o para geração de código.

No entanto, muitas das funções do Image Processing Toolbox™ não estão disponíveis para conversão do código no ambiente MATLAB Coder™ na versão 2012b, sendo assim

necessária a elaboração do código que realiza o mesmo comportamento das funções do Toolbox.

As partes do código elaborado neste trabalho que descrevem o comportamento das funções presentes no MATLAB, foram desenvolvidos a partir dos conceitos do método da teoria de processamento de imagem digital descritos no capítulo 2 com auxílio de algumas refêrencias com exemplos de códigos para processamento de imagem (BLANCHET, CHARBIT, 2001; SZELISKI, 2010; PARKER, 2011).

O HDL Coder™ sintetiza o código das funções do MATLAB para descritores de hardware Verilog ou VHDL. O código HDL gerado pode ser usado para a programação de FPGA ou ASIC. No entanto, a programação no ambiente MATLAB deve ser pensada de forma a ser implementada em hardware. Por exemplo, na geração do código em HDL não está disponível a geração do código com dados de entradas e saídas na forma de matrizes, sendo necessária a serialização da imagem representada em um dado vetor. Outra questão a ser considerada na geração do código em HDL é a conversão dos dados de ponto flutuantes para ponto fixo, sendo necessário atribuir tamanhos fixos para os dados que armazenarão os números inteiros e dos números fracionários de uma variável.

Nos códigos apresentados no Apêndice A, estão as operações implementadas juntamente com as funções equivalentes do Image Processing Toolbox™ (as funções originais aparecem no código precedidas do símbolo %, tornando-se uma linha de comentário que não será executada na simulação.

Na elaboração das operações foram necessárias adaptações para otimização do código. Por exemplo, na função defcrack.m, as operações que envolvem a obtenção do eixo maior e o eixo menor tiverem que ser adaptadas, pois no MATLAB existe a função “regionprops” que obtém estas medidas de um objeto utilizando as metricas de uma elipse possuindo os seus momentos centrais normalizados. Na adaptação utilizou-se do conceito de extremos do objeto na posição x e na posição y da imagem para obter o eixo maior. Para o eixo menor, o processo foi adaptado a fim de obter a distância em pixels de abertura do objeto, a partir do ângulo do eixo maior, que representa a inclinação da fissura detectada, são calculadas as distâncias entre os pixels na direção de 90º com margem de tolerância de 15º para mais e para menos, levando em consideração assim a natureza sinuosa da fissura, para obter a sua maior abertura em pixels.

Também na geração de código para o algoritmo baseado no filtro de partículas foram necessárias algumas adaptações, como no caso da função calc_log_likelihood, que calcula a probabilidade da partícula que irá persistir, onde atribuições como "inf", que retorna uma

representação aritmética para infinito, foram substituídas por um valor suficientemente grande para a eliminação das partículas após o cálculo probabilístico.

Para a geração do código em C é necessário atribuir-se os tipos das variáveis de entrada assim como o tamanho dos vetores. Na Figura 58 estão definidos os tipos e tamanhos de cada função para a geração das funções do algoritmo A. Um ponto importante a considerar é a possibilidade de utilizar matrizes para os dados de entrada assim como para os dados de saída, embora estes dados na linguagem C sejam tratados como vetores.

Desta maneira, na geração do código foram utilizados dois tamanhos de matrizes: o primeiro tamanho com dimensões de 1944 x 2592, para armazenamento das imagens com 5 Megapixels e o segundo com o tamanho de 1024 x 768, para as imagens de 1 Megapixels. A escolha destas dimensões justifica-se em função de que o dispositivo a ser embarcado possuí câmera modular de até 5 Megapixels. Durante as simulações no MATLAB, os dois tamanhos considerados para as imagens avaliadas não apresentaram diferenças significativas na detecção da fissura, mas sim no tempo de processamento das amostras. Alguns trabalhos relacionados a processamento de imagem, adotam imagem com dimensões consideravelmente reduzidas no sentido de obter um bom desempenho em determinada plataforma. Com a expectativa de se conseguir uma imagem que possa ter boa vizualização, neste trabalho foi adotado um tamanho mínimo de 1 Megapixels, para se obter uma imagem com boa nitidez.

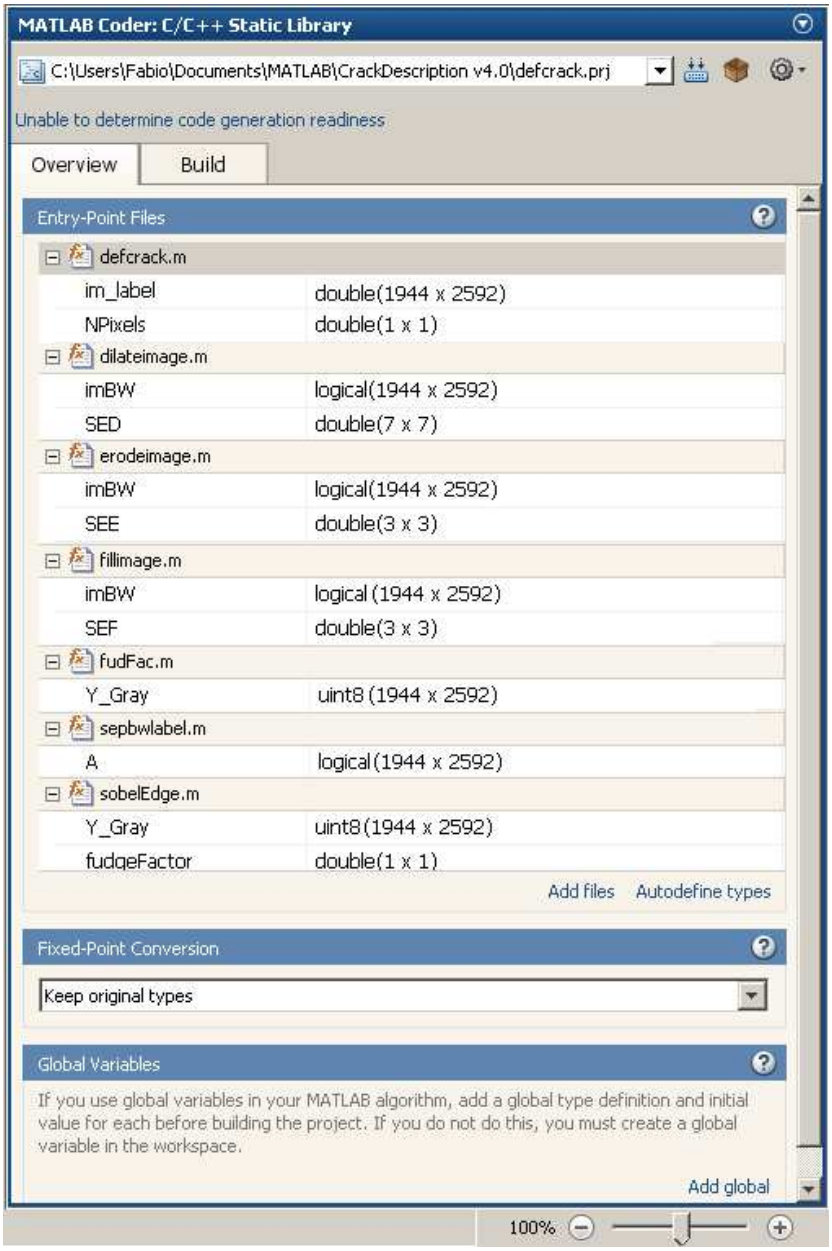


Figura 58 Definições dos tipos e tamanho dos dados de entrada do Algoritmo A

Para a declaração do tipo de dados e tamanhos os arquivos das funções com extensões (.m), devem ter estas definições dadas pela função "`assert`". Assim os dados das matrizes que representam a imagem devem ser definidos com as dimensões da imagem, no caso de imagens com 5 Megapixels são definidos na proporção de 1944 x 2592 a matriz contendo os valores da entrada dos dadosque representa a imagem. Na Tabela 4 estas definições estão relacionadas para o algoritmo A e na Tabela 5 para o algoritmo B.

Tabela 4 – Código de declaração do tamanho e tipo de dados de entrada do algoritmo A

Linhas de Código
`assert(all(size(Y_Gray) <= [1944 2592]));`
`assert(isa(Y_Gray, 'uint8'));`
`assert(all(size(fudgeFactor) == [1,1]));`
`assert(isa(fudgeFactor, 'double'));`
`assert(all(size(imBW) <= [1944 2592]))`
`assert(isa(imBW, 'logical'))`
`assert(all(size(SED) <= [9,9]))`
`assert(isa(SED, 'double'))`
`assert(all(size(SEF) <= [3,3]));`
`assert(isa(SEF, 'double'));`
`assert(all(size(SEE) <= [3,3]))`
`assert(isa(SEE, 'double'))`
`assert(all(size(A) <= [1944 2592]))`
`assert(isa(A, 'logical'))`
`assert(all(size(NPixels) == [1,1]))`
`assert(isa(NPixels, 'double'))`
`assert(all(size(im_label) <= [1944 2592]))`
`assert(isa(im_label, 'double'))`

Selecionando a opção de geração de código com biblioteca estática, a ferramenta Codegen cria os arquivos “.c” contendo o código C das funções com os respectivos headers “.h”, além de outras funções auxiliares que são necessárias para complementar as operações.

Na Figura 59 está a janela de especificação do programa MATLAB com a atribuições dos tipos e tamanhos de cada função para a geração do código em C das funções do algoritmo B.

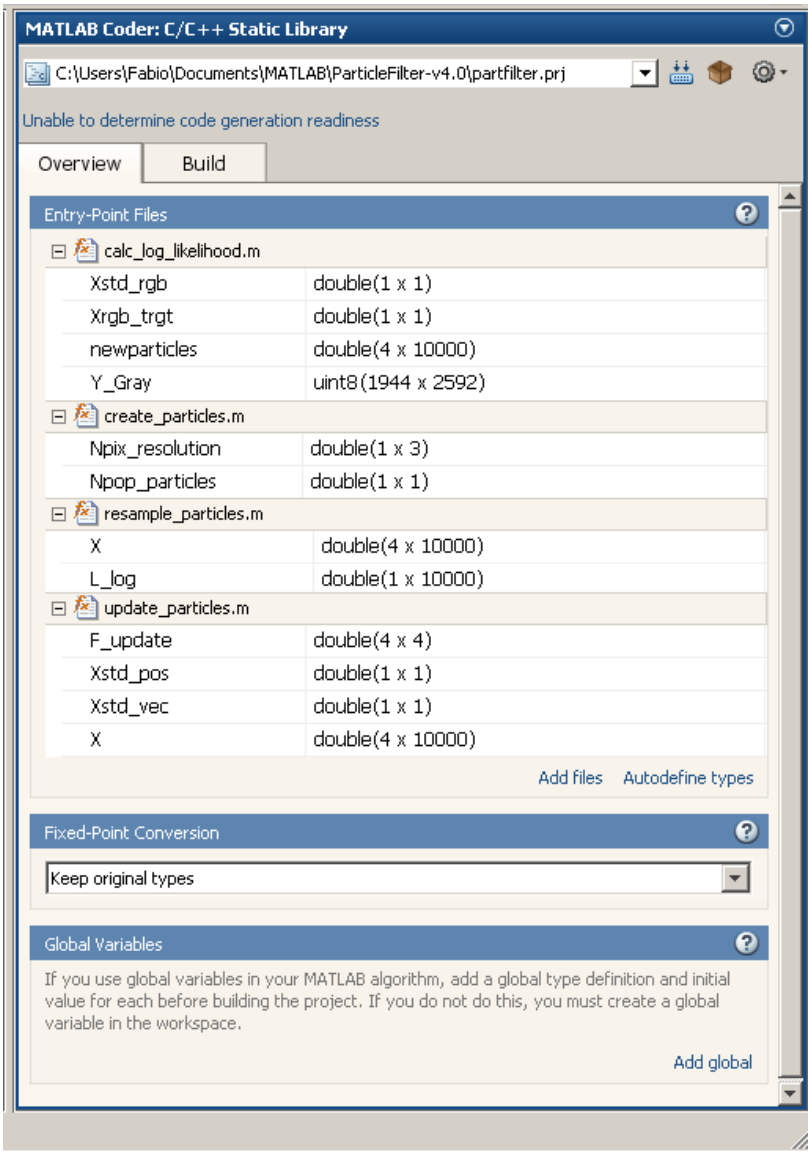


Figura 59 Definições dos tipos e tamanho dos dados de entrada do Algoritmo B

Tabela 5 – Código de declaração do tamanho e tipo de dados de entrada do algoritmo B

Linhas de Código
`assert(all(size(Npix_resolution)==[1 3]))`
`assert(isa(Npix_resolution, 'double'))`
`assert(all(size(Npop_particles)==[1 1]))`
`assert(isa(Npop_particles, 'double'))`
`assert(all(size(Xstd_pos)==[1 1]))`
`assert(isa(Xstd_pos, 'double'))`
`assert(all(size(F_update)==[4 4]))`
`assert(isa(F_update, 'double'))`
`assert(all(size(Xstd_rgb)==[1 1]))`
`assert(isa(Xstd_rgb, 'double'))`
`assert(all(size(Xrgb_trgt)==[1 1]))`
`assert(isa(Xrgb_trgt, 'double'))`
`assert(all(size(newparticles) <= [4 10000]))`
`assert(isa(newparticles, 'double'))`
`assert(all(size(vl_calc)<= [1 10000]))`
`assert(isa(vl_calc, 'double'))`

Para a execução das funções geradas no ambiente MATLAB é necessário o desenvolvimento de um programa principal na linguaguem C que irá tratar da obtenção dos dados da imagem digitalizada, assim como a utilização das funções geradas para a detecção da fissura e posterior armazenamento das informações obtidas.

Como consequência da geração do código pelo Codegen, cujos dados de entrada das funções devem ser de tipos e tamanhos definidos na ferramenta, o programa principal desenvolvido deve trabalhar com a passagem de dados nestes tamanhos assim como nos tipos de dados que a ferramenta atribui para cada variável.

No arquivo gerado pela ferramenta com o nome de rtwtypes.c estão localizados os tipos de dados, sendo alguns apresentados na Tabela 6. Estes tipos de variáveis foram declarados no programa principal para tratar da passagem de parâmetros entre as funções. No caso da necessidade da conversão entre estes tipos de dados, deve ser considerado o tamanho para não haver perda de dados.

Tabela 6 –Tipos de dados definidos na geração do Código em C

Nome de dados gerados pelo Codegen	Tipo de Dados
int8_T	signed char
uint8_T	unsigned char
int16_T	short
uint16_T	unsigned short
real_T	double
boolean_T	unsigned char
int_T	int

No Apêndice estão os dois programas principais desenvolvidos em linguagem C que utilizam a chamada das funções dos algoritmos que foram gerados pela ferramenta MATLAB para processamento das imagens, obtidas em formato jpeg. O algortimo A está disponível no Apêndice A.3 e o algoritmo B no Apêndice A.4. Nestes programas principais estão os comentários de comportamento das instruções utilizadas.

Nas plataformas utilizadas, que possuem ambiente Linux, a compilação do programa principal junto com as funções geradas, é realizada utilizando o GNU Compiler Collection

(GCC), através de programas do tipo ¨Makefiles¨, os quais incluem as regras de compilação do projeto de software. Estes programas são apresesentados nos Apêndices A.5 (geração de código para o algoritmo A) e A.6 (para o algoritmo B). A compilação gera um programa executável em conjunto com uma biblioteca estática.

Todas as imagens e arquivos utilizados neste trabalho encontram-se diponível em https://www.dropbox.com/sh/oabtq5f0r7hd78m/AAA-3P4eam6Wy1bbnSbzSYTHa?dl=0

4.4 PLATAFORMAS UTILIZADAS

A seguir estão as plataformas utilizadas para a verificação dos comportamentos dos algoritmos, tanto em um ambiente embarcável como um processamento em uma estação que ofereça um desempenho melhor.

4.4.1 – Computador com Sistema Operacional Linux

Para o teste dos algoritmos utilizando uma plataforma computacional representando uma estação em solo e não embarcada no VANT, na qual requisitos de consumo e peso não são tão restritivos, utilizou-se da seguinte plataforma:

- Intel Core Duo CPU P8400 2,27GHz
- Memória RAM 2GB
- Sistema Operacional Linux Slackware 13.37

4.4.2 – Raspberry PI

A escolha de utilização em uma plataforma multitarefa que possibilite o processamento da imagem como Raspberry Pi torna-se atraente devido ao seu tamanho reduzido e baixo consumo, na ordem de 700mA a 1000mA dependendo do modelo. Este dispositivo tem todo o seu hardware integrado em uma única placa e tem sido utilizado em uma grande quantidade de projetos. O dispositivo é baseado em um SoC integrado (SoC-System on Chip) Broadcom BCM2835, que inclui um processador ARM1176JZF-S de 700 MHz, GPU VideoCore IV e 512 MB de memória RAM. Nesta plataforma não está incluido um disco rígido como nos computadores, entretanto oferece entrada para uma memória não volátil do tipo micro SD para armazenamento de dados, onde o sistema operacional é instalado. A alimentação do dispositivo

é realizada através de uma porta micro USB, presente na placa com tensão de 5V, sendo o consumo dependente dos periféricos acoplados no mesmo, a corrente máxima nestes dispositivos pode chegar a 1,2 A(amperes) (RASPBERRY PI, 2014).

Atualmente o software de gerenciamento de instalação chamado de NOOBS (New Out Off the Box Software) oferece a possibilidade de instalação de sistemas operacionais como Raspbian, que é um sistema operacional livre baseado em Debian, otimizado para o hardware Raspberry Pi, oferecendo a facilidade de programação e compilação de sistemas baseados em Linux.

4.4.3 – FPGA com Microblaze

Para a análise dos algoritmos em uma plataforma utilizando microprocessador soft, foi empregada uma versão Microblaze juntamente com uma placa FPGA Virtex 6 da Xilinx, cujas características técnicas são apresentadas abaixo:

- MicroBlaze Versão 8.40.a, com síntese com ênfase em Performance e com Full_MMU para o Linux;
- Clock 100MHz;
- 512MB RAM;
- Com Extended Floating Point Unit;
- Divisor e Multiplicador em hardware;
- Barrel Shifter;
- FPGA Xilinx XC6VLX240T, Placa de desenvolvimento Xilinx ML605;
- Ferramenta Xilinx Platform Studio 14.2;
- Linux Kernel 3.14.

Estes dispositivos foram escolhidos como base para o processamento e a análise do desempenho dos algoritmos descritos anteriormente, os resultados obtidos em cada plataforma possibilitam uma percepção do comportamento dos algoritmos utilizados permitindo a sua utilização para uma inspeção futura. Cada plataforma possui características distintas, para uma

plataforma computacional seu conceito visa a sua utilização em uma quantidade consideravelmente superior de tarefas a executar no seu sistema operacional, pois foi projetado para atender um uso geral ao nível do usuário, o Raspberry Pi as tarefas são consideravelmente reduzidas, e suas operações são otimizadas para o seu hardware, sendo projetado para uso de desenvolvimento e aprendizado, que tem sido utilizado para algumas pesquisas, enquanto na FPGA pode ter o seu processamento dedicado a operações especificas realizadas por uma determinada tarefa, em geral repetitiva.

5 RESULTADOS

Neste capítulo serão apresentados os resultados obtidos na execução dos algoritmos, nas plataformas utilizadas.

5.1 TEMPOS DE EXECUÇÃO DOS ALGORITMOS NAS DIFERENTES PLATAFORMAS

A Figura 60 apresenta o tempo total de execução do algoritmo A no ambiente MATLAB™, detalhando o tempo de execução de cada uma das funções do algoritmo. Com o tempo total de 76,864 segundos é possível verificar que a função defcrack demanda mais tempo na execução de analise de uma imagem com porcentagem de 72,0% do total.

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
defcrack	function	1	55.374 s	72.0%	
sepbwlabel	function	1	17.921 s	23.3%	
sobelEdge	function	1	2.513 s	3.3%	
dilateimage	function	1	0.569 s	0.7%	
fillimage	function	1	0.165 s	0.2%	
erodeimage	function	1	0.085 s	0.1%	
imread	function	1	0.066 s	0.1%	
close	function	1	0.059 s	0.1%	
fudFac	function	1	0.004 s	0.0%	
Self time (built-ins, overhead, etc.)			0.108 s	0.1%	
Totals			76.864 s	100%	

Figura 60 Tempo de execução de uma Imagem de 1Mpx ambiente MATLAB™ algoritmo A

Na Figura 61 estão os tempos das tarefas do algoritmo B, com tempo total de 0,405s. A função que desempenha o calculo probabilístico da partícula foi a de maior custo computacional, demandando cerca de 59,5% do tempo total de execução.

Children (called functions)

Function Name	Function Type	Calls	Total Time	% Time	Time Plot
calc_log_likelihood	function	1	0.241 s	59.5%	
imread	function	1	0.070 s	17.3%	
close	function	1	0.007 s	1.7%	
resample_particles	function	1	0.005 s	1.2%	
update_particles	function	1	0.004 s	1.0%	
create_particles	function	1	0.003 s	0.7%	
Self time (built-ins, overhead, etc.)			0.075 s	18.5%	
Totals			0.405 s	100%	

Figura 61 Tempo de execução de uma Imagem de 1Mpx ambiente MATLAB Algoritmo B

Verifica-se que o algoritmo B possui um tempo consideravelmente menor de processamento do que o algoritmo A, o que era esperado, dado a sua menor complexidade. A Tabela 7 apresenta uma relação do número efetivo de linhas do código no ambiente MATLAB para os 2 algoritmos, onde pode-se verificar um grande numero de instruções para o algorimto Sobel.

Tabela 7 – Relação do número efetivo de linhas de instrução dos códigos no ambiente MATLAB

Environments	Lines of code (LOC)	Executable file (bytes)
Algorithm Particle Filter	118	3.977
Algorithm Sobel	392	21.882

Através da compilação do programa desenvolvido em C e das funções geradas no MATLAB, utilizando o programa GNU Compiler Collection (GCC) obteve-se um programa executável que pode ser utilizada em arquiteturas com sistema operacional Linux. Foram realizados testes de desempenho nas plataformas RaspberryPi e em um computador desktop, sendo que Figura 62 apresenta o gráfico comparativo do tempo de execução do algoritmo A nas duas plataformas. Neste estudo de caso foram usadas 49 amostras de imagem com dimensões de 1 megapixels, sendo que os tempos médios de execução no computador desktop foi de cerca de 3,98 segundos, enquanto no Raspberry Pi ficou por volta de 80,59 segundos, uma razão de aproximadamente 20 vezes maior de tempo de execução. No eixo das abscissas do gráfico está representado o numero de objetos presentes, após o tratamento inicial da

imagem, realizado pela detecção de borda, da dilatação, do preenchimento, da erosão e finalmente da identificação dos objetos. O número destes objetos influencia no tempo total devido a quantidade de verificações necessárias das descrições de cada etapa, a serem realizadas pela tarefa de descrição e eliminação dos objetos fora de padrão, determinada através dos parâmetros de área, eixo maior e eixo menor.

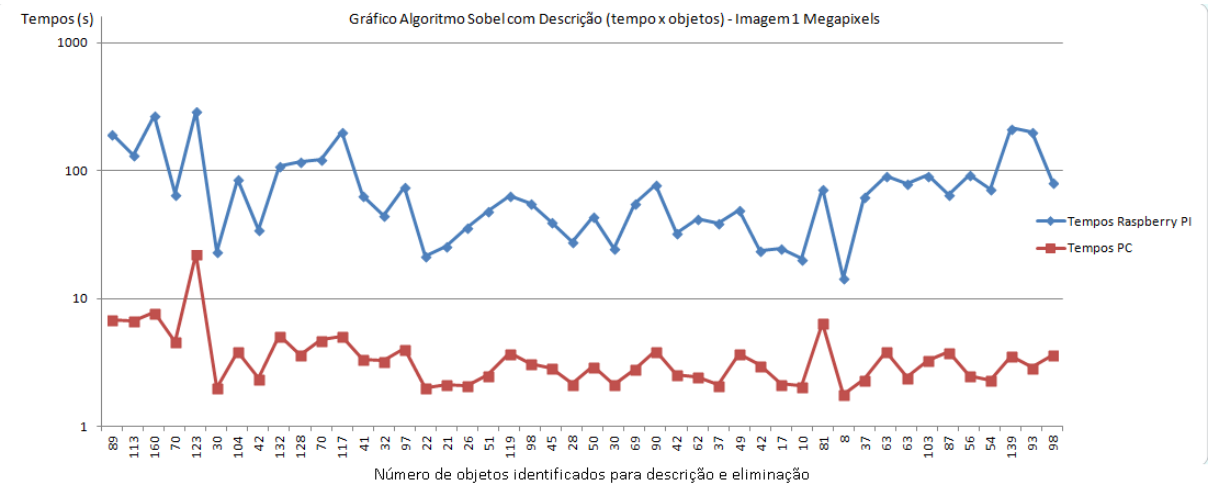


Figura 62 Gráfico comparativo de tempo de execução nas plataformas do Algoritmo A para Imagens de 1Megapixels

Com o processamento das imagens de 5 Megapixels nas duas plataformas percebe-se um acréscimo de tempo médio de um pouco mais de 5 vezes na proporção para as imagens processadas na plataforma Raspberry PI e de 6 vezes no computador, com os tempos médios de 432,60 a 26,56, respectivamente. A Figura 63 apresenta o gráfico comparativo para as 48 imagens de 5 Megapixels.

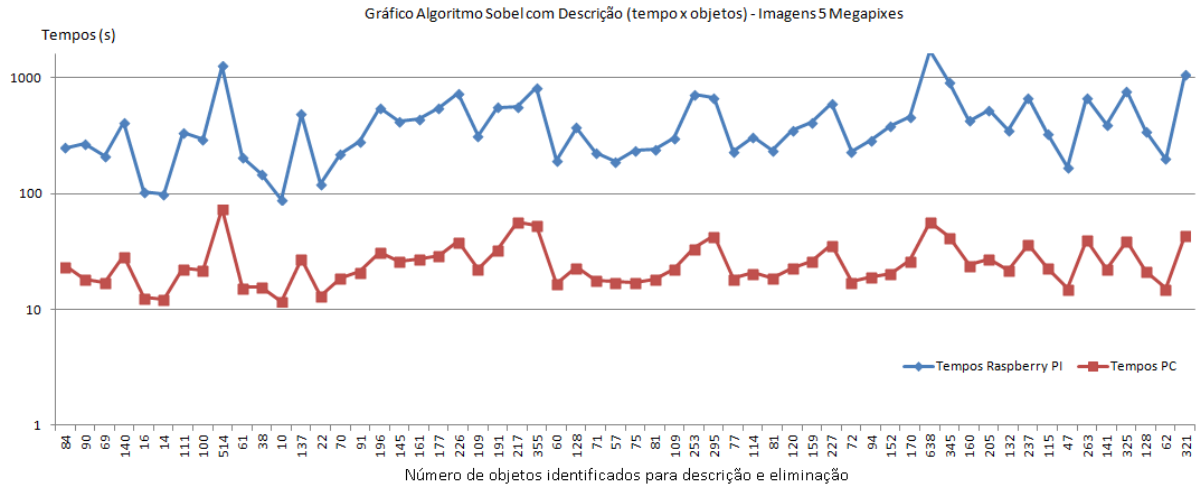


Figura 63 Gráfico comparativo de tempo de execução nas plataformas do Algoritmo A para imagens de 5 Megapixels

Novamente, no caso do algoritmo B, que utiliza o Filtro de Partículas, os tempos de execução foram consideravelmente menores. Na Figura 64 é apresentado o gráfico comparativo dos tempos de execução do algoritmo nas duas plataformas. Durante o processo de obtenção dos dados temporais nota-se a presença de alguns picos momentâneos nos tempos de execução. As variações se devem ao processamento de outras tarefas no processador, devido à arquitetura multitarefa do ambiente de processamento. A média de tempo para o computador desktop foi de 0,137 segundos enquanto no Raspberry Pi este tempo foi de 0,897 segundos, uma proporção de mais de 6 vezes superior no Raspberry em relação ao computador para imagens processadas de dimensão de 1 Megapixels.

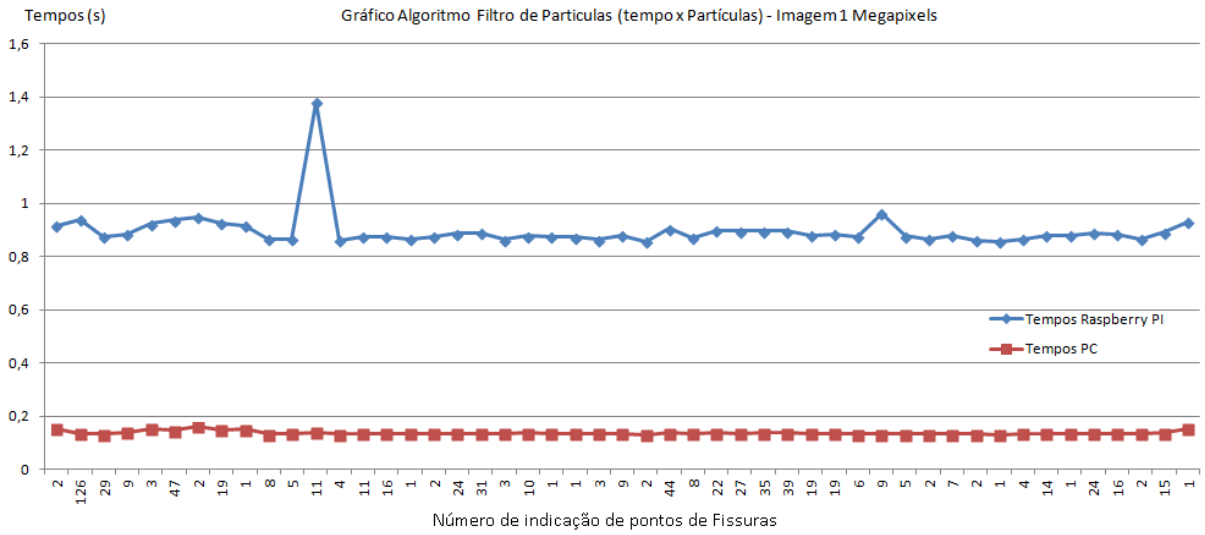


Figura 64 Gráfico comparativo de tempo de execução nas plataformas do Algoritmo B para imagens de 1 Megapixels

A comparação temporal das imagens de 5 Megapixels nas duas plataformas é apresentada na Figura 65. Neste caso a proporção de tempos médios com relação às imagens de 1 Megapixels foi de cerca de 6,7 vezes maior na plataforma Raspberry Pi e de 5,8 vezes no computador, com tempos médios de cerca de 0,802 segundos e de 6,001 segundos respectivamente.

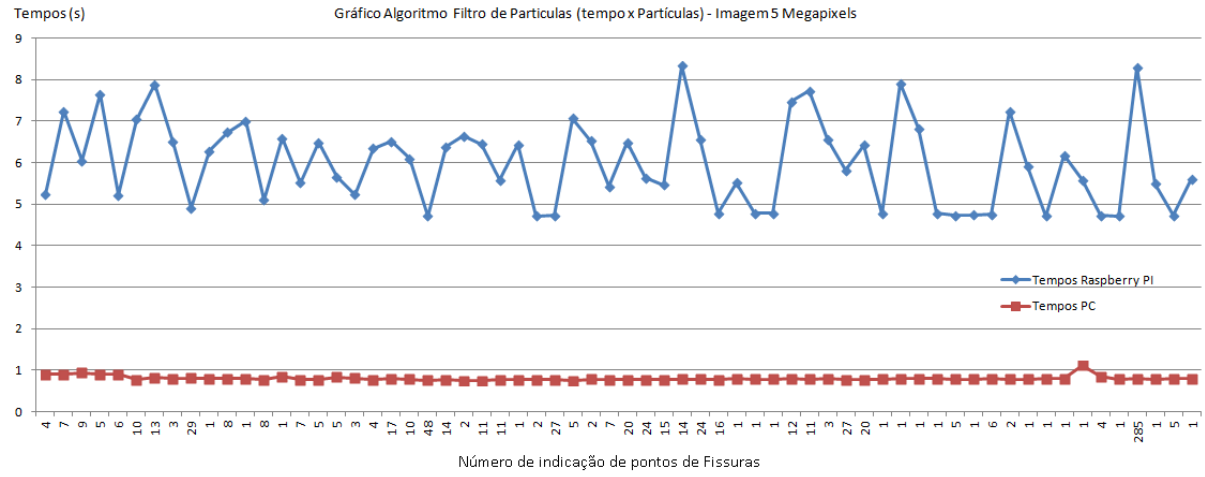


Figura 65 Gráfico comparativo de tempo de execução nas plataformas do Algoritmo B para imagens de 5 Megapixels

Comparando os gráficos anteriores a possibilidade de utilização do algoritmo Filtro de Partículas na plataforma Raspberry torna-se viável pelo melhor desempenho temporal.

Para a execução de cada etapa dos algoritmos desenvolvidos foram obtidos os tempos de execução de cada sub-programa executados. Na Figura 66 e naFigura 67, estão representados os percentuais de tempo de cada tarefa do algoritmo A, executado na plataforma Raspberry PI com imagens de 1 Megapixels e 5 Megapixels, respectivamente.

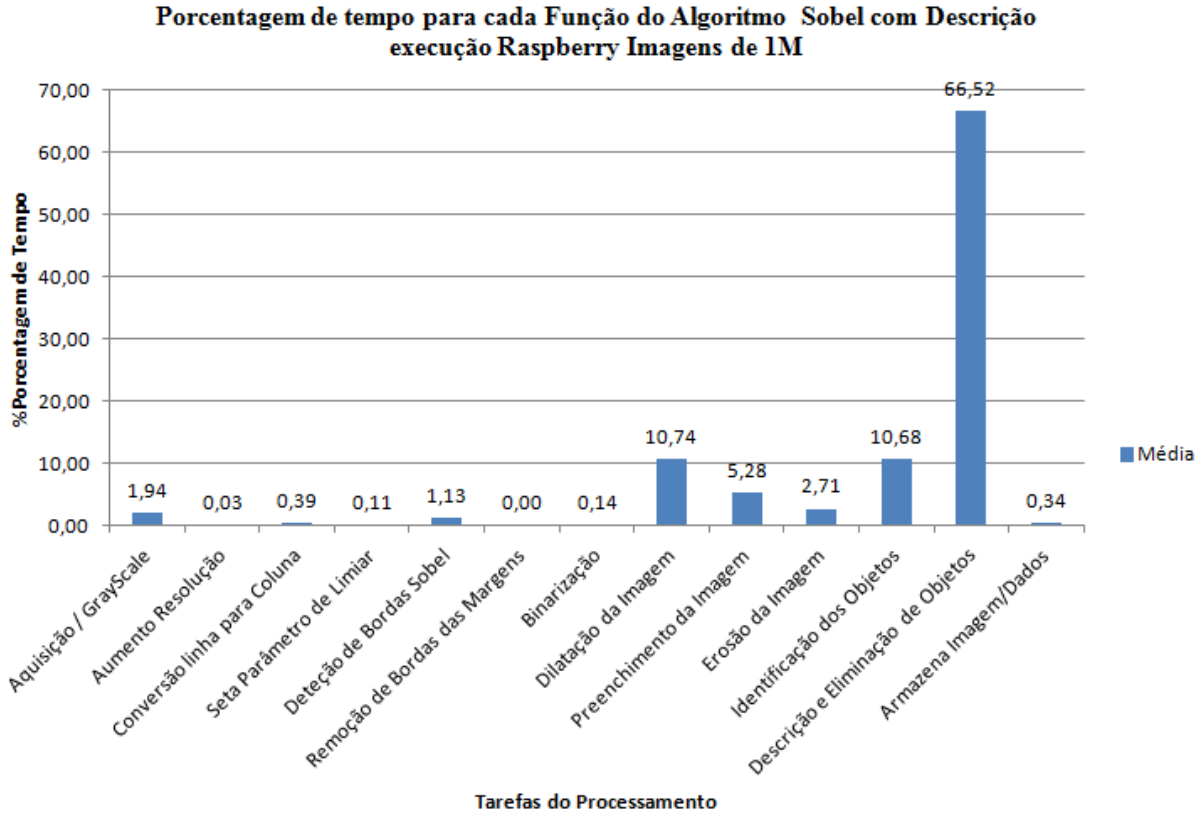


Figura 66 Gráfico do Percentual de tempo do algoritmoA na execução nas plataforma Raspberry Pi com imagens de 1 Megapixels

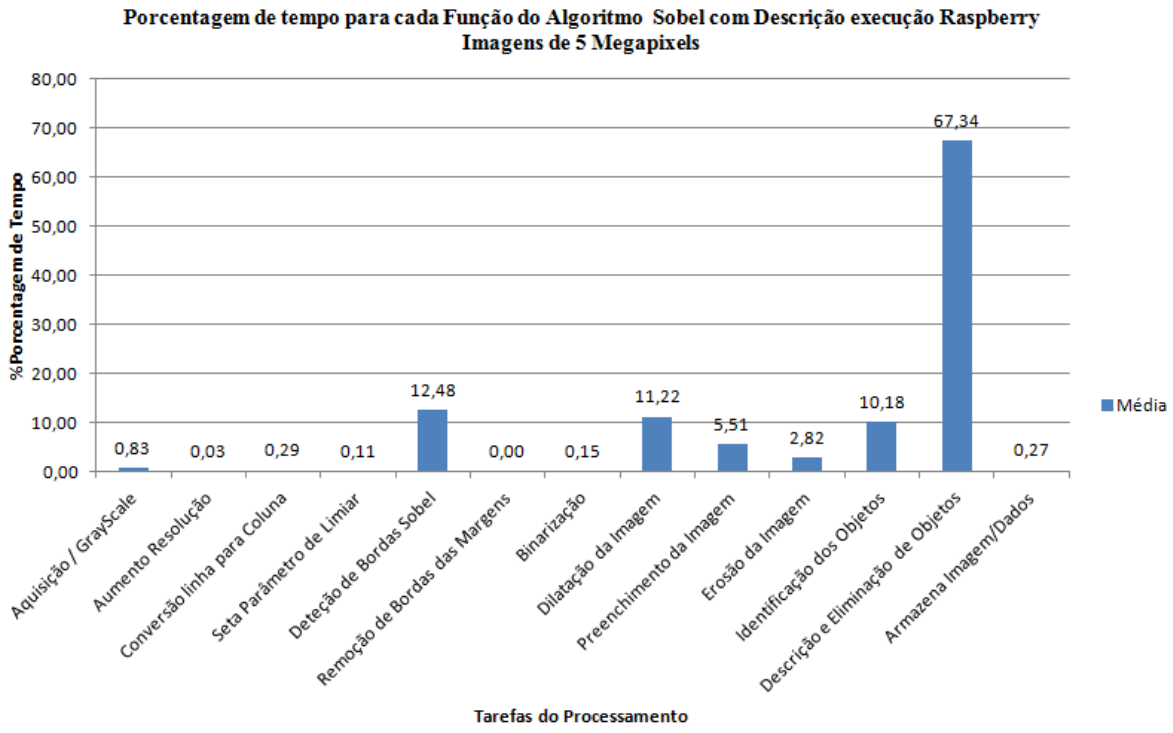


Figura 67 Gráfico do Percentual de tempo do algoritmoA na execução no Raspberry com imagens de 5 Megapixels

A função associada à tarefa que verifica a descrição dos objetos e realiza o processo de classificação, eliminando aqueles que estão fora de padrão característico de uma fissura, representa de 66 a 67% dos tempos médios de execução de todo o programa, enquanto que as tarefas que realizam o processo de morfologia matemática na imagem analisada demandam entre de 2% a 11%.

Um fato importante quanto à função de detecção de borda, é que está demandou um percentual médio de 1,13% para imagens de 1 Megapixels enquanto que para imagens de 5 Megapixels este percentual foi maior, na ordem de 12,48% do tempo total do programa.

Na Tabela 8 e na Tabela 9 estão os tempos de cada tarefa do programa de detecção do algoritmo A para a plataforma Raspberry PI. Também neste caso deve-se levar em consideração que o programa está sendo executado em conjunto com outros processos, devido ao ambiente de multitarefas. Os tempos obtidos são variáveis, na proporção da concorrência do uso do processador. Deste modo, uma análise do tempo médio de execução, juntamente com o seu desvio padrão, servem como referência para análise de cenários de empregabilidade destes algoritmos.

Existe uma grande variação de tempo, principalmente na função de descrição e eliminação dos objetos. Com os dados obtidos verificou-se que quanto maior o número de

objetos a serem analisados após o tratamento inicial da imagem com a aplicação do operador Sobel e da morfologia matemática, maior é o tempo desta função.

Tabela 8 – Tempo das Tarefas na plataformas Raspberry Pi do Algoritmo A para imagens de 1 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição/ GrayScale	0,934339	1,210044	0,267997	0,401110
Ajuste na Dimensão Imagem	0,015517	0,016920	0,015160	0,000246
Ajuste Sequencia de dados Imagem	0,186597	0,188273	0,185641	0,000776
Seta Parâmetro de Limiar	0,054593	0,056427	0,053990	0,000479
Deteção de Bordas Sobel	0,543385	0,550492	0,536765	0,003452
Remoção de Bordas das Margens	0,000215	0,000289	0,000014	0,000036
Binarização	0,068011	0,070047	0,067324	0,000717
Dilatação da Imagem	5,176647	5,205560	5,160415	0,010736
Preenchimento da Imagem	2,543720	2,575744	2,536011	0,009247
Erosão da Imagem	1,303361	1,322924	1,299613	0,005139
Identificação dos Objetos	8,671356	34,546485	0,234517	7,491773
Descrição e Eliminação de Objetos	60,942094	267,219527	2,458500	60,406209
Armazena Imagem/Dados	0,157539	1,021151	0,121665	0,138955
Tempo Total	80,597374	288,576834	14,324149	65,641125

Tabela 9 – Tempo das Tarefas na plataformas Raspberry Pi do Algoritmo A para imagens de 5 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição/ GrayScale	2,525916	3,160398	1,788016	0,368089
Ajuste na Dimensão Imagem	0,097607	0,105245	0,095568	0,001701
Ajuste Sequencia de dados Imagem	0,846157	0,872748	0,834039	0,009813
Seta Parâmetro de Limiar	0,337121	0,350846	0,335467	0,001993
Deteção de Bordas Sobel	36,839619	37,630320	36,372680	0,272929
Remoção de Bordas das Margens	0,000623	0,000952	0,000607	0,000048
Binarização	0,434475	0,439549	0,432802	0,001422
Dilatação da Imagem	33,080307	33,279390	33,012498	0,049896
Preenchimento da Imagem	16,264231	16,348534	16,222518	0,029988
Erosão da Imagem	8,317102	8,367331	8,305147	0,013293
Identificação dos Objetos	55,221219	355,054882	1,475351	70,558743
Descrição e Eliminação de Objetos	311,000392	1356,097292	20,780970	243,236511
Armazena Imagem/Dados	0,798486	0,836514	0,742237	0,020957
Tempo Total	432,607596	1733,053562	88,760146	299,848425

Nas Figura 68 e Figura 69, estão representados os percentuais de tempo de cada tarefa do algoritmo A, executado no computador com imagens de 1 Megapixels e 5 Megapixels, respectivamente.

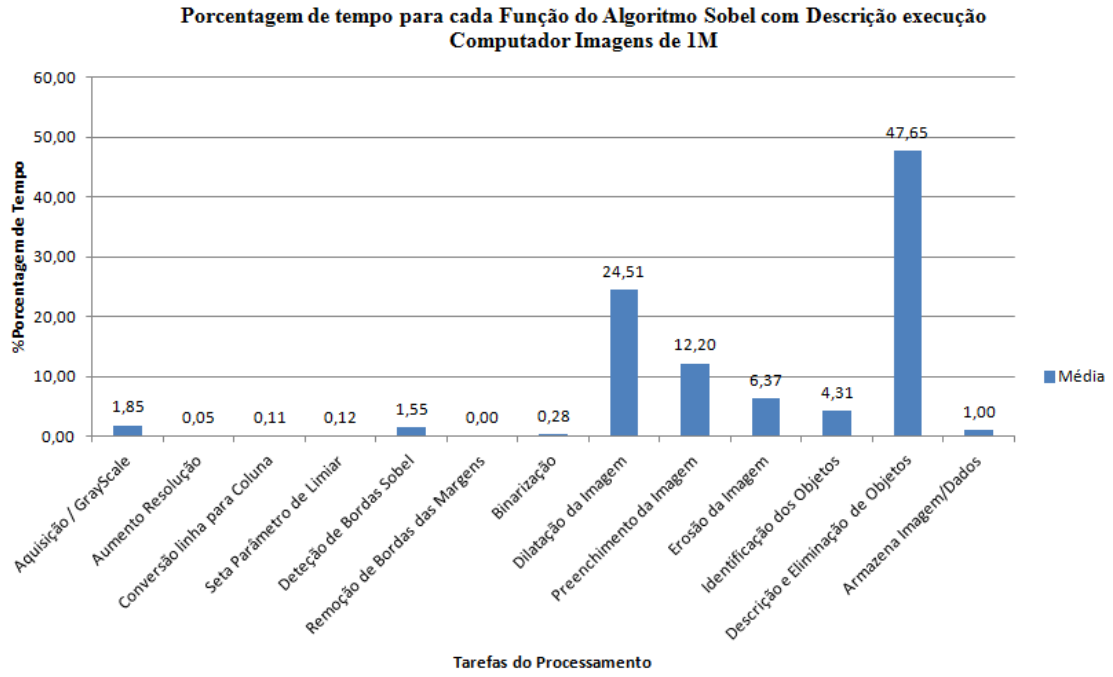


Figura 68 Gráfico do Percentual de tempo do algoritmoA na execução no computador com imagens de 1 Megapixels

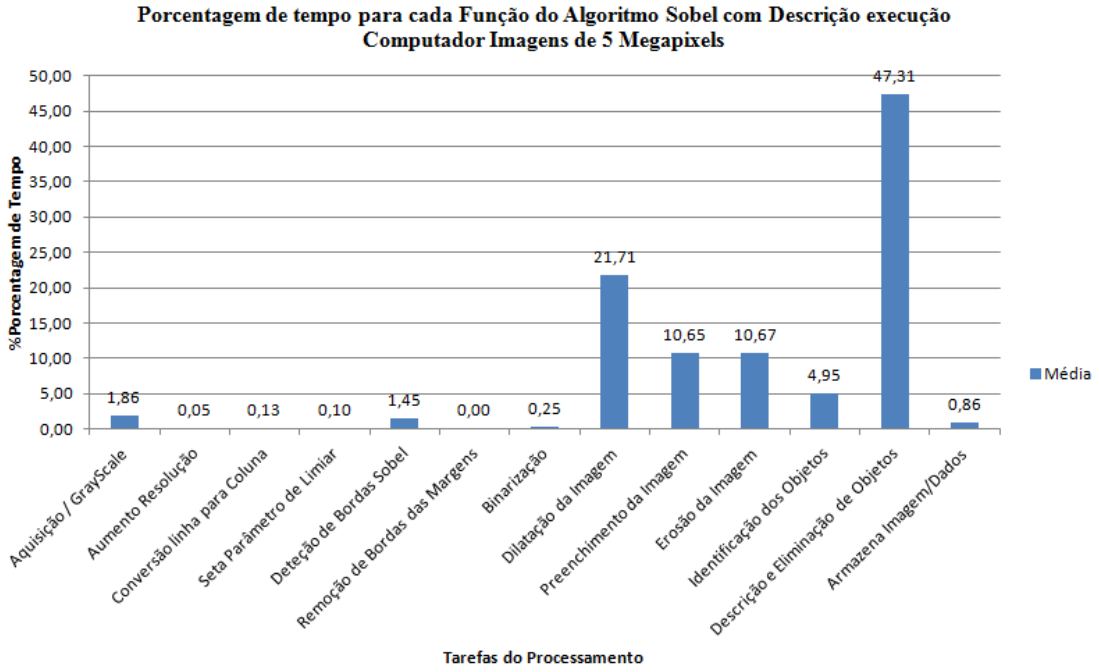


Figura 69 Gráfico do Percentual de tempo do algoritmoA na execução no computador com imagens de 5 Megapixels

A função que verifica a classificação dos objetos também ocupa considerável proporção do tempo de execução com média de 47% de toda a duração média de processamento na execução do programa. A função da dilatação da imagem também é um processo crítico na execução do programa no computador com 24% do tempo médio total.

Nas Tabela 10 e Tabela 11 são apresentados os tempos de cada tarefa do programa de detecção do algoritmo A para execução no computador desktop, observando que os tempos dexecução médios para as imagens de 5 megapixels foram cerca de 26,6 segundos, apresentando picos de tempo de 74,1 segundos (devido às interrupções das tarefas). Já nas imagens de 1 megapixels os tempos médios são de aproximadamente 3,98 segundos com picos de 22,23 segundos.

Tabela 10 – Tempos das Tarefas no Computador do Algoritmo A para imagens de 1 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição/ GrayScale	0,058000	0,073847	0,054303	0,004973
Ajuste na Dimensão Imagem	0,001597	0,001624	0,001580	0,000009
Ajuste Sequencia de dados Imagem	0,003298	0,003338	0,003276	0,000013
Seta Parâmetro de Limiar	0,003713	0,003790	0,003691	0,000020
Deteção de Bordas Sobel	0,048029	0,049356	0,047819	0,000243
Remoção de Bordas das Margens	0,000035	0,000062	0,000004	0,000007
Binarização	0,008719	0,008785	0,008691	0,000020
Dilatação da Imagem	0,760259	0,765282	0,758098	0,002202
Preenchimento da Imagem	0,378621	0,383521	0,377472	0,001175
Erosão da Imagem	0,197719	0,198903	0,197053	0,000525
Identificação dos Objetos	0,171037	0,715991	0,016259	0,152219
Descrição e Eliminação de Objetos	2,324730	20,507494	0,266006	3,146133
Armazena Imagem/Dados	0,031254	0,040854	0,027616	0,002270
Tempo Total	3,987012	22,237641	1,787196	3,203406

Tabela 11 – Tempos das Tarefas no Computador do Algoritmo A para imagens de 5 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição/ GrayScale	0,425117	0,760464	0,342609	0,131289
Ajuste na Dimensão Imagem	0,010619	0,014543	0,010240	0,000572
Ajuste Sequencia de dados Imagem	0,030048	0,044702	0,029513	0,001966
Seta Parâmetro de Limiar	0,023579	0,023654	0,023522	0,000032
Deteção de Bordas Sobel	0,326440	0,357307	0,318455	0,009941
Remoção de Bordas das Margens	0,000100	0,000263	0,000089	0,000024
Binarização	0,056134	0,059708	0,055558	0,000933
Dilatação da Imagem	4,887885	4,961764	4,863609	0,018550
Preenchimento da Imagem	2,399420	2,414473	2,387597	0,006772
Erosão da Imagem	2,401961	2,415035	2,392513	0,005666

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Identificação dos Objetos	1,562656	8,320232	0,119636	1,810072
Descrição e Eliminação de Objetos	14,242479	57,787614	1,017250	11,283246
Armazena Imagem/Dados	0,194714	0,207972	0,175544	0,007851
Tempo Total	26,561151	74,141130	11,805853	12,524462

Nas Figura 70 e Figura 71, estão representados o percentual de tempo de cada tarefa do algoritmo B, executado na plataforma Raspberry PI com imagens de 1 Megapixels e 5 Megapixels, respectivamente.

Pode-se verificar que os percentuais de tempos das tarefas do filtro de partículas são inferiores aos das tarefas de para a aquisição da imagem e do armazenamento com percentual médio de 37,6% e 30,7% respectivamente para imagens de 1 megapixels e de 52,4% e 29,6% para imagens de 5 megapixel.

O processo que realiza o ajuste na dimensão da imagem também demanda um tempo considerável de aproximadamente 20,9% do tempo total.

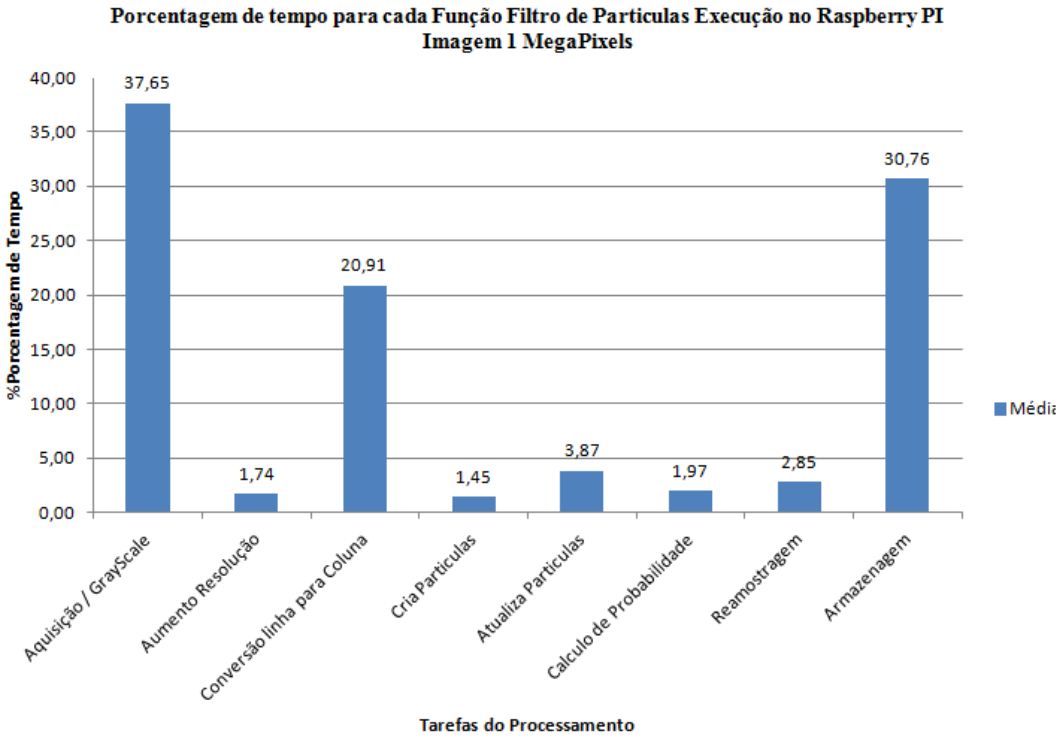


Figura 70 Gráfico do Percentual de tempo do algoritmo B na execução no Raspberry com imagens de 1 Megapixels

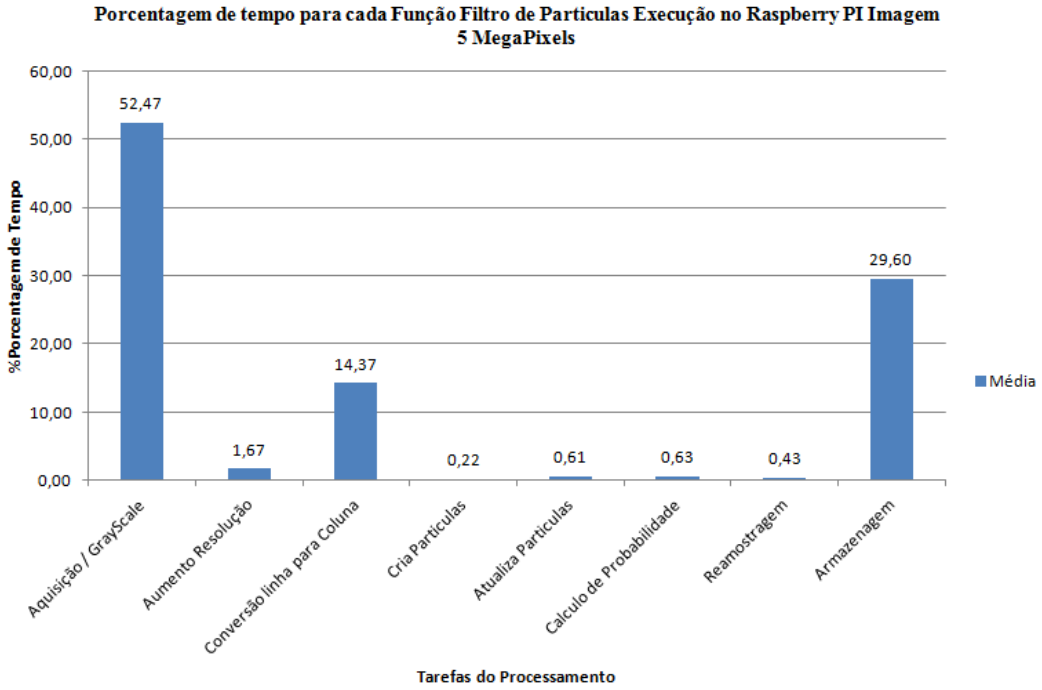


Figura 71 Gráfico do Percentual de tempo do algoritmo B na execução no Raspberry com imagens de 5 Megapixels

Nas Tabela 12 e Tabela 13 estão os tempos de cada tarefa do programa de detecção do algoritmo B para executado no Raspberry PI, observando que nos tempos médios das imagens de 5 megapixels ficaram em 6,001 segundos, apresentando picos de tempo de 8,334 segundos e para imagens de 1 megapixel com o tempo de 0,897 segundos. Tempos consideráveis na aquisição da imagem e no armazenamento, verificando desta forma a eficiência do algoritmo que utiliza a classificação por abordagem estatística (Filtro de Partículas) com relação ao algoritmo de abordagem sintática (Eliminação por Descrição), para a detecção da fissura para um sistema a ser embarcado.

Tabela 12 – Tempos das Tarefas no Raspberry PI do Algoritmo B para imagens de 1 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição / GrayScale	0,336449	0,381171	0,315434	0,014418
Ajuste na Dimensão Imagem	0,015509	0,017035	0,015185	0,000251
Ajuste Sequencia de dados Imagem	0,186673	0,190768	0,185829	0,000923
Cria Particulas	0,012985	0,014348	0,012796	0,000217
Atualiza Particulas	0,034589	0,039474	0,034219	0,000782
Calculo de Probabilidade	0,017549	0,026620	0,016995	0,001658
Reamostragem	0,025483	0,036963	0,023523	0,001891
Armazenagem	0,279443	0,899038	0,236361	0,116017
Tempo Total	0,897121	1,378949	0,855385	0,074693

Tabela 13 – Tempos das Tarefas no Raspberry PI do Algoritmo B para imagens de 5 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição / GrayScale	3,207646	5,736335	2,114009	1,040034
Ajuste na Dimensão Imagem	0,097433	0,104690	0,096686	0,001268
Ajuste Sequencia de dados Imagem	0,838667	0,860806	0,833306	0,005565
Cria Particulas	0,013052	0,013204	0,012907	0,000079
Atualiza Particulas	0,035826	0,036382	0,035445	0,000228
Calculo de Probabilidade	0,036700	0,049958	0,036141	0,001738
Reamostragem	0,024815	0,026646	0,023857	0,000626
Armazenagem	1,747087	4,229238	1,473571	0,516448
Tempo Total	6,001225	8,333832	4,714400	1,027750

Nas Figura 72 e Figura 73 são apresentados o percentual de tempo de cada tarefa do algoritmo B executado na plataforma Raspberry PI com imagens de 1 megapixels e 5 megapixels, respectivamente. Observando que a tarefa de aquisição e a tarefa de armazenamento também são consideradas como tempos críticos na execução do programa, com destaque para a tarefa de conversão de linha para coluna, que na plataforma Raspberry PI demandou um tempo significativo quando comparada as demais tarefas, ao contrário do percentual apresentado no computador desktop.

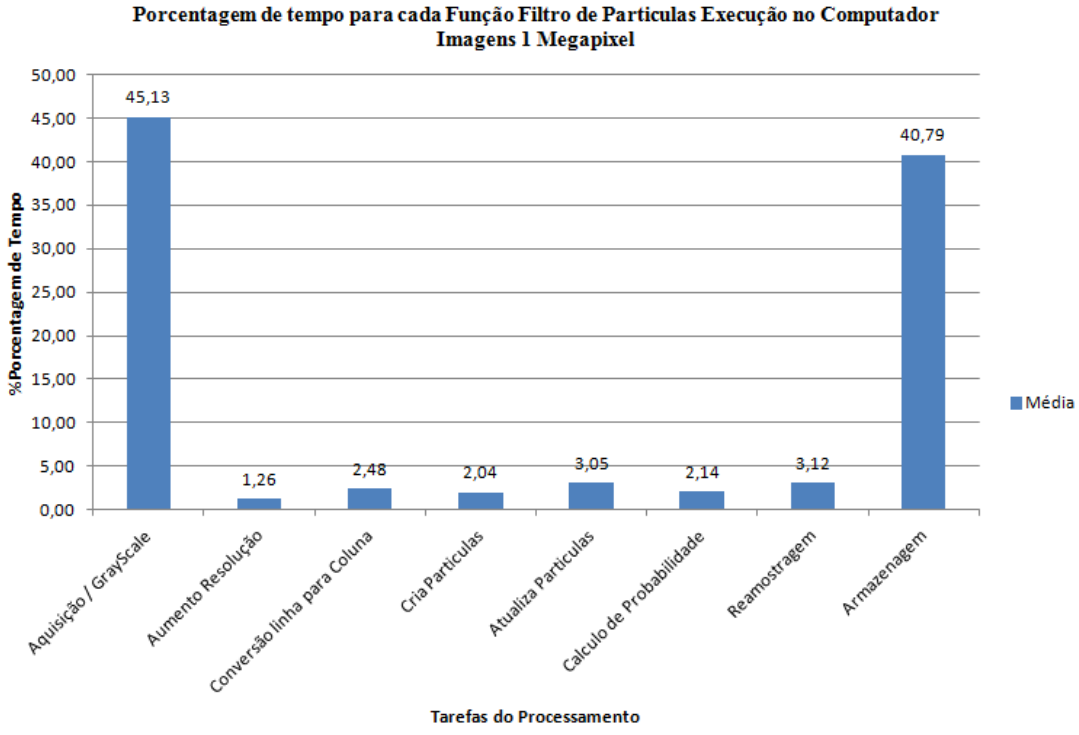


Figura 72 Gráfico do Percentual de tempo do algoritmo B com execução no Computador com imagens de 5 Megapixels

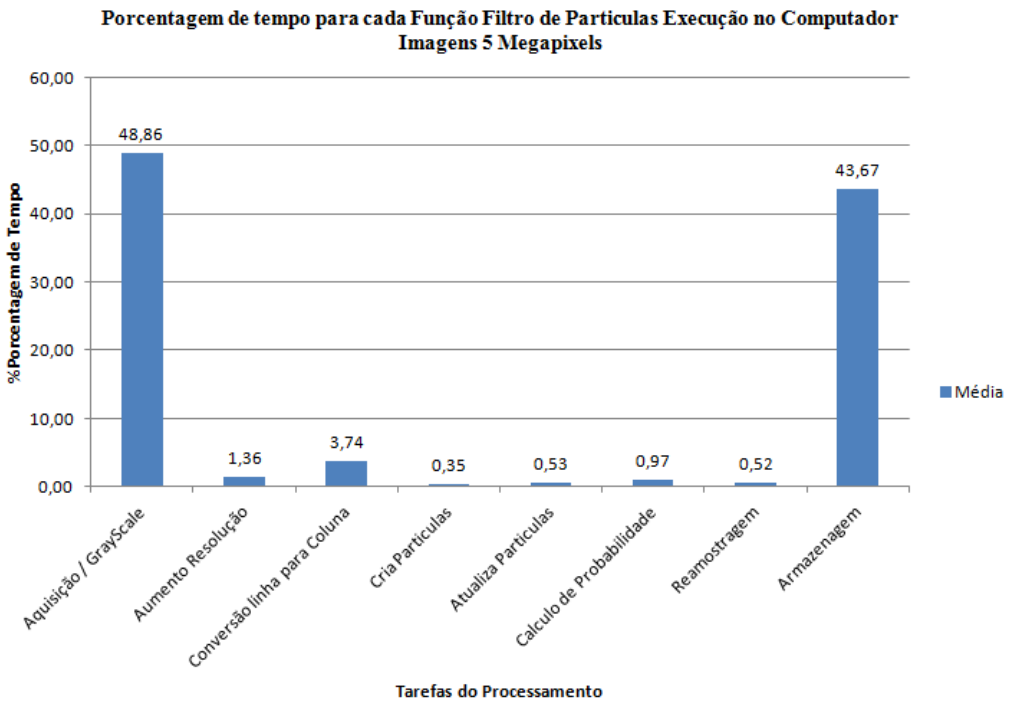


Figura 73 Gráfico do Percentual de tempo do algoritmo B com execução no Computador com imagens de 5 Megapixels

Nas Tabela 14 e Tabela 15 estão os tempos de cada tarefa do programa de detecção do algoritmo B, executados no computador, observando que nos tempos médios das imagens de 5 megapixels ficaram em cerca de 0,803 segundos, apresentando picos de tempo de 1,119 segundos e para imagens de 1 megapixel com o tempo de 0,137 segundos com picos de 0,160 segundos. Tempos consideráveis também neste caso na aquisição da imagem e no armazenamento.

Tabela 14 – Tempos das Tarefas no Computador do Algoritmo B para imagens de 1 Megapixel

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição / GrayScale	0,061828	0,076486	0,058314	0,004856
Ajuste na Dimensão Imagem	0,001718	0,001797	0,001668	0,000022
Ajuste Sequencia de dados Imagem	0,003384	0,003554	0,003321	0,000036
Cria Particulas	0,002783	0,002997	0,002702	0,000040
Atualiza Particulas	0,004168	0,004837	0,004056	0,000106
Calculo de Probabilidade	0,002918	0,003139	0,002790	0,000069
Reamostragem	0,004257	0,004737	0,004014	0,000188
Armazenagem	0,055787	0,065058	0,052340	0,002324
Tempo Total	0,136843	0,160472	0,130403	0,006654

Tabela 15 – Tempos das Tarefas no Computador do Algoritmo B para imagens de 5 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição / GrayScale	0,393039	0,619874	0,368519	0,045215
Ajuste na Dimensão Imagem	0,010860	0,018253	0,009642	0,001210
Ajuste Sequencia de dados Imagem	0,029979	0,039749	0,028879	0,001388
Cria Particulas	0,002807	0,003047	0,002731	0,000049
Atualiza Particulas	0,004277	0,012742	0,004001	0,001096
Calculo de Probabilidade	0,007722	0,015730	0,007013	0,001080
Reamostragem	0,004185	0,004984	0,003999	0,000168
Armazenagem	0,349768	0,423068	0,322294	0,015938
Tempo Total	0,802635	1,119039	0,749933	0,055505

Por último na análise realizada utilizando MicroBlaze com a FPGA Xilinx, observa-se através da Figura 74 que o tempo da tarefa de classificação com a proporção de 77% do tempo total das tarefas para o algoritmo A, levou um tempo consideravelmente maior igualmente constatado nas demais plataformas. Já no algoritmo B a tarefa que demandou maior tempo foi o de obtenção dos dados da imagem para o processamento, chegando a 92% do tempo total das tarefas.

Considerando a obtenção de imagens cujo tamanho é de 786.432 bytes, mesmo a imagem estando alocada na memória RAM do dispositivo, devido à passagem dos dados de forma serializada e também pela frequência do clock de 100 Mhz ocasionou um tempo consideravelmente elevado nesta tarefa.

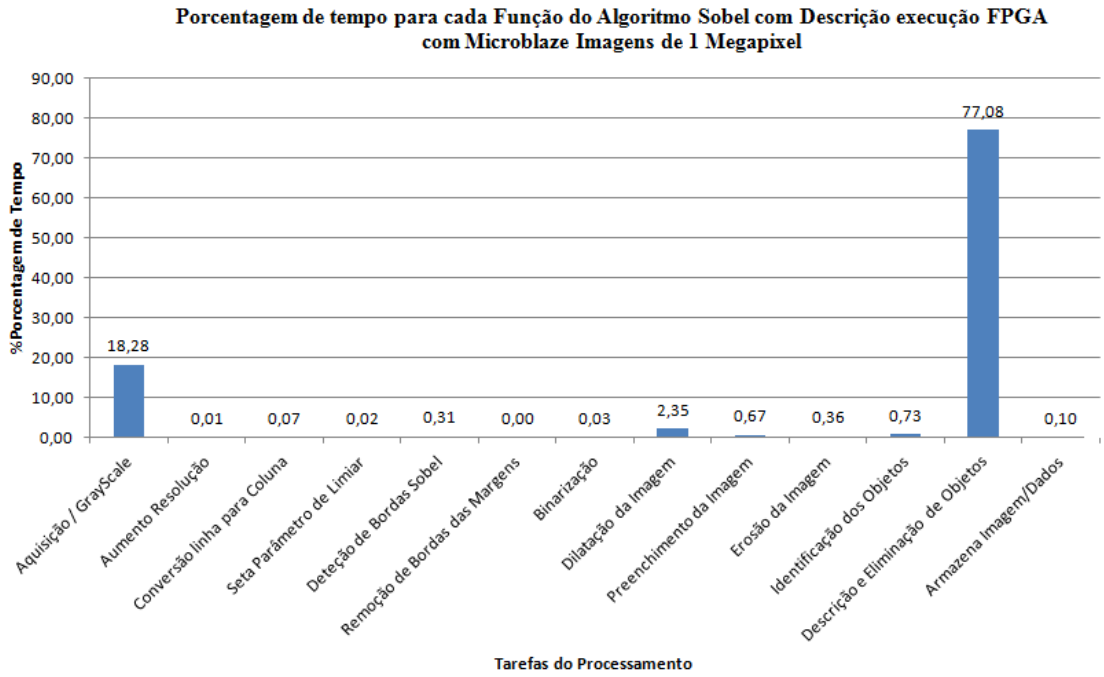


Figura 74 Gráfico do Percentual de tempo do algoritmo A com execução na FPGA com Microblaze de imagens com 1 Megapixels

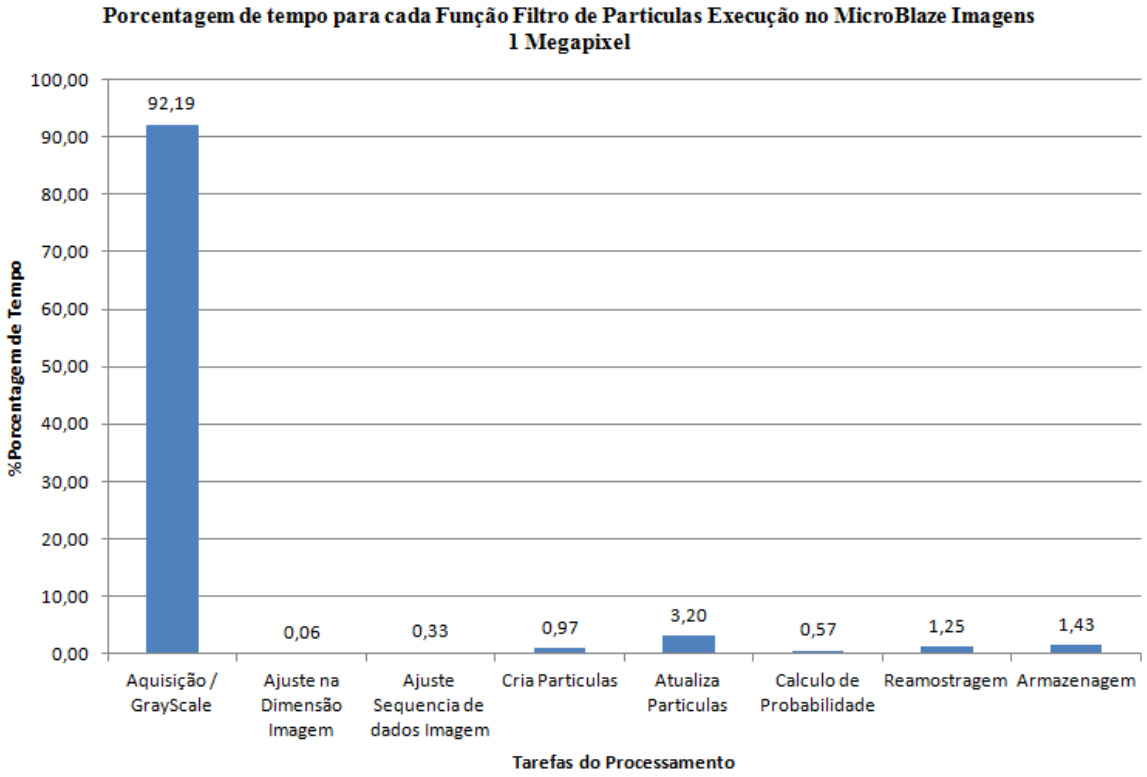


Figura 75 Gráfico do Percentual de tempo do algoritmo B com execução na FPGA com Microblaze de imagens com 1 Megapixels

Na Tabela 17 estão os tempos de cada tarefa do programa de detecção do algoritmo B, executados em FPGA utilizando um microprocessador soft MicroBlaze. Para este teste foram utilizadas as imagens de 1 megapixel devido ao tempo consideravelmente maior para imagens de 1 megapixel. Os tempos neste tamanho para cada imagem ficaram em média de 3 minutos e 54 segundos com picos de 3 minutos e 58 segundos, com pouca variação nos tempos obtidos das amostras analisadas. Percebe-se tempos consideráveis também no caso da aquisição da imagem, que demandou cerca de 3 minutos e 35 segundos, sendo que o processamento real da imagem foi realizado com um tempo médio de 19 segundos.

Tabela 16 – Tempos das Tarefas no MicroBlaze executando o Algoritmo A para imagens de 1 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição/ GrayScale	197,775828	199,171875	196,339844	1,1919181
Ajuste na Dimensão Imagem	0,137400	0,137695	0,136719	0,000370881
Ajuste Sequencia de dados Imagem	0,777842	0,779541	0,774902	0,002134873
Seta Parâmetro de Limiar	0,219676	0,220459	0,218750	0,00056534
Deteção de Bordas Sobel	3,322998	3,326172	3,320496	0,00215299

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Remoção de Bordas das Margens	0,002024	0,002197	0,001953	0,000111926
Binarização	0,282094	0,283203	0,281738	0,000570582
Dilatação da Imagem	25,516551	25,542297	25,494385	0,015834216
Preenchimento da Imagem	7,241465	7,259460	7,233643	0,009155772
Erosão da Imagem	3,853618	3,860352	3,848267	0,005171696
Identificação dos Objetos	29,746480	81,628906	2,473633	33,33733351
Descrição e Eliminação de Objetos	3594,588796	9494,705078	115,187012	4171,983184
Armazena Imagem/Dados	1,113709	1,302124	1,044922	0,100917433
Tempo Total	3864,578410	9815,699219	359,241699	4204,375351

Tabela 17 – Tempos das Tarefas no MicroBlaze executando o Algoritmo B para imagens de 1 Megapixels

Tarefas do Programa Principal	Média	Máximo	Mínimo	Desvio padrão
Aquisição / GrayScale	215,689257	220,304690	211,150873	1,167117
Ajuste na Dimensão Imagem	0,135696	0,136656	0,135033	0,000427
Ajuste Sequencia de dados Imagem	0,773396	0,775148	0,771386	0,000805
Cria Particulas	2,271609	2,306780	2,255084	0,010174
Atualiza Particulas	7,486277	7,559689	7,430923	0,028671
Calculo de Probabilidade	1,325033	1,363119	1,308415	0,011238
Reamostragem	2,930880	2,998906	2,882772	0,021176
Armazenagem	3,354410	6,560957	2,790016	0,676679
Tempo Total	233,966558	238,292744	228,997592	1,329718

Foi constatado que no processador soft, cuja arquitetura é de 32 bits, no tratamento de instruções com dados do tipo Double, mesmo com a opção de otimização para o tratamento destes dados que são de 64 bits, são necessários mais instruções do que o observado na arquitetura ARM. Na arquitetura ARM observa-se que estas instruções já são otimizadas para 64 bits que realiza melhor as operações com ponto flutuante, ao contrário da arquitetura em hardware onde há a necessidade de operações com a varíavel em ponto fixo.

A Tabela 18 resume os tempos médios obtidos e seus devios padrões, demonstrando uma grande variação temporal para o algoritmo com operador Sobel.

Tabela 18 – Resumo dos tempos obtidos na plataformas analisadas para Imagens de 1 Megapixels, dados em segundos

Tempo em Segundos para imagens de 1Mpixels			
Código	**Plataforma**	**Média**	**Desvio Padrão**
Operador Sobel (Algoritmo A)	**PC (Linux)**	3,987012	3,203406
	Raspberry	80,597374	65,641125

Tempo em Segundos para imagens de 1Mpixels			
Código	**Plataforma**	**Média**	**Desvio Padrão**
	Microblaze	5058,043518	4788,524014
Filtro de Partículas (Algoritmo B	**PC (Linux)**	0,136843	0,006654
	Raspberry	0,897121	0,074693
	Microblaze	233,966558	1,329718

Para o algoritmo A, percebe-se que a tarefa que realiza o processo de descrição do objeto é o que demanda mais tempo com relação as demais tarefas. A escolha de um processamento desta função em separado e executando diretamente em hardware torna-se uma alternativa interessante a fim de conseguir um melhor desempenho da aplicação, tornando possível realizá-la no processador embarcado no VANT. Todavia, para implementar esta função em FPGA, a programação da função teve que ser totalmente refeita, considerando fatores como permanência de variáveis necessárias para cada chamada da função além das considerações de ponto fixo nas operações, principalmente naquelas envolvendo divisão. Porém a ferramenta MATLAB gera a versão em VHDL mas a sua sintetização em uma FPGA fica inviável. Foi verificado que a ferramenta cria uma quantidade de variável de dados muito além da capacidade de memória embutida no processador. Isto se deve principalmente ao tamanho da imagem analisada, sendo necessário alocar estes dados em memória externa.

5.2 EXPLORAÇÃO DO ESPAÇO DE PROJETO

Exploração do espaço de projeto refere-se à atividade de exploração de alternativas de projeto, a fim de identificar a opção que melhor preenche os requisitos da aplicação. Utilizando-se os tempos de execução medidos e apresentados nas seções anteriores, pode-se analisar diferentes situações para a implementação do sistema de processamento de imagens para reconhecimento de fissuras em fachadas de argamassa:

i) Codificação dos algoritmos de processamento de imagem em MATLAB e execução em um ambiente com suporte a ferramenta MATLAB para um computador desktop (que será localizado em uma estação em solo, o que significa, que não estará instalado no VANT devido ao seu elevado consumo de energia e de peso);

ii) Execução do programa compilado em C / C ++ aplicado em um ambiente de tempo de execução com suporte Linux, que pode ser embarcado no VANT ou então em uma estação em solo).

iii) Emprego do código compilado em C / C++ em um processador soft utilizando uma placa FPGA.

Para uma inpeção, o inspetor deseja uma forma de saber em quais locais estão presentes fissuras em uma fachada, com o auxílio de um mecanismo que sinalize a presença de fissuras além de uma ferramenta de forma a mensurar a fissura, sendo uma informação importante num levantamento para um relatório para medidas de reparos futuros.

Estas implementações podem ser utilizadas nos seguintes cenários de aplicação em uma inspeção predial.

_ Cenário A: O VANT captura as imagens e armazena-as para posterior processamento em computador de maior desempenho e instalado em uma estação em terra. Este cenário pode ser ainda subdividido em dois cenários: no primeiro cenário, denominado de A1, o VANT leva apenas uma câmera digital com recursos de memória suficiente para o armazenamento de todas as imagens, que serão transferidas para um computador em solo após o retorno do VANT à sua base. Outro subcenário, denominado de A2, no qual o VANT captura as imagens e após digitalizá-las e armazená-las em memória, inicia a transmissão das mesmas via um link de comunicação sem fio para o computador em solo. Neste cenário o processamento das imagens inicia antes mesmo do VANT retornar ‘a estação base, aumentando assim o desempenho geral da aplicação. Desta maneira pode-se considerar a captura de imagem tanto com dimensões de 1 megapixels quanto com imagens de 5 megapixels, no entanto deve-se considerar o tempo de transmissão de dados da imagem.

Figura 76 - Cenário A1: O VANT captura as imagens e armazena-as para posterior processamento em computador de maior desempenho e instalado em uma estação em terra.

Figura 77 - Cenário A2: O VANT captura as imagens e após digitalizá-las e armazená-las em memória, inicia a transmissão via um link de comunicação sem fio para o computador em solo.

_ Cenário B: O VANT captura as imagens e realiza o processamento em uma plataforma embarcada, ainda durante o voo. A imagem capturada é armazenada em um buffer que é então processada sequencialmente armazenando então os dados obtidos após o processamento, possibilitando após a varredura a disponibilidade dos dados ainda no local do levantamento. Com os tempos obtidos anteriormente na plataforma Raspberry PI a opção de captura e processamento para imagens de 1 megapixel neste caso é a melhor opção quando comparado ao tempo de processamento das imagens de 5 megapixels, dado que o elevado tempo de processamento destas últimas restringe o número de imagens processadas no tempo de autonomia de voo do VANT.

Figura 78 - Cenário B: O VANT captura as imagens e realiza o processamento em uma plataforma embarcada, ainda durante o voo.

_ No último cenário C, o VANT equipado com um FPGA a bordo pode também apresentar dois subcenários: uma na qual a maior parte do tempo as partes de algoritmos consumindo são codificados em VHDL e executado na FPGA (Cenário C1) e outro quando o programa gerado C/C ++ é portado para um FPGA com um processador soft Microblaze apoiado em um Sistema Operacional Linux (Cenário C2).

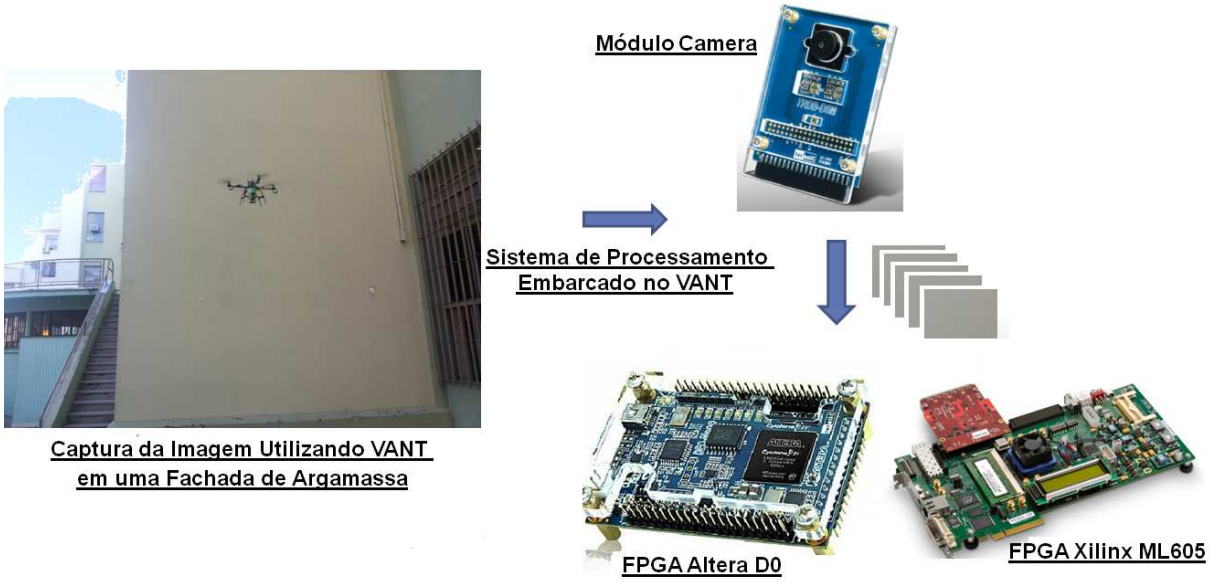


Figura 79 - Cenário C: VANT equipado com um FPGA a bordo realiza o processamento em uma plataforma embarcada, ainda durante o voo.

Na captura da imagem de 1 megapixel de uma câmera modular da plataforma Raspberry PI a uma distância entre 1,00 metro e 1,50 metro abrange uma área de aproximadamente 1,05 metro por 0,79 metro e 1,57 metro por 1,18 metro, respectivamente.

Considerando um levantamento de uma edificação com uma das faces com dimensões de 8 metros de largura e 25 metros de altura, possuindo então uma área de 200 metros quadrados e prevendo uma captura da imagem a uma distância mínima de 1 metro da fachada a ser analisada, são necessárias 8 imagens a serem capturadas na horizontal e 33 imagens na vertical, perfazendo um total de 264 imagens a serem processadas. A Figura 80 ilustra a aplicação.

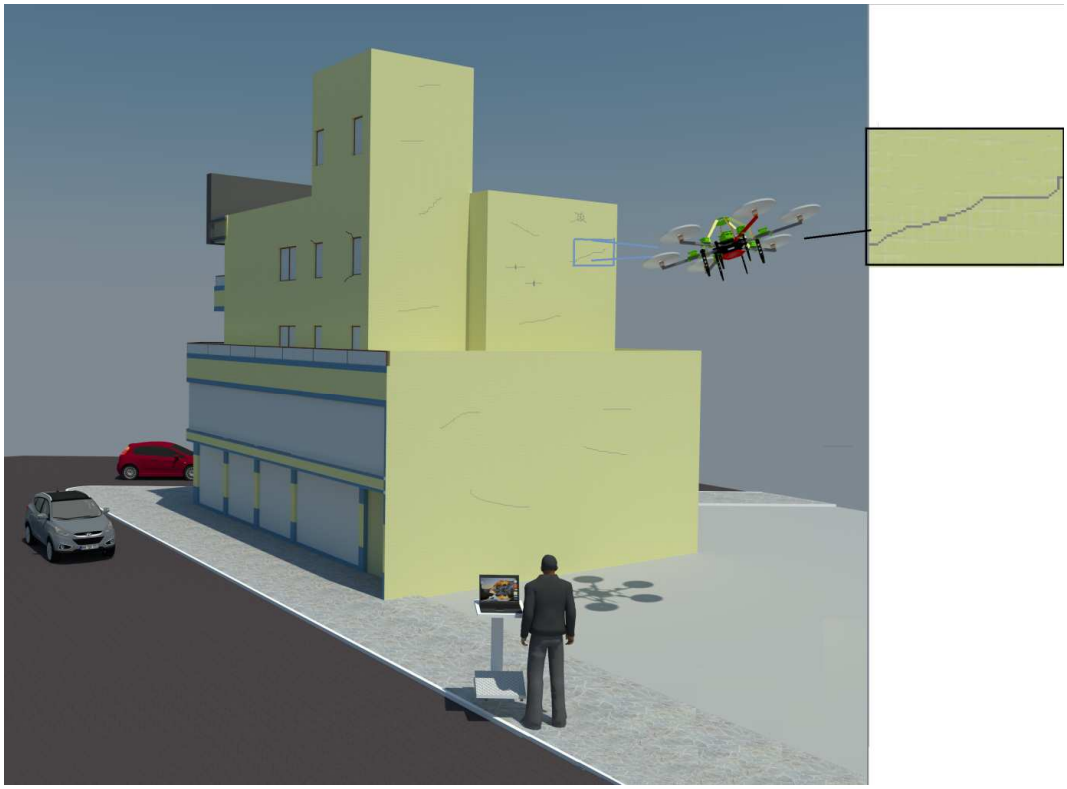


Figura 80 Ilustração de uma Inspeção para Detecção de fissuras em uma fachada de argamassa utilizando um VANT.

A Tabela 19 apresenta um resumo dos tempos considerados conforme a seguir:

t1 – O tempo de viagem do VANT entre a decolagem até a localização da fachada, devendo ser considerada também o tempo de retorno;

t2 – O tempo de aquisição de imagem, assumindo que o VANT pecorrerá a uma velocidade máxima de 0,5 m / s que permitirá uma captura de imagem a cada 2 segundos cobrindo então a uma distância de 1,05m na horizontal da fachada, de modo que cada imagem corresponderá a uma área de 1,05 x 0,79 m² em uma distância de aquisição de 1,00m do prédio.

t3 - Tempo para transferir as imagens do VANT a uma estação no solo via rede sem fios (assumindo um link com 110 kbps de dados eficazes e uma distância de até 400 m);

t4 – Tempo de processamento de todas as imagens a ser realizado no levantamento, que é dependente do algoritmo utilizado e da plataforma computacional onde o algoritmo executa.

Tabela 19 – Cenários para a Tarefa de Detecção de Fissuras do Algoritmo A

Situações	VANT Voo até a Fachada	Captura de 264 Imagem	Transf. Imagem via Rede sem Fio	Tempo de Processamento da Imagem	Tempo Total Estimado	Consumo de Energia do Sistema
	t1	t2	t3	t4	t1 / t2 / t3 / t4	
Cenario A1	00:05:00	00:08:48	-	00:17:32	00:46:20	180,22 Watts
Cenario A2			02:20:24	00:17:32	02:20:28	202,64 Watts
Cenario B			-	05:54:05	05:54:06	10,50 Watts
Cenario C			-	15 dias 10 hr 55 min	15 dias 10 hr 55 min	-

No Cenário A1 pode-se prever um tempo aproximado de varredura com captura de imagem a cada 2 segundos, com uma velocidade média do VANT a 0,53m/s é possível obter as 264 imagens para o processamento posterior. O tempo de varredura será um pouco superior a 8 minutos para este caso, considerando ainda um tempo de posicionamento e retorno de aproximadamente 10 minutos. O tempo de voo neste caso não deverá ser maior que 19 minutos. Considerando o tempo de processamento no computador para imagens de 5 megapixels o tempo total do processo deverá ser de um pouco mais de 2 horas, devendo neste caso ainda considerar o tempo que levará para transferir todas as imagens para um computador. O recurso essencial está na quantidade de memória disponível para armazenamento de todas as imagens. Para uma imagem de 5 megapixels com compressão em JPEG, os dados podem variar em torno de 2,6 MB(Megabyte), dependendo dos parâmetros da captura da imagem, gerando um volume de dados de aproximadamente 692MB considerando 200m². Para imagens menores como 1Mpx o tamanho aproximado de imagens serão de cerca de 108Megabytes. O consumo de energia considerado é do dispositivo de captura de imagem assim como o armazenamento. Este dispositivo consome em média 2,5Watts/hora e de um computador portátil, no exemplo, para um processamento da imagem, com consumo aproximado de 90Watts/hora, para o cenário A2, os tempos de aproximadamente 8 minutos de captura mais de quase 2 horas de processamento no computador, o consumo total será de cerca de 180 Watts no sistema.

No cenário A2, deve-se considerar o tempo de envio de pacotes de imagem através de rede sem fio. Nos dispositivos disponíveis para transmissão de dados digitais há uma taxa de envio na ordem de 250kbps (kilo bits por segundo). Considerando um dispositivo como o ZigBee que é um comunicador wireless robusto gerido por ZigBee Alliance baseado no padrão IEEE 802.15.4 da camada física e Mac. Uma taxa de transferência de pacote de dados efetiva que pode ser considerado de 110kbps, o tempo de envio de um dado, porém, de dimensão maior que o pacote suportado por um processo de transmissão pode variar bastante, dependendo de

fatores como interferências que podem levar a reenvio de pacotes perdidos. Em [BURCHFIELD, 2006] existe uma abordagem de envio de dados, empregando métodos de envio, baseado neste artigo a consideração do envio de dados utilizando o ZigBee.

Assim para uma imagem de 5 megapixels o tempo de transferência é dada na razão da equação 31.

$$T = \frac{Tamanho\ de\ dados\ da\ imagem}{taxa\ efetiva\ de\ transferencia\ de\ dados} \tag{31}$$

Considerando um tamanho de aproximadamente 2,6 MBytes, sendo que 1 Mega Bytes é igual a 1.048.576 bytes e 1 byte é igual a 8 bits. O tamanho médio de uma imagem de 5 megapixels é de aproximadamente 21.810.381 bits. Assim temos o tempo dado pela equação 32.

$$T = \frac{21.810.381}{110.000} = 198{,}276 \text{ segundos} = 3{,}304 \text{ minutos} \tag{32}$$

O tempo estimado de envio de uma imagem de 5 megapixels gira em torno de 3 minutos e 18 segundos.

No caso de uma imagem de 1 megapixel, o tamanho médio é de 428KB(Kilobytes), convertendo para bits temos 3.506.176. Deste modo o tempo de envio de uma imagem de 1 megapixel data pela equação 33.

$$T = \frac{3.506.176}{110.000} = 31{,}874 \text{ segundos} \tag{33}$$

Para uma imagem de 5 megapixel percebe-se que o tempo de envio é consideravelmente superior a de 1 megapixels, para o cenário proposto pode-se considerar o envio de imagem de 1 megapixel, já que o tempo de processamento na ordem 4 segundos, o tempo critico será o de envio da imagem.

Considerando um tempo de espera para o envio da imagem, com processamento de cada imagem o tempo de varredura levaria cerca de 2 horas e 21 minutos. Para um VANT, onde sua autonomia gira em torno de 40 minutos de voo, seria necessárias 3 missões aéreas para a captura das imagens, utilizando somente o envio das imagens.

Percebe-se que para este cenário, o consumo de energia será maior, devido ao tempo de voo aliado ao consumo de energia na transmissão de dados, do dispositivo de captura de imagem e do computador para o processamento. Para um dispositivo como um XBee-Pro™,

que possui consumo de transmissão por volta de 0,9 Watts/hora, receptor e transmissor, o consumo total neste cenário ficaria por volta de 202,6 Watts.

No cenário B, utilizando o sistema embarcado no Raspberry PI, o tempo é bem superior, tornando-se inviável a utilização para o processamento neste algoritmo, a melhor opção que atenderá a utilização do sistema embarcado será no cenário A2, uma vez que durante o voo pode-se obter todas as imagens, sendo então enviado para o processamento em uma estação em solo. Neste caso pode-se considerar que o tempo de obtenção dos resultados pode ser mais rápido do que no caso do cenário A1, uma vez que as imagens capturadas ainda deverão ser extraídas para um computador e serem processada. O consumo aproximado neste caso será o aproximado do Raspberry PI em torno de 1,48 Watts/hora em máximo processamento e de 1,27 Watts/hora, possuindo um consumo total de aproximadamente 10,5 Watts.

Na Tabela 20 estão as informações dos tempos e consumos no caso da execução do algoritmo B.

Tabela 20 – Cenários para a Tarefa de Detecção de Fissuras do Algoritmo B

Situações	VANT Voo até a Fachada	Captura da Imagem	Transf. Imagem via Rede sem Fio	Tempo de Processamento da Imagem	Tempo Total Estimado	Consumo de Energia do Sistema
	t1	t2	t3	t4	t1 / t2 / t3 / t4	
Scenario A1	00:05:00	00:08:48	-	00:00:36	00:29:24	5,33 Watts
Scenario A2			02:20:24	00:00:36	02:25:26	6,36 Watts
Scenario B			-	00:03:56	00:18:48	0,30 Watts
Scenario C			-	17:07:57	17:12:59	

No algoritmo B, os quatro cenários apresentam tempos que estão compatíveis com a autonomia de voo do VANT, no cenário A1 a diferenças para o algoritmo A no tempo de processamento que ficará em média de aproximadamente 3 minutos e 31 segundos para a imagens de 5 megapixels e 36 segundos para imagens de 1 megapixels. Considerando uma captura de imagem a cada 2 segundos em uma velocidade de 0,5 m/s chega-se a um tempo de varredura de 8 minutos e 48 segundos, verificando a necessidade de capacidade de armazenamento de 692 megabytes para imagens de 5 megapixels e 108 megabytes para imagens de 1 megapixels. O consumo neste caso ficou em torno de 5,33 Watts, somando-se o tempo total de captura das imagens com o tempo de processamento no computador.

No cenário A2, entretanto, com o tempo de processamento de imagem reduzido, o envio de mensagem para processamento no computador não é mais uma boa opção, devido ao tempo considerado de 31,2 segundos para o envio de imagem via rede sem fio e também pelo consumo maior de energia neste caso.

No cenário B considerando o tempo de processamento embarcado no Raspberry PI pode-se realizar uma varredura de aproximadamente 3 minutos e 51 segundos considerando somente o tempo de processamento com uma razão de velocidade de 1,31m/s para imagens de 1 megapixels. A memória disponível deverá ser de aproximadamente 108 megabytes, neste caso existe a viabilidade de uso do algoritmo embarcado no VANT, com a vantagem do consumo baixo de aproximadamente 0,30 Watts.

No cenário C, pode ser observado os tempos consideravelmente superiores, não sendo uma boa opção de processamento para o emprego no VANT, para o algoritmo A, o tempo de processamento pode levar dias e para o algoritmo B pode levar horas.

6 CONCLUSÕES E TRABALHOS FUTUROS

Através do estudo de manifestações patológicas, especificamente a presença de fissuras em fachadas e de processamento de imagem visando a detecção de fissuras e rachaduras, foi pesquisado trabalhos relacionados ao tema. Nesta dissertação foram utilizados alguns métodos destes trabalhos a fim de chegar a uma boa margem de precisão nestas detecções, a fim de verificar requisitos temporais para o embarque no VANT, utilizando para isto dois algoritmos, um envolvendo classificação estatística e outro a classificação sintática. Estes algoritmos desenvolvidos através do ambiente de simulação utilizaram preliminarmente funções disponíveis na ferramenta MATLAB. Alterações importantes nos métodos utilizados para o processamento de imagem levaram em considerações especialmente o seu tempo de execução, uma vez que impactuariam durante o seu emprego em uma plataforma embarcável. Atráves do estudo de trabalhos relacionados a processamento de imagens, verificou-se diferentes técnicas que possibilitassem a detecção de fissuras de forma automática, porém na medida do desenvolvimento e das observações destas técnicas alguns métodos demonstraram-se bastante custosos no requisito temporal e com baixo desempenho na detecção das fissuras. Um dos métodos analisados, por exemplo, foi o filtro gaussiano, que dependendo do parâmetro sigma, pode aumentar consideravelmente o tempo de execução não levando a detecções satisfatórias observadas visualmente. No desenvolvimento observou-se que o Operador Sobel enfatizou melhor a borda principalmente para imagens diagonais.

Para a geração do código observou-se a necessidade de realizar operações que descrevessem o mesmo comportamento das funções disponíveis na ferramenta MATLAB. Isto se deve à não geração automática de algumas funções na versão MATLAB utilizada.

Dos algoritmos utilizados, são bem visíveis as diferenças dos tempos de execução dos algoritmos nas plataformas utilizadas. O algoritmo que utiliza a abordagem estatística, o filtro de partículas, possuiu tempos de execução que permitem o seu emprego diretamente em um VANT, porém não oferece descrições de modo a permitir uma classificação do objeto detectado como no algoritmo que utiliza Operador Sobel.

Através dos cenários conclui-se que a melhor escolha de utilização do algoritmo A é a de efetuar o seu emprego no processamento no computador, com transmissão de dados, a custo de um consumo de energia maior do sistema, sendo também necessárias missões de voo a mais para cobrir uma área de aproximadamente 200 metros quadrados, por exemplo, devido a autonomia do VANT. A opção de utilização no processamento embarcado pode ser também considerada, no caso do cenário B após a captura de todas as imagens, com o início do

processamento, mas o tempo elevado de processamento, no entanto, reduzirá o uso do dispositivo no caso de necessidade de outro levantamento em um curto espaço de tempo. Porém a utilização do programa em uma plataforma como o Microblaze é possível devido a utilização da memória externa disponível, mas verificou-se que o tempo de processamento se tornou consideravelmente alto. Conforme apresentado, os tempos foram bem superiores: uma análise nas instruções do programa compilado neste ambiente demonstrou que várias instruções são necessárias para o tratamento dos dados em 64 bits, observando no programa tanto compilado para Raspberry PI que utiliza intruções RISC como para o PC estas instruções são reduzidas. Na implementação junto a um processador soft, verificou-se um aumento considerável no tempo de processamento da imagem, principalmente no que tange o carregamento da imagem na memória interna do processador, concluindo que para a aplicação de processamento de imagem que envolve alto número de transferência de dados, não é uma boa alternativa na implementação da aplicação proposta.

Para o algoritmo B, a utilização em todos os cenários foi satisfatória, no cenário B é atrativa sua utilização devido ao menor número de equipamentos necessários no transporte para um levantamento, com um fornecimento de informações em tempo hábil para um operador do sistema.

Por fim, a combinação do uso de ambos os algoritmos apresenta-se como uma alternativa interessante. O algoritmo B poderia ser executado durante o voo no processador embarcado no VANT e caso o mesmo detectasse uma provável existência de fissura na imagem adquirida, esta seria armazenada para posterior processamento no computador desktop em solo rodando o algoritmo A. Assim utiliza-se o filtro de partículas para a detecção preliminar da fissura em um sistema embarcado como o Raspberry PI, reduzindo de tal modo o número de mensagens a serem transmitidas para um processamento do algoritmo que irá descrever a fissura em um computador não embarcado com melhor desempenho, viabilizando desta maneira um sistema que detecte a fissura em um ponto e obtenha os dados que serão utilizados na classificação futura deste tipo de manifestação patológica.

Um aspecto importante a se ponderar é que os tempos considerados médios podem variar, sendo dependentes de aspectos como o número de objetos detectados para analise e o número de tarefas computacionais sendo executadas simultaneamente.

Este trabalho enfatizou os aspectos do processamento de imagem para detecção de fissuras em fachadas de argamassa e obtenção de dados de dimensões destas fissuras, outras implementações ao sistema são necessárias para a aplicação em um VANT.

Para os trabalhos futuros, é essencial o empregado no VANT de sensores de distância, primeiro servindo com auxílio na navegação, evitando colisão com obstáculos, de modo a manter uma distância apropriada da fachada e em segundo para aquisição correta da imagem, sendo importante manter uma margem de distância para se ter imagens uniformes e de forma paralela a fachada. O valor da distância na hora da captura da imagem será essencial para obter a real dimensão no caso da detecção da fissura.

Deve ser considerada uma interface de comunicação entre o sistema de processamento de imagem e os dados de posicionamento disponíveis no VANT, servindo de referência na localização de determinada imagem capturada na fachada. Informações como orientação, altura e distância percorrida na horizontal ou na vertical serão importantes na descrição da localização de um objeto encontrado.

Com um sistema que torne possível o controle do VANT próximo ao objeto de captura, uma próxima evolução do algoritmo através dos dados extraídos, poderá ser a de classificação da manifestação patológica com auxílio de um profissional que possua experiência na área, parametrizando de forma assistida o algoritmo. Sistemas especialistas que auxiliam na classificação através de características conhecidas, pode ser utilizadas na evolução do algoritmo.

REFERÊNCIAS

ABDEL-QAER, I.; ABUDAYYEH, O.; KELLY, M. Analysis of Edge-Detection Techniques for Crack Identification in Bridges. **Journal of Computing in Civil Enginnering**, [S.l.] , v.17, n.4, p. 255–263, Spet. 2003. Disponível em: <http://dx.doi.org/10.1061/(ASCE)0887-3801(2003)17:4(255)> Acesso em: 11 dez. 2014.

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

ANDRADE, J. J. O.; DAL MOLIN, D. C. C. Considerações quanto aos trabalhos de levantamento de manifestações patológicas e formas de recuperação em estruturas de concreto armado. In: CONGRESSO IBEROAMERICANO DE PATOLOGIA DAS CONSTRUÇÕES (CONPAT), 4., 1997, Porto Alegre. **Anais**... Porto Alegre: UFRGS, 1997. v. 1, p. 321-327.

ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. **NBR 15575**: Edificações habitacionais - desempenho. Parte 1: requisitos gerais. Rio de Janeiro, 2013. Disponível em: <http://www.abntcatalogo.com.br/norma.aspx?ID=195611> Acesso em: 18 maio 2014.

BLANCHET, G.; CHARBIT M. **Digital Signal and Image Processing using Matlab**. Newport Beach: Iste, 2001. 764p.

BITTNER, K.; SPENCE, I. **Use case modeling**: UML. Boston: Pearson Education, 2003. 348p.

BOSA, J. L. **Sistema embarcado para a manutenção inteligente de atuadores elétricos**. 2009. 169 p. Dissertação (Mestrado em Ciência da Computação) - Programa de Pós-Graduação em Computação, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2009.

BURCHFIELD, T. R.; VENKATESAN, S.; WEINER, D. Maximizing through put in ZigBee wireless networks through analysis, simulations and implementations. In.: INTERNATIONAL WORKSHOP ON LOCALIZED ALGORITHMS AND PROTOCOLS FOR WIRELESS SENSOR NETWORK, 2007, Santa Fe, New Mexico. **Proceedings...** . Athens: CTI PRESS, Jun. 2007. p. 15 - 29. Disponível em: <http://dslab.utdallas.edu/~ryanb/pubs/ZigBee_Throughput.pdf> Acesso em: 24 nov. 2014.

CARRO, L. **Projeto e prototipação de sistemas digitais**. Porto Alegre: Editora da UFRGS, 2001. v. 1, 234 p.

CHEN L.C. et al. Measuring System for Cracks in Concrete Using Multitemporal Images. **Journal of Surveying Engineering**, Reston, v.132, n.2, p.77–82, May 2006. Disponível em: < http://ascelibrary.org/doi/abs/10.1061/%28ASCE%290733-9453%282006%29132%3A2%2877%29> Acesso em: 22 dez 2014.

CHEN, S.J.; et al. **Hardware software co-design of a multimedia SOC platform**. [S.l.]: Springer, 2009. 163p.

CORNELIS B., et al. Bayesian crack detection in ultra high resolution multimodal images of paintings. **Computer Science, Computer Vision and Pattern Recognition**, New York, v.2, p. 1 - 11, July 2013. Disponível em: <http://arxiv.org/abs/1304.5894> Acesso em: 05 jun 2014.

COSTA F.N. **Processo de produção de revestimento de fachada de argamassa problemas e oportunidades de melhoria**. 2005. 180p. Dissertação (Mestrado em Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

CREMONINI, R. A. **Incidência de manifestações patológicas em unidades escolares na Região de Porto Alegre**: recomendações para projeto, execução e manutenção. 1988. 169p. Dissertação (Mestrado em Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1988.

DAL MOLIN, D.C.C. **Fissura em estruturas de concreto armado, analise de manifestações típicas e levantamento de casos ocorridos no Estado do Rio Grande do Sul**. 1988. 238p. Dissertação (Mestrado em Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1988.

DENG, C. et al. Unmanned aerial vehicles for power line inspection: a cooperative way in platforms and communications. **Journal of Communications**. [S.l.], v. 9, n. 9, p. 687-692, Sept. 2014. Disponível em: <http://www.jocm.us/uploadfile/2014/0918/20140918110959273.pdf> Acesso em: 22 nov. 2014.

DONG, G. et al. Inspecting transmission lines with an unmanned fixed: wings aircraft. In: INTERNATIONAL CONFERENCE ON APPLIED ROBOTICS FOR THE POWER INDUSTRY (CARPI), 2, 2012, Zurich. **Proceedings...** New York: IEEE, 2012. p. 173 – 174. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6473355>. Acesso em: 11 nov. 2014.

DRAPER, M. et al. Evaluation of synthetic vision overlay concepts for UAV sensor operations: landmark cues and picture-in-picture. In.: HUMAN FACTOR & ERGONOMICS SOCIETY ANNUAL MEETING, 2006, San Franscisco. **Proceedings...** Washington, DC: DTIC, 2006. p. 38.

DUARTE, R. B. **Fissuras em alvenarias:** causas principais, medidas preventivas e técnicas de recuperação. Porto Alegre: CIENTEC, 1998. Boletim técnico, 25.

THOMAZ, E. **Trincas em edifícios**: causas, prevenção e recuperação. São Paulo: Pini; USP-IPT, 1989. 194 p.

ELDRIDGE, H. J. **Common defects in buildings**. London: Crown, 1982. 486p.

ESCHMANN, C. et al. Unmanned aircraft systems for remote building inspection and monitoring. In: European Workshop on Structural Health Monitoring, 6, 2012, Dresden. **Proccedings…** Berlin: DGZfP, 2012. Th.2.B.1. Disponível em: <http://www.ewshm2012.com/Proceedings/Th2B-Civil> Acesso em: 10 set 2013.

FURBER, S. **Arm system-on-chip architecture.** São Paulo: Addison_Wesley, 2000. 419p.

FURTADO, F. **Reconhecimento de rachaduras por visão computacional utilizando um filtro de partículas**. 2012. 41p. Monografia (Graduação Engenharia de Controle e Automação) – Universidade Federal do Ouro Preto, Ouro Preto, 2012. Disponível em: <http://www.em.ufop.br/cecau/monografia_2012.php>. Acesso em: 5 maio 2014.

FURTADO, V. H. et al. Aspectos de segurança na integração de veículos aéreos não tripulados (VANT) no Espaço Aéreo Brasileiro. In: SITRAER - SIMPÓSIO DE TRANSPORTE AÉREO, 7,2008, Rio de Janeiro. **Anais...**. Rio de Janeiro : E-PAPERS, 2008. v. 1 p. 506-517.

GONÇALVES, L.F. **Desenvolvimento de um Sistema de Manutenção Inteligente Embarcado**. 2011. 233 p. Tese (Doutorado em Engenharia Elétrica) - Programa de Pós-Graduação em Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2009.

GONZALEZ, R. C.; WOODS, R. E. **Digital image processing**. 2nd. ed. São Paulo: Pearson-Prentice Hall, 2002. 780p.

GNU COMPILER COLLECTION (GCC). **Home**. Disponível em: <https://gcc.gnu.org> Acesso em: 14 out. 2014.

GUEDES, G.T.A. **UML 2:** uma abordagem prática. Porto Alegre: Novatec, 2009. 488p.

HALMILTON, K, MILES R. **Learning UML 2.0**. Brasília: O'Reilly, 2006. 286 p.

HARALICK, R.; SHAPIRO L.; **Computer and robot vision**. [S.l.]:Addison-Wesley, 1992. v. 1. 672p.

HOLMES, J. et al. **Development of an automated pavement crack sealing system**. 2010.125p. Monograph (Reserch and development) – Department of Transportation of Georgia Institute of Technology, Atlanta. 2010. Disponível em: <http://trid.trb.org/view.aspx?id=1106013> Acesso em: 22 maio 2014.

IYER, S. SINHA, S. K. A Robust Approach for Automatic detection and Segmentation of Cracks in Undergound Pipelines Image. **ELSEVIER** Image and Vision Computing, [S.l.], v. 23, n. 10, p. 921–933, Sept. 2005. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0262885605000764# > Acesso em: 5 jun. 2014.

JENSEN, A. M. et al. In-Situ unmanned aerial vehicle (UAV) sensor calibration to improve automatic image. In: IEEE INTERNATIONAL GEOSCIENCE AND REMOTE SENSING SYMPOSIUM (IGARSS), 2010, Honolulu. **Proceedings...** . [S.l.]: IEEE, 2010. p. 596–599. July 2010. Disponível em: <http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=5652989> Acesso em: 19 dez. 2014.

KIEL, G. **Diversidade bacteriana em biofilmes de superfícies externas de prédios históricos na cidade de Porto Alegre**. 2005. 90p. Dissertação (Mestrado em Microbiologia Agrícola e do Ambiente) – Faculdade de Agronomia, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

LANGE, S.; S¨UNDERHAUF, N.; PROTZEL, P. A Vision Based Onboard, Approach for Landing and Position Control of an Autonomous Multirotor UAV in GPS-Denied Environments. In.: INTERNATIONAL CONFERENCE ON ADVANCED ROBOTICS (ICAR 2009), 2009, Munich. **Proceedings....**[S.l.]: IEEE, 2009. p. 1 - 6. Disponível em: < http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=5174709> Acesso em: Acesso em: 22 nov. 2014.

LAZZARETTI, E. P. **Avaliação de desempenho de implementações em hardware e software de algoritmos para aplicações de manutenção inteligente.** 2012. 96 p. Dissertação (Mestrado em Engenharia Elétrica) – Universidade Federal do Rio Grande do Sul, Porto Alegre, 2012.

LI, Z; et al. Knowledge-based Power Line Detection for UAV Surveillance and Inspection Systems. In: INTERNATIONAL CONFERENCE IMAGE AND VISION COMPUTING, 23., 2008, Christchurch. **Proceedings…** .[S.l.]: IEEE, 2008. p. 1-6.

MAGALHÃES, E. F. **Fissuras em Alvenarias**: configurações típicas e levantamento de incidências no Estado do Rio Grande do Sul. 2004. 180p. Dissertação (Mestrado em Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2004.

MARTINS, A.P.; PIZOLATO JUNIOR, J.C.; BELINI, V.L. Método com base em imagem para o monitoramento da abertura de fissura em alvernaria e concreto usando plataforma móvel. **Revista Ibracon de Estruturas e Materiais, IBRACON Structures and Materials Journal,** [S.1.], v.6, n. 3, p. 414-435, June 2013.

MATHWORKS. **MATLAB Coder™**: getting started guide. [S. l], 2012a. 125 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/coder/coder_gs.pdf>. Acesso em: 05 Mar. 2014.

MATHWORKS. **Embedded Coder™**: getting started guide. [S. l]. 2012b. 123 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/ecoder/ecoder_gs.pdf>. Acesso em: 05 Mar. 2014.

MATHWORKS. **HDL Coder™**: getting started guide. [S. l], 2012c. 66 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_gs.pdf>. Acesso em: 05 Mar. 2014.

MATHWORKS. **HDL Verifier™**: getting started guide. [S. l], 2012d. 15 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/hdlverifier/hdlv_gs_book.pdf>. Acesso em: 5 de Mar. de 2014.

MATHWORKS. **HDL Coder™**: user's guide. [S. l], 2012e. 1128 p. Disponível em: <http://www.mathworks.com/help/pdf_doc/hdlcoder/hdlcoder_ug.pdf >. Acesso em: 5 de Mar. de 2014.

MAXFIELD, C. **The design Warrior's guide to FPGAs**. Orlando: Academic Press, 2004. 542p.

METNI N.; HAMEL T. A UAV for bridge inspection: visual servoing control law with orientation limits. **Elsevier Automation in Construction,** Amsterdam, v. 17, n. 1, p. 3–10 22, Dec. 2006 – Disponível em:

<http://www.sciencedirect.com/science/article/pii/S0926580507000052> Acesso em: 24 out. 2014.

MOON, H.G.; KIM J.H. Inteligent Crack Detecting Algorithm On The Concrete Crack Image Using Neural Network. In: INTERNACIONAL ASSOCIATION FOR AUTOMATION AND ROBOTICS IN CONSTRUCTION (ISARC), Seoul, 2011. **Proceedings**... [S.l.]: ISARC, 2011. p. 1461-1467.

NIEMUELLER, T. **Automatic detection and segmentation of cracks in underground pipelines images**. 2006. 27p. Seminar: Medical Image Processing Summer. Disponível em: <http://www.niemueller.de/uni/crackdet/crackdet.pdf>. Acesso em: 13 abr. 2014.

OLIVEIRA, H.; CORREIA, P.L. Supervised Strategies for Cracks Detection in Images of Road Pavement Flexible Surfaces. In:EUROPEAN SIGNAL PROCESSING (EUSIPCO), 16, Lausanne, 2008. **Proceedings...** [S.l.]: EURASIP, 2008. p.25-29.

ORDONEZ, E.D.M et al. **Projeto, desempenho e aplicações de sistemas digitais em circuitos programáveis (FPGAs)**. São Paulo: Pompéia, 2003. 300p.

OBJECT MANAGEMENT GROUP. **OMG Unified Modeling Language (OMG UML), Superstructure**. Needham, 2011. 748p. Disponível em: <http://www.omg.org/spec/UML/2.4.1/>. Acesso em: 22 May 2014.

PASQUALOTTO, N. **Mapeamento de manifestações patológicas em edificação histórica**: estudo no prédio do Observatório Astronômico da UFRGS. 2012. 139p. Projeto de Diplomação (Graduação em Engenharia Civil) - Universidade Federal do Rio Grande do Sul, Porto Alegre, 2012.

PARKER, J.R. **Algorithms for Image Processing and Computer Vision**. Indianapolis: Wiley, 2011. 506p.

PEDRINI, H.; SCHWARTZ, W.R. **Análise de imagens digitais**: princípios, algoritmos e aplicações. São Paulo: Thomson, 2008. 508p.

PRASANNA, P.; et al. Computer Vision Based Crack Detection and Analysis. In.: CONFERENCE SENSORS AND SMART SCTRUCTURE TECHONOLOGIES FOR CIVIL, MECHANICAL, AND AEROSPACE SYSTEMS, 2012, San Diego. **Proceedings…**. [S.1]: SPIE, 2012, v.8345, p. 834542-1. Disponível em: < http://proceedings.spiedigitallibrary.org/proceeding.aspx?articleid=1314546> Acesso em: 19 jan 2014.

RASPBERRY PI. **Documentation and downloads for an ARM based computer running under GNU/Linux**. Disponível em: <http://www.raspberrypi.org/documentation/ > Acesso em: 5 Jun. 2014.

RATHINAM, S.; KIM Z. W.;SENGUPTA, R. **Vision-Based following of structures using an unmanned aerial vehicle (UAV)**: research report. Berkeley: University of California at Berkeley, 2006. Disponível em: <http://its.berkeley.edu/publications/UCB/2006/RR/UCB-ITS-RR-2006-1.pdf > Acesso em: 24 de out. 2014.

RIDLER, T.W.; CALVARD, S. Picture thresholding using an iterative selection method, **IEEE Transaction on System, Man, and Cybernetics,** New York,v.8, n. 8, p. 630-632. Aug.1978.

RITTER, G. X., WILSON J. N. **Handbook of computer vision algorithms in image Algebra**. 2nd ed. Boca Raton: CRC Press, 2000. 425p.

SANTHI, B. et al. Automatic Detection of Cracks in Pavements using edge Detectin Operator. **Journal of Theoretical and Applied Information Technology**, Islamabad, v.36, n.2, p.199-205, Feb 2012.

SILVA, A. F. **Manifestações patológicas em fachadas com revestimentos argamassados**: estudo de caso em edifícios em Florianópolis. 2007. 190p. Dissertação (Mestrado em Arquitetura e Urbanismo) - Universidade Federal de Santa Catarina, Florianópolis, 2007.

SILVA, L. M. B. **Desenvolvimento de um sistema especialista para diagnóstico de fissuras em concreto armado**. 1996. 180p. Dissertação (Mestrado em Engenharia) no Curso de Pós- Graduação em Engenharia Civil, Universidade Federal do Rio Grande do Sul, Porto Alegre, 1996.

SOBEL, I. Na Isotropic 3 x 3 Image Gradient Operator. In.:Freeman, H. (ed.) **Machine vision for three-dimensional scenes**. London: Academic Press, Generalized and Separable Sobel Operatorsm, 1990. p. 376-379.

SZELISKI, R. **Computer Vision: Algorithms and Applications.** [S.l.]:Springer, 2010. 979p. Disponível em: <http://szeliski.org/Book/> Acesso em: 10 jun 2014.

THRUN, S. **Probabilistic robotics**. Cambridge: MIT Press, c2006. 647 p.

VASSÁNYI, I. Implementing processor arrays on FPGAs. **Field-Programmable logic and applications from FPGAs to computing paradigm lecture notes in computer science**, Estonia, v. 1482, p. 446-450 27. Aug. 1998.

VERÇOZA, E. J. **Patologia das edificações**. Porto Alegre: Sagra, 1991. 173 p.

VIEIRA A. A. **Influência dos detalhes arquitetônicos no estado de conservação das fachadas de edificações do patrimônio cultural do Centro Histórico de Porto Alegre**: Estudo de Caso. 2005. 163p. Dissertação (Mestrado em Engenharia Civil) – Programa de Pós-Graduação em Engenharia Civil, Universidade Federal do Rio Grande do Sul, Porto Alegre, 2005.

ZHAO, J. et al.. An Adaptive Threshold Neural-Network Scheme for Rotorcraft UAV Sensor Failure Diagnosis. **Advances in Neural Networks**, Nanjing, v. 4493, n.4 p. 589-596. June 2007.

ZIGBEE ALLIANCE. **Home**: ZigBee Alliance. Disponível em: <http://zigbee.org/> Acesso em: 24 nov. 2014.

APÊNDICE : ALGORITMOS

A.1 ALGORITMO A – AMBIENTE MATLAB

```
% Programa Principal Matlab main_matlab.m

close all
clear all
clc

fudgeFactor=zeros(1,1);

%% Inicializa a camera de captura
% vid = videoinput('winvideo',1,'RGB24_320x240');   %Dado obtido de um
quadro da camera

%% Image obtained by file (Study)
%
% vid = VideoReader('c:\Videos_Fissuras\SDC10164.AVI');  %Tables Get a mp4
video (H.264 codec installation Needed encoded video (. Mp4,. M4v)
%
% fps = vid.FrameRate;
% duration = vid.Duration;
%
% if vid.NumberOfFrames >= 0
%     nFrames=vid.NumberOfFrames;       %nFrames = get(vid,
'NumberOfFrames');
% else
%     nFrames = fps * duration;
% end
%
% num = round(nFrames/fps)-1;


%% Analise de uma imagem obtida em arquivo
im = imread('C:\Imagens_1024x768\Fissura_1024x768_01.jpg');
%im = imread('Fissura03.jpg');

%Step shows Image Original
figure(1), imshow(im), title('Image Capturada');
num=1;

for k=1:num
    %% Inicio do tratamento da Imagem

    %Transformação RGB para Escala de Cinza
    %%Equivalente a:   % Y_Gray = rgb2gray(im);
    Y_Gray = .2989*im(:,:,1) + .5870*im(:,:,2) + .1140*im(:,:,3);

    %Tamanho da Imagem
    [row,col] = size(Y_Gray);
    size_r = row*col;


    %% Detecção de Bordas usando Sobel
```

```
    %%Equivalent a:  % [~, threshold] = edge(Y_Gray, 'sobel'), fudgeFactor
= 2.5, B = edge(Y_Gray,'sobel', threshold*fudgeFactor);

    fudgeFactor=fudFac(Y_Gray)    %Seleção do Fator para detecção de borda

    B = sobelEdge(Y_Gray, fudgeFactor);

    %Conversão para imagem binária
    imBW=logical(B);

    %Step shows the Edge detected Image
    figure(2), imshow(imBW), title('Detecção de Borda da Imagem');

    %% Dilated Gradient Mask
    % equivalent: % SE = strel('octagon',7); dilatImage = imdilate(imBW,
[SE]);

    %Structuring element octagon 7 x 7
    SED = [0 0 1 1 1 0 0; 0 1 1 1 1 1 0; 1 1 1 1 1 1 1;  1 1 1 1 1 1 1; 1 1
1 1 1 1 1; 0 1 1 1 1 1 0; 0 0 1 1 1 0 0 ];

    dilatImage = dilateimage(imBW,SED);

    %Step shows the dilated gradient mask
    figure(3), imshow(dilatImage), title('Dilatação da Imagem');


    %% Fill Holes of Image
    %Equivalent:   fillHoles = imfill(dilatImage, 'holes');

    %Structuring element diamant 3 x 3
    SEF = [ 0 1 0 ; 1 1 1; 0 1 0];

    fillHoles = fillimage(dilatImage,SEF);

    figure(4), imshow(fillHoles);
    title('Preenchimento de lacunas na Imagem');

    %% Erose Image
    %%Equivalent: seE = strel('diamant',3), erodeImage =
imerode(fillHoles,seE);

    %Structuring element
    %SEE=[0 1 1 1 0; 1 1 1 1 1; 1 1 1 1 1; 1 1 1 1 1; 0 1 1 1 0];
    SEE = [ 0 1 0 ; 1 1 1; 0 1 0 ];

    erodeImage =  erodeimage(fillHoles,SEE);

    figure(5), imshow(erodeImage), title('Erosão da Imagem');


    %% Function that separates the cracks detected and returns parameters
of each crack
    %%Equivalent: im_label = bwlabel(erodeImage,8);

    im_label = sepbwlabel(erodeImage);
```

```
    figure(6), imshow(im_label), title('Image label');


    %% Função que separa as fissuras detectadas e devolve parametros de
cada fissura

    [NumCrack, center, Area, Orientation, MajorAxisLength,
MinorAxisLength,imcrack] = defcrack(im_label, size_r);


    figure(7), imshow(imcrack), title('Imagem Binária das Fissuras
Destacadas');

    %Gera Arquivo de Bitmap
    filename = sprintf('ImagemcomFissura_%d',k);
    print ('-dbitmap', filename);


    figure(8), imshow(Y_Gray)  %Plota objeto de interesse

    %Plota Imagem com indicação da fissura
    hold on
    if NumCrack ~=0
        for num=1:NumCrack
            plot(center(1,1,num), center(1,2,num), '*', 'color','r')
        end
        title('Indicação das Fissuras Detectadas')
        hold off

        %Gera Arquivo de Bitmap
        filename = sprintf('IndicaçãoFissura_%d',k);
        print ('-dbitmap', filename);

        figure(9)                                                  %Plota
o angulo de orientação do individuo
        hold on
        for num=1:NumCrack
            plot(num, Orientation(num), '*', 'color','b')
        end
        title('Orientação angular das Fissuras')
        hold off
    end

    %Gera Arquivo de Bitmap
    filename = sprintf('AngulosFissura_%d',k);
    print ('-dbitmap', filename);

end


-------------------------------------------------------------------------

% Função fudFac.m


function [fudgeFactor] = fudFac(Y_Gray) %#codegen

assert(all(size(Y_Gray) <= [1944 2592]));
```

```
assert(isa(Y_Gray, 'uint8'));

fudgeFactor = zeros(1,1);

major_val = max(max(Y_Gray));
minor_val = min(min(Y_Gray));

if (major_val / minor_val) > 5
    fudgeFactor(1,1) = 0.40;
elseif (major_val / minor_val) > 3
    fudgeFactor(1,1) = 0.35;
elseif (major_val / minor_val) > 2
    fudgeFactor = 0.30;
elseif (major_val / minor_val) > 1
    fudgeFactor = 0.25;
elseif (major_val / minor_val) > 0.75
    fudgeFactor = 0.20;
elseif (major_val / minor_val) > 0.5
    fudgeFactor = 0.15;
else
    fudgeFactor = 0.10;
end
```

```
% Função sobelEdge.m

function B = sobelEdge(Y_Gray,fudgeFactor) %#codegen

assert(all(size(Y_Gray) <= [1944 2592]));
assert(isa(Y_Gray, 'uint8'));
assert(all(size(fudgeFactor) == [1,1]));
assert(isa(fudgeFactor, 'double'));

%% Edge With Sobel
% equivalent: [~, threshold] = edge(Y_Gray, 'sobel'); fudgeFactor = 1.0; B
= edge(Y_Gray,'sobel', threshold*fudgeFactor);

%Initialize a matrix of matrix size ImSobel with zeros
Gray_D=double(Y_Gray);
ImSobel = zeros(size(Gray_D));

for i=1:size(Gray_D,1)-2
    for j=1:size(Gray_D,2)-2
        %Sobel mask for x-direction:
        Gx=((Gray_D(i+2,j)+2*Gray_D(i+2,j+1)+Gray_D(i+2,j+2))-
(Gray_D(i,j)+2*Gray_D(i,j+1)+Gray_D(i,j+2)));
        %Sobel mask for y-direction:
        Gy=((Gray_D(i,j+2)+2*Gray_D(i+1,j+2)+Gray_D(i+2,j+2))-
(Gray_D(i,j)+2*Gray_D(i+1,j)+Gray_D(i+2,j)));
        %The gradient of the image
        ImSobel(i,j)=abs(Gx)+abs(Gy);   %or
ImSobel(i,j)=sqrt(Gx.^2+Gy.^2);
    end
end

T=fudgeFactor*(min(ImSobel(:))+max(ImSobel(:)));

Thresh=round(T);
```

```
B=max(ImSobel,Thresh);
B(B==round(Thresh))=0;
```

```
% Função dilateimage.m

function dilatBW = dilateimage(imBW, SED) %#codegen

assert(all(size(imBW) <= [1944 2592]));
assert(isa(imBW, 'logical'));
assert(all(size(SED) <= [7,7]));
assert(isa(SED, 'double'));

imBW=double(imBW);

%% Dilated Gradient Mask
% equivalent: % SE = strel('octagon',7); dilatImage = imdilate(B, [SE]);

%Dimension to the calculation of sums of structuring elements
[rowimBW, colimBW]=size(imBW);
[rowSED, colSED]=size(SED);
%Sides of the structuring element;
mlSED=floor(rowSED/2);
mcSED=floor(colSED/2);
dilatImage=double(imBW);
for i=1:rowSED
    for j=1:colSED
        if SED(i,j)==1
            pixels1=max(0, i-mlSED-1);
            pixels2=max(0, mlSED-i+1);
            pixels3=max(0, j-mcSED-1);
            pixels4=max(0, mcSED-i+1);

dilatImage=max(dilatImage,[zeros(pixels1,colimBW);zeros(rowimBW-pixels1-
pixels2,pixels3),imBW(1+pixels2:rowimBW-pixels1,1+pixels4:colimBW-
pixels3),zeros(rowimBW-pixels2-pixels1,pixels4);zeros(pixels2,colimBW)]);
        end
    end
end

dilatBW=logical(dilatImage);
```

```
% Função fillimage.m

function [fillHolesBW] = fillimage(imBW, SEF) %#codegen

assert(all(size(imBW) <= [1944 2592]));
assert(isa(imBW, 'logical'));
assert(all(size(SEF) <= [3,3]));
assert(isa(SEF, 'double'));

%% Fill Holes of Image
```

```
    %Equivalent:    fillHoles = imfill(dilatImage, 'holes');

imBW=double(imBW);

%Dimension to the calculation of sums of structuring elements
[rowimBW, colimBW]=size(imBW);
[rowSEF, colSEF]=size(SEF);
%Sides of the structuring element;
mlSEF=floor(rowSEF/2);
mcSEF=floor(colSEF/2);

%first dilation
%Initialize a matrix of matrix size imBW with zeros
imBWF=imBW;
for i=1:rowSEF
    for j=1:colSEF
        if SEF(i,j)==1
            pixels1=max(0, i-mlSEF-1);
            pixels2=max(0, mlSEF-i+1);
            pixels3=max(0, j-mcSEF-1);
            pixels4=max(0, mcSEF-j+1);
            imBWF=max(imBWF,[zeros(pixels1,colimBW);zeros(rowimBW-pixels1-
pixels2,pixels3),imBW(1+pixels2:rowimBW-pixels1,1+pixels4:colimBW-
pixels3),zeros(rowimBW-pixels2-pixels1,pixels4);zeros(pixels2,colimBW)]);
        end
    end
end

%after erosion
fillHoles=imBWF;
for i=1:rowSEF
    for j=1:colSEF
        if SEF(i,j)==1
            pixels1=max(0, i-mlSEF-1);
            pixels2=max(0, mlSEF-i+1);
            pixels3=max(0, j-mcSEF-1);
            pixels4=max(0, mcSEF-j+1);
            fillHoles=min(imBWF,[zeros(pixels1,colimBW);zeros(rowimBW-
pixels1-pixels2,pixels3),fillHoles(1+pixels2:rowimBW-
pixels1,1+pixels4:colimBW-pixels3),zeros(rowimBW-pixels2-
pixels1,pixels4);zeros(pixels2,colimBW)]);
        end
    end
end

fillHolesBW=logical(fillHoles);
```

```
% Função erodeimage.m


function [erodeImageBW] = erodeimage(imBW,SEE) %#codegen

assert(all(size(imBW) <= [1944 2592]));
assert(isa(imBW, 'logical'));
assert(all(size(SEE) <= [3,3]));
assert(isa(SEE, 'double'));
```

```
%% Erose Image
%%Equivalent: seE = strel('octagon',5), erodeImage =
imerode(fillHoles,seE);

imBW=double(imBW);
[rowimBW, colimBW]=size(imBW);
[rowSEE, colSEE]=size(SEE);
%Sides of the structuring element;
mlSEE=floor(rowSEE/2);
mcSEE=floor(colSEE/2);
erodeImage=imBW;
for i=1:rowSEE
    for j=1:colSEE
        if SEE(i,j)==1
            pixels2=max(0, i-mlSEE-1);
            pixels1=max(0, mlSEE-i+1);
            pixels4=max(0, j-mcSEE-1);
            pixels3=max(0, mcSEE-j+1);

erodeImage=min(erodeImage,[zeros(pixels1,colimBW);zeros(rowimBW-pixels1-
pixels2,pixels3),imBW(1+pixels2:rowimBW-pixels1,1+pixels4:colimBW-
pixels3),zeros(rowimBW-pixels2-pixels1,pixels4);zeros(pixels2,colimBW)]);
        end
    end
end

erodeImageBW=logical(erodeImage);
```

```
% Função sepwlabel.m


function [Label] = sepbwlabel(A) %#codegen

assert(all(size(A) <= [1944 2592]));
assert(isa(A, 'logical'));

[row, col]=size(A);
Label=zeros(row,col);
Conect=zeros(1,3);
k=zeros(1,1);
N=zeros(1,1);
Value=zeros(1,1);

for i=1:row     %Rastreamento da matriz de imagem
    for j=1:col
        if A(i,j)~=0          %Verifica a existência de pixel na varredura

            %Verifica a existência de pixel nas vizinhanças
            if i>1 && j>1
                if A(i-1,j-1)~=0
                    Value=Label(i-1,j-1);
                end
            end
            if i>1
                if A(i-1,j)~=0
```

```
                if Value==0;
                    Value=Label(i-1,j);
                elseif Value<Label(i-1,j)
                    k=k+1;
                    Conect(1,k)=Label(i-1,j);
                elseif Value>Label(i-1,j)
                    k=k+1;
                    Conect(1,k)=Value;
                    Value=Label(i-1,j);
                end
            end
        end
        if i>1 && j<col
            if A(i-1,j+1)~=0
                if Value==0;
                    Value=Label(i-1,j+1);
                elseif Value<Label(i-1,j+1)
                    k=k+1;
                    Conect(1,k)=Label(i-1,j+1);
                elseif Value>Label(i-1,j+1)
                    k=k+1;
                    Conect(1,k)=Value;
                    Value=Label(i-1,j+1);
                end
            end
        end
        if j>1
            if A(i,j-1)~=0
                if Value==0;
                    Value=Label(i,j-1);
                elseif Value<Label(i,j-1)
                    k=k+1;
                    Conect(1,k)=Label(i,j-1);
                elseif Value>Label(i,j-1)
                    k=k+1;
                    Conect(1,k)=Value;
                    Value=Label(i,j-1);
                end
            end
        end
        % Atualiza o valor sequencial
        if Value==0
            N=N+1;
            Label(i,j)=N;
        else         %Atribui o valor do pixel na vizinhança
            Label(i,j)=Value;
            if k>0      % Atualiza os valores do matriz de referencia
               for l=1:i
                    for m=1:col
                        for z=1:k
                            if Label(l,m)==Conect(1,z)
                                Label(l,m)=Value;
                            end
                        end
                    end
               end
                k=0;
            end
            Value=0;
        end
    end
```

```
    end
end
% Label2=zeros(row,col);
% cont=zeros(1,1);
% num=zeros(1,1);
%% Sequencia os objetos
% for int=N:-1:1
%     for i=1:row     %Rastreamento da matriz de imagem
%         for j=1:col
%             if Label(i,j)==int
%                 if cont == 0
%                     num=num+1;
%                     cont = 1;
%                 end
%                 Label2(i,j) = num;
%             end
%         end
%     end
%     cont=0;
% end
```

```
% Função defcrack.m

function [NCrack, CENTER, AREA, Deg, MajorAxisLenght, MinorAxisLength,
imCrack] = defcrack(im_label,NPixels) %#codegen

assert(all(size(NPixels) == [1,1]));
assert(isa(NPixels, 'double'));
assert(all(size(im_label) <= [1944 2592]));
assert(isa(im_label, 'double'));

NumObject = max (max(im_label))               %Gets the value of the number
of segments

%% Description of the crack

MaxPixels = NPixels * 0.00015;
MinPixels = NPixels * 0.009;
MinArea = NPixels * 0.0015;
MaxArea = NPixels * 0.09;

%% Variables
NCrack = zeros(1);
% MajorAL=zeros(1);
% MinorAL=zeros(1);
% Area=zeros(1);

min_px_in = zeros(1);
min_px_en = zeros(1);
min_py_in = zeros(1);
min_py_en = zeros(1);

% major_row = zeros(1);
% minor_row = zeros(1);
% major_col = zeros(1);
```

```
% minor_col = zeros(1);
% ind_majrow = zeros(1);
% ind_minrow = zeros(1);
% ind_majcol = zeros(1);
% ind_mincol = zeros(1);

ConjArea = zeros(1,NumObject);
MajAL = zeros(1,NumObject);
MinAL = zeros(1,NumObject);
ConjDeg=zeros(1,NumObject);
mid_point_MajorAxis=zeros(1,2);
% mid_point_MinorAxis=zeros(1,2);
ConjCENTER=zeros(1,2,NumObject);

%% Matrix with the same dimension with the original separate images
[row, col]=size(im_label);                          %Size of the original
matrix
imCrack = zeros(row,col);

pos=zeros(2,NPixels);

for m=1:NumObject                                                    %
Acquired data from separate images
    Im_n = zeros(row,col);              %Creates array of zeros with the
dimensions of the original matrix
    pos=zeros(2,NPixels);
    Area=0;
    for l=1:row
        for j=1:col
            if im_label(l,j)== m
                Im_n(l,j) = 1;
                Area=Area+1;                           %Get the area of each
object
                pos(1,Area) = l;
                pos(2,Area) = j;
            end
        end
    end

    if Area >=  MinArea && Area <=  MaxArea % Elimination of non-standard
objects

        [major_row,ind_majrow] = max(pos(1,1:Area));
        [minor_row,ind_minrow] = min(pos(1,1:Area));
        [major_col,ind_majcol] = max(pos(2,1:Area));
        [minor_col,ind_mincol] = min(pos(2,1:Area));

        row_n = major_row - minor_row + 1;
        col_n = major_col - minor_col + 1;

        Im_reduz = zeros(row_n,col_n);

        for l=1:(row_n)
            for j=1:(col_n)
                Im_reduz(l,j)=Im_n(l+minor_row-1,j+minor_col-1);
            end
        end

        %% Description of the Region
```

```
        % Equivalent: Reduz = regionprops(Im_reduz, 'Image'); Bloco =
Reduz(:,:).Image; Bloco = logical(Bloco); Area = bwarea(Bloco);  EM =
regionprops(Bloco, 'MajorAxisLength'); MajorAL = EM.MajorAxisLength; Em =
regionprops(Bloco, 'MinorAxisLength'); MinorAL = Em.MinorAxisLength;

%         dif1 = zeros(1);
%         dif2 = zeros(1);
%         dif3 = zeros(1);
%         dif4 = zeros(1);
%         dif5 = zeros(1);
%         dif6 = zeros(1);
%         dist1 = zeros(1);
%         dist2 = zeros(1);
%         dist3 = zeros(1);
%         dist4 = zeros(1);
        dist5 = zeros(1);
        ang = zeros(1);
        ang1 = zeros(1);
        ang2 = zeros(1);
%         ang3 = zeros(1);

        %Gets the major axis of each image based on the largest distance
        dif1 = pos(1,ind_minrow)-pos(1,ind_mincol);
        dif2 = pos(2,ind_minrow)-pos(2,ind_mincol);
        dif3 = pos(1,ind_minrow)-pos(1,ind_majrow);
        dif4 = pos(2,ind_minrow)-pos(2,ind_majrow);
        dif5 = pos(1,ind_minrow)-pos(1,ind_majcol);
        dif6 = pos(2,ind_minrow)-pos(2,ind_majcol);
        dist1 = sqrt(dif1*dif1+dif2*dif2);
        dist2 = sqrt(dif3*dif3+dif4*dif4);
        dist3 = sqrt(dif5*dif5+dif6*dif6);

        [MajorAL,ind] = max([dist1,dist2,dist3]);

        if MajorAL>= MaxPixels % Elimination of non-standard objects, major
axis

            if ind == 1
                ang = atan(double((dif1)/(dif2)));
            elseif ind == 2
                ang = atan(double((dif3)/(dif4)));
            elseif ind == 3
                ang = atan(double((dif5)/(dif6)));
            end


            %range of angles of the minor axis to obtain
            if ang < 0
                ang1 = ang + 1.83;
                ang2 = ang + 1.31;
            end
            if ang > 0
                ang2 = ang - 1.83;
                ang1 = ang - 1.31;
            end

            %Check points within the range of the angle for storing the
minor axis position of these points
```

```
            for i=1:Area
                for j=i:Area
                    dif1 = pos(1,j)-pos(1,i);
                    dif2 = pos(2,j)-pos(2,i);
                    ang3 = atan(dif1/dif2);
                    if  (ang3>ang2) && (ang3<ang1)
                        dist4=sqrt((dif1*dif1)+(dif2*dif2));
                        if (dist4 > dist5)
                            dist5 = dist4;
                            min_px_in = pos(2,i);
                            min_px_en = pos(2,j);
                            min_py_in = pos(1,i);
                            min_py_en = pos(1,j);
                        end
                    end
                end
            end

            MinorAL = dist5;

            if MinorAL<= MinPixels  % Elimination of non-standard objects
Elimination of non-standard objects, minor axis

                NCrack = NCrack + 1;
                %% Matrix de cracks detection
                for l=1:row_n
                    for j=1:col_n
                        imCrack(l+minor_row-1,j+minor_col-1)=Im_reduz(l,j);
                    end
                end

                %% Data of the cracks

                ConjArea(1,NCrack) = Area;
                MajAL(1,NCrack) = MajorAL;
                MinAL(1,NCrack) = MinorAL;
                %Equivalent:          s = regionprops(Im_reduz,
'centroid'); centro = s.Centroid; CENTER(:,:,z) = centro(1,:);

                if ind == 1
                    mid_point_MajorAxis =
[(pos(2,ind_minrow)+pos(2,ind_mincol))/2
(pos(1,ind_minrow)+pos(1,ind_mincol))/2];
                elseif ind == 2
                    mid_point_MajorAxis =
[(pos(2,ind_minrow)+pos(2,ind_majrow))/2
(pos(1,ind_minrow)+pos(1,ind_majrow))/2];
                elseif ind == 3
                    mid_point_MajorAxis =
[(pos(2,ind_minrow)+pos(2,ind_majcol))/2
(pos(1,ind_minrow)+pos(1,ind_majcol))/2];
                end
                mid_point_MinorAxis = [(min_px_en+min_px_in)/2
(min_py_en+min_py_in)/2];    % Ponto mï¿½dio de um segmento de reta eixo
menor
                ConjCENTER(:,:,NCrack) =
[(mid_point_MajorAxis(1,1)+mid_point_MinorAxis(1,1))/2
(mid_point_MajorAxis(1,2)+mid_point_MinorAxis(1,2))/2];
```

```
                %Equivalent:  Reduz = regionprops(Im_n, 'Image');  Bl =
Reduz(:,:).Image;   Bl = logical(Bl);  O = regionprops(Bl, 'Orientation');
Deg(1,z) = O.Orientation;
                ConjDeg(1,NCrack)=(ang)*180/pi;
            end
        end
    end
end

AREA=zeros(1,NCrack);
CENTER=zeros(1,2,NCrack);
Deg=zeros(1,NCrack);
MajorAxisLenght=zeros(1,NCrack);
MinorAxisLength=zeros(1,NCrack);

for z=1:NCrack
    AREA(1,z)=ConjArea(1,z);
    MajorAxisLenght(1,z) = MajAL(1,z);
    MinorAxisLength(1,z) = MinAL(1,z);
    CENTER(:,:,z) = ConjCENTER(:,:,z);
    Deg(1,z) = ConjDeg(1,z);
end
```

A.2 ALGORITMO B - AMBIENTE MATLAB

```
%Programa Principal  - ParticleFilterCrack.m

close all
clear all
clc


%% inicialização da câmera

% vid = videoinput('winvideo',1,'RGB24_320x240');   Inicializa camera
instalada no computador ou dispositivo:

%% Imagem obtida por arquivo (Estudo)
%
% vid = VideoReader('c:\Videos_Fissuras\SDC10001.AVI');  %Obter Quadros de
um video mp4 (Necessario instalação do codec H.264 encoded video (.mp4,
.m4v)
%
% fps = vid.FrameRate;
% duration = vid.Duration;
%
% if vid.NumberOfFrames >= 0
%     nFrames=vid.NumberOfFrames;       %nFrames = get(vid,
'NumberOfFrames');
% else
%     nFrames = fps * duration;
% end
%
% num = round(nFrames/fps)-1;
```

```
%Analise de Image de um Arquivo de Foto
im = imread('C:\Imagens_1024x768\Fissura_1024x768_01.jpg');
figure(1)
imshow(im)
num=1;

for k=1:num
    %% Inicio dos Parâmetros Filtro de Particulas

    F_update = [1 0 1 0; 0 1 0 1; 0 0 1 0; 0 0 0 1];
    Npop_particles = 10000;

    Xstd_rgb = 10;
    Xstd_pos = 50;
    Xstd_vec = 50;
    Xrgb_trgt = 0;

    Npix_resolution = size(im);

    %Transformação RGB para Escala de Cinza
    %%Equivalente a:   % Y_Gray = rgb2gray(im);
    Y_Gray = .2989*im(:,:,1) + .5870*im(:,:,2) + .1140*im(:,:,3);


    %% Rastreamento do Objeto pelo Filtro de Partículas
    particles = create_particles(Npix_resolution, Npop_particles);

    % Previsão
    newparticles = update_particles(F_update, Xstd_pos, Xstd_vec,
particles);

    % Calculando probabilidade
    vl_calc = calc_log_likelihood(Xstd_rgb, Xrgb_trgt, newparticles,
Y_Gray);

    % Reamostrando
    positions = resample_particles(newparticles, vl_calc);

    % Mostrando imagem com particulas detectadas
    figure(2)
    imshow(im)
    title('Pontos da Particulas')
    hold on
    plot(positions(2,:), positions(1,:), '*','color','r')
    hold off
    drawnow

    %Gera Arquivo de Bitmap
    filename = sprintf('ImagemcomFissura_%d',k);
    print ('-dbitmap', filename);



end
```

```
%Função  crate_particles.m


function particles = create_particles(Npix_resolution, Npop_particles)
%#codegen

assert(all(size(Npix_resolution)==[1 3]));
assert(isa(Npix_resolution, 'double'));
assert(all(size(Npop_particles)==[1 1]));
assert(isa(Npop_particles, 'double'));

% Cria Npop_particles numeros aleatórios variando entre 0 e resolução na
Horizontal/Coluna (resoluçao no eixo X)
X1 = randi(Npix_resolution(2), 1, Npop_particles);
% Cria Npop_particles numeros aleatórios variando entre 0 e resolução na
Vertical/Linha(resoluçao no eixo Y)
X2 = randi(Npix_resolution(1), 1, Npop_particles);
%Cria um vetor de 2 dimensões com Npop_particles elementos cada de valor
zero.
X3 = zeros(2, Npop_particles);
% Vetor de vetores (matrix) -> X = [X1[4]; X2[10000]; X3[1]]
% X1[Npop_particles] ----- X2[Npop_particles] ------
X3[10000][Npop_particles]
particles = [X1; X2; X3];
```

```
%Função update_particles.m

function newparticles = update_particles(F_update, Xstd_pos, Xstd_vec,
particles) %#codegen

assert(all(size(Xstd_pos)==[1 1]));
assert(isa(Xstd_pos, 'double'));
assert(all(size(F_update)==[4 4]));
assert(isa(F_update, 'double'));
assert(all(size(particles) <= [4 10000]));
assert(isa(particles, 'double'));
assert(all(size(Xstd_vec) == [1 1]));
assert(isa(Xstd_vec, 'double'));

% Numero de elementos na segunda posição do vetor (10000)
N = size(particles, 2);
% X1 + X3(1); X2 + X3(2); X3(1); X3(2)
% F_update consiste na rotina da multiplicação da matriz
% para que tenha a configuração dada acima.
newparticles = F_update * particles;
% Adição da incerteza na posição da particula
newparticles(1:2,:) = newparticles(1:2,:) + Xstd_pos * randn(2, N);
% Adição da incerteza no deslocamento da particula
newparticles(3:4,:) = newparticles(3:4,:) + Xstd_vec * randn(2, N);
```

```
% Função calc_log_linkelihood.m

function vl_calc = calc_log_likelihood(Xstd_rgb, Xrgb_trgt, newparticles,
Y_Gray) %#codegen
```

```
assert(all(size(Xstd_rgb)==[1 1]));
assert(isa(Xstd_rgb, 'double'));
assert(all(size(Xrgb_trgt)==[1 1]));
assert(isa(Xrgb_trgt, 'double'));
assert(all(size(newparticles) <= [4 10000]));
assert(isa(newparticles, 'double'));
assert(all(size(Y_Gray) <= [1944 2592]));
assert(isa(Y_Gray, 'uint8'));


% Armazena o numero de pixels na coluna do frame em Npix_h.
Npix_h = size(Y_Gray, 1);
% Armazena o numero de pixels na linha do frame em Npix_w.
Npix_w = size(Y_Gray, 2);
% Numero de particulas.
N = size(newparticles,2);
% Cria um vetor de 0 com 4000 elementos.
vl_calc = zeros(1,N);
% Muda o vetor de escala de cinza para a primeiro posição do vetor da
% imagem, a resolução da coluna de pixels para a segunda posição e a
% resolução da linha de pixels para a terceira posição.
Y = permute(Y_Gray, [3 1 2]);
% Calculos de imprecisão
A = -log(sqrt(2 * pi) * Xstd_rgb);
B = - 0.5 / (Xstd_rgb.^2);
% Arredonda os valores double de X para inteiros.
X = round(newparticles);
for k = 1:N
    % Armazena em "m" a posição da particula na coluna de pixels
    m = X(1,k);
    % Armazena em "n" a posição da particula na linha de pixels
    n = X(2,k);
    % Verdadeiro caso o valor de m esteja no intervalor possivel da
    % coluna de pixels.
    I = (m >= 1 & m <= Npix_h);
    % Verdadeiro caso o valor de n esteja no intervalo possivel da
    % linha de pixels.
    J = (n >= 1 & n <= Npix_w);
    % Certifica-se que as operações somente serão validas para as
particulas
    % contidas no frame do video.
    if I && J
        F = 1;
        Z=all(Y(:, m, n));
        if Z >= 0
            F = 2;
        end
        % Faz cast nas dimensoes da imagem para double.
        C = double(Y(:, m, n));
        % Subtrai do canal o target Color.
        D = C - Xrgb_trgt;
        % Cria a tendencia da particula corresponder ao target color de
acordo,
        % sendo particulas com menor valor correspondentes a maior
tendencia.
        % (no método seguinte há uma equação que melhora a compreensão).
        D2 = D' * D;
        % Introduz a equação que determina o intervalo possível para o
target color.
        vl_calc(k) = A + B * D2/F;
    else
```

```
        % Elimina particulas fora da área da imagem.
        vl_calc(k) = -1000000;
    end
end
```

```
% Função resample_particles.m

function positions = resample_particles(newparticles, vl_calc) %#codegen

assert(all(size(newparticles) <= [4 10000]));
assert(isa(newparticles, 'double'));
assert(all(size(vl_calc)<= [1 10000]));
assert(isa(vl_calc, 'double'));

% Calculo da distribuição cumulativa
% Exponencial da subtração do maior valor da probabilidade
% da partícula
L = exp(vl_calc - max(vl_calc));
% Divide cada termo da probabilidade pelo somatório da mesma
Q = L / sum(L, 2);
% Faz a soma cumulativa da distribuição da probabilidade
R = cumsum(Q, 2);
% Generating Random Numbers
N = size(newparticles, 2);
T = rand(1, N);
% Reamostrando
% Armazena a posição das particulas
[~, I] = histc(T, R);
% Armazena somenta as particulas mais aptas a sobrevivencia
positions = newparticles(: , I + 1);
```

A.3 PROGRAMA PRINCIPAL EM C – ALGORITMO A

```
Programa Principal :
dtcrackdesc.c
Data: 14/10/2014
*************************************************************************
***************************/

#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>

#include "rt_nonfinite.h"
#include "sobelEdge.h"
#include "defcrack.h"
#include "dilateimage.h"
#include "erodeimage.h"
#include "fillimage.h"
#include "sepbwlabel.h"
#include "defcrack_emxutil.h"
#include "fudFac.h"
```

```
#include <jpeglib.h>
#define IMAGE_NCHANNELS 1
#define RESOL_W 1024
#define RESOL_H 768

/* struct to store a grayscale raw image */
struct raw_img {
    uint8_t *img;
    uint8_t num_comp;
    uint32_t width;
    uint32_t height;
    real_T factor;
    uint8_T *imgdim;
    uint8_T *imgswap;
    real_T *imgsobel;
    boolean_T *imgBW;
};

    real_T sobel_img[RESOL_W*RESOL_H];
    boolean_T dilatImage[RESOL_W*RESOL_H];
    boolean_T fillHoles[RESOL_W*RESOL_H];
    boolean_T erodeImage[RESOL_W*RESOL_H];
    real_T label[RESOL_W*RESOL_H];
    real_T imcrack[RESOL_W*RESOL_H];

/**
 * jpeg_decompress - open a jpeg image, decompress it and convert
 * to a raw grayscale image
 * @param file name of jpeg file
 * @param raw pointer to struct raw_img
 */
static int jpeg_decompress(const char *file, struct raw_img *raw)
{
    FILE *jpeg_file;                            //Cria ponteiro File com o
nome do arquivo da imagem
    struct jpeg_decompress_struct jdec;
    struct jpeg_error_mgr jerr;
    JSAMPROW row_pointer[1];
    float r;
    float g;
    float b;
    int err;
    int i;
    int j;

    err = 0;
    j = 0;

    jpeg_file = fopen(file, "rb");              //Abre o arquivo de imagem
    if (jpeg_file == NULL) {
        printf("Error opening file %s\n!", file);
        err = -1;
        goto error;
    }

    jdec.err = jpeg_std_error(&jerr);
    jpeg_create_decompress(&jdec);
    jpeg_stdio_src(&jdec, jpeg_file);
    jpeg_read_header(&jdec, TRUE);
```

```
    raw->width = jdec.image_width;              //Obtem o valor da
quantidade de pixels horizontal
    raw->height = jdec.image_height;            //Obtem o valor da
quantidade de pixels vertical
    raw->num_comp = jdec.num_components;        //Obtem o numero de
componentes da imagem

    if (raw->num_comp != 3) {                   //Verifica se o numero de
componentes da imagem é compatível
        printf("Cannot convert a jpeg file with %d componentes per "
                "pixel\n", raw->num_comp);
        err = -1;
        goto error;
    }

    printf("JPEG File Information:\n");         //Exibe na tela as
informações da imagem
    printf("\tResolution: %d x %d pixels\n", raw->width, raw->height);
//Exibe na tela a dimensão da imagem
    printf("\tColor components per pixel: %d\n", raw->num_comp);
//Exibe na tela o numero de componentes da imagem

    jpeg_start_decompress(&jdec);               //Função de descompressão
da imagem .jpeg

    raw->img = (uint8_t *) malloc(raw->width * raw->height);
//Locação da memória para armazenar a imagem original em img
    row_pointer[0] = (unsigned char *) malloc(raw->width * raw->num_comp);
//Alocação da memória para o ponteiro das 3 componentes

    while (jdec.output_scanline < jdec.image_height) {
        jpeg_read_scanlines(&jdec, row_pointer, 1);
        for (i = 0; i < raw->width * raw->num_comp; i += 3) {
            r = (float) row_pointer[0][i];
            g = (float) row_pointer[0][i + 1];
            b = (float) row_pointer[0][i + 2];
            raw->img[j] = (uint8_t) (0.2989 * r + 0.5870 * g + 0.1140 * b);
// Conversão para Escala de Cinza
            j++;
        }
    }

    jpeg_finish_decompress(&jdec);              //Finaliza a descompressão
da imagem
    jpeg_destroy_decompress(&jdec);             //Libera a memoria jdec
    free(row_pointer[0]);                       //Libera a memoria
row_pointer
    fclose(jpeg_file);                          //Fecha arquivo da imagem

error:
    return err;
}

/**
 * jpeg_compress - compress a raw grayscale image to a jpeg file
 * @param file name of jpeg file
 * @param raw pointer to struct raw_img
 */
```

```
static int jpeg_compress(const char *file, struct raw_img *raw)
{
    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;
    JSAMPROW row_ptr[1];
    FILE* out_jpeg;
    uint8_t *image;
    int row_stride;
    int err;

    err = 0;
    image = raw->img;                           //Atribui a variavel image
a imagem original
    out_jpeg = fopen(file, "w+");               //Abre o arquivo para
armazenar a imagem
    if (out_jpeg == NULL) {
        printf("Could no open '%s' file\n", file);
        err = -1;
        goto error;
    }


    jpeg_create_compress(&cinfo);               //Cria o ponteiro para
compressão da imagem
    jpeg_stdio_dest(&cinfo, out_jpeg);          //Aponta o ponteiro do
arquivo da imagem

    cinfo.image_width = raw->width;             //Armazena a dimensão
horizontal da imagem
    cinfo.image_height = raw->height;           //Armazena a dimensão
vertical da imagem
    cinfo.input_components = IMAGE_NCHANNELS;   //Armazena os componentes
da imagem
    cinfo.in_color_space = JCS_GRAYSCALE;       //Seta parametro para
indicar as cores da imagem
    cinfo.err = jpeg_std_error(&jerr);          //Cria variavel de erro

    jpeg_set_defaults(&cinfo);                  //Seta os parametros
padrões de compressão da imagem
    jpeg_start_compress(&cinfo, TRUE);          //Inicia processo de
compressão
    row_stride = raw->width * IMAGE_NCHANNELS;  //Determina a dimensão
horizontal da imagem com o numero de componentes para o armazenamento da
matriz

    while (cinfo.next_scanline < cinfo.image_height) {      //Efetua a
montagem da imagem com os compontentes a cada linha da imagem
        row_ptr[0] = &image[cinfo.next_scanline * row_stride];
        jpeg_write_scanlines(&cinfo, row_ptr, 1);
    }

    jpeg_finish_compress(&cinfo);               //Finaliza o processo de
compressão
    jpeg_destroy_compress(&cinfo);              //Libera a memória do
ponteiro cinfo
    fclose(out_jpeg);                           //Fecha arquivo da imagem

error:
    return err;
}
```

```
/**
 * Preenchimento do vetor para passagem de parâmetro para o Matlab na
dimensão correta.
  * @param raw pointer to a struct raw_img with imgdim
 */

static void img_fill_dimen(struct raw_img *raw)
{
    int j;
    int k;
    int m;
    int n;

    j=0;
    m=0;

    raw->imgdim = (uint8_T *) malloc(RESOL_H*RESOL_W);

    while (j < RESOL_W*RESOL_H) {
            for (n=0; n < raw->width; n++){
            raw->imgdim[j] = (uint8_T)raw->img[m];
            j++;
            m++;
        }
        for (k=0; k < RESOL_W - raw->width; k++) {
                raw->imgdim[j] = 0;
                j++;
        }
        if (j == RESOL_W * raw->height){
            for (k=0; k < (RESOL_H - raw->height) * RESOL_W; k++) {
                raw->imgdim[j] = 0;
                j++;
            }
        }
    }
}


/**
 * Trocando a sequência do vetor da imagem de vertical para horizontal
 * Motivo: Matlab analisa o vetor por linha sequenciando pelo espaçamento
na vertical
 * @param raw pointer to a struct raw_img with imgswap
 */
static void img_swap_wtoh(struct raw_img *raw)
{
    int i;
    int j;
    int m;

    i=0;

    raw->imgswap = (uint8_T *) malloc(RESOL_H*RESOL_W);

    for (j=0 ; j < RESOL_W; j++) {
        for(m=0; m < RESOL_H; m++){
            raw->imgswap[i] = raw->imgdim[j + RESOL_W * m];
```

```
            i++;
        }
    }
}


/**
 * Ajuste da variavel fudgeFactor para detecção de borda
   dependente da variação de intensidade de pixels da imagem
 */
static void img_factor(struct raw_img *raw)
{
    int i;
    float major;
    float minor;

    major = 0;
    minor = 255;

    for (i=0; i<raw->width * raw->height; ++i){
        if (raw->img[i]>major) major=raw->img[i];
        if (raw->img[i]<minor) minor=raw->img[i];
    }

    printf("\tMaior Intensidade de Pixel: %d \n",(int) major);
    printf("\tMenor Intensidade de Pixel: %d \n",(int) minor);

    if (major / minor > 5) raw->factor = 0.35;
    else if (major / minor > 3) raw->factor = 0.30;
    else if (major / minor > 2) raw->factor = 0.25;
    else if (major / minor > 1) raw->factor = 0.20;
    else if (major / minor > 0.9) raw->factor = 0.15;
    else if (major / minor > 0.75) raw->factor = 0.10;
    else raw->factor = 0.10;
}

/**
 * Remoção da borda após detecção das bordas
 */

static void img_rem_resid(struct raw_img *raw)
{
    int k;
    int j;

    for (k=0; k < RESOL_W; k++){
        for (j=(raw->height-3)+RESOL_H*k; j < RESOL_H*(k+1); j++) {
            raw->imgsobel[j] = 0;
        }
    }

    for (j=(raw->width-3)*RESOL_H; j < RESOL_H*RESOL_H; j++) {
            raw->imgsobel[j] = 0;
    }
}

/**
 * Binarização da Imagem
 */
```

```
static void img_BW(struct raw_img *raw)
{
    int k;

    raw->imgBW = (boolean_T *) malloc(RESOL_H*RESOL_W);

    for (k=0; k < RESOL_H*RESOL_W; k++){
        if ((int8_T)raw->imgsobel[k] != 0){
            raw->imgBW[k] = 1;
        }
        else if ((int8_T)raw->imgsobel[k] == 0){
            raw->imgBW[k] = 0;
        }
    }
}

/** Função Principal **/

int main(int argc, char **argv)
{
    struct raw_img raw_img;
    int err;
    int i;
    int j;
    int k = 0;
    int Data[2];
    real_T size_r;

    real_T *NCrack;
    emxArray_real_T *CENTER;
    emxArray_real_T *AREA;
    emxArray_real_T *Deg;
    emxArray_real_T *MajorAxisLenght;
    emxArray_real_T *MinorAxisLength;

    struct timeval tv1,tv2;
    double ti=0.0;
    double tf=0.0;
    double ttotal=0.0;
    double t[13];

    int pos=0;
    int major=0;

    /** Parâmetros para o processamento da Imagem
************************************************************
    * @param fudgeFactor %Ratio parameter for threshold of magnetude
gradient edge (variation between 0.4 to 0.2)
    * @param SED Structuring element octagon 7 x 7 for Dilate Image
    * @param SEF Structuring element 3 x 3 for Fill Image
    * @param SEF Structuring element 3 x 3 for Erode Image */

    real_T fudgeFactor;
    real_T SED[49] = {0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
          1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0};
    real_T SEF[9] = {0, 1, 0, 1, 1, 1, 0, 1, 0};
    real_T SEE[9] = {0, 1, 0, 1, 1, 1, 0, 1, 0};
```

```
    /**
*************************************************************************
*******************************/
    err = 0;

    if (argc != 3) {
        printf("Usage: %s <Image In> <Image Out> \n", argv[0]);
        err = -1;
        goto error;
    }

    /** Passo 1 - Obtem a Imagem **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    err = jpeg_decompress(argv[1], &raw_img); //Retorna imagem em forma de
vetor atraves de um ponteiro img
    if (err == -1) goto error;
    size_r = (raw_img.width * raw_img.height);
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual
    printf("\tNúmero de Pixels da Imagem: %d \n",(int)size_r);

    /** Passo 2 - Preenche a imagem para dimensão de passagem entre as
funções **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    img_fill_dimen(&raw_img);       //Função que Preenche a imagem para
dimensão de passagem entre as funções
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 3 - Troca na sequencia do vetor da imagem de linhas para
colunas **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    img_swap_wtoh(&raw_img);        //Função que troca a sequencia de
linhas da imagem para colunas
    free(raw_img.imgdim);           //Libera memória que armazena a imagem
com ajuste de dimensão
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 4 - Seta valor de Limiar para Sobel**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
//  fudgeFactor=fudFac(raw_img.imgswap);      //função que obter o
parametro de limiar para sobel matlab
    img_factor(&raw_img);                       //Função que obtem o
parametro de limiar para sobel local
```

```
    fudgeFactor = raw_img.factor;               //Fator para Detecção de
Bordas pelo Operador SOBEL baseado na propoção de intensidade de pixels
    printf("\tFactor: %f \n",fudgeFactor);
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 5 - Detecção de Bordas usando o Operador Sobel**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    sobelEdge(raw_img.imgswap, fudgeFactor, sobel_img); //Detecção de
Bordas pelo Operador SOBEL (Gradientes horizontais e verticais)
    free(raw_img.imgswap);          //Libera memória que armazena a imagem
de troca
    raw_img.imgsobel = sobel_img;   //Armazena na estrutura raw_img o vetor
de imagem com borda
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    printf("\tProcessing...\n");

    /** Passo 6 - Remoção da bordas nos limites da dimensão da Imagem**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    img_rem_resid(&raw_img);        //Remove a bordas da dimensão da imagem
caso seja menordetecção das laterais da imagem menor
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual
    printf("\tRemove Detecção Laterais ok...\n");

    /** Passo 7 - Transforma a Imagem em Escala de Cinza para Binária**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    img_BW(&raw_img); //Transforma a imagem para Binária
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual
    printf("\tBinarização ok...\n");

    /** Passo 8 - Dilatação da Imagem**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    dilateimage(raw_img.imgBW,SED, dilatImage); // Dilatação da Imagem
    free(raw_img.imgBW);            //Libera memória que armazena a imagem
binaria
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
```

```
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual
    printf("\tDilatação ok...\n");

    /** Passo 9 - Preenchimento da Imagem**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    fillimage(dilatImage, SEF, fillHoles); // Preenchimento da Imagem
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual
    printf("\tPreenchimento ok...\n");

    /** Passo 10 - Erosão da Imagem**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    erodeimage(fillHoles, SEE, erodeImage); // Erosão da Imagem
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual
    printf("\tErosão ok...\n");

    /** Passo 11 - Divisão e Identificação dos Objetos**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    sepbwlabel(erodeImage, label); // Função para a Separação e
Identificação dos objetos da imagem
    for (i=0; i<RESOL_W*RESOL_H; ++i){
        if (label[i]>major) major= label[i];
    }
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    printf("\tNumero de objetos: %d \n",major);

    if(major > 1000) {
        printf("Numeros de Objetos Superior a 1000, qualidade de imagem
ruim!\n");
        goto error;
    }
    /** Passo 12 - Descrição e Eliminação dos objetos fora de padrão de
Fissura**/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    /* Locação de Memória Variáveis de Dados do Objetos*/
    NCrack = (real_T *) malloc(major);
    b_emxInit_real_T(&CENTER, 2);
```

```
    b_emxInit_real_T(&Deg, 2);
    b_emxInit_real_T(&AREA, 2);
    b_emxInit_real_T(&MajorAxisLenght, 2);
    b_emxInit_real_T(&MinorAxisLength, 2);

    defcrack(label, size_r, NCrack, CENTER, AREA, Deg, MajorAxisLenght,
MinorAxisLength, imcrack); // Função da Detecções de Fissura na imagem

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual


    /** Passo 13 **/    //Verifica a existência de dados para armazenamento
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução
    for (i=0; i < AREA->allocatedSize / AREA->numDimensions; i++ ){
        for (j=0; j < AREA->numDimensions; j++){
                Data[j] = AREA->data[i*AREA->numDimensions+j];
            if (Data[j]!=0){
                k++;
            }
        }
    }

//    if (k > 0){
        /** Armazena os dados e imagem caso exista Fissura **/
        jpeg_compress(argv[2], &raw_img);
        free(raw_img.img);                  //Libera memória que armazena a
imagem original
        FILE *fp1;                          //Cria ponteiro para File
        fp1 = fopen("Dados.dat","a+");       //Abre arquivo Dados.dat
        fprintf(fp1,"%s\t", argv[1]);       //Escreve o Nome da Imagem
atual
        fprintf(fp1,"%d\t",(int)size_r);    //Escreve o Tamanho da Imagem
        fprintf(fp1,"%d\t",major);          //Escreve o numero de objetos
detectados
        fprintf(fp1,"%d\t", k);             //Escreve a quantidade de
indicações de fissura

        for (i=0; i < k*2; i++ ) {
            for (j=0; j < CENTER->numDimensions; j++){
                Data[j] = CENTER->data[i*CENTER->numDimensions+j];
//Obtem as coordenadas de indicação da Fissura na Imagem
            }
            if (Data[0]!=0.0 || Data[1]!=0.0){
                printf("%dº Centers %d \t%d\n", i+1, Data[0], Data[1]);
//Exibe na tela as coordenadas de indicação da Fissuraa na Imagem
                fprintf(fp1,"%d\t%d\t", Data[0], Data[1]);
//Armazena a quantidade de indicações de fissura
            }
        }
        for (i=0; i < k; i++ ) {
            for (j=0; j < AREA->numDimensions; j++){
                Data[j] = AREA->data[i*AREA->numDimensions+j];      //Obtem
a area em pixels da Fissura na Imagem
                if (Data[j]!=0){
```

```
                    printf("%dº Crack Area:  \t%d\n", j+1, Data[j]);//Exibe
na tela a area em pixels da Fissura na Imagem
                    fprintf(fp1,"%d\t", Data[j]);
//Armazena a area em pixels da Fissura na Imagem
                    k++;
                }
            }
        }
        for (i=0; i < k; i++ ){
            for (j=0; j < k; j++){
                Data[j] = Deg->data[i*Deg->numDimensions+j];        //Obtem
a orientação do eixo maior da Fissura na Imagem
                if (Data[j]!=0){
                    printf("%dº Crack Deg:  \t%d\n", j+1, Data[j]); //Exibe
na tela a orientação do eixo maior da Fissura na Imagem
                    fprintf(fp1,"%d\t", Data[j]);
//Armazena a orientação do eixo maior da Fissura na Imagem
                }
            }
        }
        for (i=0; i < k; i++ ){
            for (j=0; j < MajorAxisLenght->numDimensions; j++){
                Data[j] = MajorAxisLenght->data[i*MajorAxisLenght-
>numDimensions+j];     //Obtem o eixo maior da Fissura na Imagem
                if (Data[j]!=0){
                    printf("%dº Crack Comprimento:  \t%d\n", j+1, Data[j]);
//Exibe na tela o eixo maior da Fissura na Imagem
                    fprintf(fp1,"%d\t", Data[j]);
//Armazena o eixo maior da Fissura na Imagem
                }
            }
        }
        for (i=0; i < k; i++ ){
            for (j=0; j < MinorAxisLength->numDimensions; j++){
                Data[j] = MinorAxisLength->data[i*MinorAxisLength-
>numDimensions+j];//Obtem a abertura da Fissura na Imagem
                if (Data[j]!=0){
                    printf("%dº Maior Fenda:  \t%d\n", j+1, Data[j]);
//Exibe na tela a abertura da Fissura na Imagem
                    fprintf(fp1,"%d\t", Data[j]);
//Armazena a abertura da Fissura na Imagem
                }
            }
        }
        fprintf(fp1,"\n");
        int fclose(FILE *fp1);          //Fecha o arquivo Dados.txt

        gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
        ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
        tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
        pos++;
        t[pos] = (tf-ti);     //Armazena variável de tempo para o passo
atual
 //   }

    /** Armazenagem dos tempos de execução **/
    FILE *fp2;
    fp2 = fopen("tempos.dat","w");
```

```
    fprintf(fp2,"Tempos de execução das Funções de Detecção de Fissura da
Imagem: %s\n", argv[1]);
    for (i=0; i < pos+1; i++){
        ttotal = ttotal + t[i];
        fprintf(fp2,"Tempo de execução do passo %d \t%f segundos\n", i+1,
t[i]);
    }
    fprintf(fp2,"Tempo de total de execução \t%f segundos\n", ttotal);
    int fclose(FILE *fp2);

    FILE *fp3;
    fp3 = fopen("tempos_corridos.dat","a+");
    fprintf(fp3,"%s\t", argv[1]);
    fprintf(fp3,"%d\t",(int)size_r);
    fprintf(fp3,"%d\t",major);
    fprintf(fp3,"%d\t", k);
    for (i=0; i < pos+1; i++){
        fprintf(fp3,"%f\t", t[i]);
    }
    fprintf(fp3,"%f\t", ttotal);
    fprintf(fp3,"\n");

    int fclose(FILE *fp3);

    printf("\tConcuído\n");

    return 0;

error:
    return err;
}
```

A.4 PROGRAMA PRINCIPAL EM C – ALGORITMO B

```
Programa Principal :
dtcrackfiltro.c
Algoritmo baseado em Filtro de Particulas para detecção de Fissuras
Data: 14/10/2014
*************************************************************************
*****************************/

#include <stdio.h>
#include <math.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <sys/time.h>
#include <jpeglib.h>

#include "calc_log_likelihood.h"
#include "create_particles.h"
#include "resample_particles.h"
#include "update_particles.h"
#include "parfiltercrack_emxAPI.h"
#include "parfiltercrack_emxutil.h"
```

```
#define IMAGE_NCHANNELS 3
#define RESOL_W 1024
#define RESOL_H 768

/* struct to store a grayscale raw image */
struct raw_img {
    uint8_t num_comp;
    uint32_t width;
    uint32_t height;
    uint8_t *img;
    uint8_T *imgdim;
    uint8_T *imgswap;
    uint8_t *imgnorm;
};

/**
 * jpeg_decompress - open a jpeg image, decompress it and convert
 * to a raw grayscale image
 * @param file name of jpeg file
 * @param raw pointer to struct raw_img
 */
static int jpeg_decompress(const char *file, struct raw_img *raw)
{
    FILE *jpeg_file;
    struct jpeg_decompress_struct jdec;
    struct jpeg_error_mgr jerr;
    JSAMPROW row_pointer[1];
    float r;
    float g;
    float b;
    int err;
    int i;
    int j;
    int k;
    int z;

    err = 0;
    j = 0;
    z = 0;

    jpeg_file = fopen(file, "rb");
    if (jpeg_file == NULL) {
        printf("Error opening file %s\n!", file);
        err = -1;
        goto error;
    }

    jdec.err = jpeg_std_error(&jerr);
    jpeg_create_decompress(&jdec);
    jpeg_stdio_src(&jdec, jpeg_file);
    jpeg_read_header(&jdec, TRUE);

    raw->width = jdec.image_width;
    raw->height = jdec.image_height;
    raw->num_comp = jdec.num_components;

    if (raw->num_comp != 3) {
        printf("Cannot convert a jpeg file with %d componentes per "
                "pixel\n", raw->num_comp);
        err = -1;
```

```
        goto error;
    }

    printf("JPEG File Information:\n");
    printf("\tResolution: %d x %d pixels\n", raw->width, raw->height);
    printf("\tColor components per pixel: %d\n", raw->num_comp);

    jpeg_start_decompress(&jdec);

    raw->img = (uint8_t *) malloc(raw->width * raw->height);
    raw->imgnorm = (uint8_t *) malloc(raw->width * raw->height * raw-
>num_comp);
    row_pointer[0] = (unsigned char *) malloc(raw->width * raw->num_comp);

    while (jdec.output_scanline < jdec.image_height) {
        jpeg_read_scanlines(&jdec, row_pointer, 1);
        for (k = 0; k < raw->width * raw->num_comp ; k++){
            raw->imgnorm[z] = row_pointer[0][k];
            z++;
        }
        for (i = 0; i < raw->width * raw->num_comp; i += 3) {
            r = (float) row_pointer[0][i];
            g = (float) row_pointer[0][i + 1];
            b = (float) row_pointer[0][i + 2];
            /* here convert to grayscale */
            raw->img[j] = (uint8_t) (0.2989 * r + 0.5870 * g + 0.1140 * b);
            j++;
        }
    }

    jpeg_finish_decompress(&jdec);
    jpeg_destroy_decompress(&jdec);
    free(row_pointer[0]);
    fclose(jpeg_file);

error:
    return err;
}


/**
 * jpeg_compress - compress a raw grayscale image to a jpeg file
 * @param file name of jpeg file
 * @param raw pointer to struct raw_img
 */
static int jpeg_compress(const char *file, struct raw_img *raw)
{
    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;
    JSAMPROW row_ptr[1];
    FILE* out_jpeg;
    uint8_t *image;
    int row_stride;
    int err;

    err = 0;
    image = raw->imgnorm;
    out_jpeg = fopen(file, "w+");
    if (out_jpeg == NULL) {
```

```
        printf("Could no open '%s' file\n", file);
        err = -1;
        goto error;
    }


    jpeg_create_compress(&cinfo);
    jpeg_stdio_dest(&cinfo, out_jpeg);

    cinfo.image_width = raw->width;
    cinfo.image_height = raw->height;
    cinfo.input_components = IMAGE_NCHANNELS;
    cinfo.in_color_space = JCS_RGB;
    cinfo.err = jpeg_std_error(&jerr);

    jpeg_set_defaults(&cinfo);
    jpeg_start_compress(&cinfo, TRUE);
    row_stride = raw->width * IMAGE_NCHANNELS;

    while (cinfo.next_scanline < cinfo.image_height) {
        row_ptr[0] = &image[cinfo.next_scanline * row_stride];
        jpeg_write_scanlines(&cinfo, row_ptr, 1);
    }

    jpeg_finish_compress(&cinfo);
    jpeg_destroy_compress(&cinfo);
    fclose(out_jpeg);


error:
    return err;
}

/**
 * Preenchimento do vetor para passagem de parâmetro para o Matlab na
resolução correta.
  * @param raw pointer to a struct raw_img with imgdim
 */

static void img_fill_dimen(struct raw_img *raw)
{
    int j;
    int k;
    int m;
    int n;

    j=0;
    m=0;

    raw->imgdim = (uint8_T *) malloc(RESOL_H*RESOL_W);

    while (j < RESOL_W*RESOL_H) {
            for (n=0; n < raw->width; n++){
            raw->imgdim[j] = (uint8_T)raw->img[m];
            j++;
            m++;
        }
        for (k=0; k < RESOL_W - raw->width; k++) {
                raw->imgdim[j] = 255;
```

```
                j++;
            }
            if (j == RESOL_W * raw->height){
                for (k=0; k < (RESOL_H - raw->height) * RESOL_W; k++) {
                    raw->imgdim[j] = 255;
                    j++;
                }
            }
        }
}


/**
 * Trocando a sequência do vetor da imagem da montagem de altura para
largura
 * Motivo: Matlab analisa o vetor por linha sequenciando pelo espaçamento
da altura
 * @param raw pointer to a struct raw_img with imgswap
 */
static void img_swap_wtoh(struct raw_img *raw)
{
    int i;
    int j;
    int m;

    i=0;

    raw->imgswap = (uint8_T *) malloc(RESOL_H*RESOL_W);
    for (j=0 ; j < RESOL_W; j++) {
        for(m=0; m < RESOL_H; m++){
            raw->imgswap[i] = raw->imgdim[j + RESOL_W * m];
            i++;
        }
    }
}


int main(int argc, char **argv)
{
    struct raw_img raw_img;
    int err;
    int i;
    int j;
    int k;
    int m;
    int flag1;
    int flag2;
    real_T Data[40000];

    struct timeval tv1,tv2;
    double ti=0.0;
    double tf=0.0;
    double ttotal=0.0;
    double t[20];
    int pos=0;

    real_T Npop_particles;
    real_T Xstd_rgb;
    real_T Xstd_pos;
    real_T Xstd_vec;
```

```
    real_T Xrgb_trgt;

    real_T Npix_resolution[3];
    emxArray_real_T *particles;
    real_T newparticles[40000];
    real_T Likelihood[10000];
    real_T Positions[40000];
    real_T dataparticles[40000];

    /** Parâmetros para o processamento da Imagem
*****************************************************************
    * @param F_update
    * @param Npop_particles Numero de particulas para update
    * @param Xstd_rgb Variação do alvo RGB
    * @param Xstd_pos Variação da posição do alvo
    * @param Xstd_vec Vetor de distancia do alvo
    * @param Xrgb_trgt  Cor alvo do objeto **/

    real_T F_update[16] = {1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 0, 0, 1};
    Npop_particles = 10000;

    Xstd_rgb = 5;
    Xstd_pos = 50;
    Xstd_vec = 50;
    Xrgb_trgt = 0;
    /**
*************************************************************************
*******************************/
    err = 0;

    if (argc != 3) {
        printf("Usage: %s <Image In> <Image Out> \n", argv[0]);
        err = -1;
        goto error;
    }

    /** Passo 1 - Obtem a Imagem **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    err = jpeg_decompress(argv[1], &raw_img); //Retorna imagem em forma de
vetor atraves de um ponteiro img
    if (err == -1) goto error;

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    printf("\tProcessing....\n");

    /** Passo 2 - Preenche a imagem para dimensão de passagem entre as
funções **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    img_fill_dimen(&raw_img); //Preenche a imagem para resolução de
passagem entre as funções
```

```
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 3 - Troca na sequencia do vetor da imagem de linhas para
colunas **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    img_swap_wtoh(&raw_img); //Passagem das linhas da imagem para colunas
    free(raw_img.imgdim);    //Libera memória que armazena a imagem com
ajuste de dimensão

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 4 - Cria os Vetores de Particulas **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    Npix_resolution[1] = raw_img.width;
    Npix_resolution[0] = raw_img.height;
    Npix_resolution[2] = (real_T) raw_img.num_comp;

    emxInit_real_T(&particles, 2);
    create_particles(Npix_resolution,  Npop_particles, particles);
//Rastreamento do Objeto pelo Filtro de Partículas

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 5 - Atualiza as Particulas **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    for (i=0; i < 4*Npop_particles; i++){
        dataparticles[i] = particles->data[particles->numDimensions+i];
    }

    update_particles(F_update, Xstd_pos, Xstd_vec, dataparticles,
newparticles);  // Atualização da Particulas para previsão

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
```

```
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 6 - Calculo da Probabilidade **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    calc_log_likelihood(Xstd_rgb, Xrgb_trgt, newparticles, raw_img.imgswap,
Likelihood);  //Calculo de probabilidade

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 7 - Reamostragem da Particula **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

    resample_particles(newparticles, Likelihood, Positions);  //
Reamostragem

    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Passo 8 - ** Verifica as coordenadas das particulas existentes e
armazena **/
    gettimeofday(&tv1, NULL);       //Obtem o tempo atual de execução

                FILE *fp1;
                fp1 = fopen("Dados.dat","a+");
                fprintf(fp1,"%s\t", argv[1]);
                fprintf(fp1,"%d\t",(int) raw_img.width*raw_img.height);

    k=0;
    j=0;
    for (i=0; i < 4*Npop_particles; i+=4 )
    {
        if (Positions[i+2]<raw_img.width && Positions[i+3]<raw_img.height
&& Positions[i+2]>0 && Positions[i+3]>0)
        {
            if (k==0)
            {
                Data[j] = Positions[i+2];
                Data[j+1] = Positions[i+3];
                j+=2;
                k++;
                printf("%dº Centers %f \t%f\n", k, Positions[i+2],
Positions[i+3]);
                fprintf(fp1,"%d\t%d\t",(int)Positions[i+2],
(int)Positions[i+3]);
            }
            else
            {
```

```
                flag1 = 0;
                flag2 = 0;
                for (m=j; m>=0; m-=2)
                {
                    if (Data[m] != Positions[i+2]) flag1=1;
                    if (Data[m] == Positions[i+2]) flag2=1;
                }
                if (flag1 == 1 && flag2 == 0)
                {
                    Data[j] = Positions[i+2];
                    Data[j+1] = Positions[i+3];
                    j+=2;
                    k++;
                    printf("%dº Centers %f \t%f\n", k, Positions[i+2],
Positions[i+3]);
                    fprintf(fp1,"%d\t%d\t",(int)Positions[i+2],
(int)Positions[i+3]);
                }
            }

        }
    }

            fprintf(fp1,"%d\n",k);
        int fclose(FILE *fp1);


    //if (k>0){
        jpeg_compress(argv[2], &raw_img);
//    }
    free(raw_img.img);          //Libera memória que armazena a imagem
original
    gettimeofday(&tv2, NULL);       //Obtem o tempo atual de execução
    ti = (double)(tv1.tv_sec) + (double)(tv1.tv_usec)/ 1000000.00;
//Tratamento do Tempo Inicial segundos e microsegundos
    tf = (double)(tv2.tv_sec) + (double)(tv2.tv_usec)/ 1000000.00;
//Tratamento do Tempo Final segundos e microsegundos
    pos++;
    t[pos] = (tf-ti);     //Armazena variável de tempo para o passo atual

    /** Armazena os tempos de processamento **/
    FILE *fp2;
    fp2 = fopen("tempos.dat","w");
    for (i=0; i < pos+1; i++){
        ttotal = ttotal + t[i];
        fprintf(fp2,"Tempo de execução da passo %d \t%f segundos\n", i+1,
t[i]);
    }
    fprintf(fp2,"Tempo de total de execução \t%f segundos\n", ttotal);
    int fclose(FILE *fp2);

    FILE *fp3;
    fp3 = fopen("tempos_corridos.dat","a+");
    fprintf(fp3,"%s\t", argv[1]);
    fprintf(fp3,"%d\t",(int) raw_img.width*raw_img.height);
    for (i=0; i < pos+1; i++){
        fprintf(fp3,"%f\t", t[i]);
    }
    fprintf(fp3,"%f\t", ttotal);
    printf("Numero de Objetos detectados\t%d\n", k);
```

```
    fprintf(fp3,"%d\n",k);
    int fclose(FILE *fp3);

    printf("\tConcuído\n");

    return 0;

error:
    return err;
}
```

A.5 MAKEFILE DA COMPILAÇÃO ALGORITIMO A

```
CFLAGS=-O2 -Wall
CINCLUDE=-I./include -I /usr/src/jpeg-6b/
CLIBDIR=-L.. -L /usr/src/jpeg-6b/
CLIBS=-l jpeg

CPPFLAGS=
CPPINCLUDE=-I./include/cpp
CPPLIBDIR=-L ./lib
CPPLIBS=-l sfunc -lm

INCLUDE=${CINCLUDE} ${CPPINCLUDE}
FLAGS= ${CFLAGS} ${CPPFLAGS} -DDEBUG
LIBDIR=${CLIBDIR} ${CPPLIBDIR}
LIBS=${CLIBS} ${CPPLIBS}

CMP=gcc
CMPFLAGS=${FLAGS} ${INCLUDE}
LDFLAGS=${LIBDIR} ${LIBS}

all: cleanant prepara libsfunc.a dtcrackdesc clean

prepara:
    sudo mkdir -p ./include/cpp     #Cria a pasta include/cpp
    sudo mkdir -p ./lib     #Cria a pasta lib
    sudo cp *.h ./include/cpp       #Copia os headers do diretorio atual
    sudo mkdir -p ./ImagemProcess/

dtcrackdesc: dtcrackdesc.c
    ${CMP} ${CMPFLAGS} -o $@ $^ ${LDFLAGS}
#${DMAGICK}

libsfunc.a: sobelEdge.o defcrack.o dilateimage.o erodeimage.o fillimage.o
sepbwlabel.o defcrack_emxAPI.o defcrack_emxutil.o defcrack_initialize.o
defcrack_terminate.o rt_nonfinite.o rtGetInf.o rtGetNaN.o fudFac.o
    ar rv $@ $^ #Cria Biblioteca EstÃ¡tica
    mv $@ ./lib #Move o Biblioteca EstÃ¡tica para a pasta lib

fudFac.o: fudFac.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

sobelEdge.o: sobelEdge.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@
```

```
defcrack.o: defcrack.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

dilateimage.o: dilateimage.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

erodeimage.o: erodeimage.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

fillimage.o: fillimage.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

sepbwlabel.o: sepbwlabel.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

defcrack_emxAPI.o: defcrack_emxAPI.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

defcrack_emxutil.o: defcrack_emxutil.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

defcrack_initialize.o: defcrack_initialize.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

defcrack_terminate.o: defcrack_terminate.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

rt_nonfinite.o: rt_nonfinite.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

rtGetInf.o: rtGetInf.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

rtGetNaN.o: rtGetNaN.c
	${CMP} ${CMPFLAGS} -c $^ -o $@

cleanant:
	sudo rm -f -v dtcrackdesc
	sudo rm -f -r -v ./include/cpp      #Limpa a pasta include/cpp
	sudo rm -f -r -v ./lib      #Limpa a pasta lib
#   rm -r ./ImagemProcess/      #Limpa as imagens processadas

clean:
	rm -f  *~ *.bak *.o *.jpeg *.dat

distclean: clean
	rm -f tempos.dat

install:
```

A.6 MAKEFILE DA COMPILAÇÃO ALGORITIMO B

```
CFLAGS=-O2 -Wall
```

```
CINCLUDE=-I./include -I /usr/src/jpeg-6b/
CLIBDIR=-L.. -L /usr/src/jpeg-6b/
CLIBS=-l jpeg

CPPFLAGS=
CPPINCLUDE=-I./include/cpp
CPPLIBDIR=-L ./lib
CPPLIBS=-l sfunc -lm

INCLUDE=${CINCLUDE} ${CPPINCLUDE}
FLAGS= ${CFLAGS} ${CPPFLAGS} -DDEBUG
LIBDIR=${CLIBDIR} ${CPPLIBDIR}
LIBS=${CLIBS} ${CPPLIBS}

CMP=gcc
CMPFLAGS=${FLAGS} ${INCLUDE}
LDFLAGS=${LIBDIR} ${LIBS}

all: cleanant prepara libsfunc.a dtcrackfiltro clean


prepara:
    sudo mkdir -p ./include/cpp     #Cria a pasta include/cpp
    sudo mkdir -p ./lib     #Cria a pasta lib
    sudo cp *.h ./include/cpp       #Copia os headers do diretorio atual
    sudo mkdir -p ./ImagemProcess/

dtcrackfiltro: dtcrackfiltro.c
    ${CMP} ${CMPFLAGS} -o $@ $^ ${LDFLAGS}

libsfunc.a: create_particles.o update_particles.o calc_log_likelihood.o
resample_particles.o parfiltercrack_data.o parfiltercrack_emxAPI.o
parfiltercrack_emxutil.o parfiltercrack_initialize.o
parfiltercrack_terminate.o permute.o rand.o randn.o rt_nonfinite.o
rtGetInf.o rtGetNaN.o cumsum.o exp.o histc.o round.o sum.o
    ar rv $@ $^ #Cria Biblioteca EstÃ¡tica
    mv $@ ./lib #Move o Biblioteca EstÃ¡tica para a pasta lib

create_particles.o: create_particles.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

update_particles.o: update_particles.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

calc_log_likelihood.o: calc_log_likelihood.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

resample_particles.o: resample_particles.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

parfiltercrack_data.o: parfiltercrack_data.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

parfiltercrack_emxAPI.o: parfiltercrack_emxAPI.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

parfiltercrack_emxutil.o: parfiltercrack_emxutil.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@
```

```
parfiltercrack_initialize.o: parfiltercrack_initialize.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

parfiltercrack_terminate.o: parfiltercrack_terminate.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

permute.o: permute.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

rand.o: rand.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

randn.o: randn.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

rt_nonfinite.o: rt_nonfinite.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

rtGetInf.o: rtGetInf.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

rtGetNaN.o: rtGetNaN.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

cumsum.o: cumsum.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

exp.o: exp.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

histc.o: histc.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

round.o: round.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

sum.o: sum.c
    ${CMP} ${CMPFLAGS} -c $^ -o $@

cleanant:
    rm -f dtcrackfiltro
    rm -f -r -v ./include/cpp       #Limpa a pasta include/cpp
    rm -f -r -v ./lib       #Limpa a pasta lib
    rm -r ./ImagemProcess/      #Limpa as imagens processadas

clean:
    rm -f  *~ *.bak *.o *.jpeg *.dat

distclean: clean
    rm -f tempos.dat
install:
```

A.7 Relação de Amostras de Fissuras em FAchadas com Revestimento de Argamassa


Amostra 1.1


Amostra 1.2


Amostra 1.3


Amostra 1.4


Amostra 1.5

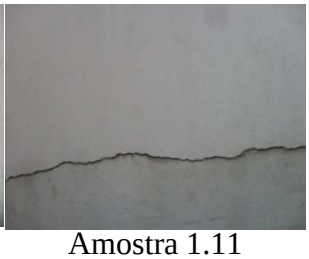

Amostra 1.6


Amostra 1.7


Amostra 1.8


Amostra 1.9


Amostra 1.10


Amostra 1.11


Amostra 1.12


Amostra 1.13


Amostra 1.14


Amostra 1.15


Amostra 1.16


Amostra 1.17


Amostra 1.18


Amostra 1.19


Amostra 1.20


Amostra 1.21


Amostra 1.22


Amostra 1.23


Amostra 1.24

Amostra 1.25 Amostra 1.26 Amostra 1.27 Amostra 1.28

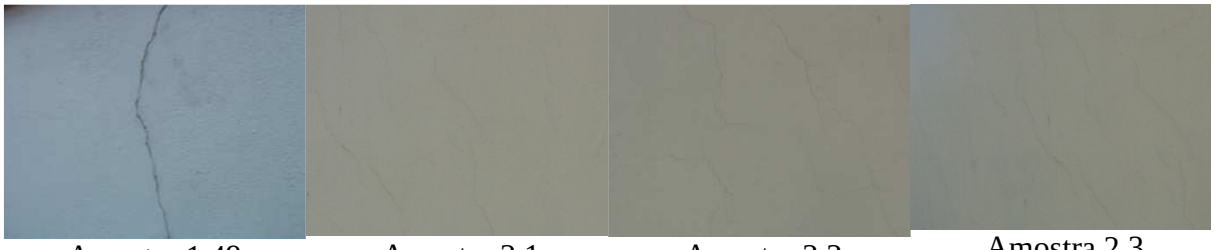
Amostra 1.29 Amostra 1.30 Amostra 1.31 Amostra 1.32

Amostra 1.33 Amostra 1.34 Amostra 1.35 Amostra 1.36

Amostra 1.37 Amostra 1.38 Amostra 1.39 Amostra 1.40

Amostra 1.41 Amostra 1.42 Amostra 1.43 Amostra 1.44

Amostra 1.45 Amostra 1.46 Amostra 1.47 Amostra 1.48



Amostra 1.49 Amostra 2.1 Amostra 2.2 Amostra 2.3


Amostra 2.4


Amostra 2.5


Amostra 2.6


Amostra 2.7

ANEXO: MODELOS DE CORES

Considerações Iniciais

Segundo (PEDRINI H., SCHWARTZ W.R., 2008) os modelos ou espaços de cores permitem a especificação de cores em um formato padronizado para atender a diferentes dispositivos gráficos ou aplicações que requerem a manipulação de cores. Um modelo de cor é essencialmente uma representação tridimensional na qual cada cor é especificada por m ponto no sistema de coordenadas tridimensionais. Os modelos de cores podem ser aditivos ou subtrativos. Nos modelos aditivos, a cor é gerada pela combinação de vários comprimentos de onda luminosa. A Cor branca é grada pela adição das cores primárias verde, vermelha e azul. A cor preta indica que nenhuma luz está sendo transmitida. Exemplos de modelos aditivos incluem XYZ, RGB, HSV e HLS e nos modelos subtrativos incluem CMY e CMYK. A figura A.1 ilustra a combinação de cores nos modelos aditivos e subtrativos.

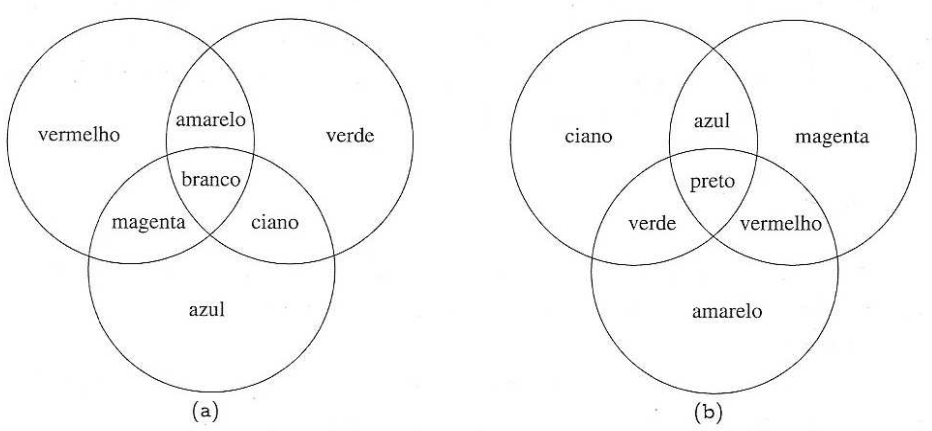


Figura A1 – Modelos de cores. (a) aditivo; (b) subtrativo.

A seguir alguns modelos mais utilizados para representação de cores.

Modelos de Cores

Modelo RGB

Modelo de cores é baseado em um sistema de coordenadas cartesianas, em que o espaço de cores é um cubo, como mostrado na figura A.2. As cores primárias, vermelha (R, red), verde (G, Green) e azul (B, blue) estão em três vértices no cubo, as cores primárias complementares, ciano, magenta e amarelo estão em outros três vértices, o vértice junto à origem é o preto e o mais afastado da origem corresponde à cor branca.

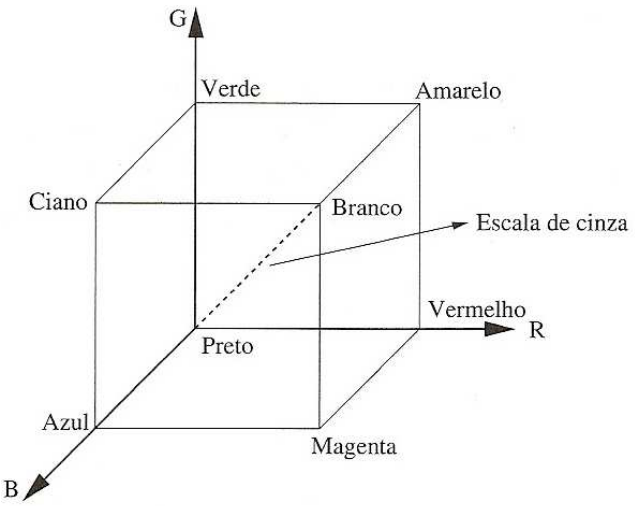


Figura A2 – Modelo RGB.

No modelo RGB, a escala de cinza se estende através da diagonal do cubo, ou seja, a reta que une a origem (preto) até o vértice mais distante (branco). Por conveniência, geralmente assume-se que valores R, G e B estão normalizados entre 0 e 1. O modelo RGB é muito utilizado em dispositivos como monitores e câmeras de vídeo.

Modelo YIQ

Neste modelo, o componente Y corresponde à luminância e os componentes I (matiz) e Q (saturação) juntos codificam as informações de crominância.

A conversão do modelo RGB para YIQ é definida como

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.523 & 0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{(A.1)}$$

em que $0 \leq R,G,B \leq 1$. A soma dos elementos da primeira linha da matriz é igual a 1, enquanto a soma das duas outras linhas é igual a 0. Assim, para uma imagem tom de cinza, em que todos os componentes R, G e B são iguais, os componentes I e Q são 0.

O modelo YIQ é utilizado para transmissão de sinal de televisão a cores. O uso do modelo YIQ possui a vantagem de que o sinal de luminância Y pode ser utilizado diretamente pelos aparelhos de televisão em preto e branco, mantendo a compatibilidade entre sistemas de televisão colorida e em preto e branco. O modelo YIQ é utilizado pelo padrão americano NTSC (*National Television System Committee*). (PEDRINI H., SCHWARTZ W.R., 2008).

Modelo YUV

O modelo YUV é utilizado para representar cores nos padrões de televisão PAL (do inglês, Phase Alternation by Line) e SECAM (do francês, Séquentiel Couleur à Mémoire). O componente Y correspondente à luminância e os componentes U e V codificam as informações de crominância. (PEDRINI H., SCHWARTZ W.R., 2008).

A transformação do sistema RGB para YUV é dada por

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{(A.2)}$$

em que $0 \leq R,G,B \leq 1$.

Modelo YCbCr

O modelo YCbCr é largamente utilizada em vídeos digitais. Neste modelo, a informação de luminância é representada pelo componente Y, enquanto a informação de cor é armazenada nos componentes Cb e Cr. O componente Cb é a diferença entre a cor azul e uma valor de referência, o componente Cr é a diferença entre a cor vermelha e um valor de referência. (PEDRINI H., SCHWARTZ W.R., 2008).

A conversão do sistema RGB para YCbCr é dada por

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.147 & -0.289 & 0.436 \\ 0.615 & -0.515 & -0.100 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \quad \text{(A.3)}$$

Modelo HSV

O modelo HSV é definido pelos parâmetros matiz(H, hue), saturação (S, saturation) e luminância (V, value). A conversão do modelo RGB para o modelo HSV pode ser realizada por meio das seguintes equações:

$$H = \begin{cases} 60\dfrac{(G - B)}{(M - m)}, & \text{se } M = R \\ 60\dfrac{(B - R)}{(M - m)} + 120, & \text{se } M = G \\ 60\dfrac{(R - G)}{(M - m)} + 240, & \text{se } M = B \end{cases}$$

$$S = \begin{cases} \dfrac{(M - m)}{M}, & \text{se } M \neq 0 \\ 0, & \text{caso contrário} \end{cases} \quad \text{(A.4)}$$

$$V = M$$

em que m = min(R, G, B) e M = Max(R,G, B). A luminância V e a saturação S estão normalizadas entre 0 e 1. O matiz H varia entre 0 e 360 graus. Pode-se observar a partir da equação A.4 que se a saturação S for igual a 0, então o matiz H é definida, ou seja, a cor do ponto situa-se ao longo da escala de cinzas. Se o valor V for igual a 0, ou seja, M=0, então a saturação S é indefinida. (PEDRINI H., SCHWARTZ W.R., 2008).