

**UNIVERSIDADE FEDERAL DE PELOTAS**  
Centro de Desenvolvimento Tecnológico  
Curso de Bacharelado em Ciência da Computação



Trabalho de Conclusão de Curso

**Estudo e Aplicação de Métodos de Segmentação de Imagens para o  
Diagnóstico de Falhas na Fabricação de Equipos**

**Leandro Soares Guedes**

Pelotas, 2014

**Leandro Soares Guedes**

**Estudo e Aplicação de Métodos de Segmentação de Imagens para o Diagnóstico de Falhas na Fabricação de Equipos**

Trabalho de Conclusão de Curso apresentado ao Centro de Desenvolvimento Tecnológico da Universidade Federal de Pelotas, como requisito parcial à obtenção do título de Bacharel em Ciência da Computação

Orientador: Prof. Dr. Marilton Sanhotene de Aguiar  
Co-orientador: Prof. MSc. Alexandre Solon Nery

Pelotas, 2014

Universidade Federal de Pelotas / Sistema de Bibliotecas  
Catalogação na Publicação

G924e Guedes, Leandro Soares

Estudo e aplicação de métodos de segmentação de imagens para o diagnóstico de falhas na fabricação de equipos / Leandro Soares Guedes ; Marilton Sanchotene de Aguiar, orientador ; Alexandre Solon Nery, coorientador. — Pelotas, 2014.

78 f. : il.

Trabalho de Conclusão de Curso (Graduação em Ciência da Computação) — Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, 2014.

1. Processamento de imagens. 2. Segmentação de imagens. 3. Detecção de objetos. 4. Automação industrial. 5. Equipos médicos. I. Aguiar, Marilton Sanchotene de, orient. II. Nery, Alexandre Solon, coorient. III. Título.

CDD : 006.6

Leandro Soares Guedes

**Estudo e Aplicação de Métodos de Segmentação de Imagens para o Diagnóstico de Falhas na Fabricação de Equipos**

Trabalho de Conclusão de Curso aprovado, como requisito parcial, para obtenção do grau de Bacharel em Ciência da Computação, Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas.

Data da Defesa: 06 de Fevereiro de 2014

Banca Examinadora:



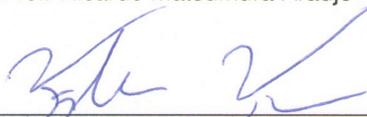
Prof. Marilton Sanchotene de Aguiar – Orientador



Prof. Paulo Roberto Ferreira Jr



Prof. Ricardo Matsumura Araujo



Fabrício Neitzke Ferrreira

**Dedico este trabalho aos meus pais, Nara e Gilmar Guedes.  
Obrigado por todo o apoio e carinho.**

## **AGRADECIMENTOS**

Quero agradecer a todos que contribuíram de alguma forma para que este projeto e o curso ao todo obtivessem êxito. Primeiramente, agradeço ao orientador Marilton e ao Co-orientador Alexandre por todo o auxílio prestado; aos meus pais e todos os familiares que me apoiaram durante todo o curso; aos meus amigos de Palmeira das Missões, que compartilham comigo inúmeros momentos memoráveis durante toda minha trajetória até o final da graduação: Lucas, Roberta, Rodrigo e Vanessa; aos meus colegas e amigos da universidade que, sem dúvida, fizeram parte dessa história, entre eles: Eduardo, Gustavo, Heitor, John, Luan, Lucas, Luis, Marlon, Renata, Ricardo, Thaieni, Vinícius e Willian; e, também não posso deixar de citar, meus amigos do intercâmbio, que fizeram minha vida em Portugal muito melhor, entre eles: Bruno, Isabele, Juliana, Pedro, Raquel e Thales. A todos vocês citados diretamente ou indiretamente, aqui fica o meu muito obrigado!

**A coisa mais indispensável a um homem é reconhecer o uso que deve  
fazer do seu próprio conhecimento.**

— PLATÃO

## RESUMO

GUEDES, Leandro Soares. **Estudo e Aplicação de Métodos de Segmentação de Imagens para o Diagnóstico de Falhas na Fabricação de Equipos**. 2014. 78 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2014.

O uso de técnicas antigas que não utilizam da tecnologia para a produção, acabam aumentando o custo e tempo de fabricação de produtos, entre eles, estão os equipos utilizados na infusão intravenosa.

Para resolver estes problemas, em uma indústria que produz equipos em grande escala, por exemplo, pode-se aplicar técnicas computacionais para aumentar a eficiência e qualidade de uma produção.

Uma dessas técnicas é o Processamento Digital de Imagens, onde através de uma câmera e um computador (ou outro dispositivo proposto), consegue-se processar e detectar defeitos na fabricação de produtos que antes eram feitos a olho nu, levando tempo e esforços humanos.

Sendo assim, este trabalho estuda métodos tradicionais de segmentação de imagens e propõe métodos específicos para o diagnóstico de falhas na fabricação de equipos utilizados na infusão intravenosa.

Os algoritmos Detector de Bordas Sobel, Detector de Bordas Canny, Detector de Círculos e de Linhas Hough, Histograma, Comparação de Histograma, Convolução e Casamento de Padrões foram estudados e aplicados a fim de solucionar os problemas específicos deste trabalho, que são: detectar se o papel filtrante encontra-se em perfeito estado em um filtro de solução e se o cilindro de borracha de um injetor lateral está corretamente posicionado.

Como resultado, este trabalho detecta o filtro de solução e se o papel filtrante está em perfeito estado, em um dos casos, no outro, se o cilindro de borracha está corretamente posicionado ou qual seu ângulo de inclinação em relação ao eixo.

**Palavras-chave:** Processamento de Imagens, Segmentação de Imagens, Detecção de Objetos, Automação Industrial, Equipos Médicos.

## ABSTRACT

GUEDES, Leandro Soares. **Study and Application of Image Segmentation Methods for Diagnosing Problems in Manufacturing Equipments.** 2014. 78 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Centro de Desenvolvimento Tecnológico, Universidade Federal de Pelotas, Pelotas, 2014.

The use of old techniques that don't use technology on the production, actually increase the cost and time of manufacturing products, such as, equipments used in the intravenous infusion.

To solve these problems in a factory that produces equipments in large scale, for example, we can apply computational techniques to increase the efficiency and quality of the production.

One of these techniques is the Digital Image Processing, where through a camera and a computer (or another proposed device), is possible to process and detect defects in manufacturing products that were once made with the naked eye, taking time and human efforts.

Thus, this work shows the study of traditional image segmentation methods and proposes specific methods for the diagnosis of problems in manufacturing equipments used in the intravenous infusion.

The algorithms Sobel Edge Detector, Canny Edge Detector, Hough Circle and Line Detectors, Histogram, Histogram Comparison, Convolution and Template Matching were studied and applied to solve specific problems of this work, which are: to detect if the filter paper is in perfect condition in a filter solution and if the rubber cylinder of the side injector is properly positioned.

As a result, this work detects the filter solution and if the filter paper is in perfect condition, in one case, in the other one, if the disc is correctly positioned or what is the inclination angle in relation to the axis.

**Keywords:** Image Processing, Image Segmentation, Object Detection, Industrial Automation, Medical Equipments.

## LISTA DE FIGURAS

Figura 1	Exemplo de equipo utilizado na administração de infusões parenterais. . . . .	15
Figura 2	Componentes de um equipo parenteral. . . . .	16
Figura 3	Componentes de um equipo. . . . .	17
Figura 4	Figura à esquerda após aplicação do Sobel resultando na Figura à direita. . . . .	22
Figura 5	Aplicação do Detector de Bordas Canny em diferentes níveis. . . . .	24
Figura 6	Detecção de círculos no OpenCV. A esquerda a imagem original e a direita depois de tratada. . . . .	26
Figura 7	Diferentes resultados dos métodos do DLH aplicado a uma imagem. . . . .	27
Figura 8	Gráfico de Frequência x Intensidade de pixels no Histograma da imagem a direita. . . . .	29
Figura 9	Equalização da imagem obtida na Figura 8. . . . .	30
Figura 10	Respectivamente: Imagem original, Tratada com Blur e Tratada com Edge . . . . .	32
Figura 11	Métodos aplicados a imagem do Injetor. . . . .	35
Figura 12	Peça A em diferentes estados de conservação. . . . .	37
Figura 13	Simulação da Peça A em uma linha de montagem. . . . .	38
Figura 14	Aplicação do Sobel à Peça A. . . . .	39
Figura 15	Detecção do círculo na Peça A. . . . .	39
Figura 16	Multi-deteção: Reconhecimento de dois filtros simultaneamente. . . . .	40
Figura 17	Imagens geradas pelo crop após a detecção do círculo. . . . .	40
Figura 18	Peça B com diferentes posicionamentos do cilindro de borracha. . . . .	42
Figura 19	Simulação da Peça B em uma linha de montagem. . . . .	43
Figura 20	Aplicação do Casamento de Padrões a diferentes posições do cilindro de borracha. . . . .	44
Figura 21	Resultado do DLH aplicado individualmente a cada imagem da Figura 20. . . . .	45
Figura 22	Um microcontrolador PIC18F8720 num encapsulamento TQFP de 80 pinos. . . . .	47
Figura 23	FPGA fabricado pela Altera modelo Stratix IV. . . . .	48
Figura 24	<i>Smartphone</i> Samsung S4 Mini com o sistema operacional Android. . . . .	48
Figura 25	<i>Tablet</i> Apple iPad Air com o sistema operacional iOS. . . . .	49

## LISTA DE TABELAS

Tabela 1	Resultado do Histogram Comparison aplicado à Figura 17. . .	40
Tabela 2	Resultado da Comparação de Histogramas na Peça B. . . . .	43
Tabela 3	Resultado do DLH aplicado à Figura 21. . . . .	46

## LISTA DE ABREVIATURAS E SIGLAS

CD	Compact Disc
CDF	Cumulative Distribution Function
CH	Comparação de Histogramas
CP	Casamento de Padrões
DBC	Detector de Bordas Canny
DBS	Detector de Bordas Sobel
DCH	Detector de Círculos Hough
DLH	Detector de Linhas Hough
FPGA	Field Programmable Gate Array
GVF	Gradient Vector Flow
GAC	Geodesic Active Contour
GET	Geodesic Edge Tracing
HIS	Hybrid Intelligent Systems
HSV	Hue-Saturation-Value Color Model
OpenCV	Open Source Computer Vision Library
PC	Personal Computer
PVC	Polyvinyl chloride
RGB	Red Green Blue Color Model
ROI	Region of Interest

# SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO</b>	15
1.1	Motivação	17
1.2	Objetivos	18
1.3	Organização do Trabalho	18
<b>2</b>	<b>REFERENCIAL TEÓRICO</b>	19
2.1	Introdução	19
2.2	Detector de Bordas Sobel	20
2.3	Detector de Bordas Canny	22
2.4	Detector de Círculos Hough	25
2.5	Detector de Linhas Hough	26
2.5.1	Detector de Linhas Hough Padrão	27
2.5.2	Detector de Linhas Hough Probabilístico	28
2.6	Histograma	28
2.6.1	Equalização	29
2.7	Comparação de Histogramas	30
2.7.1	Correlação	30
2.7.2	Chi-quadrado	31
2.7.3	Intersecção	31
2.7.4	Bhattacharyya	31
2.8	Convolução	31
2.9	Casamento de Padrões	33
<b>3</b>	<b>APLICAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS PARA A DETECÇÃO DE FALHAS DE FABRICAÇÃO</b>	36
3.1	Introdução	36
3.2	O caso da Peça A	36
3.3	O caso da Peça B	41
3.4	Proposta dos Aspectos de Infra-estrutura	46
3.4.1	Microcontrolador	47
3.4.2	FPGA	47
3.4.3	Smartphone	48
3.4.4	Tablet	49
<b>4</b>	<b>CONCLUSÕES</b>	50
	REFERÊNCIAS	51
	ANEXO A DETECTOR DE BORDAS SOBEL	54

<b>ANEXO B</b>	<b>DETECTOR DE BORDAS CANNY</b>	<b>56</b>
<b>ANEXO C</b>	<b>DETECTOR DE CÍRCULOS HOUGH</b>	<b>58</b>
<b>ANEXO D</b>	<b>DETECTOR DE LINHAS HOUGH</b>	<b>60</b>
<b>ANEXO E</b>	<b>HISTOGRAMA</b>	<b>64</b>
<b>ANEXO F</b>	<b>COMPARAÇÃO DE HISTOGRAMA</b>	<b>66</b>
<b>ANEXO G</b>	<b>CONVOLUÇÃO</b>	<b>68</b>
<b>ANEXO H</b>	<b>CASAMENTO DE PADRÕES</b>	<b>75</b>

# 1 INTRODUÇÃO

Equipo é um dispositivo (vide Figura 1<sup>1</sup>) que transporta soluções líquidas, como medicamentos ou alimentos, de um reservatório para o paciente. Equipos são utilizados para as mais variadas situações clínicas na administração de infusões, seja parenteral ou enteral.



Figura 1: Exemplo de equipo utilizado na administração de infusões parenterais.

A nutrição parenteral serve para complementar ou substituir completamente a alimentação oral ou enteral. Uma pessoa que não pode, não consegue ou não deve alimentar-se utilizando seu aparelho digestivo necessita de uma outra maneira de alimentação que o mantenha com um estado nutricional adequado, pois o paciente desnutrido enfrenta muito mal as enfermidades e invariavelmente evolui para óbito quando não é revertida esta situação.

No caso específico de fármacos a administração enteral pode ser feita i) pela boca; ii) por tubo gástrico, como tubo de alimentação duodenal ou gastrostomia; ou, iii) pelo reto. Já a administração parenteral, com efeito sistêmico, recebe-se a substância por outra forma que não pelo trato digestivo como, por exemplo,

---

<sup>1</sup><http://www.g3h.com.br/wp-content/uploads/2011/02/Equipo-Foto-Sensivel.jpg>

injeções i) intravenosa; ii) intra-arterial; iii) intramuscular; iv) intracardíaca; v) subcutânea; vi) intraóssea; vii) intradérmica; viii) intraperitoneal; etc.

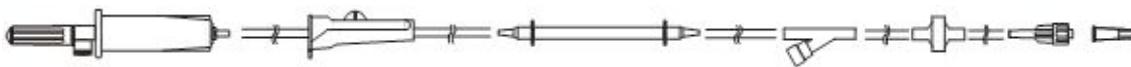


Figura 2: Componentes de um equipo parenteral.

Usualmente, um equipo é formado pelos seguintes componentes (como pode ser observado esquematicamente na Figura 2, da esquerda para a direita):

- Câmara de gotejamento: destinada a formação e cadência de gotas, com tubo gotejador que proporciona relação de gotas por milímetro, localizada entre a ponta perfurante e o tubo do equipo normalmente é confeccionada em PVC ou polipropileno. Pode conter filtro de solução (normalmente de 15 micra) para a retenção de partículas de suspensão.
- Segmento de silicone: Responsável por conectar os componentes permitindo a passagem das substâncias desejadas.
- Regulador de fluxo: neste caso do tipo rolete, destinado ao controle do processo de gotejamento, feito através de uma pinça rolete. Localizada sobre o tubo do equipo, entre a câmara gotejadora e o segmento de silicone, normalmente confeccionado em ABS grau alimentar de cor branca.
- Tubo: que une os componentes terminais (extremidades) e destinado a transportar o líquido do recipiente de solução até o paciente. Pode ser confeccionado em PVC ou poliuterano (PVC *free*), dependendo do modelo de equipo.
- Injetor lateral: do tipo ipsilone (Y), destinado a aplicação de múltiplas injeções de drogas no interior do equipo, através de uma membrana auto-vedante. Localizado junto ao tubo, próximo ao conector terminal do equipo e é confeccionado em policarbonato.
- Protetor: é um acessório adaptável aos componentes terminais do equipo de infusão para proteção. Localizado sobre componentes terminais e confeccionado em polietileno.
- Conector terminal: do tipo graduado, destinado a ligação do equipo ao paciente, sendo adaptado à via de acesso venoso e/ou enteral do paciente.

Este projeto procura atender a detecção de falhas no processo de fabricação da câmara de gotejamento e do injetor lateral do tipo ipsilone, detalhados na Figura 3.

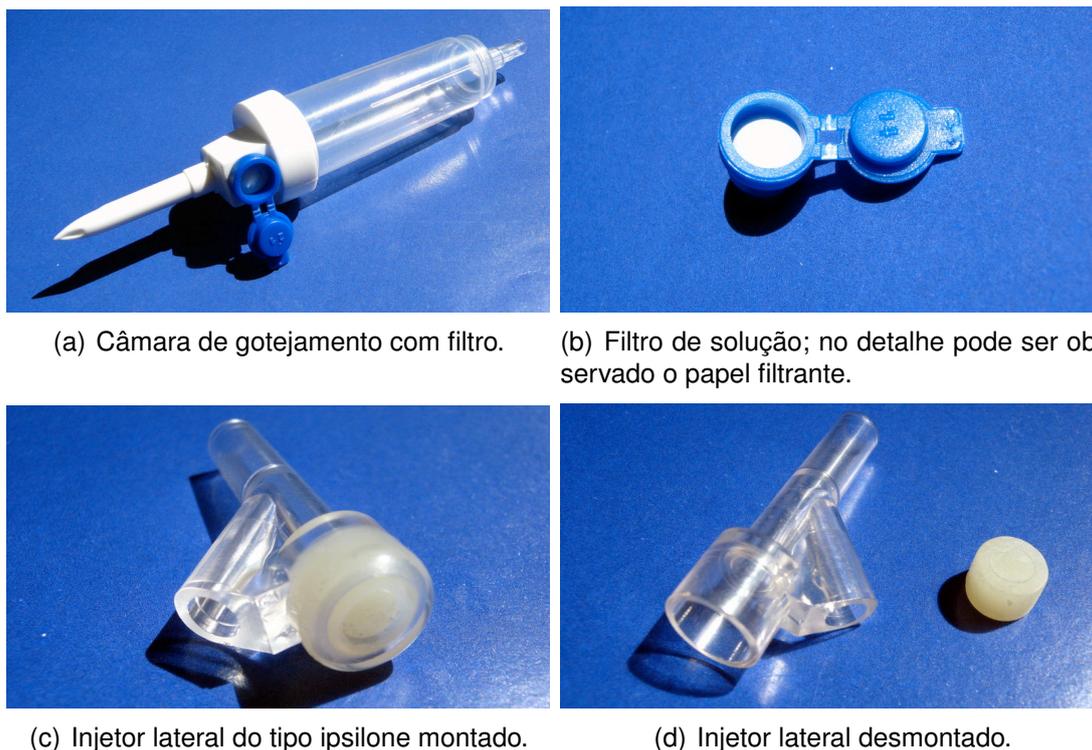


Figura 3: Componentes de um equipo.

No âmbito da câmara de gotejamento, mostrada na Figura 3(a), interessa-se na produção adequada do filtro de ar, Figura 3(b), cujo principal desafio a ser enfrentado é a verificação do correto posicionamento do papel filtrante (círculo branco interno ao filtro). Caso o papel esteja posicionado incorretamente ou rasgado pode ser observado o vazamento da medicação, ocasionando reações alérgicas ou danos na superfície da pele do paciente.

Esta verificação permitirá a separação dos filtros com defeito antes do processo de montagem da câmara final, reduzindo a probabilidade de perda de produção e parada da linha de montagem. Esta peça também será chamada de **Peça A** durante este trabalho.

No âmbito do injetor lateral, mostrado na Figura 3(c), tem-se como desafio a observação do posicionamento da membrana auto-vedante (cilindro de borracha) no interior do injetor, Figura 3(d), antes do procedimento de junção das partes. Ou, alternativamente, na separação do conjunto montado erroneamente antes de encaminhá-lo para a próxima etapa da linha de produção. Esta peça também será chamada de **Peça B** durante este trabalho.

## 1.1 Motivação

Os dois desafios apontados por este projeto podem ser entendidos como problemas de processamento digital de imagens. Onde um mecanismo de observa-

ção do processo de fabricação destes componentes pode capturar imagens que serão processadas por um computador para posterior verificação e tomada de decisão.

## 1.2 Objetivos

Este Projeto propõe estudar e desenvolver métodos de segmentação e avaliá-los quanto a sua aplicabilidade nas duas peças citadas anteriormente. São objetivos específicos deste Projeto:

1. Estudar algoritmos e filtros de processamento digital de imagens convencionais mais apropriados para a aplicação em questão;
2. Desenvolver um conjunto de métodos necessários para a segmentação e detecção de falhas na produção das referidas peças;
3. Produzir uma avaliação separada dos métodos quando aplicados a diferentes tipos de componentes do equipo.

Dados os objetivos acima apresentados, este projeto tem como metas e indicadores:

1. A implementação de algoritmos para a segmentação de imagens.
2. O desenvolvimento de uma ferramenta protótipo de aquisição e processamento de imagens dos componentes do equipo.

## 1.3 Organização do Trabalho

Este trabalho divide-se em 4 capítulos. O Capítulo 1 contém a Introdução, onde é apontado o problema existente. No Capítulo 2 é apresentada uma visão geral da área de estudo deste trabalho, o Processamento de Imagens, para então ser aplicado ao problema em questão: controle de qualidade em uma linha de produção de Equipos, além de que, são apresentados os principais algoritmos estudados e utilizados para a realização deste projeto. Em seguida, no Capítulo 3, é onde mostra a aplicação dos algoritmos, análise, e resultados obtidos ao aplicar aos problemas propostos, além de apresentar alternativas de Infraestrutura para o problema em questão. Por fim, no Capítulo 4 são mostradas as conclusões relativas ao trabalho apresentado.

## 2 REFERENCIAL TEÓRICO

### 2.1 Introdução

Para solucionar problemas com o processamento de imagens digitais, diversos algoritmos têm sido propostos. Em geral, estes algoritmos podem ser classificados em métodos baseados em limiares (*thresholding*), bordas ou regiões. Um exemplo de *thresholding* pode ser encontrado em (GANSTER et al., 2001), onde foi utilizada uma fusão global de *thresholding*, *thresholding* adaptativo e técnicas de *clustering*. Os métodos de *thresholding* alcançam bons resultados quando existe bom contraste entre as partes, portanto quando o histograma da imagem é bimodal, mas usualmente falham quando os modos de duas regiões se sobrepõem.

Abordagens baseadas em bordas, como as utilizadas em (RUBEGNI et al., 2001), buscam encontrar pontos onde trocam os sinais de funções Laplacianas ou Gaussianas (*zero-crossing*) e podem utilizar vários métodos de contornos ativos como o GVF utilizado em (ERKOL et al., 2005), o modelo GAC e o GET descritos em (CHUNG; SAPIRO, 2000). Abordagens como estas têm fraca performance quando os limites não são bem definidos, por exemplo quando a transição entre os padrões é suave. Nestas situações, as bordas tem *gaps* e o contorno se perde nestas discontinuidades. Uma outra dificuldade é a presença de pontos espúrios que não pertencem à borda do objeto. Estes pontos espúrios são resultantes de artefatos como pelos/fiapos, reflexões especulares ou mesmo irregularidades na textura das partes e podem impedir que o contorno chegue à borda do objeto.

Abordagens baseadas em regiões também têm sido utilizadas. Alguns exemplos incluem o crescimento de regiões em multi-escala descrita em (CELEBI; ASLANDOGAN; BERGSTRESSER, 2005), o *flooding* morfológico utilizado em (SCHMID, 1999), o algoritmo baseado em cadeias de Markov multirresolução (GAO; ZHANG; FLEMING, 2000) e a união estatística de regiões (CELEBI et al., 2008). Estas abordagens apresentam dificuldades quando os objetos são textu-

rizados ou tem a presença de diferentes cores levando à sobre-segmentação.

A matemática intervalar associada ao processamento de imagens digitais tem como objetivo obter um controle rigoroso de diversos tipos de erros que envolvem representações contínuas de valores reais. A ideia considerada por (LYRA, 2003) foi representar imagens digitais como imagens digitais intervalares com o objetivo de controlar erros adquiridos no momento da digitalização da imagem. Conforme (TAKAHASHI, 2005), na etapa de digitalização ocorrem dois momentos nos quais são necessários realizar discretizações nas imagens tanto espacialmente quanto em amplitude. É natural que ocorram aproximações nesta etapa, acarretando erros de aproximação na imagem digital, e a segmentação resultante possivelmente não corresponderia fielmente à imagem original. A quantificação da imagem digital, bem como a informação da precisão da discretização efetuada é um problema que pode ser tratado pela matemática intervalar (LYRA, 2003).

Esta área é conhecida como Sistemas Inteligentes Híbridos (HIS), que é um campo de pesquisa extremamente promissor, de onde a próxima geração de sistemas inteligentes será baseada (IJHIS, 2011). Recentemente, HIS tornou-se mais popular pois está sendo utilizada em problemas complexos reais, envolvendo imprecisão e incerteza. A possibilidade da integração de técnicas de matemática intervalar trará robustez para os modelos baseados em HIS.

Este trabalho de conclusão está inserido no contexto de um projeto maior, em desenvolvimento no grupo de pesquisa, que tem o interesse de propor métodos baseados em HIS para a segmentação e classificação de imagens digitais. Por isso, o escopo deste projeto está no levantamento de métodos convencionais que poderiam ser aplicados neste contexto de fabricação de Equipos.

Os algoritmos e métodos estudados ficaram centrados em Detector de Bordas Sobel, Detector de Bordas Canny, Detector de Círculos Hough, Detector de Linhas Hough, Histograma, Comparação de Histogramas, Convolução e Casamento de Padrões, pois estes se apresentaram com maior contribuição ao projeto em desenvolvimento pelo grupo de pesquisa.

## **2.2 Detector de Bordas Sobel**

O Detector de Bordas Sobel, também chamado apenas de Sobel, é uma operação utilizada em processamento de imagem, similar ao DBC, aplicada sobretudo em algoritmos para detecção de bordas. É uma das mais importantes convoluções para a computação de derivadas. Esse algoritmo existe na ordem de qualquer derivada assim como em derivadas parciais (BRADSKI; KAEHLER, 2008).

Resumidamente, esse operador calcula diferenças finitas, dando uma aproximação do gradiente da intensidade dos pixels da imagem. Em cada ponto da imagem, o resultado da aplicação do filtro Sobel devolve o gradiente ou a norma deste vetor.

As derivadas de Sobel têm a boa propriedade de que elas podem ser definidas para kernels de qualquer tamanho, e esses kernels podem ser construídos de forma rápida e iterativa. Os kernels maiores dão uma melhor aproximação à derivada, pois, quanto menor o kernel, mais sensível ao ruído fica (HUAMÁN, 2014a).

Para entender isso mais exatamente, devemos perceber que uma derivada Sobel não é exatamente uma derivada. Isto é porque o operador de Sobel é definido em um espaço discreto. O que o operador Sobel representa, na verdade, é um ajuste de um polinômio. Ou seja, o Sobel derivado de segunda ordem na direção  $x$  não é realmente uma segunda derivada, é um ajuste local para uma função parabólica. Isso explica porque usar um kernel grande: o maior kernel está computando o ajuste ao longo de um número maior de pixels.

Matematicamente este operador utiliza duas matrizes  $3 \times 3$  que são convoluídas com a imagem original para calcular aproximações das derivadas, uma para as variações horizontais e uma para as verticais. Sendo  $I$  a imagem inicial então,  $G_x$  e  $G_y$  serão duas imagens que em cada ponto contêm uma aproximação às derivadas horizontal e vertical de  $I$ . Os passos matemáticos de execução são apresentados a seguir:

1. Cálculo da primeira derivada: Mudanças horizontais. É computado ao convoluir  $I$  (imagem a ser operada) com um kernel  $G_x$

$$\mathbf{G}_x = \begin{pmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{pmatrix} * I$$

2. Cálculo da segunda derivada: Mudanças verticais. É computado ao convoluir  $I$  com um kernel  $G_y$

$$\mathbf{G}_y = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix} * I$$

3. A cada ponto da imagem, é calculada uma aproximação do gradiente naquele ponto combinando os dois resultados acima:

$$G = \sqrt{G_x^2 + G_y^2}$$

Embora, às vezes, a seguinte equação mais simples seja utilizada:

$$G = |G_x| + |G_y|$$

O filtro Sobel calcula o gradiente da intensidade da imagem em cada ponto, dando a direção da maior variação de claro para escuro e a quantidade de variação nessa direção. Assim, obtém-se uma noção de como varia a luminosidade em cada ponto, de forma mais suave ou abrupta.

Com isto consegue-se estimar a presença de uma transição claro-escuro e de qual a orientação desta. Como as variações claro-escuro intensas correspondem a fronteiras bem definidas entre objetos, consegue-se fazer a detecção de bordas.

Foi feita uma implementação do Sobel por (HUAMÁN, 2014a) e adaptada ao projeto (como pode ser observada no Anexo A). Essa implementação carrega uma imagem, aplica um *Gaussian Blur* para reduzir o ruído, converte para cinza, gera as derivadas de x e y, exibe o gradiente total aproximado, e por fim, exibe a imagem resultante. Na Figura 14 pode-se ver um exemplo deste algoritmo.

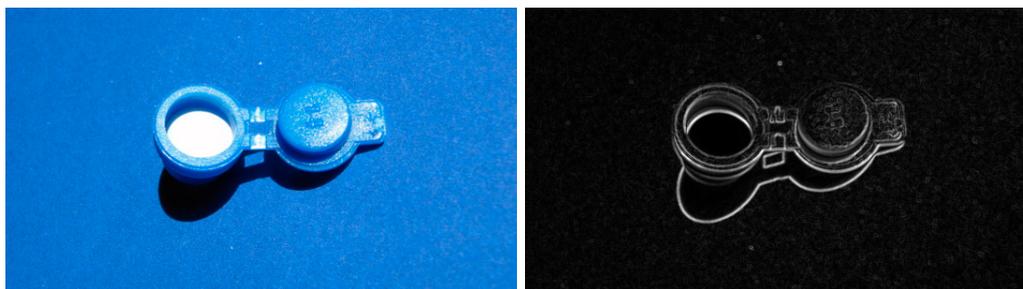


Figura 4: Figura à esquerda após aplicação do Sobel resultando na Figura à direita.

### 2.3 Detector de Bordas Canny

O Detector de Bordas Canny, do inglês *Canny Edge Detector*, também conhecido como *Optimal Detector*, é um algoritmo de detecção de bordas, similar ao DBS, que usa uma implementação multi-etapas para detectar uma quantidade variada de bordas em uma imagem. Esse algoritmo foi desenvolvido por John F. Canny em 1986.

O algoritmo possui três características principais (HUAMÁN, 2014b):

1. Baixa taxa de erro: significando uma boa detecção somente das bordas existentes.
2. Boa localização: a distância entre pixels detectados de borda e pixels reais de borda reais têm de ser minimizadas.
3. Resposta mínima: apenas uma resposta do detector por aresta.

Este algoritmo possui 4 passos de execução:

1. Filtrar qualquer ruído. O filtro de Gauss é utilizado para esta finalidade. Um exemplo de um núcleo gaussiano de tamanho 5 que pode ser usado é o seguinte:

$$\mathbf{K} = \frac{1}{159} \begin{pmatrix} 2 & 4 & 5 & 4 & 2 \\ 4 & 9 & 12 & 9 & 4 \\ 5 & 12 & 15 & 12 & 5 \\ 4 & 9 & 12 & 9 & 4 \\ 2 & 4 & 5 & 4 & 2 \end{pmatrix}$$

2. Encontrar o gradiente da intensidade da imagem. Para isso, segue-se um procedimento análogo ao Sobel:

(a) Aplicar um par de máscaras de convolução (nas direções x e y):

$$\mathbf{G}_x = \begin{pmatrix} -1 & 0 & +1 \\ -2 & 0 & +2 \\ -1 & 0 & +1 \end{pmatrix}$$

$$\mathbf{G}_y = \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{pmatrix}$$

(b) Encontrar a força de gradiente e direção com:

$$G = \sqrt{G_x^2 + G_y^2} \text{ e } \theta = \arctan\left(\frac{G_y}{G_x}\right)$$

A direção é arredondada para um dos quatro ângulos possíveis (chamados 0, 45, 90 ou 135).

3. Supressão não-máxima é aplicada. Isto remove os pixels que não são considerados como sendo parte de uma aresta. Assim, apenas linhas finas (arestas candidatas) permanecerão.

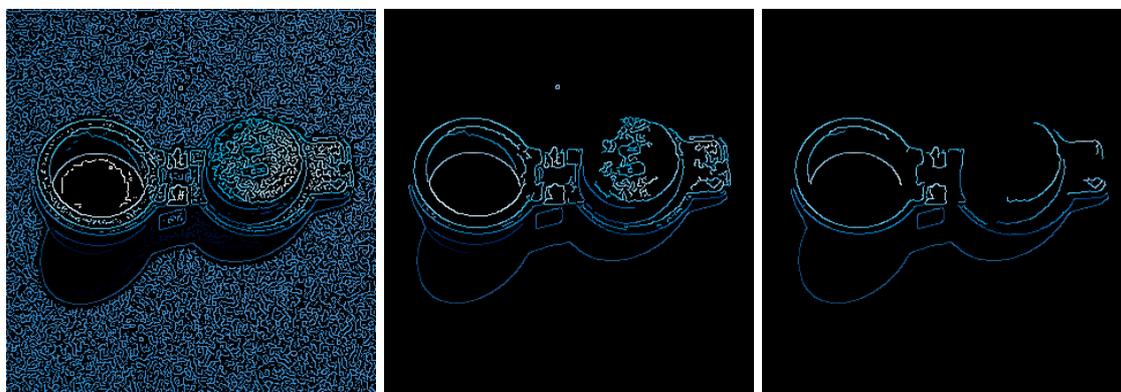
4. Histerese: *Canny* usa dois limites (superior e inferior) para explorar o fato de que pixels de borda são normalmente conectados. Se fosse utilizado apenas um nível (limite) poderia permitir muito ruído em um limite baixo ou perderia importantes bordas caso estivesse com um limite alto, assim, com os limites superior e inferior permitimos uma melhor detecção de bordas em uma imagem (YU, 2014). Nesses dois limites:

- (a) Se um gradiente do pixel é maior do que o limite superior, o pixel é aceito como uma borda.
- (b) Se um valor de gradiente de pixel for inferior ao limiar inferior, então é rejeitado.
- (c) Se o gradiente de pixel está entre os dois limiares, então ele irá ser aceito se está ligado a um elemento de imagem que está acima do limite superior.

*Canny* recomendou uma taxa superior:inferior entre 2:1 e 3:1.

Este método foi implementado no OpenCV por (HUAMÁN, 2014b) e encontra-se no Anexo B uma versão adaptada ao projeto. Pode-se definir o limite do DBC e escolher diversos valores entre o limite (nível) mínimo e o máximo de aplicação.

O programa gera uma máscara, em fundo preto, onde linhas brilhantes representam as bordas detectadas. Esta máscara é aplicada a imagem original e o resultado é exibido em uma janela. Na Figura 5(a) pode-se observar o equipo apresentado na Figura 3(b) após aplicação do nível mínimo do *Canny*. Na Figura seguinte, 5(b), o resultado exibido é do DBC no limite médio, já eliminando ruídos e deixando as bordas mais visíveis. Depois de executado no limite máximo, obtém-se a Figura 5(c) como resultado, onde fica visível a eficiência do algoritmo para a detecção de bordas.



(a) Limite mínimo.

(b) Limite médio.

(c) Limite máximo.

Figura 5: Aplicação do Detector de Bordas Canny em diferentes níveis.

## 2.4 Detector de Círculos Hough

O Detector de Círculos Hough, do inglês *Hough Circle Detector*, detecta círculos nas imagens e está disponível no OpenCV (HUAMÁN, 2014c).

Para fazer isso, necessita-se definir um círculo que é caracterizado por três parâmetros: Eixo X central, Eixo Y central e o raio  $r$ . Onde o  $(X_{centro}, Y_{centro})$  define o centro do círculo, que é caracterizado por um ponto verde, e o raio  $r$  que nos permite delimitar a circunferência do círculo.

Para aplicação específica no reconhecimento do equipamento apresentado na Figura 3(b) e no cilindro de borracha da Figura 3(d) é requerido que a câmera esteja capturando imagens exatamente a frente do centro dos eixos. Isso faria com que esse algoritmo reconheça a presença de um círculo.

De forma geral o algoritmo funciona deste modo:

1. Carrega uma imagem.
2. Realiza um *blur* para reduzir o ruído.
3. Aplica a transformada DCH na imagem.
4. Exibe o resultado com a detecção encontrada.

Detalhadamente, o algoritmo possui as seguintes etapas em OpenCV (BRADSKI; KAEHLER, 2008):

1. Carrega uma imagem.
2. A imagem transmitida é passada por uma fase de detecção de borda.
3. Em cada ponto diferente de zero na imagem de borda, o gradiente local é considerado (a inclinação é calculada pela primeira computação da primeira ordem de derivadas Sobel x- e y-). Usando esse gradiente, todos os pontos ao longo da linha indicada por este declive são incrementados no acumulador. Ao mesmo tempo, a localização de cada um destes pixels diferentes de zero da borda da imagem é reconhecida.
4. Os candidatos a centro são então selecionados a partir dos pontos neste acumulador (bidimensional) que estão acima de um determinado limite e maiores do que todos os seus vizinhos imediatos. Estes candidatos serão classificados em ordem decrescente de seus valores no acumulador, para que os centros com os pixels mais suportados apareçam em primeiro lugar.
5. Em seguida, para cada centro, todos os pixels diferentes de zero calculados anteriormente são considerados. Estes pixels são classificados de acordo com sua distância do centro.

6. Trabalhando a partir das menores distâncias para o raio máximo, um único raio (que é melhor suportado pelos pixels diferentes de zero) é selecionado.
7. Um centro é mantido se tiver apoio suficiente dos pixels não nulos da imagem de borda e se for uma distância suficiente de qualquer centro selecionado anteriormente.
8. Exibe o resultado com a detecção encontrada.

Esta implementação em OpenCV permite que o algoritmo execute muito mais rápido e, talvez ainda mais importante, ajude a superar o problema da população esparsa de um acumulador tridimensional, o que levaria a uma grande quantidade de ruído e a tornar os resultados instáveis.

O algoritmo é, então, muito eficiente para detectar um ou mais círculos e possui um rápido tempo de execução. No Anexo C pode-se ver uma versão do código adaptada ao projeto. Um exemplo é mostrado na Figura 6 onde o algoritmo detecta um círculo presente na imagem de um CD.



Figura 6: Detecção de círculos no OpenCV. A esquerda a imagem original e a direita depois de tratada.

## 2.5 Detector de Linhas Hough

O Detector de Linhas Hough, do inglês *Hough Lines Detector*, detecta linhas nas imagens e está disponível no OpenCV (HUAMÁN, 2014d).

Sabe-se que uma linha, no espaço de imagem, pode ser expressada com duas variáveis. Por exemplo: Parâmetros:  $(m, b)$  no sistema de coordenadas cartesianas; E parâmetros:  $(r, \theta)$  No sistema de coordenadas polares.

Nas transformadas Hough, é possível expressar linhas no sistema Polar. Assim, uma equação de linha pode ser escrita como:  $y = \left(-\frac{\cos \theta}{\sin \theta}\right)x + \left(\frac{r}{\sin \theta}\right)$ . Organizando os termos:  $r = x \cos \theta + y \sin \theta$ .

Em geral, para cada ponto  $(x_0, y_0)$ , pode-se definir a família de linhas que passa por esse ponto como:  $r_\theta = x_0 \cdot \cos \theta + y_0 \cdot \sin \theta$ . O que significa que cada ponto  $(x_0, y_0)$  representa cada linha que passa por  $(x_0, y_0)$ .

Se para um determinado  $(x_0, y_0)$  plotarmos a família de linhas que passa por ele, teremos uma senóide. Pode-se fazer a mesma operação acima para todos os pontos de uma imagem. Se as curvas de dois pontos diferentes se cruzam no plano  $\theta - r$ , isso significa que ambos os pontos pertencem a uma mesma linha.

Isso tudo significa que, em geral, uma linha pode ser detectada por encontrar o número de intersecções entre curvas. Quanto mais curvas intersectando, significa que a linha representada pela intersecção tem mais pontos. Em geral, pode-se definir um limiar do número mínimo de intersecções para detectar uma linha.

O DLH mantém o controle da intersecção entre as curvas de cada ponto da imagem. Se o número de intersecções é acima de certo valor, então ele declara-o como uma linha com os parâmetros  $(\theta, r_\theta)$  do ponto de intersecção.

A técnica do DLH foi implementada em linguagem C++ e adaptada ao projeto. O código pode ser observado no Anexo D. Ele carrega uma imagem, transforma em tons de cinza, aplica o DBC, realiza a transformada padrão Hough (`vector<Vec2f> lines`);, calcula (de acordo com a função desejada) e exibe o resultado das linhas geradas.

Na Figura 7 pode-se observar como essas funções retornam resultados diferentes. Na Figura 7(b) as linhas cortam toda a imagem enquanto na Figura 7(c) as linhas são traçadas apenas no espaço que existirem.

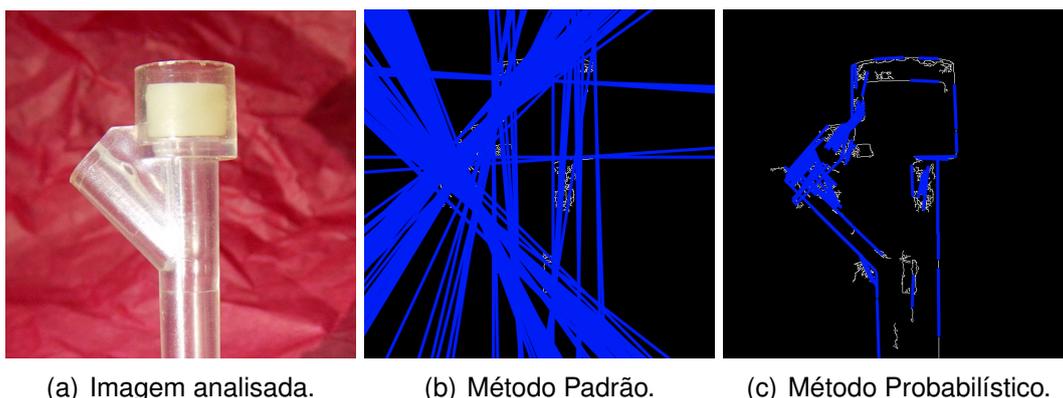


Figura 7: Diferentes resultados dos métodos do DLH aplicado a uma imagem.

Os métodos comparados na Figura 7 são detalhados abaixo:

### 2.5.1 Detector de Linhas Hough Padrão

Consiste praticamente no que foi abordado durante essa seção. Gera como resultado um vetor  $(\theta, r_\theta)$ . É implementado em OpenCV pela função `HoughLines`

(dst, lines, 1, CV\_PI/180, 100, 0, 0); com os seguintes argumentos:

dst: Saída do detector de borda. Deve ser uma imagem em tons de cinza

lines: Um vetor que irá armazenar os parâmetros  $(r, \theta)$  das linhas detectadas

rho : A resolução do parâmetro  $r$  em pixels (foi usado 1 pixel)

theta: A resolução do parâmetro  $\theta$  em radianos, foi usado 1 grau (CV\_PI/180)

threshold: O número mínimo de intersecções para detectar uma linha

srn e stn: Parâmetro padrão em zero.

### 2.5.2 Detector de Linhas Hough Probabilístico

Uma implementação mais eficiente do Detector de Linhas Hough. Gera como saída os extremos das linhas detectadas  $(x_0, y_0, x_1, y_1)$ . É implementado em OpenCV pela função `HoughLinesP` (dst, lines, 1, CV\_PI/180, 50, 50, 10); com os seguintes argumentos:

dst: Saída do detector de borda. Deve ser uma imagem em tons de cinza

lines: Um vetor que irá armazenar os parâmetros  $(x_{start}, y_{start}, x_{end}, y_{end})$  das linhas detectadas

rho : A resolução do parâmetro  $r$  em pixels (foi usado 1 pixel)

theta: A resolução do parâmetro  $\theta$  em radianos, foi usado 1 grau (CV\_PI/180)

threshold: O número mínimo de intersecções para detectar uma linha

minLinLength: O número mínimo de pontos que pode gerar uma linha. Linhas com número menor que esse de pontos serão desconsideradas

maxLineGap: A diferença máxima entre dois pontos a serem considerados na mesma linha.

## 2.6 Histograma

Um histograma representa graficamente a distribuição de frequências a partir de um conjunto de medições. Um exemplo simples de histograma é a representação da quantidade de indivíduos por faixa etária de uma população. Um eixo representa as possíveis idades dos indivíduos, que podem variar de 1 a 130 anos, ou mais, enquanto que um outro eixo representa quantas vezes uma determinada idade se repete na população, ou seja, a frequência.

No processamento de imagens, um histograma representa a informação da quantidade de vezes que uma dada cor se repete na imagem, geralmente com base em uma foto na escala de cinza, segundo mostra a Figura 8.

Uma imagem possui um único histograma, entretanto, a recíproca não é em geral verdadeira, pois um histograma não contém informação espacial, apenas valores de intensidade (PEDRINI; SCHWARTZ, 2008).

A partir do histograma de uma imagem é possível, por exemplo, determinar

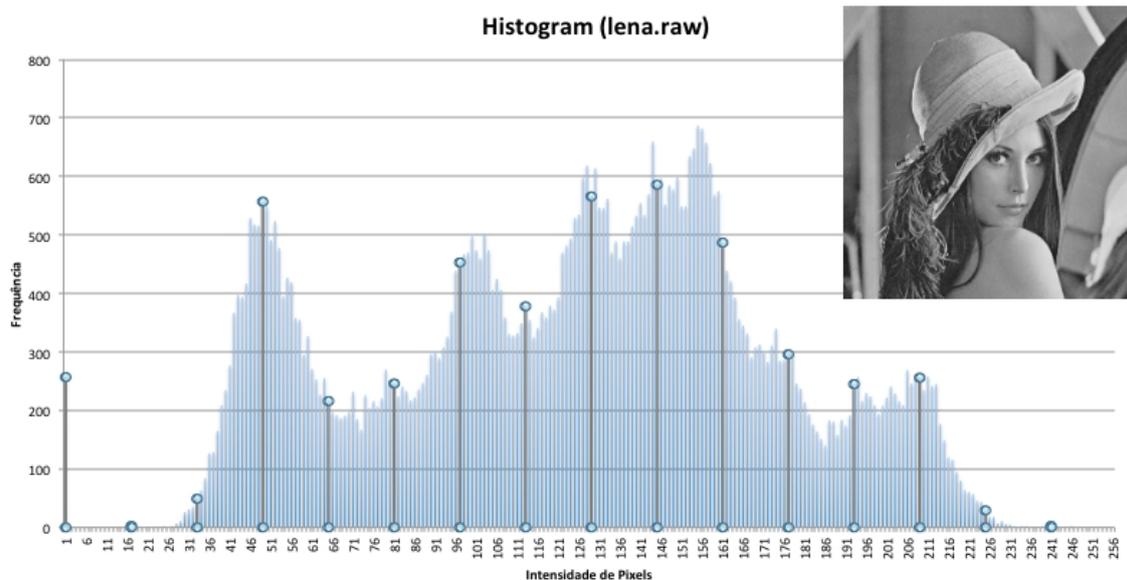


Figura 8: Gráfico de Frequência x Intensidade de pixels no Histograma da imagem a direita.

regiões indesejadas da imagem, analisando picos ou vales do histograma para então delimitar tais regiões.

Também podem ser implementadas transformações e equalizações, melhorando brilho e contraste, e consequentemente obter dados diferenciados que são úteis para identificar objetos na imagem processada.

### 2.6.1 Equalização

É um método que melhora o contraste de uma imagem, de modo a esticar a faixa de intensidade, ou seja, esticar o intervalo com maior incidência de pixels detectados (WANNER, 2014).

Na Figura 8 pode-ver ver a imagem com o Histograma aplicado, mas essa imagem pode ser equalizada e gera a imagem que pode ser vista na Figura 9.

Equalização implica um mapeamento de distribuição (o histograma dado) para uma outra distribuição (uma distribuição mais ampla e uniforme de valores de intensidade), de modo os valores de intensidade são espalhados em toda a gama. Para conseguir o efeito de equalização, o remapeamento deve ser a função de distribuição cumulativa (CDF).

No projeto utiliza-se essas técnicas para modificar a imagem original com a finalidade de auxiliar no processo final de mapeamento e detecção de bordas.

Foi feita uma implementação da técnica do histograma (como pode ser observada no Anexo E) que lê uma imagem em escala de cinza e calcula o respectivo histograma.



Figura 9: Equalização da imagem obtida na Figura 8.

## 2.7 Comparação de Histogramas

Primeiramente introduzidos por Swain e Ballard (SWAIN; BALLARD, 1991) e posteriormente generalizado por Schiele e Crowley (SCHIELE; CROWLEY, 1996), é a habilidade de comparar dois Histogramas em determinados critérios específicos de similaridade.

Existe em OpenCV a função `cvCompareHist(const SparseMat& H1, const SparseMat& H2, int method)` disponível em (HUAMÁN, 2014e) que faz exatamente isso. Os dois primeiros argumentos são os histogramas a serem comparados (que devem ser do mesmo tamanho) e o terceiro é uma das métricas desejadas. As seguintes métricas são utilizadas:

### 2.7.1 Correlação

$$d_{\text{correlacao}}(\mathbf{H}_1, \mathbf{H}_2) = \frac{\sum_i H'_1(i) \cdot H'_2(i)}{\sqrt{\sum_i H_1'^2(i) \cdot H_2'^2(i)}}$$

onde  $H'_k(i) = H_k(i) - (1/N)(\sum_j H_k(j))$  e  $N$  é o número de frequência do histograma. Para a correlação, quanto maior o valor em uma escala de 1 a -1, melhor a combinação encontrada. Uma combinação perfeita tem valor 1, uma combinação sem correlação possui o valor 0 e uma falta completa de combinação possui o valor -1.

### 2.7.2 Chi-quadrado

$$d_{\text{chi-quadrado}}(\mathbf{H}_1, \mathbf{H}_2) = \sum_i \frac{(H_1(i) - H_2(i))^2}{H_1(i) + H_2(i)}$$

Para o chi-quadrado, um valor pequeno representa uma melhor combinação do que um valor grande. Uma combinação perfeita é 0 e não há limite máximo para uma completa falta de combinação.

### 2.7.3 Intersecção

$$d_{\text{interseccao}}(\mathbf{H}_1, \mathbf{H}_2) = \sum_i \min(H_1(i), H_2(i))$$

Para a intersecção, valores altos representam uma boa combinação e baixos indicam uma combinação ruim. Se ambos os histogramas estiverem normalizados para 1, então uma intersecção perfeita é 1 e uma completa falta de intersecção é 0.

### 2.7.4 Bhattacharyya

$$d_{\text{bhattacharyya}}(\mathbf{H}_1, \mathbf{H}_2) = \sqrt{1 - \sum_i \frac{\sqrt{H_1(i) \cdot H_2(i)}}{\sqrt{\sum_i H_1(i) \cdot \sum_i H_2(i)}}}$$

Para a combinação Bhattacharyya, um valor pequeno representa uma melhor combinação do que um valor maior. Uma combinação perfeita é 0 e não há limite máximo para uma completa falta de combinação.

Foi feita uma implementação da técnica de Comparação de Histogramas por (HUAMÁN, 2014f) e adaptada ao projeto (como pode ser observada no Anexo F). Essa implementação, carrega duas imagens (a original e a imagem a ser comparada), converte para HSV, aplica intervalos distintos de Hue e Saturação, calcula os histogramas para as imagens HSV de acordo com os quatro métodos de comparação explicados acima, e por fim, exibe os resultados numéricos das comparações.

## 2.8 Convolução

O processo de convolução utiliza uma imagem de entrada  $\mathbf{f}$  com dimensões  $M \times N$  pixels e uma máscara  $\mathbf{w}$  com dimensões  $m \times n$  pixels. A saída é uma imagem  $\mathbf{g}$  com dimensões  $M \times N$  pixels (PEDRINI; SCHWARTZ, 2008).

Uma convolução permite fazer muitas coisas, como calcular derivadas, detec-

tar bordas, aplicar *blur*, entre outras coisas (FARID, 2012).

Isso tudo é feito com um "kernel de convolução". Esse kernel nada mais é do que uma pequena matriz. Esta matriz tem números em cada célula e tem um ponto central. Este kernel desliza sobre uma imagem e faz sua parte. O ponto central é utilizado para determinar a posição do núcleo do kernel em relação à imagem analisada (SINHA, 2014).

A maneira mais direta para calcular a convolução seria a utilização de múltiplos laços. Mas isso provoca uma série de cálculos repetidos. E, como o tamanho da imagem e do kernel aumentam, o tempo para calcular a convolução aumenta drasticamente também.

Técnicas têm sido desenvolvidas para calcular convoluções rapidamente. Uma dessas técnicas é o uso da Transformada Discreta de Fourier. Ela converte toda a operação de convolução em uma multiplicação simples.

Abaixo, na Figura 10 tem-se uma imagem tratada com esse algoritmo. A imagem da Lena original à esquerda, foi tratada com o *Blur* e resultou na imagem central, e ao ser tratada com o *Edge* (detecção de bordas) resultou na imagem à direita.



Figura 10: Respectivamente: Imagem original, Tratada com Blur e Tratada com Edge

Para o trabalho proposto, esse algoritmo é uma base para ajustar a imagem recebida com o objetivo de facilitar a detecção dos objetos. A técnica de convolução foi implementada em linguagem C (como pode ser observado no Anexo G) onde se carrega uma imagem, aloca o tamanho de linhas e colunas necessários, realiza um histograma e a convolução, e por fim, libera a memória e exhibe os resultados.

## 2.9 Casamento de Padrões

Esta técnica, chamada de Casamento de Padrões, do inglês *Template Matching*, consiste em detectar a área de maior correspondência (casamento) entre duas imagens.

As duas imagens, são os componentes principais: Imagem Fonte (F): A imagem em que esperamos encontrar uma correspondência para a imagem molde; Imagem Molde (M): A imagem que será comparada com a Imagem Fonte. Para identificar a área de casamento entre essas duas imagens, deve-se comparar a imagem molde contra a imagem fonte, deslizando-a.

Deslizar, neste caso, significa mover a imagem fonte um pixel de cada vez, da esquerda para a direita e de cima para baixo. Em cada local, uma métrica é calculada para representar quão "bom" ou "ruim" é a correspondência naquele local (ou quão similar a imagem molde é daquela área específica da imagem fonte).

Para cada local de M sobre F, é armazenada esta métrica na matriz resultado (R). Cada local (x,y) em R contém a métrica da imagem fonte.

Os locais mais brilhantes na matriz resultado, indicam os maiores casamentos. Na prática, pode-se utilizar a função `minMaxLoc` para localizar o valor mais elevado (ou inferior, dependendo do tipo de método correspondente) na matriz R. Existem 6 métodos disponíveis em OpenCV para calcular o CP que são apresentados a seguir (HUAMÁN, 2014g):

### Método 0

Interpretado com o valor 0 em OpenCV, este método se chama `CV_TM_SQDIFF` e é realizado pela seguinte equação:

$$\mathbf{R}(x, y) = \sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2$$

### Método 1

Interpretado com o valor 1 em OpenCV, este método se chama `CV_TM_SQDIFF_NORMED` e é realizado pela seguinte equação:

$$\mathbf{R}(x, y) = \frac{\sum_{x', y'} (T(x', y') - I(x + x', y + y'))^2}{\sqrt{\sum_{x', y'} (T(x', y'))^2 \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

## Método 2

Interpretado com o valor 2 em OpenCV, este método se chama `CV_TM_CCORR` e é realizado pela seguinte equação:

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) = \sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))$$

## Método 3

Interpretado com o valor 3 em OpenCV, este método se chama `CV_TM_CCORR_NORMED` e é realizado pela seguinte equação:

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{x', y'} (T(x', y') \cdot I(x + x', y + y'))}{\sqrt{\sum_{x', y'} (T(x', y')^2) \cdot \sum_{x', y'} I(x + x', y + y')^2}}$$

## Método 4

Interpretado com o valor 4 em OpenCV, este método se chama `CV_TM_CCOCOEFF` e é realizado pela seguinte equação:

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) = \sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))$$

onde  $T'(x', y') = T(x', y') - 1/(w \cdot h) \cdot \sum_{x'', y''} T(x'', y'')$  e  $I'(x + x', y + y') = I(x + x', y + y') - 1/(w \cdot h) \cdot \sum_{x'', y''} I(x + x'', y + y'')$

## Método 5

Interpretado com o valor 5 em OpenCV, este método se chama `CV_TM_CCOCOEFF_NORMED` e é realizado pela seguinte equação:

$$\mathbf{R}(\mathbf{x}, \mathbf{y}) = \frac{\sum_{x', y'} (T'(x', y') \cdot I'(x + x', y + y'))}{\sqrt{\sum_{x', y'} (T'(x', y')^2) \cdot \sum_{x', y'} I'(x + x', y + y')^2}}$$

Depois que a função termina a comparação, os melhores casamentos podem ser encontrado como mínimos globais (quando `CV_TM_SQDIFF` for utilizado) ou máximos (quando for utilizado `CV_TM_CCORR` ou `CV_TM_CCOCOEFF`) usando a função `minMaxLoc` (`const SparseMat& a, double* minVal, double*`

`maxVal, int* minIdx=0, int* maxIdx=0`). No caso de uma imagem colorida, a soma do molde no numerador e cada soma no denominador são feitas ao longo de todos os canais, e os valores médios separados são utilizados para cada canal. Ou seja, a função pode ter imagens fonte e molde coloridas. O resultado ainda será uma imagem de um único canal, o que é mais fácil de analisar.

A Figura 11 abaixo exhibe o resultado dos métodos que localizaram com sucesso o cilindro de borracha (passado como molde) para a figura do Injetor (Figura 11(e)).

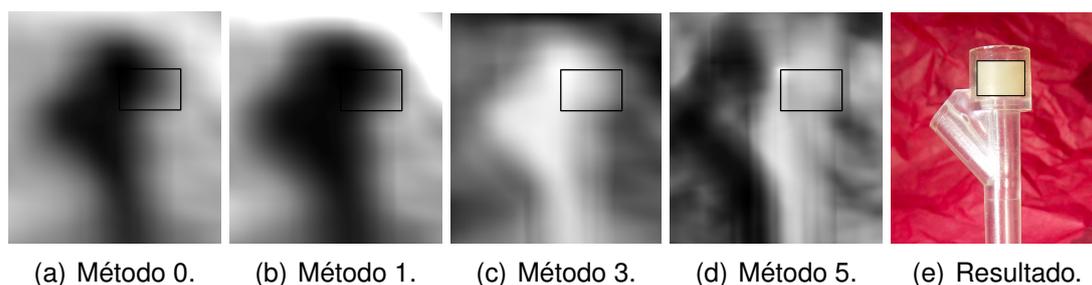


Figura 11: Métodos aplicados a imagem do Injetor.

A técnica de CP vista na figura acima foi implementada em linguagem C++ para o OpenCV por (HUAMÁN, 2014g) e adaptada ao projeto. O código pode ser observado no Anexo H. Ele carrega duas imagens, sendo que uma delas (passada pelo segundo parâmetro) deve ser de tamanho menor para ser comparada com a primeira. Após isso, é criada uma matriz para exibir os resultados, o algoritmo localiza a melhor combinação com o `minMaxLoc`, calcula e recorta na imagem original a nova imagem a ser exibida, por fim, salva a imagem recortada.

## **3 APLICAÇÃO DE TÉCNICAS DE PROCESSAMENTO DE IMAGENS PARA A DETECÇÃO DE FALHAS DE FABRICAÇÃO**

### **3.1 Introdução**

Após estudos preliminares, o projeto passou a ser desenvolvido exclusivamente em OpenCV devido à eficiência desta biblioteca para aplicação em uma linha de montagem, onde o resultado necessita ser aplicado em tempo real. O OpenCV é uma biblioteca de visão computacional de código aberto que possui desde funções simples até funções complexas tais como as que são apresentadas neste trabalho. Esta biblioteca foi desenvolvida inicialmente pela Intel Corporation e pode ser encontrada gratuitamente na Internet (OPENCV, 2014).

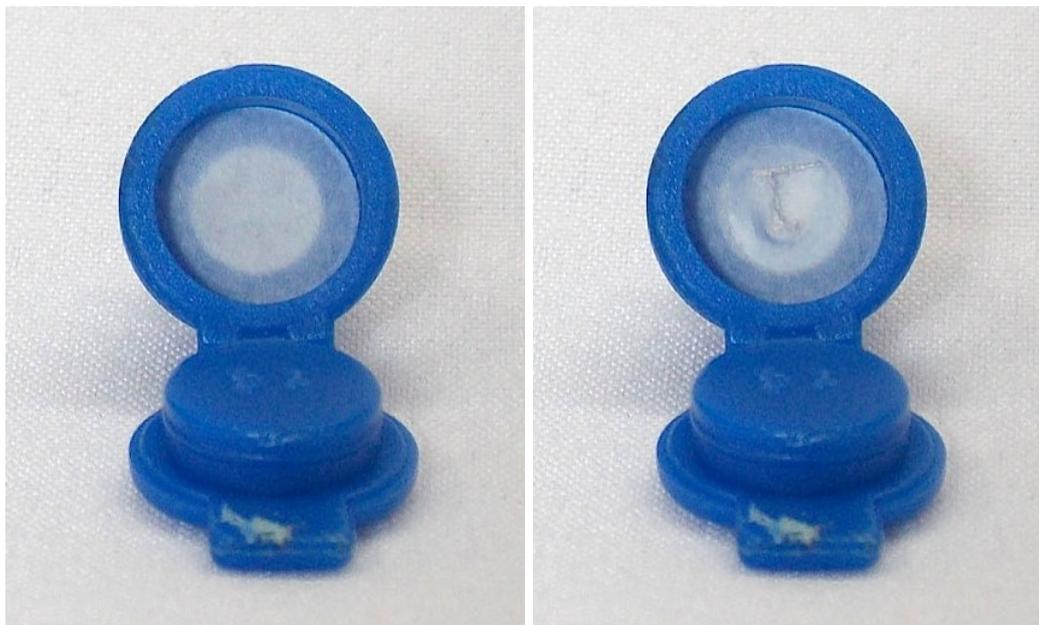
Foram feitas imagens dos equipos em cenários hipotéticos, com iluminação controlada, adequando ao contexto real a fim de extrair com êxito um resultado satisfatório dos algoritmos que foram apresentados. Existem intervalos de segurança para essa abordagem hipotética, pois em linhas de montagem reais, com maior velocidade de produção, há a necessidade de um grupo de controle maior para solucionar as diferenças de iluminação existentes.

O projeto, dividido entre Peça A e Peça B, passou a ter dois problemas distintos de segmentação de imagens para serem solucionados. De um lado, para a Peça A, necessitava a verificação do correto posicionamento e estado do papel filtrante (círculo branco interno ao filtro), e de outro, para a Peça B, o correto posicionamento da membrana auto-vedante (cilindro de borracha).

### **3.2 O caso da Peça A**

A Peça A, trata-se do filtro de solução. Na Figura 12, pode-se ver algumas imagens que foram utilizadas no projeto. Os filtros exibidos nessa Figura possuem apenas 1cm de diâmetro cada, e o papel filtrante interno a cada filtro, apenas 0.8cm. São objetos bastante pequenos e exigem condições especiais

para uma análise correta.



(a) Papel filtrante em bom estado.

(b) Papel filtrante rasgado.

Figura 12: Peça A em diferentes estados de conservação.

Esses filtros, quando produzidos em uma linha de montagem sem automação, levam ao trabalho humano de análise individual de suas condições. Este projeto procura solucionar este problema de maneira automatizada. O papel filtrante durante a produção pode se encontrar: i) correto, ii) rasgado, ou, iii) inexistente. A condição i) é a esperada, enquanto as outras duas devem ser rejeitadas.

O primeiro desafio encontrado para a Peça A foi como delimitar a área referente ao papel filtrante para fazer uma análise do posicionamento e estado do mesmo. Para delimitar essa área diversas técnicas foram estudadas, mas, a de detecção de círculos foi a mais coerente com o caso. O problema é que, dependendo do ângulo da peça, não é formado um círculo, mas sim um elipse. Por esse motivo, foi designado um padrão de posicionamento para a linha de montagem: a peça deve estar exatamente em frente a câmera, formando um ângulo reto. Um exemplo de cenário hipotético (sem considerar fielmente as proporções reais dos equipamentos) é representado na Figura 13 onde o resultado esperado da captura é semelhante ao exibido na Figura 12.

Para solucionar o caso, a primeira etapa, tratou de detectar o filtro, e, mais precisamente o papel filtrante. Diversas técnicas de processamento de imagens foram testadas e, para a situação definida e os cenários hipotéticos fotografados, apenas um detector de círculos propriamente dito não funcionava com precisão.

Passou a ser necessário utilizar algoritmos de detecção de bordas. Dois algoritmos, entre os principais da literatura, foram amplamente testados e estudados,

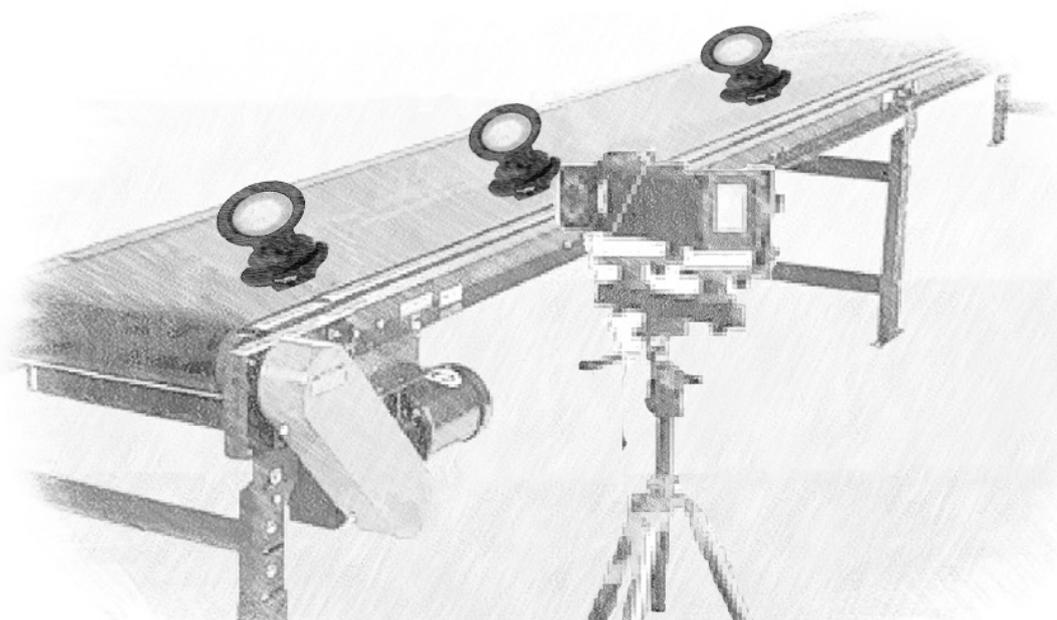


Figura 13: Simulação da Peça A em uma linha de montagem.

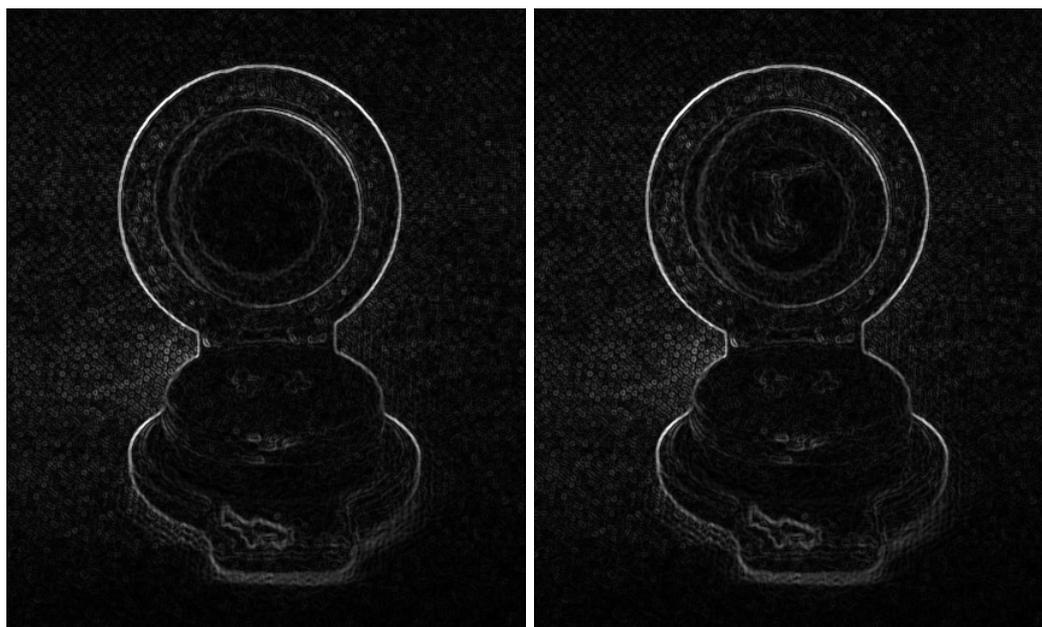
e, também foram explicados neste projeto: o Detector de Bordas Canny e o Sobel. Ambos possuem visualmente um resultado excelente, mas, pelos testes feitos com a Peça A, o Sobel foi mais eficiente para o algoritmo de detecção de círculos.

Pode-se ver na Figura 14, a aplicação do Sobel na Figura 12(a) e 12(b), respectivamente. Claramente, o algoritmo delimita as regiões esperadas de borda da imagem, em tons brancos, e regiões lisas ficam em tons pretos.

Agora, com a detecção de bordas feita corretamente, novamente alguns algoritmos de detecção de círculos existentes na literatura foram testados. Entre eles, o que mais se encaixou no projeto foi o escolhido, trata-se do Detector de Círculos Hough. Esse algoritmo testado e exibido na Figura 15, localiza o círculo do filtro com sucesso, independente do estado do papel filtrante. Entre o processo resultante na Figura 15(a) e na 15(b) há a realização do Sobel exibido na Figura 14(a). A Figura 15(b) exhibe o círculo na Imagem fonte pois o algoritmo detecta o mesmo na imagem gerada pelo Sobel que é passada como entrada, mas o resultado detectado é exibido na Imagem fonte, que tem as mesmas dimensões.

Além da detecção de um círculo, o DCH possui a multi-deteção. O processo e resultado de duas peças simultâneas é exibido na Figura 16. Na Figura 16(a) pode-se observar a imagem dois equipos passadas como parâmetro, na Figura 16(b) pode-se observar o resultado da detecção de bordas Sobel, e por fim, na Figura 16(c) pode-se ver os círculos detectados na Figura 16(b).

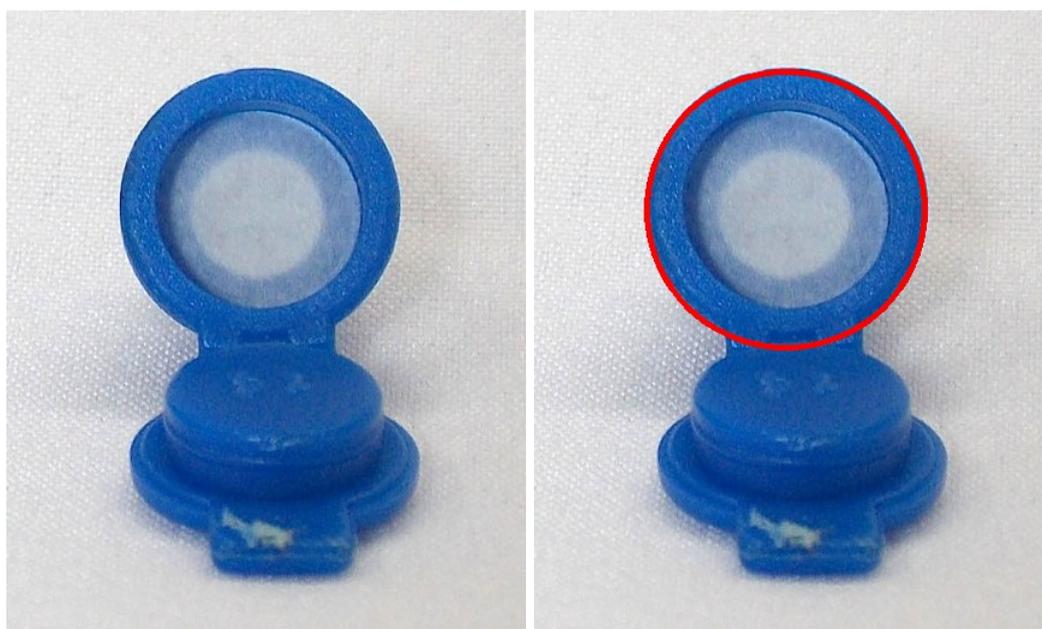
De nada adiantaria apenas detectar o círculo sem recorta-lo. Na Figura 17



(a) Papel filtrante em bom estado.

(b) Papel filtrante rasgado.

Figura 14: Aplicação do Sobel à Peça A.



(a) Imagem fonte.

(b) Após aplicar o DCH.

Figura 15: Detecção do círculo na Peça A.

pode-se observar o resultado do recorte da imagem. Isso é feito ao delimitar a ROI (região de interesse) da imagem, no caso, implementado no Anexo C, cria-se um retângulo com o diâmetro do círculo:  $Rect(center.x-radius, center.y-radius, 2*radius, 2*radius)$ . Isso acaba delimitando exatamente a região que precisa-se analisar.

Agora que o filtro já está recordado e delimitado, há a necessidade de saber

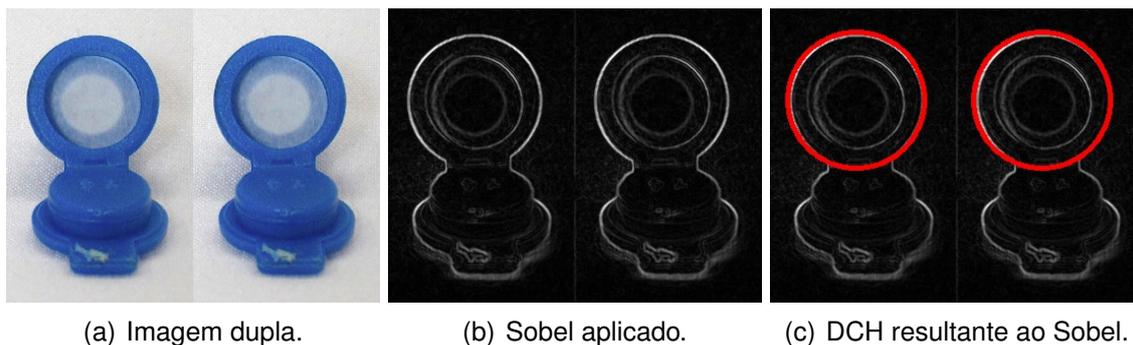


Figura 16: Multi-deteccção: Reconhecimento de dois filtros simultaneamente.



Figura 17: Imagens geradas pelo crop após a detecccção do círculo.

qual o estado do papel filtrante. Como a Peça A seria sempre fotografada na mesma posição e com as mesmas condições de iluminação, a Comparação de Histogramas pareceu o método mais simples e eficiente. Nesse algoritmo, qualquer alteração simples de pixels resulta em diferenças no cálculo dos métodos já apresentados. A Figura 17(a) e a Figura 17(b) foram comparadas com um resultado padrão, respectivamente, na Tabela 1.

Tabela 1: Resultado do Histogram Comparison aplicado à Figura 17.

Método	Resultado padrão	Filtro em bom estado	Filtro defeituoso
Correlação	1.000000	1.000000	0.857419
Chi-quadrado	0.000000	0.000000	7.043819
Intersecção	22.473220	22.473220	17.786991
Bhattacharyya	0.000000	0.000000	0.201498

Pode-se perceber que uma imagem de um filtro padrão, aceito na linha de

montagem, é analisado na coluna Resultado Padrão (o resultado do método Intersecção é diferente de 0 ou 1 por não estar normalizado), e depois, respectivamente, esses resultados são comparados à peça com o filtro em bom estado (Figura 17(a)) e com a peça rasgada (Figura 17(b)). Esses resultados nos indicam que o filtro em bom estado está idêntico ao filtro padrão, e que o filtro defeituoso está diferente.

Após diversos testes, chega-se a conclusão que a Peça A pode ser classificada como **Aceita** caso tenha os Métodos com valores idênticos ao da peça base (Imagem esperada) ou como **Rejeitada** caso tenha esses valores (de um ou mais Métodos) diferentes.

Todo esse processo é realizado em tempo desprezível se comparado ao tempo de uma câmera normal capturar e salvar uma imagem de alta qualidade, ou seja, ganha-se o desempenho e agilidade que uma linha de montagem necessita.

### 3.3 O caso da Peça B

A Peça B, trata-se do injetor lateral do tipo ipsilone. Na Figura 18, pode-se ver algumas imagens que foram utilizadas no projeto. O injetor possui 3.5cm de comprimento e a membrana auto-vedante (cilindro de borracha) apenas 0.8cm de diâmetro, sendo assim, exigem condições especiais para uma análise correta assim como a Peça A.

Esses injetores, quando produzidos em uma linha de montagem sem automação, levam ao trabalho humano de análise individual de suas condições. Este projeto procura solucionar este problema de maneira automatizada. O cilindro de borracha, que é o objeto de interesse a ser analisado nessa peça, pode se encontrar durante a produção com o posicionamento: i) correto, ii) inclinado em relação a base, ou, iii) inexistente. A condição i) é a esperada, enquanto a ii) deve ser analisada qual o ângulo de inclinação e a iii) rejeitada.

Um exemplo de cenário hipotético (sem considerar fielmente as proporções reais dos equipos) para a análise dessa peça é representado na Figura 19 onde o resultado esperado da captura é semelhante ao exibido na Figura 18(a).

O primeiro desafio encontrado para a Peça B foi como detectar o cilindro de borracha interno ao injetor para a análise do posicionamento e estado do mesmo. Para detectar essa peça diversas técnicas foram pensadas, mas não havia nenhum detector que pudesse ter um resultado eficiente semelhante ao que foi feito na Peça A com o DCH. O cilindro de borracha, visualizado lateralmente como na Figura 18(a) é próximo a um retângulo, mas um detector de retângulos através das técnicas existentes não gerava o reconhecimento do cilindro de borracha,

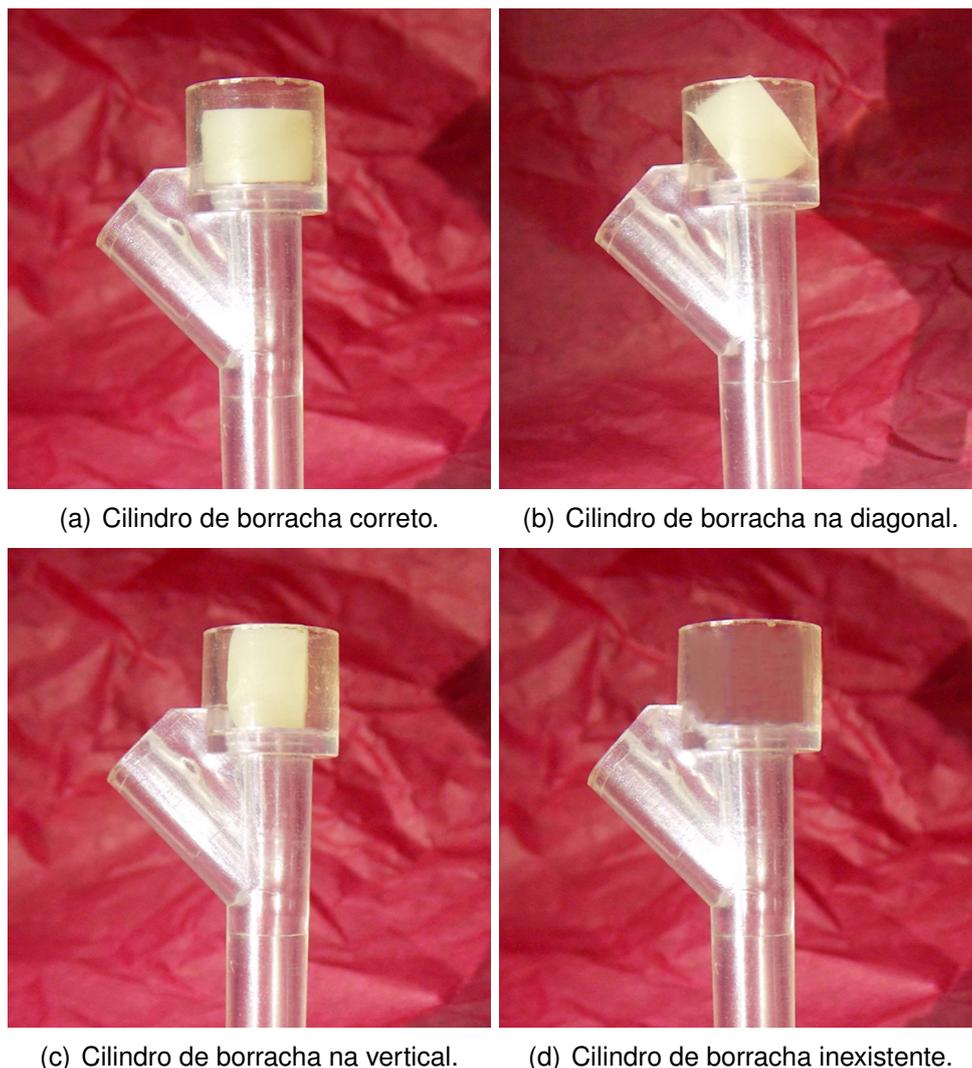


Figura 18: Peça B com diferentes posicionamentos do cilindro de borracha.

nem depois de aplicar algoritmos de detecção de bordas. Então, foi pensado em um algoritmo de detecção de cores, mas mais uma vez sem sucesso, pois pelo padrão RGB e HSV, algumas regiões do material translúcido do injetor geravam os mesmos valores de cor encontrados no cilindro de borracha.

Foi então que a técnica de Comparação de Histogramas foi aplicada à peça e gerou bons resultados. Ela é útil para aceitar rapidamente um injetor e não precisar calcular seu ângulo de inclinação. O funcionamento já abordado na outra peça continua o mesmo, a imagem de um injetor aceito, com o cilindro de borracha corretamente posicionado, tem seus métodos calculados e comparados com a imagem gerada pela câmera no momento da produção. Na Tabela 2, abaixo, pode-se conferir a Figura 18(a) sendo utilizada como o Resultado padrão, a Figura 18(b) comparada na Análise 1, a Figura 18(c) comparada na Análise 2 e a Figura 18(d) comparada na Análise 3.

Conclui-se que em nenhuma das três análises os equipamentos estão aceitos,

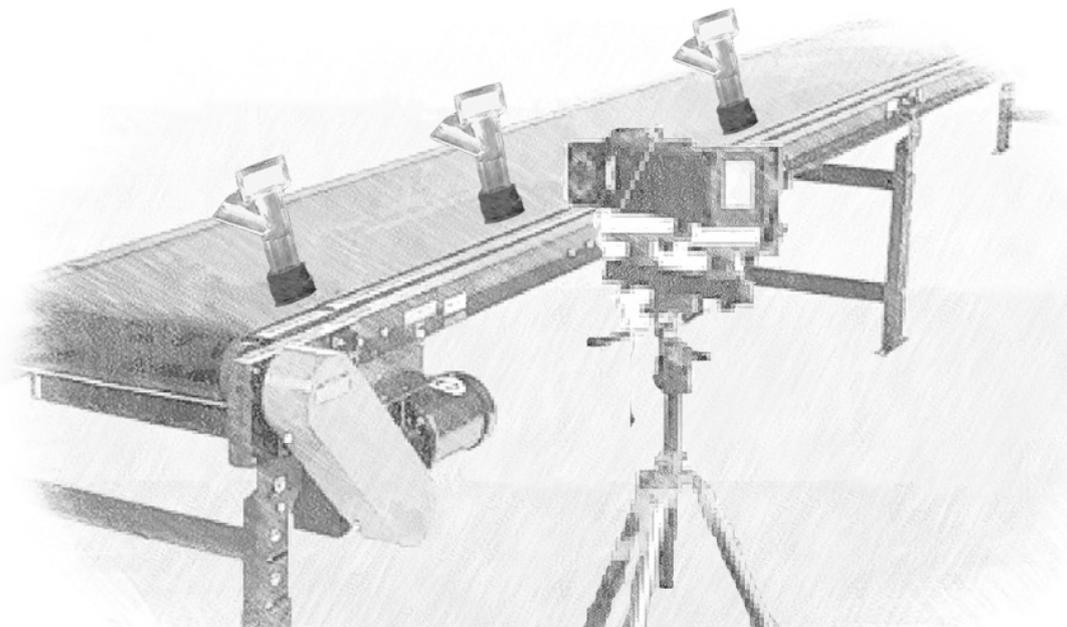


Figura 19: Simulação da Peça B em uma linha de montagem.

Tabela 2: Resultado da Comparação de Histogramas na Peça B.

Método	Resultado padrão	Análise 1	Análise 2	Análise 3
Correlação	1.000000	0.965170	0.981092	0.964091
Chi-quadrado	0.000000	3.572390	2.837325	10.560724
Intersecção	17.425112	12.694655	14.035284	15.338859
Bhattacharyya	0.000000	0.188681	0.160143	0.225030

como era esperado. A próxima etapa, então, ainda era de detectar o cilindro de borracha. Uma técnica simples mas poderosa foi aplicada: Casamento de Padrões. Nessa técnica em que compara-se uma Imagem Molde à uma Imagem Fonte, consegue-se facilmente detectar com precisão, na Peça B, onde está o cilindro de borracha devido a um fator importante: a região mais semelhante é detectada. Na Figura 20 pode-se ver diversos testes dessa técnica na detecção do cilindro de borracha.

A Figura 20(a) foi a Imagem Molde passada ao algoritmo, recortada da Figura 18(a). Os resultados seguintes foram gerados ao analisar outras imagens. As Imagens Fonte que geraram os resultados exibidos pelas Figuras 20(b), 20(b) e 20(i) podem ser vistas, respectivamente, nas Figuras 18(b), 18(c) e 18(d). Como pode-se perceber, apenas a Figura 20(i) é um resultado sem exibir um cilindro de borracha, e era exatamente o esperado já que a Imagem Fonte é a de um injetor lateral vazio. As outras imagens utilizadas neste projeto podem ser visualizadas



Figura 20: Aplicação do Casamento de Padrões a diferentes posições do cilindro de borracha.

no servidor da instituição<sup>1</sup>.

Ainda sobre a implementação do Casamento de Padrões, o algoritmo, adaptado ao projeto, já exhibe o resultado recortado (e não destacado como na versão do Referencial Teórico), e salva esse resultado como uma nova imagem.

Agora que já se sabe se o cilindro de borracha está corretamente posicionado ou onde está o cilindro de borracha na imagem analisada, a próxima etapa da Peça B, é saber se o cilindro de borracha está realmente no injetor ou qual a sua posição em relação ao eixo  $x$  (base do injetor). Novamente diversas técnicas foram analisadas e testadas, mas a de detectar as bordas e usar um detector de linhas foi a que gerou melhores resultados.

O detector de bordas utilizado na implementação foi o Canny e o detector de linhas foi o Hough. Combinando variações do DBC e os dois métodos do DLH, chegou-se a conclusão de que para as imagens analisadas, o método Padrão do DLH gera os resultados esperados, pois, a linha definida intercepta o eixo

<sup>1</sup><http://minerva.ufpel.edu.br/~marilton/GuedesLS/>

de análise. Na Figura 21 pode-se ver o resultado gráfico dessa implementação, aplicado a cada uma das imagens, respectivamente, da Figura 20.

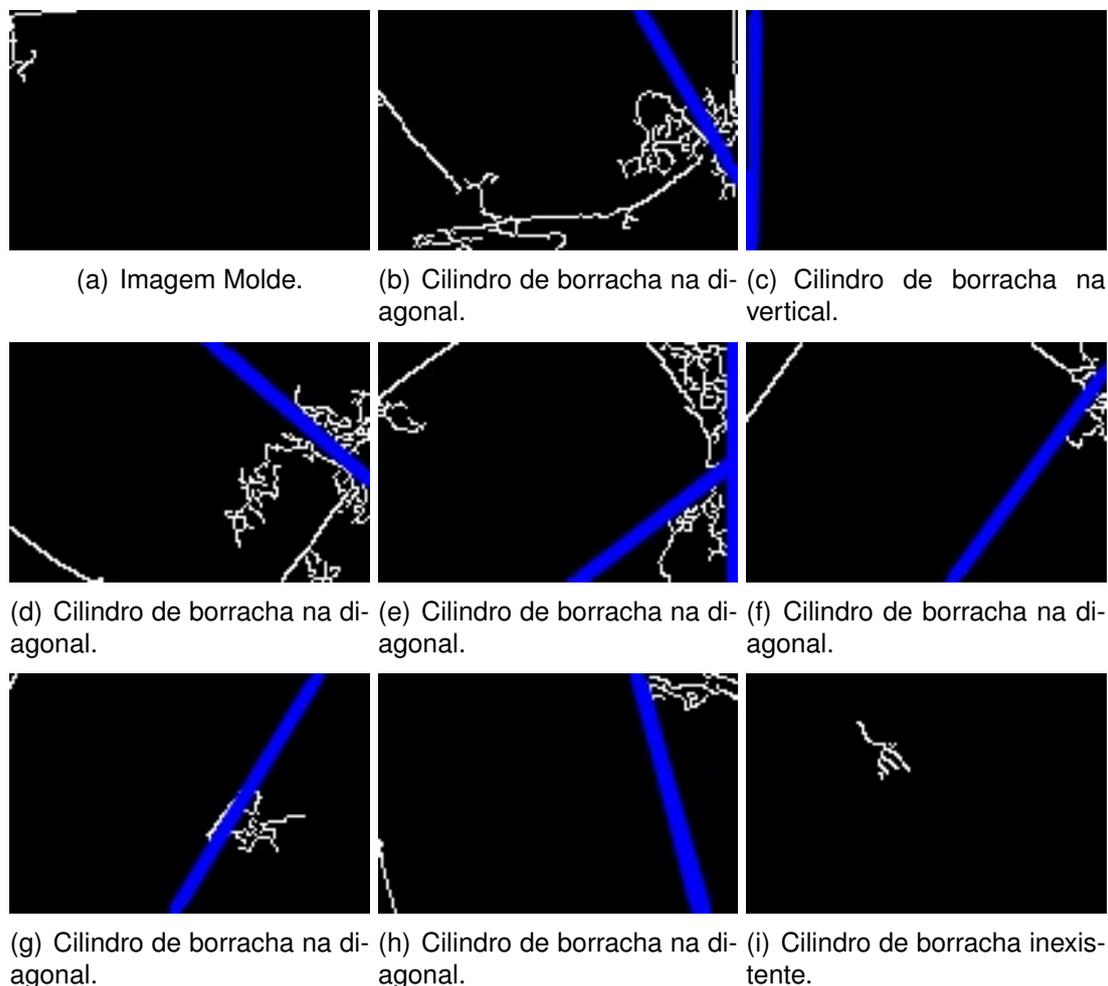


Figura 21: Resultado do DLH aplicado individualmente a cada imagem da Figura 20.

Como esperado, a Figura 21(a) que é a Imagem Molde, não exibe nenhuma linha detectada pelo algoritmo, assim como a Figura 21(i) que é do cilindro de borracha inexistente. As outras imagens detectam com sucesso linhas do cilindro de borracha. Essas linhas podem ser de qualquer um dos lados, já que a inclinação não precisa ser específica em relação a eles, mas sim, um resultado em relação ao eixo. Os casos da Figura 21(e) e da Figura 21(h) detectaram mais de uma linha, então, o resultado final passa a ser o menor ângulo gerado.

A linha é definida por dois pontos, o Ponto Inicial, definido no código em OpenCV como `Point ponto_i (pontocvRound(x0 + alpha*(-sin_t)), cvRound(y0 + alpha*cos_t));` e o Ponto Final, como `Point ponto_f (cvRound(x0 - alpha*(-sin_t)), cvRound(y0 - alpha*cos_t));`, depois, é chamada a função `line` que desenha a ligação entre esses pontos. E o ângulo encontrado, é gerado por  $(\text{ponto\_f.y} - \text{ponto\_i.y}) / (\text{ponto\_f.x} - \text{ponto\_i.x})$ .

i . x).

Na Tabela 3 pode-se ver os resultados numéricos das retas localizadas, assim como o ângulo de inclinação gerado em relação ao eixo  $x$ .

Tabela 3: Resultado do DLH aplicado à Figura 21.

Imagem analisada	Ponto Inicial	Ponto Final	Ângulo de Inclinação
Figura 21(a)	-	-	-
Figura 21(b)	(-427,-908)	(573,824)	59.999271
Figura 21(c)	(-14,1000)	(20,-1000)	89.026062
Figura 21(d)	(-723,-693)	(787,619)	40.986534
Figura 21(e)*	(-729,694)	(868,-510)	37.013134
Figura 21(f)	(-496,876)	(680,-742)	53.989471
Figura 21(g)	(-431,908)	(599,-807)	59.011688
Figura 21(h)*	(-187,-987)	(364,936)	74.011322
Figura 21(i)	-	-	-

A Figura 21(a) e a Figura 21(i), como já explicado, não apresentam linhas, conseqüentemente não apresentam um ângulo de Inclinação. Também pode-se perceber que a 21(e) e a Figura 21(h) possuem um asterisco na Tabela 3 pois essas imagens são as que possuem mais de uma linha, sendo assim, os pontos e ângulos exibidos, são da menor inclinação encontrada.

Como a Figura 21(a) não seria analisada em aplicações reais pois é retirada do injetor na posição ideal, que já seria **Aceito** na primeira etapa da CH, essa imagem pode ser ignorada, o que conseqüentemente mostra, que toda imagem que chegar a essas etapas e não possuir linhas, é uma imagem de um cilindro de borracha inexistente, ou seja, **Rejeitado**. Assim como, todas as outras que tiverem linhas, são classificadas como **Mal Posicionadas** e mostram o ângulo gerado.

Assim como na Peça A, todo esse processo é realizado em tempo desprezível se comparado ao tempo de uma câmera normal capturar e salvar uma imagem de alta qualidade, ou seja, ganha-se o desempenho e agilidade que uma linha de montagem necessita.

### 3.4 Proposta dos Aspectos de Infra-estrutura

Atualmente há uma forte recomendação de utilizar produtos específicos para solucionar problemas digitais que não exigem o processamento de um computador com todos os seus periféricos. Com essa ação, um projeto reduz custos e espaço físico. No problema abordado neste projeto, por se tratar de uma linha de montagem, poderíamos adaptar para situações específicas que reduziriam

custo e espaço. Abaixo são mostradas alternativas que podem ser aplicadas ao projeto.

### 3.4.1 Microcontrolador

Um microcontrolador é um computador-num-chip, contendo um processador, memória e periféricos de entrada/saída. É um microprocessador que pode ser programado para funções específicas, em contraste com outros microprocessadores de propósito geral (como os utilizados nos PCs). Eles são embarcados no interior de algum outro dispositivo (geralmente um produto comercializado) para que possam controlar as funções ou ações do produto. Pode-se ver na Figura 22 um exemplo de Microcontrolador<sup>2</sup>.



Figura 22: Um microcontrolador PIC18F8720 num encapsulamento TQFP de 80 pinos.

### 3.4.2 FPGA

Um FPGA é um hardware reconfigurável nos quais uma matriz de blocos lógicos configuráveis (*Configurable Logic Blocks* - CLBs), conectados por canais também programáveis, pode ser configurada para desempenhar um conjunto de funções lógicas e dar origem a qualquer tipo de sistema digital (WILSON, 2007). Esta alternativa, assim como o Microcontrolador, reduziria a necessidade de um computador dedicado e seria configurado exclusivamente para resolver o problema abordado no projeto. Pode-se ver na Figura 23 um exemplo de FPGA<sup>3</sup>.

<sup>2</sup><http://upload.wikimedia.org/wikipedia/commons/1/18/PIC18F8720.jpg>

<sup>3</sup>[http://upload.wikimedia.org/wikipedia/commons/f/fa/Altera\\_StratixIVGX\\_FPGA.jpg](http://upload.wikimedia.org/wikipedia/commons/f/fa/Altera_StratixIVGX_FPGA.jpg)



Figura 23: FPGA fabricado pela Altera modelo Stratix IV.

### 3.4.3 Smartphone

Extremamente popular hoje em dia, um *Smartphone* é um telefone celular com funcionalidades avançadas que podem ser estendidas por meio de programas executados por seu sistema operacional. Os sistemas operacionais dos *Smartphones* permitem que possam ser adicionados diversos aplicativos de funções específicas. Essa alternativa seria válida para uso artesanal, verificando a qualidade de peças fabricadas, mas dependendo das configurações em nada deixaria a desejar se comparado a um computador dedicado, além da portabilidade dos aparelhos assim como a redução do espaço físico. Pode-se ver na Figura 24 um exemplo de um Smartphone<sup>4</sup>.



Figura 24: *Smartphone* Samsung S4 Mini com o sistema operacional Android.

<sup>4</sup><http://www.samsung.com/br/consumer/cellular-phone/cellular-phone-tablets/smartphones/GT-I9195ZWLZVV>

### 3.4.4 *Tablet*

O *Tablet* também é uma alternativa, similar ao *Smartphone* este produto possui funcionalidades avançadas que podem ser estendidas por meio de programas executados por seu sistema operacional. Apresenta uma tela sensível ao toque que é o dispositivo de entrada principal. Este produto possui uma tela maior que do *Smartphone*, sendo útil, por exemplo, em casos de visualização de estatísticas dos equipos analisados simultaneamente a análise em tempo real, permitindo maior interação com as informações. Pode-se ver na Figura 25 o exemplo de um *Tablet*<sup>5</sup>.



Figura 25: *Tablet* Apple iPad Air com o sistema operacional iOS.

---

<sup>5</sup><http://www.apple.com/ipad-air/specs/>

## 4 CONCLUSÕES

O presente trabalho analisou e estudou o caso específico de qualidade de uma linha de produção de equipamentos médicos, chamados *Equipos*. Aplicando-se algoritmos de Processamento de Imagens, é possível avaliar a qualidade destes equipamentos, e seus componentes, por meio de uma câmera e um computador (ou outro dispositivo proposto). Isto possibilita a detecção de defeitos inerentes a todo processo de fabricação.

Os algoritmos aplicados no trabalho ficaram centrados em: Detector de Bordas Sobel, Detector de Bordas Canny, Detector de Círculos e de Linhas Hough, Histograma, Comparação de Histograma, Convolução e Casamento de Padrões.

Para a Peça A, houve a verificação do correto posicionamento e estado do papel filtrante (círculo interno ao filtro), sendo classificado como Aceito ou Rejeitado. Os testes mostraram a eficiência dos algoritmos para a solução do problema, que foram: Detector de Bordas Sobel, Detecção de Círculos Hough e Comparação de Histograma.

Para a Peça B, houve a verificação do correto posicionamento da membrana auto-vedante (cilindro de borracha), sendo classificado como Aceito, Rejeitado, ou Mal Posicionado, exibindo o grau de inclinação encontrado. Os testes mostraram a eficiência dos algoritmos para a solução do problema, que foram: Comparação de Histograma, Casamento de Padrões e Detecção de Linhas Hough.

Os próximos passos deste trabalho são: (i) buscar e avaliar outros algoritmos de processamento de imagem consistentes com o processo de detecção de falhas para Equipos (e afins); (ii) avaliar o custo do processo de análise e o desempenho da ferramenta de análise; (iii) estudar e aplicar a uma linha de montagem real; (iv) analisar sistemas de iluminação automática.

## REFERÊNCIAS

BRADSKI, G.; KAEHLER, A. **Learning OpenCV**. New Jersey, USA: O'Reilly Media Inc., 2008.

CELEBI, M.; ASLANDOGAN, Y.; BERGSTRESSER, P. Unsupervised border detection of skin lesion images. In: INTERNATIONAL CONFERENCE ON INFORMATION TECHNOLOGY: CODING AND COMPUTING (ITCC 2005), 2005. **Proceedings...** [S.l.: s.n.], 2005. v.2, p.123 – 128.

CELEBI, M. E.; KINGRAVI, H. A.; LEE, J.; STOECKER, W. V.; MALTERS, J. M.; IYATOMI, H.; ASLANDOGAN, Y. A.; MOSS, R.; MARGHOOB, A. A. **Fast and Accurate Border Detection in Dermoscopy Images Using Statistical Region Merging**.

CHUNG, D. H.; SAPIRO, G. Segmenting skin lesions with partial-differential-equations-based image processing algorithms. **IEEE Trans. Medical Imaging**, [S.l.], v.19, n.7, p.763–767, July 2000.

ERKOL, B.; MOSS, R. H.; STANLEY, R. J.; STOECKER, W. V.; HVATUM, E. Automatic lesion boundary detection in dermoscopy images using gradient vector flow snakes. **IEEE Trans. Medical Imaging**, [S.l.], v.11, n.1, p.17–26, 2005.

FARID, H. **Fundamentals of Image Processing**. URL: <http://www.cs.dartmouth.edu/farid/downloads/tutorials/fip.pdf/> [Online, último acesso em 25 de janeiro de 2014.].

GANSTER, H.; PINZ, P.; ROHRER, R.; WILDLING, E.; BINDER, M.; KITTLER, H. Automated melanoma recognition. **IEEE Trans. Medical Imaging**, [S.l.], v.20, n.3, p.233–239, Mar. 2001.

GAO, J.; ZHANG, J.; FLEMING, M. A novel multiresolution color image segmentation technique and its application to dermatoscopic image segmentation. In: INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 2000. **Proceedings...** [S.l.: s.n.], 2000. v.3, p.408 –411.

HUAMÁN, A. **Sobel Derivatives**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel\\_derivatives/sobel\\_derivatives.html](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/sobel_derivatives/sobel_derivatives.html) [Online, último acesso em 25 de janeiro de 2014.].

HUAMÁN, A. **Canny Edge Detector**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny\\_detector/canny\\_detector.html](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/canny_detector/canny_detector.html) [Online, último acesso em 25 de janeiro de 2014.].

HUAMÁN, A. **Hough Circle Transform**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough\\_circle/hough\\_circle.html](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html) [Online, último acesso em 25 de janeiro de 2014.].

HUAMÁN, A. **Hough Lines**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough\\_lines/hough\\_lines.html](http://docs.opencv.org/doc/tutorials/imgproc/imgtrans/hough_lines/hough_lines.html) [Online, último acesso em 25 de janeiro de 2014.].

HUAMÁN, A. **Histogram Comparison - compareHist**. URL: <http://docs.opencv.org/modules/imgproc/doc/histograms.html?highlight=comparehist#comparehist> [Online, último acesso em 25 de janeiro de 2014.].

HUAMÁN, A. **Histogram Comparison**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram\\_comparison/histogram\\_comparison.html](http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_comparison/histogram_comparison.html) [Online, último acesso em 25 de janeiro de 2014.].

HUAMÁN, A. **Template Matching**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/histograms/template\\_matching/template\\_matching.html#template-matching](http://docs.opencv.org/doc/tutorials/imgproc/histograms/template_matching/template_matching.html#template-matching) [Online, último acesso em 25 de janeiro de 2014.].

IJHIS. **The international journal of hybrid intelligent systems**. URL: <http://ijhis.hybridsystem.com/> [Online, último acesso em 25 de janeiro de 2014.].

LYRA, A. **Uma fundamentação matemática para o processamento de imagens digitais intervalares**. 2003. Tese (Doutorado em Ciência da Computação) — .

OPENCV. **OpenCV**. URL: <http://www.opencv.org> [Online, último acesso em 25 de janeiro de 2014.].

PEDRINI, H.; SCHWARTZ, W. **Análise de Imagens Digitais: Princípios, Algoritmos e Aplicações**. São Paulo, Brasil: Thomson Learning, 2008.

RUBEGNI, P.; FERRARI, A.; CEVENINI, G.; PICCOLO, D.; BURRONI, M.; PEROTTI, R.; PERIS, K.; TADDEUCCI, P.; BIAGIOLI, M.; DELL'EVA, G.; CHIMENTI,

S.; ANDREASSI, L. Differentiation between pigmented spitz naevus and melanoma by digital dermoscopy and stepwise logistic discriminant analysis. In: OF, 2001. **Proceedings...** [S.l.: s.n.], 2001. v.11, n.1, p.37–44.

SCHIELE, B.; CROWLEY, J. L. Object recognition using multidimensional receptive field histograms. In: EUROPEAN CONFERENCE ON COMPUTER VISION, 1996. **Anais...** [S.l.: s.n.], 1996. p.610–619.

SCHMID, P. Lesion detection in dermatoscopic images using anisotropic diffusion and morphological flooding. In: IMAGE PROCESSING, 1999. ICIP 99. PROCEEDINGS. 1999 INTERNATIONAL CONFERENCE ON, 1999. **Anais...** [S.l.: s.n.], 1999. v.3, p.449–453 vol.3.

SINHA, U. **Convolutions**. URL: <http://www.aishack.in/2010/08/convolutions> [Online, último acesso em 25 de janeiro de 2014.].

SWAIN, M. J.; BALLARD, D. H. Color indexing. In: INTERNATIONAL JOURNAL OF COMPUTER VISION 7, 1991. **Anais...** [S.l.: s.n.], 1991. p.11–32.

TAKAHASHI, A. **Extensões intervalares do método de segmentação de imagens digitais por k-means: estudos comparativos e de caso**. 2005. Tese (Doutorado em Ciência da Computação) — .

WANNER, J. **Histogram Equalization**. URL: [http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram\\_equalization/histogram\\_equalization.html](http://docs.opencv.org/doc/tutorials/imgproc/histograms/histogram_equalization/histogram_equalization.html) [Online, último acesso em 25 de janeiro de 2014.].

WILSON, P. **Design Recipes for FPGAs**. Burlington, MA, UK: Newnes, 2007.

YU, C. **Canny Edge Detection and Hough Transform**. URL: [http://www.indiana.edu/~d11/B657/B657\\_lec\\_hough.pdf](http://www.indiana.edu/~d11/B657/B657_lec_hough.pdf) [Online, último acesso em 25 de janeiro de 2014.].

## ANEXO A DETECTOR DE BORDAS SOBEL

```
1 #include "opencv2/imgproc/imgproc.hpp"
2 #include "opencv2/highgui/highgui.hpp"
3 #include <stdlib.h>
4 #include <stdio.h>
5 #include <sstream>
6
7 using namespace cv;
8 using namespace std;
9
10 int main( int , char** argv )
11 {
12
13     Mat src , src_gray;
14     Mat grad;
15     int scale = 1;
16     int delta = 0;
17     int ddepth = CV_16S;
18     string nick = (argv[2]);
19
20     /// Carrega uma imagem
21     src = imread( argv[1] );
22
23     if( !src.data )
24         { return -1; }
25
26     /// Aplica um Gaussian Blur para reduzir o ruído
27     GaussianBlur( src , src , Size(3,3), 0, 0, BORDER_DEFAULT );
28
29     /// Converte para cinza
30     cvtColor( src , src_gray , COLOR_RGB2GRAY );
31
32     /// Gera as "derivadas" de x e y
33     Mat grad_x , grad_y;
34     Mat abs_grad_x , abs_grad_y;
35
```

```
37  /// Gradiente X
Sobel( src_gray, grad_x, ddepth, 1, 0, 3, scale, delta,
      BORDER_DEFAULT );
convertScaleAbs( grad_x, abs_grad_x );
39
41  /// Gradiente Y
Sobel( src_gray, grad_y, ddepth, 0, 1, 3, scale, delta,
      BORDER_DEFAULT );
convertScaleAbs( grad_y, abs_grad_y );
43
45  /// Gradiente total aproximado
addWeighted( abs_grad_x, 0.5, abs_grad_y, 0.5, 0, grad );
47
49  /// Mostra os resultados
namedWindow( "Sobel – Detector de Bordas", WINDOW_AUTOSIZE );
imshow( "Sobel – Detector de Bordas", grad );

51  stringstream ss;
string type = ".jpg";
53  ss << nick << type;
string saida = ss.str();
55  imwrite(saida, grad);

57  waitKey(0);
return 0;
59
}
```

Listing A.1: Sobel em linguagem C++ para OpenCV.

## ANEXO B DETECTOR DE BORDAS CANNY

```
1 #include "opencv2/imgproc/imgproc.hpp"
2 #include "opencv2/highgui/highgui.hpp"
3 #include <stdlib.h>
4 #include <stdio.h>
5
6 using namespace cv;
7
8 /// Variáveis Globais
9 Mat src , src_gray;
10 Mat dst , detected_edges;
11 int edgeThresh = 1;
12 int lowThreshold;
13 int const max_lowThreshold = 100;
14 int ratio = 3;
15 int kernel_size = 3;
16 char* window_name = "Canny";
17
18 void CannyThreshold(int , void*)
19 {
20     /// Reduz ruídos com uma matriz 3x3
21     blur( src_gray , detected_edges , Size(3,3) );
22
23     /// Canny detector
24     Canny( detected_edges , detected_edges , lowThreshold , lowThreshold*
25           ratio , kernel_size );
26
27     /// Gera a saída como uma máscara, e mostramos o resultado
28     dst = Scalar::all(0);
29     src.copyTo( dst , detected_edges );
30     imshow( window_name , dst );
31 }
32
33 int main( int argc , char** argv )
34
```

```
{
36  /// Carrega uma imagem
    src = imread( argv[1] );
38
40  if( !src.data )
    { return -1; }
42
    /// Cria uma matrix do mesmo tipo e tamanho da origem
    dst.create( src.size(), src.type() );
44
    /// Converte a imagem para tons de Cinza
46  cvtColor( src, src_gray, CV_BGR2GRAY );
48
    /// Cria uma janela
    namedWindow( window_name, CV_WINDOW_AUTOSIZE );
50
    /// Cria uma Trackbar para o usuário selecionar o nível
52  createTrackbar( "Nivel:", window_name, &lowThreshold,
        max_lowThreshold, CannyThreshold );
54
    /// Mostra a imagem
    CannyThreshold(0, 0);
56
    /// Espera até alguma tecla ser digitada para fechar o programa
58  waitKey(0);
60
    return 0;
}
```

Listing B.1: Detector de Bordas Canny em linguagem C++ para OpenCV.

## ANEXO C DETECTOR DE CÍRCULOS HOUGH

```
1 #include "opencv2/highgui/highgui.hpp"
2 #include "opencv2/imgproc/imgproc.hpp"
3 #include <iostream>
4 #include <stdio.h>
5 #include <sstream>
6
7 using namespace cv;
8 using namespace std;
9
10 int main(int argc, char** argv)
11 {
12     Mat src, ori, src_gray;
13     string nick = (argv[3]);
14
15     /// Le as imagens
16     src = imread( argv[1], 1 );
17     ori = imread( argv[2], 1 );
18
19     if(argc != 4)
20     {
21         cout << "Uso: " << argv[0] << " <sobel file> <original file> <
22             output name>" << endl;
23         return -1;
24     }
25
26     /// Converte para cinza
27     cvtColor( src, src_gray, CV_BGR2GRAY );
28
29     /// Reduz o ruído para não detectar falsos círculos
30     GaussianBlur( src_gray, src_gray, Size(9, 9), 2, 2 );
31
32     vector<Vec3f> circles;
33
34     /// Aplica a Transformada Hough para encontrar os círculos
```

```
HoughCircles( src_gray , circles , CV_HOUGH_GRADIENT, 1, src_gray .
    rows/8 , 200, 100, 0, 0 );
35
Mat image_roi;
37 // Desenha o circulo detectado
for( size_t i = 0; i < circles.size(); i++ )
39 {
    Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
41 int radius = cvRound(circles[i][2]);
    circle( ori , center , radius , Scalar(0,0,255), 3, 8, 0 );
43 Rect region_of_interest = Rect(center.x-radius ,center.y-radius
    ,2*radius,2*radius);
    image_roi = ori(region_of_interest);
45 }

47 // Mostra os resultados
namedWindow( "Hough – Transformada de Decteccao de Circulos" ,
    CV_WINDOW_AUTOSIZE );
49 imshow( "Hough – Transformada de Decteccao de Circulos" , image_roi)
    ;

51 stringstream ss;
    string type = ".jpg";
53 ss << nick << type;
    string saida = ss.str();
55 imwrite(saida , image_roi);

57 waitKey(0);
    return 0;
59 }
```

Listing C.1: Detector de Círculos Hough em linguagem C++ para OpenCV.

## ANEXO D DETECTOR DE LINHAS HOUGH

```
1 #include "opencv2/highgui/highgui.hpp"
2 #include "opencv2/imgproc/imgproc.hpp"
3 #include <iostream>
4 #include <stdio.h>
5 #include <sstream>
6
7 using namespace cv;
8 using namespace std;
9
10 /// Variaveis Globais
11 Mat src , edges;
12 Mat src_gray;
13 Mat standard_hough , probabilistic_hough;
14 int min_threshold = 50;
15 int max_trackbar = 150;
16
17 const char* standard_name = "Deteccao de Linhas Hough Padrao";
18 const char* probabilistic_name = "Deteccao de Linhas Hough
19     Probabilistica";
20
21 /// Trackbar inicia em 0
22 int s_trackbar = 0;
23 int p_trackbar = max_trackbar;
24
25 /// Cabecalho de funcoes
26 void help();
27 void Standard_Hough( int , void* );
28 void Probabilistic_Hough( int , void* );
29
30 /// Funcao principal
31 int main( int , char** argv )
32 {
33     /// Le a imagem
34     src = imread( argv[1], 1 );
```

```

35  if( src.empty() )
    { help();
37      return -1;
    }

39
41  /// Transforma a imagem em tons de cinza
    cvtColor( src, src_gray, COLOR_RGB2GRAY );

43  /// Aplica a DBC
    Canny( src_gray, edges, 1, 300, 3 );

45
47  /// Cria a trackbar para os limites
    char thresh_label[50];
    sprintf( thresh_label, "Varia de %d ate %d", min_threshold,
49         max_trackbar );

51  namedWindow( standard_name, WINDOW_AUTOSIZE );
    createTrackbar( thresh_label, standard_name, &s_trackbar,
        max_trackbar, Standard_Hough );

53  //namedWindow( probabilistic_name, WINDOW_AUTOSIZE );
    //createTrackbar( thresh_label, probabilistic_name, &p_trackbar,
        max_trackbar, Probabilistic_Hough );

55
57  /// Inicializa
    Standard_Hough(0, 0);
    //Probabilistic_Hough(0, 0);
59  waitKey(0);
    return 0;
61 }

63 /// Funcao de ajuda em caso de erro
void help()
65 {
    printf("\n\tDeteccao de Linhas Hough \n ");
67     printf("\tUso: ./algoritmo <nome_da_imagem>\n\n");
}

69
71 /// Hough Padrao
void Standard_Hough( int, void* )
{
73     vector<Vec2f> s_lines;
    cvtColor( edges, standard_hough, COLOR_GRAY2BGR );

75
77     /// Transformada padrao
    HoughLines( edges, s_lines, 1, CV_PI/180, min_threshold + s_trackbar,
        0, 0 );

```

```

79  /// Calcula e exhibe o resultado
float m, r;
81  for( size_t i = 0; i < s_lines.size(); i++ )
    {
83      float r = s_lines[i][0], t = s_lines[i][1];
double cos_t = cos(t), sin_t = sin(t);
85      double x0 = r*cos_t, y0 = r*sin_t;
double alpha = 1000;

87      Point ponto_i( cvRound(x0 + alpha*(-sin_t)), cvRound(y0 + alpha*
cos_t) );
89      Point ponto_f( cvRound(x0 - alpha*(-sin_t)), cvRound(y0 - alpha*
cos_t) );
line( standard_hough, ponto_i, ponto_f, Scalar(255,0,0), 3,
CV_AA);
91      m = ((float) (ponto_f.y - ponto_i.y))/((float) (ponto_f.x -
ponto_i.x));
r = fabs(atan(m)*180.0f)/3.14159265359f;
93      printf("Linha: ponto_i(%d,%d) e ponto_f(%d,%d)\n",ponto_i.x,
ponto_i.y, ponto_f.x,ponto_f.y);
printf("Angulo = %f\n",r);
95    }
imshow( standard_name, standard_hough );
97
/// Salva a imagem resultante
99  stringstream ss;
string type = ".jpg";
101  ss << "yXvhlc" << type;
string saida = ss.str();
103  imwrite(saida, standard_hough);
}
105
/// Hough Probabilistico
107 void Probabilistic_Hough( int, void* )
{
109     vector<Vec4i> p_lines;
cvtColor( edges, probabilistic_hough, COLOR_GRAY2BGR );
111
/// Transformada probabilistica
113 HoughLinesP( edges, p_lines, 1, CV_PI/180, min_threshold + p_trackbar
, 30, 10 );

115 /// Calcula e exhibe o resultado
for( size_t i = 0; i < p_lines.size(); i++ )
117     {
Vec4i l = p_lines[i];

```

```
119         line( probabilistic_hough , Point(l[0], l[1]), Point(l[2], l[3]),  
              Scalar(255,0,0), 3, CV_AA);  
121     }  
    //imshow( probabilistic_name , probabilistic_hough );  
}
```

Listing D.1: Detector de Linhas Hough em linguagem C++ para OpenCV.

## ANEXO E HISTOGRAMA

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4 #include <string.h>
5 #define WIDTH 507
6 #define HEIGHT 384
7 #define TRUE (0==0)
8 #define FALSE (0==1)
9
10 int loadRawFile(char * filename, int x, int y, unsigned char **data);
11 int loadPPMGrayFile(char * filename, int x, int y, unsigned char **data
12 );
13 int initData(unsigned char **data, unsigned char **frame, int imageX,
14 int imageY);
15 int freeData(unsigned char **data, unsigned char **frame, int imageX,
16 int imageY);
17
18 int main(int argc, char **argv)
19 {
20     unsigned char **data;
21     unsigned char **frame;
22
23     int i=0;
24     int j=0;
25     int imageX = WIDTH;
26     int imageY = HEIGHT;
27
28     initData(data, frame);
29
30     /// Carrega a imagem de um arquivo
31     if (!loadRawFile("lena.raw", WIDTH,HEIGHT, data))
32         return FALSE;
33
34     int histogram[256];
35     /// Calcula the histogram
```

```
34 for(i = 0; i < (WIDTH) ; i++)
    {
36     for(j = 0 ; j < (HEIGHT) ; j++)
        {
38         histogram[data[i][j]]++;
        }
    }
40
42 printf("\nHistogram [intensity 0–255] : [pixel count] \n");
44 for(i = 0 ; i < 256 ; i++)
    {
46     printf("%d : %d \n",i ,histogram[i]);
    }
48
49 /// Libera memoria
50 if (!freeData(data , frame ,WIDTH,HEIGHT))
51     return FALSE;
52 return TRUE;
53 }
```

Listing E.1: Implementação do Histograma em linguagem C++.

## ANEXO F COMPARAÇÃO DE HISTOGRAMA

```
1 #include "opencv2/highgui/highgui.hpp"
2 #include "opencv2/imgproc/imgproc.hpp"
3 #include <iostream>
4 #include <stdio.h>
5
6 using namespace std;
7 using namespace cv;
8
9
10 int main( int argc, char** argv )
11 {
12     Mat src_base, hsv_base;
13     Mat src_test1, hsv_test1;
14
15     /// Carrega a imagem original e a imagem a ser comparada
16
17     src_base = imread( argv[1], 1 );
18     src_test1 = imread( argv[2], 1 );
19
20     if( argc < 2 )
21     {
22         printf("Erro. Uso: ./algoritmo <imagem original> <imagem
23             testada>\n");
24         return -1;
25     }
26
27     /// Converte para HSV
28     cvtColor( src_base, hsv_base, CV_BGR2HSV );
29     cvtColor( src_test1, hsv_test1, CV_BGR2HSV );
30
31     /// Usando 30 intervalos para hue e 32 para saturacao
32     int h_bins = 50; int s_bins = 60;
33     int histSize[] = { h_bins, s_bins };
34
35     /// Hue varia de 0 a 256, saturacao de 0 a 180
36     float h_ranges[] = { 0, 256 };
```

```

float s_ranges[] = { 0, 180 };
36
const float* ranges[] = { h_ranges, s_ranges };
38
/// Usa canal 0 e 1
40 int channels[] = { 0, 1 };
42
/// Histogramas
MatND hist_base;
44 MatND hist_test1;
46
/// Calcula os histogramas para as imagens HSV
calcHist( &hsv_base, 1, channels, Mat(), hist_base, 2, histSize,
         ranges, true, false );
48 normalize( hist_base, hist_base, 0, 1, NORM_MINMAX, -1, Mat() );
50
calcHist( &hsv_test1, 1, channels, Mat(), hist_test1, 2, histSize,
         ranges, true, false );
normalize( hist_test1, hist_test1, 0, 1, NORM_MINMAX, -1, Mat() );
52
54
/// Aplica o metodo de comparacao
for( int i = 0; i < 4; i++ )
56 { int compare_method = i;
    double base_base = compareHist( hist_base, hist_base,
        compare_method );
58    double base_test1 = compareHist( hist_base, hist_test1,
        compare_method );
60
    printf( "\n Metodo [%d] \n Imagem original: %f \n Imagem
        testada: %f \n\n", i, base_base, base_test1);
}
62
return 0;
64 }

```

Listing F.1: Comparação de Histograma em linguagem C++ para OpenCV.

## ANEXO G CONVOLUÇÃO

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <math.h>
4
5 #define KernelX 3
6 #define KernelY 3
7 #define WIDTH 256
8 #define HEIGHT 256
9 #define TRUE (0==0)
10 #define FALSE (0==1)
11
12 int loadRawFile(char * filename, int x, int y, unsigned char **data);
13 int simpleConvolution(unsigned char **data, unsigned char **frame, int
    imageX, int imageY);
14 int initData(unsigned char **data, unsigned char **frame, int imageX,
    int imageY);
15 int freeData(unsigned char **data, unsigned char **frame, int imageX,
    int imageY);
16 float computeConv(int window[][KernelY], float kernel[][KernelY], int
    inv);
17 float sum2(float a, float b, float c);
18 float mul3(int inv, int w, float k);
19 float computeConv2(int a, int b, int c, int d, int e, int f, int g, int
    h, int i, float af, float bf, float cf, float df,
20 float ef, float ff, float gf, float hf, float iff, int inv);
21 int histogram[256];
22
23 int main(int argc, char **argv)
24 {
25     unsigned char **data;
26     unsigned char **frame;
27
28     int i=0;
29     int j=0;
```

```
30 int imageX = WIDTH + ceil((KernelX/2.0f)); /// A imagem eh
    carregada com 2 linhas e colunas extras de pixels
31 int imageY = HEIGHT + ceil((KernelY/2.0f)); /// A imagem eh
    carregada com 2 linhas e colunas extras de pixels
32
33 /// Aloca dados
34 data = (unsigned char **) malloc(imageY*sizeof(unsigned char *));
35 if(data == NULL)
36 {
37     fprintf(stderr, "Nao consegue alocar dados\n");
38     return FALSE;
39 }
40
41 /// Aloca frames
42 frame = (unsigned char **) malloc((imageY - 2)*sizeof(unsigned char
    *));
43 if(frame == NULL)
44 {
45     fprintf(stderr, "Nao consegue alocar dados\n");
46     return FALSE;
47 }
48
49 /// Aloca dados da coluna
50 for( i = 0 ; i < imageY ; i++)
51 {
52     data[i] = (unsigned char *) malloc(imageX*sizeof(unsigned char
    ));
53     if(data[i] == NULL)
54     {
55         fprintf(stderr, "Nao consegue alocar dados\n");
56         return FALSE;
57     }
58
59     for(j = 0 ; j < imageX ; j ++ )
60     {
61         data[i][j] = 0;
62     }
63 }
64
65 /// Aloca frames da coluna
66 for( i = 0 ; i < (imageY - 2) ; i++)
67 {
68     frame[i] = (unsigned char *) malloc((imageX -2 )*sizeof(
    unsigned char));
69     if(frame[i] == NULL)
70     {
71         fprintf(stderr, "Nao consegue alocar dados\n");
```

```

72         return FALSE;
73     }
74
75     for(j = 0 ; j < (imageX -2) ; j ++ )
76     {
77         frame[i][j] = 0;
78     }
79 }
80
81 /// Carrega a imagem
82 if (!loadRawFile("lena.raw", WIDTH,HEIGHT, data))
83     return FALSE;
84
85 /// Computa o histograma
86 for(i = 0; i < (HEIGHT) ; i++)
87 {
88     for(j = 0 ; j < (WIDTH) ; j++)
89     {
90         histogram[data[i][j]]++;
91     }
92 }
93
94 /// Convolucao
95 simpleConvol(data , frame ,WIDTH,HEIGHT) ;
96
97 /// Retorna a imagem em PPM
98 fprintf(stdout, "P6\n%d %d\n255\n",WIDTH,HEIGHT) ;
99 for(i = 0 ; i < (HEIGHT) ; i ++ )
100 {
101     for(j = 0 ; j < (WIDTH) ; j ++ )
102     {
103         fputc(frame[i][j], stdout) ;
104         fputc(frame[i][j], stdout) ;
105         fputc(frame[i][j], stdout) ;
106     }
107 }
108 fflush(stdout) ;
109
110 /// Liberar dados
111 if (!freeData(data , frame ,WIDTH,HEIGHT))
112     return FALSE;
113
114 return TRUE;
115 }
116
117 int freeData(unsigned char **data , unsigned char **frame , int imageX ,
118             int imageY)

```

```
118 {
    int i;
120
    /// Liberar dados
122 for(i = 0 ; i < imageY ; i++)
        free(data[i]);
124
    free(data);
126
    for(i = 0 ; i < imageY - 2 ; i++)
128         free(frame[i]);
130
    free(frame);
132
    return TRUE;
}
134
int initData(unsigned char **data, unsigned char **frame, int imageX,
int imageY)
136 {
    int i,j;
138
    /// Aloca dados
140 data = (unsigned char **) malloc(imageY*sizeof(unsigned char *));
    if(data == NULL)
142     {
        fprintf(stderr, "Nao consegue alocar dados\n");
144         return FALSE;
    }
146
    /// Aloca dados
148 frame = (unsigned char **) malloc((imageY - 2)*sizeof(unsigned char
        *));
    if(frame == NULL)
150     {
        fprintf(stderr, "Nao consegue alocar dados\n");
152         return FALSE;
    }
154
    /// Aloca dados da coluna
156 for( i = 0 ; i < imageY ; i++)
    {
158         data[i] = (unsigned char *) malloc(imageX*sizeof(unsigned char
            ));
        if(data[i] == NULL)
160         {
            fprintf(stderr, "Nao consegue alocar dados\n");
```

```

162         return FALSE;
163     }
164
165     for(j = 0 ; j < imageX ; j ++ )
166     {
167         data[i][j] = 0;
168     }
169 }
170
171 /// Aloca frames da coluna
172 for( i = 0 ; i < (imageY - 2) ; i++)
173 {
174     frame[i] = (unsigned char *) malloc((imageX -2 )*sizeof(
175         unsigned char));
176     if(frame[i] == NULL)
177     {
178         fprintf(stderr, "Nao consegue alocar dados\n");
179         return FALSE;
180     }
181
182     for(j = 0 ; j < (imageX -2) ; j ++ )
183     {
184         frame[i][j] = 0;
185     }
186 }
187
188 return TRUE;
189 }
190
191 int simpleConvol(unsigned char **data, unsigned char **frame, int
192 imageX, int imageY)
193 {
194     /// Diferentes tipos de kernel para testar...
195     float kernel[KernelX][KernelY] = {{0,0,0},{0,1,0},{0,0,0}};
196     int i, j;
197     int window[KernelX][KernelY];
198
199     // int sum = 0;
200     float sum = 0;
201
202     int inv = -1;
203
204     for(i = 1 ; i <= imageX ; i++)
205     {
206         for(j = 1 ; j <= imageY ; j++)
207         {
208             window[0][0] = data[i-1][j-1];

```

```

208     window[0][1] = data[i-1][j];
        window[0][2] = data[i-1][j+1];

210     window[1][0] = data[i][j-1];
        window[1][1] = data[i][j];
212     window[1][2] = data[i][j+1];

214     window[2][0] = data[i+1][j-1];
        window[2][1] = data[i+1][j];
216     window[2][2] = data[i+1][j+1];

218     sum = computeConv(window, kernel, inv);

220     if (sum > 1.0)
        sum = 1.0;

222     frame[i-1][j-1] = (unsigned char) sum*255;
224 }
    }
226 return TRUE;
}

228 float computeConv(int window[][KernelY], float kernel[][KernelY], int
inv)
230 {
    int a = window[0][0];
232     int b = window[1][0];
        int c = window[2][0];
234     int d = window[0][1];
        int e = window[1][1];
236     int f = window[2][1];
        int g = window[0][2];
238     int h = window[1][2];
        int i = window[2][2];

240     float af = kernel[0][0];
242     float bf = kernel[1][0];
        float cf = kernel[2][0];
244     float df = kernel[0][1];
        float ef = kernel[1][1];
246     float ff = kernel[2][1];
        float gf = kernel[0][2];
248     float hf = kernel[1][2];
        float iff = kernel[2][2];

250     return computeConv2(a,b,c,d,e,f,g,h,i,af,bf,cf,df,ef,ff,gf,hf,iff,
inv);

```

```
252 }
254
float computeConv2(int a, int b, int c, int d, int e, int f, int g, int
    h, int i, float af, float bf, float cf, float df,
256 float ef, float ff, float gf, float hf, float iff, int inv)
{
258     return mul3(a, inv, af) + mul3(d, inv, df) + mul3(g, inv, gf) +
        mul3(b, inv, bf) + mul3(e, inv, ef) + mul3(h, inv, hf) +
260     mul3(c, inv, cf) + mul3(f, inv, ff) + mul3(i, inv, iff);
}
262
float sum2(float a, float b, float c)
264 {
    return a + b + c;
266 }
268
float mul3(int inv, int w, float k)
{
270     return w*inv*k;
}
272
// lena.raw 256x256
274 int loadRawFile(char * filename, int x, int y, unsigned char **data)
{
276     FILE *fp;
    int i;
278     int j;

    fp = fopen(filename, "r");
    if (fp == NULL)
282         return FALSE;

    for (i = 1 ; i <= x ; i++)
    {
286         for (j = 1 ; j <= y ; j ++)
            {
288                 data[i][j] = fgetc(fp);
            }
    }
290
    fclose(fp);
292
    return TRUE;
294
}
```

Listing G.1: Implementação da Convolução em linguagem C.

## ANEXO H CASAMENTO DE PADRÕES

```
1 #include "opencv2/highgui/highgui.hpp"
2 #include "opencv2/imgproc/imgproc.hpp"
3 #include <iostream>
4 #include <stdio.h>
5
6 using namespace std;
7 using namespace cv;
8
9 /// Variaveis Globais
10 Mat img; Mat templ; Mat result;
11 const char* image_window = "Imagem original";
12 const char* result_window = "Janela com resultados";
13
14 int match_method;
15 int max_Trackbar = 5;
16
17 /// Cabecalho da funcao
18 void MatchingMethod( int , void* );
19
20
21 int main( int , char** argv )
22 {
23     /// Carrega a imagem e o template
24     img = imread( argv[1], 1 );
25     templ = imread( argv[2], 1 );
26
27     /// Cria as janelas
28     namedWindow( image_window , WINDOW_AUTOSIZE );
29     namedWindow( result_window , WINDOW_AUTOSIZE );
30
31     /// Cria uma trackbar
32     //const char* trackbar_label = "Method: \n 0: SQDIFF \n 1: SQDIFF
33     NORMED \n 2: TM CCORR \n 3: TM CCORR NORMED \n 4: TM COEFF \n 5:
34     TM COEFF NORMED";
```

```

33 //createTrackbar( trackbar_label , image_window , &match_method ,
    max_Trackbar , MatchingMethod );

35 MatchingMethod( 0, 0 );

37 waitKey(0);
    return 0;
39 }

41
void MatchingMethod( int , void* )
43 {

45     Mat img_display;
    //Mat dst;
47     img.copyTo( img_display );

49     /// Cria a matriz de resultados
    int result_cols = img.cols - templ.cols + 1;
51     int result_rows = img.rows - templ.rows + 1;

53     result.create( result_cols , result_rows , CV_32FC1 );

55     /// Faz o match e normaliza
    matchTemplate( img, templ, result, match_method );
57     normalize( result, result, 0, 1, NORM_MINMAX, -1, Mat() );

59     /// Localiza a melhor combinacao com o minMaxLoc
    double minVal; double maxVal; Point minLoc; Point maxLoc;
61     Point matchLoc;

63     minMaxLoc( result, &minVal, &maxVal, &minLoc, &maxLoc, Mat() );

65

    /// Para SQDIFF e SQDIFF_NORMED, a melhor combinacao sao de valores
        pequenos. Para os outros metodos, quanto maior, melhor.
67     if( match_method == TM_SQDIFF || match_method == TM_SQDIFF_NORMED )
        { matchLoc = minLoc; }
69     else
        { matchLoc = maxLoc; }

71

    /// Exibe o retangulo na imagem
73     //rectangle( img_display, matchLoc, Point( matchLoc.x + templ.cols ,
        matchLoc.y + templ.rows ), Scalar(0,0,255), 2, 8, 0 );
    //rectangle( result, matchLoc, Point( matchLoc.x + templ.cols ,
        matchLoc.y + templ.rows ), Scalar(0,0,255), 2, 8, 0 );

75

```

```

77 Mat roi( img_display , Rect( matchLoc.x , matchLoc.y , templ.cols , templ.
    rows ) );
78
79 //imshow( image_window , img_display );
80 imshow( image_window , img_display );
81 imshow( result_window , roi );
82
83 stringstream ss;
84 string type = ".jpg";
85 //ss << match_method << type;
86 ss << "crop" << type;
87 string saida = ss.str();
88 //imwrite( saida , img_display );
89 imwrite( saida , roi );
90
91 return;
92 }

```

Listing H.1: Casamento de Padrões em linguagem C++ para OpenCV.

**Estudo e Aplicação de Métodos de Segmentação de Imagens para o Diagnóstico de Falhas na Fabricação de Equipos – Leandro Soares Guedes**



**UNIVERSIDADE FEDERAL DE PELOTAS**  
Centro de Desenvolvimento Tecnológico  
Curso de Bacharelado em Ciência da Computação



Trabalho de Conclusão de Curso

**Estudo e Aplicação de Métodos de Segmentação de Imagens para o Diagnóstico de Falhas na Fabricação de Equipos**

**LEANDRO SOARES GUEDES**

Pelotas, 2014