

## **CURSO DE ENGENHARIA ELÉTRICA**

Augusto Abling

**SISTEMA DE VISÃO COMPUTACIONAL PARA INSPEÇÃO MULTIEÇAS EM  
LINHA DE PRODUÇÃO UTILIZANDO OPENCV**

Santa Cruz do Sul  
2017

Augusto Abling

**SISTEMA DE VISÃO COMPUTACIONAL PARA INSPEÇÃO MULTIEÇAS EM  
LINHA DE PRODUÇÃO UTILIZANDO OPENCV**

Trabalho de Conclusão de Curso II, modalidade monografia, apresentada ao curso de Engenharia Elétrica da Universidade de Santa Cruz do Sul, UNISC, como requisito parcial para obtenção do título de Bacharel em Engenharia Elétrica.

Orientador: Prof. Me. Adriano José Bombardieri

Santa Cruz do Sul

2017

## **AGRADECIMENTOS**

Gostaria de agradecer primeiramente a minha família que sempre deu incentivo para seguir em frente com determinação e visão para o futuro. Meu irmão, Adriano Abling pela experiência e conhecimentos técnicos compartilhados, juntamente com sua ajuda nos momentos necessários ao longo destes anos. Meu pai, Astor Abling pela personalidade séria e realista assim adotada por mim para realizar qualquer trabalho. A minha mãe, Dalva Teresinha Abling pelo companheirismo e apoio em todos momentos, bons e ruins, com palavras sabias que dão forças para levar minhas ideias à frente com fé.

Agradeço também a todos outros familiares, amigos e colegas que fazem ou já fizeram parte destes anos de graduação, onde nestes vivenciei muitas coisas construtivas para meu caráter e maneira de pensar, agradeço pelas fases passadas e momentos vividos.

Também presto agradecimento a meu orientador e amigo de longa data, professor Adriano José Bombardieri, que a muitos anos fornece conselhos valiosos e compartilha conhecimentos que servem para evolução tanto pessoal quanto profissional.

“Só podemos ver uma curta distância a frente,  
mas podemos ver muito lá que precisa ser feito.”

— Alan Turing

## RESUMO

Analizando a necessidade de flexibilidade na automação de processos complexos que ainda utilizam a operação humana, viu-se na visão computacional uma oportunidade de investimento nos estudos pela alta aplicabilidade em diversas áreas, porém para este projeto a área específica escolhida foi de automação industrial. Este trabalho se baseia no estudo do processamento digital de imagens voltado a um sistema de visão computacional para inspeção de peças em uma linha de produção simulada para extração de informações de classificação, aprovação e produtividade, fornecendo de forma contínua a localização das peças sobre uma esteira transportadora. Para compor o sistema foi necessário utilizar ferramentas recomendadas para tal finalidade, como a utilização de duas câmeras para obtenção das imagens, sendo uma para inspeção e outra para rastreamento das peças, juntamente com uma ferramenta computacional, composta pelo hardware e software, sendo um notebook para execução e desenvolvimento do programa, onde este foi elaborado em ambiente voltado à linguagem de programação C/C++, com o emprego de funções disponíveis na biblioteca OpenCV para processamento das imagens, juntamente com o desenvolvimento da interface gráfica de usuário para visualização do sistema em funcionamento. Ao final do trabalho foi possível notar a eficácia de algumas técnicas implementadas e outras que não ocorreram da mesma forma, também foram notáveis as necessidades de melhoria em alguns pontos que na elaboração de um sistema industrial seriam essenciais para garantir confiabilidade.

**Palavras-chave:** Automação industrial. Processamento digital de imagens. Visão computacional. OpenCV.

## ABSTRACT

Analyzing the need for flexibility in the automation of complex processes that still use human operation, it was seen in the computer vision an investment opportunity in studies due to the high applicability in several areas, but for this project the specific area chosen was industrial automation. This work is based on the study of digital image processing aimed at a computer vision system for inspection of parts in a simulated production line for extraction of classification, approval and productivity information, continuously providing the location of the parts on a conveyor belt. To compose the system, it was necessary to use tools recommended for this purpose, such as the use of two cameras to obtain the images, one for inspection and another for tracking the parts, together with a computational tool, composed of hardware and software, being a notebook for the execution and development of the program, where it was developed in an environment oriented to the C/C++ programming language, using functions available in the OpenCV library for image processing, along with the development of the graphical user interface for system visualization in functioning. At the end of the work it was possible to notice the effectiveness of some techniques implemented and others that did not occur in the same way, there were also notable improvement needs at some points that in the design of an industrial system would be essential to ensure reliability.

**Keywords:** Industrial automation. Digital image processing. Computer vision. OpenCV.

## LISTA DE ILUSTRAÇÕES

<b>Figura 1 – Exemplo de ferramenta de controle estatístico de processo: gráfico de controle ....</b>	<b>20</b>
<b>Figura 2 – Linha do tempo de algumas das principais pesquisas em visão computacional .....</b>	<b>23</b>
<b>Figura 3 – Processamento de imagem de uma tomografia do cérebro humano .....</b>	<b>24</b>
<b>Figura 4 – Leitura de código QR por visão computacional .....</b>	<b>24</b>
<b>Figura 5 – Sistema de identificação de caracteres em placas automotivas .....</b>	<b>24</b>
<b>Figura 6 – Reconhecimento biométrico da face .....</b>	<b>25</b>
<b>Figura 7 – Segmentação de regiões por sensoriamento remoto (geoprocessamento) .....</b>	<b>25</b>
<b>Figura 8 – Espectro eletromagnético em função da energia por fóton em elétron-volts (eV). ...</b>	<b>26</b>
<b>Figura 9 – Imagens de um pulsar gerado por uma estrela de nêutrons (Crab Pulsar) em seus espectros eletromagnéticos .....</b>	<b>26</b>
<b>Figura 10 – Sistema de inspeção em linha básico.....</b>	<b>27</b>
<b>Figura 11 – Leitura de cena formando a imagem na matriz de sensores .....</b>	<b>28</b>
<b>Figura 12 – CCD x CMOS leitura por varredura em linha.....</b>	<b>29</b>
<b>Figura 13 – Relações trigonométricas formadas por lente delgada .....</b>	<b>30</b>
<b>Figura 14 – Distância focal em uma câmera.....</b>	<b>31</b>
<b>Figura 15 – Modelos de iluminação para uso em visão computacional: a) Superior; b) De fundo; c) Lateral; d) Estruturada; e) Estroboscópica. .....</b>	<b>32</b>
<b>Figura 16 – Diferença entre a temperatura de cor para lâmpada LED .....</b>	<b>33</b>
<b>Figura 17 – Diferença no nível do índice de reprodução de cor .....</b>	<b>34</b>
<b>Figura 18 – Conectores FireWire e USB 2.0 respectivamente.....</b>	<b>36</b>
<b>Figura 19 – Conectores USB3 Vision, GigE Vision, Camera Link, Camera Link HS e CoaXpress respectivamente.....</b>	<b>37</b>
<b>Figura 20 – Matriz de imagem em escala de cinza.....</b>	<b>40</b>
<b>Figura 21 – Imagem dividida em uma matriz 9X10 .....</b>	<b>41</b>
<b>Figura 22 – Comparação imagem colorida (RGB) e monocromática .....</b>	<b>42</b>
<b>Figura 23 – Modelo de cores RGB, CMY e HSV consecutivamente.....</b>	<b>42</b>
<b>Figura 24 – Mudança dos níveis de cinza no processamento por sensoriamento remoto.....</b>	<b>43</b>
<b>Figura 25 – Principais passos para o processamento de imagem .....</b>	<b>44</b>
<b>Figura 26 – Vizinhança-4 e vizinhança-8 de um pixel .....</b>	<b>45</b>
<b>Figura 27 – Tipos de filtros no domínio da frequência e do espaço respectivamente .....</b>	<b>45</b>
<b>Figura 28 – Máscaras para aplicação de filtro de média de tamanhos 3x3, 5x5 e 7x7 respectivamente.....</b>	<b>46</b>
<b>Figura 29 – Respostas da aplicação de filtro de média de tamanhos 3x3, 5x5 e 7x7 respectivamente a partir da imagem original .....</b>	<b>47</b>
<b>Figura 30 – Obtenção da resposta de um filtro mediana .....</b>	<b>47</b>
<b>Figura 31 – Exemplo de aplicação de filtro passa-baixa: a) Imagem com ruído; b) Imagem com filtro de média; c) Imagem com filtro mediana .....</b>	<b>48</b>
<b>Figura 32 – Imagem 3D da função gaussiana bidimensional com média (0,0) e <math>\sigma = 1</math>. ....</b>	<b>48</b>

Figura 33 – Triângulo de Pascal .....	49
Figura 34 – Resposta a duas máscaras bidimensionais.....	49
Figura 35 – Aplicação da máscara bidimensional do filtro Gaussiano .....	50
Figura 36 – Aplicação de filtro bilateral, imagem original e processada respectivamente.....	51
Figura 37 – Mascaras de filtro Laplaciano .....	52
Figura 38 – Aplicação do filtro Laplaciano .....	52
Figura 39 – Aplicação de filtro high-boost. a) Original. b) Processada.....	53
Figura 40 – Mascaras vertical e horizontal de Sobel .....	54
Figura 41 – Aplicação do filtro de Sobel.....	54
Figura 42 – Operadores cruzados de Roberts 2x2 .....	55
Figura 43 – Aplicação do operador de Roberts 2x2 .....	56
Figura 44 – Exemplo de limiar global automático com resultado em 125 após 3 iterações .....	57
Figura 45 – Limiarização pelo método triangular .....	59
Figura 46 – Segmentação por subtração de fundo.....	59
Figura 47 – Subtração de fundo com uso de limiar.....	60
Figura 48 – Máscara 5X5 de uma aproximação pelo método Marr-Hildreth .....	61
Figura 49 – Segmentação pelo método Marr-Hildreth utilizando $\sigma=4$ e $n=25$ .....	62
Figura 50 – Comparação entre detecção por borda de Marr-Hildreth (c) e Canny (d) .....	63
Figura 51 – Formas comuns de cantos em imagens digitais .....	64
Figura 52 – Resultados da aplicação do detector de Harris .....	65
Figura 53 – Divisão de regiões por decomposição regular. a) Imagem original. b) Imagem em matriz binária. c) Imagem particionada em quadrantes. d) Quadtree correspondente.....	66
Figura 54 – Reconhecimento de objetos por aproximação poligonal .....	67
Figura 55 – Ponto de referência em um objeto .....	68
Figura 56 – Aplicação da transformada de Hough para detecção de reta na separação de hemisférios esquerdo e direto do cérebro.....	69
Figura 57 – Dilatação de uma região de interesse.....	70
Figura 58 – Erosão de uma região de interesse.....	70
Figura 59 – Exemplo de transformação morfológica por abertura e fechamento respectivamente.....	71
Figura 60 – Parte de um código em assembly para microcontrolador Motorola MC6800 .....	72
Figura 61 – Descrição em diagrama esquemático .....	73
Figura 62 – Computador DEC-PDP-11 .....	74
Figura 63 – Exemplo de código em Python .....	75
Figura 64 – Exemplo de código em linguagem de programação Java.....	75
Figura 65 – Linha do tempo fase inicial de desenvolvimento da biblioteca OpenCV .....	76
Figura 66 – Aplicações com OpenCV para detecção de objeto conhecido utilizando SURF detector, rastreamento de tráfego, detecção facial e rastreamento de pessoas, respectivamente.....	77

<b>Figura 67 – Cápsulas Nescafé Dolce Gusto .....</b>	<b>79</b>
<b>Figura 68 – Produto promocional de cápsulas .....</b>	<b>79</b>
<b>Figura 69 – Funcionamento do sistema de inspeção e rastreamento.....</b>	<b>80</b>
<b>Figura 70 – Câmera Logitech C270 .....</b>	<b>81</b>
<b>Figura 71 – Estrutura do protótipo inicial.....</b>	<b>83</b>
<b>Figura 72 – Anel de LEDs para iluminação.....</b>	<b>84</b>
<b>Figura 73 – Luminária de LED embutida.....</b>	<b>85</b>
<b>Figura 74 – Barreira de luz .....</b>	<b>85</b>
<b>Figura 75 – Estrutura do protótipo final.....</b>	<b>86</b>
<b>Figura 76 – Algoritmo de captura de imagens das câmeras .....</b>	<b>88</b>
<b>Figura 77 – Descrição da função cvtColor .....</b>	<b>88</b>
<b>Figura 78 – Função de conversão de imagem RGB em escala de cinza e funções de filtragem com máscaras 7x7.....</b>	<b>89</b>
<b>Figura 79 – Amostra de imagem capturada colorida (RGB) e em escala de cinza .....</b>	<b>89</b>
<b>Figura 80 – Comparação dos resultados da aplicação de filtro passa-baixa de média, mediana, Gaussiano e bilateral respectivamente, com máscaras 7x7.....</b>	<b>90</b>
<b>Figura 81 – Algoritmo do filtro Laplaciano .....</b>	<b>90</b>
<b>Figura 82 – Algoritmo filtro passa-alta High-boost.....</b>	<b>91</b>
<b>Figura 83 – Algoritmo do gradiente de Sobel.....</b>	<b>91</b>
<b>Figura 84 – Algoritmo do gradiente de Roberts.....</b>	<b>92</b>
<b>Figura 85 – Respostas da aplicação dos filtros Laplaciano, High-Boost, Sobel e Roberts após a suavização por filtro Gaussiano respectivamente .....</b>	<b>92</b>
<b>Figura 86 – Algoritmo de segmentação e detecção de objetos por limiarização e aproximação poligonal com uso do filtro de Sobel.....</b>	<b>93</b>
<b>Figura 87 – Respostas do algoritmo de limiarização e aproximação poligonal com uso do filtro de Sobel. Imagens da limiarização, detecção de bordas e resultados finais respectivamente.....</b>	<b>94</b>
<b>Figura 88 – Algoritmo de segmentação e detecção de objetos por limiarização e aproximação poligonal sem uso do filtro de Sobel.....</b>	<b>95</b>
<b>Figura 89 – Respostas do algoritmo de limiarização e aproximação poligonal sem uso do filtro de Sobel. Imagens da limiarização, detecção de bordas e resultados finais respectivamente.....</b>	<b>95</b>
<b>Figura 90 – Algoritmo de segmentação e detecção de objetos pela transformada de Hough ...</b>	<b>96</b>
<b>Figura 91 – Amostras das respostas do algoritmo de segmentação e detecção de objetos pela transformada de Hough .....</b>	<b>96</b>
<b>Figura 92 – Algoritmo de subtração de fundo absoluta.....</b>	<b>97</b>
<b>Figura 93 – Resultado da operação de subtração de fundo. Imagem original, de fundo e resultante respectivamente .....</b>	<b>97</b>
<b>Figura 94 – Algoritmos de pré-processamento e segmentação dos objetos em escala de cinza e RGB .....</b>	<b>98</b>

Figura 95 – Respostas de limiar para imagem em escala de cinza e colorida .....	98
Figura 96 – Algoritmo de processamento morfológico para abertura e fechamento .....	99
Figura 97 – Resposta da aplicação de abertura e fechamento morfológico respectivamente ...	99
Figura 98 – Respostas da aplicação do algoritmo de subtração de fundo e processamento morfológico .....	100
Figura 99 – Aquisição do ponto central e raio de cada peça .....	100
Figura 100 – Geometria de referência para cálculo de média aritmética das cores .....	101
Figura 101 – Inspeção de tamanho e cor ok.....	102
Figura 102 – Inspeção com peças fora do tamanho e não registrada como padrão .....	103
Figura 103 – Numeração das peças iniciais .....	103
Figura 104 – Numeração das peças seguintes .....	104
Figura 105 – Linhas limites da esteira com recuo de 26 pixels e ângulo de 73,5º .....	105
Figura 106 – Resultado da operação de subtração de fundo da câmera 2. Imagem original, de fundo e resultante respectivamente .....	105
Figura 107 – Máscara aplicada e resposta ao filtro Gaussiano respectivamente .....	106
Figura 108 – Algoritmo de limiarização por cor .....	106
Figura 109 – Soma das imagens limiarizadas .....	107
Figura 110 – Respostas do processamento morfológico de abertura (2 iterações) e fechamento (7 iterações).....	107
Figura 111 – Respostas da detecção dos objetos de rastreamento.....	108
Figura 112 – Referências para cálculo da coordenada real em milímetros .....	109
Figura 113 – Detecção e fornecimento das coordenadas das cápsulas com as marcas da correção em azul.....	110
Figura 114 – Algoritmo do cálculo de FPS .....	111
Figura 115 – Cápsulas padrões registradas para simulação .....	112
Figura 116 – Captura 1 da simulação .....	112
Figura 117 – Captura 2 da simulação .....	112
Figura 118 – Captura 3 da simulação .....	113
Figura 119 – Tela inicial da GUI .....	114
Figura 120 – Janela da seleção do método de entrada .....	114
Figura 121 – Janela das configurações de entrada .....	115
Figura 122 – Tela das configurações de processamento .....	116
Figura 123 – Janela para adicionar padrões .....	116
Figura 124 – Tela da simulação do processamento .....	117
Figura 125 – Tela da simulação do processamento seguinte .....	117
Fluxograma 126 – Lógica do processamento das imagens do sistema .....	119

## **LISTA DE TABELAS**

<b>Tabela 1 – Pontos fortes e características entre humanos e máquinas.....</b>	<b>18</b>
<b>Tabela 2 – Aparência de cor e suas classificações .....</b>	<b>33</b>
<b>Tabela 3 – Comparação geral de hardwares .....</b>	<b>35</b>
<b>Tabela 4 – Comparação entre padrões de interface de comunicação .....</b>	<b>38</b>
<b>Tabela 5 – Especificações técnicas do hardware utilizado .....</b>	<b>82</b>
<b>Tabela 6 – Relação de alguns módulos presentes na biblioteca OpenCV.....</b>	<b>87</b>
<b>Tabela 7 – Lista de alguns softwares, bibliotecas, APIs e suas aplicações .....</b>	<b>124</b>

## LISTA DE ABREVIATURAS

AIA	Automated Imaging Association
API	Application Programming Interface
APU	Accelerated Processing Unit
ASIC	Application Specific Integrated Circuits
CCD	Charge-Coupled Device
CLP	Controlador Lógico Programável
CUDA	Compute Unified Device Architecture
CMOS	Complementary Metal Oxide Semiconductor
CMU	Carnegie Mellon University
CMY	Cyan, Magenta and Yellow
CPU	Central Processing Unit
DLL	Dynamic-Link Library
DSP	Digital Signal Processor
EMVA	European Machine Vision Association
FPGA	Field Programmable Gate Array
GPU	Graphics Processing Unit
GUI	Graphical User Interface
HSV	Hue, Saturation and Value
IEEE	Instituto de Engenheiros Eletricistas e Eletrônicos
JIIA	Japan Industrial Imaging Association
LED	Light Emitting Diode
MIT	Massachusetts Institute of Technology
NASA	National Aeronautics and Space Administration
QR	Quick Response
RGB	Red, Green and Blue

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>14</b>
1.1	Justificativa.....	14
1.2	Área de limitação do tema .....	15
1.3	Objetivos .....	16
1.3.1	Objetivo geral.....	16
1.3.2	Objetivos específicos.....	16
<b>2</b>	<b>REFERENCIAL TEÓRICO.....</b>	<b>17</b>
2.1	Processos industriais .....	17
2.1.1	Automação industrial.....	18
2.1.2	Inspeção de qualidade .....	19
2.1.3	Interfaces gráficas para supervisório.....	21
2.1.4	Estratégias para melhoria da qualidade .....	21
2.2	Visão computacional.....	22
2.2.1	Breve histórico de estudos na área .....	22
2.2.2	Aplicações diversas com visão computacional .....	23
2.2.3	Sistema de inspeção visual em linha de produção .....	26
2.2.4	Sensores ópticos.....	28
2.2.5	Illuminação.....	31
2.2.6	Hardwares .....	34
2.2.7	Interfaces de comunicação .....	36
2.2.8	Softwares .....	39
2.3	Processamento digital de imagens.....	39
2.3.1	Fundamentos do processamento de imagens .....	40
2.3.2	Passos do processamento .....	43
2.3.3	Pré-processamento .....	44
2.3.4	Segmentação e reconhecimento de objetos.....	56
2.3.5	Processamento morfológico .....	69
2.4	Linguagens de programação .....	71
2.4.1	Assembly .....	71
2.4.2	VHDL.....	72
2.4.3	C/C++ .....	73
2.4.4	Python .....	74

<b>2.4.5</b>	<b>Java .....</b>	<b>75</b>
<b>2.5</b>	<b>OpenCV .....</b>	<b>76</b>
<b>3</b>	<b>METODOLOGIA .....</b>	<b>78</b>
<b>3.1</b>	<b>Estudo da necessidade de aplicação .....</b>	<b>78</b>
<b>3.2</b>	<b>Descrição do funcionamento .....</b>	<b>80</b>
<b>3.3</b>	<b>Equipamentos e ferramentas para o desenvolvimento .....</b>	<b>80</b>
<b>4</b>	<b>DESENVOLVIMENTO E APLICAÇÃO .....</b>	<b>83</b>
<b>4.1</b>	<b>Estrutura do protótipo .....</b>	<b>83</b>
<b>4.1.1</b>	<b>Correção da iluminação.....</b>	<b>84</b>
<b>4.2</b>	<b>Implementação das técnicas e algoritmos.....</b>	<b>86</b>
<b>4.2.1</b>	<b>Módulos utilizados .....</b>	<b>86</b>
<b>4.2.2</b>	<b>Obtenção e conversão das imagens .....</b>	<b>87</b>
<b>4.2.3</b>	<b>Câmera 1 (Inspeção) - Processamento digital de imagens .....</b>	<b>88</b>
<b>4.2.3.1</b>	<b>Câmera 1 - Pré-processamento do sistema .....</b>	<b>89</b>
<b>4.2.3.2</b>	<b>Câmera 1 - Segmentação e detecção dos objetos do sistema.....</b>	<b>93</b>
<b>4.2.3.3</b>	<b>Câmera 1 - Extração de características dos objetos.....</b>	<b>100</b>
<b>4.2.3.4</b>	<b>Câmera 1 - Reconhecimento e interpretação das informações .....</b>	<b>101</b>
<b>4.2.4</b>	<b>Câmera 2 (Rastreamento) - Processamento digital de imagens .....</b>	<b>104</b>
<b>4.2.4.1</b>	<b>Câmera 2 - Calibração.....</b>	<b>104</b>
<b>4.2.4.2</b>	<b>Câmera 2 - Pré-processamento do sistema .....</b>	<b>105</b>
<b>4.2.4.3</b>	<b>Câmera 2 - Segmentação e detecção dos objetos do sistema.....</b>	<b>106</b>
<b>4.2.4.4</b>	<b>Câmera 2 - Extração de características dos objetos.....</b>	<b>108</b>
<b>4.2.4.5</b>	<b>Câmera 2 - Reconhecimento e interpretação das informações .....</b>	<b>108</b>
<b>4.2.5</b>	<b>Visualização dos resultados .....</b>	<b>111</b>
<b>4.3</b>	<b>Desenvolvimento da interface gráfica .....</b>	<b>113</b>
<b>4.4</b>	<b>Resultados obtidos e conclusões.....</b>	<b>118</b>
<b>4.5</b>	<b>Oportunidades de melhoria.....</b>	<b>119</b>
<b>REFERÊNCIAS.....</b>		<b>121</b>
<b>APÊNDICE A – Tabela de softwares .....</b>		<b>124</b>
<b>ANEXO A – Algoritmo Da Transformada De Hough Generalizada.....</b>		<b>125</b>

## 1 INTRODUÇÃO

A visão humana desperta curiosidade já que é um dos cinco sentidos que mais estamos vinculados, pois este fornece informações importantes para o funcionamento, interação e compreensão de tudo a nossa volta, gerando uma enorme quantidade de dados, seja identificando objetos, cenas ou símbolos, ela nos auxilia também a tomar decisões e notar quando algo está errado ao ter estes dados processados pelo nosso cérebro (DAVIES, 2012).

Não é diferente no ambiente industrial onde a maioria dos processos ainda são manuais e dependentes da inspeção visual do produto para sua manipulação. Como sugerem R. Seleme e R. B. Seleme (2012, p. 169), uma das maneiras de automatizar um processo nesse quadro de atividade é a utilização de sistemas de visão computacional ou visão de máquina, quando está associado à ação de máquinas.

Conforme alguns autores descrevem (BOYLE; HLAVAC; SONKA, 2008; GONZALES; WOODS, 2010; PRATT, 2001), desde os primeiros passos na introdução do processamento de imagem, a partir da introdução dos computadores nos anos 60, esta área tem crescido com a evolução de tecnologias de computação e sensoriamento cada vez mais eficientes, gerando interesse de estudo e inovação em aplicações diversas como na biomedicina, segurança, trânsito, sensoriamento remoto, processos industriais e também na área do entretenimento.

OpenCV é a ferramenta que tem auxiliado milhares de estudantes e profissionais da área a aprimorar e tornar a visão computacional uma tecnologia acessível e barata, disponibilizando gratuitamente a maioria de suas funções e manuais para aquele que deseja se empenhar para trazer inovação e ajuda a sociedade como um todo (BRADSKI; KAEHLER, 2008).

### 1.1 Justificativa

Olhando para os dias atuais e o futuro das novas tecnologias, a visão computacional se destaca por estar em uma época de amadurecimento onde muitos entusiastas e profissionais da área vêm se dedicando para sua implementação na automação de serviços diversos. OpenCV é a ferramenta que

tem ótimo desempenho no processamento de imagens, pois é possível utilizar o potencial que os hardwares de hoje oferecem, assim como a flexibilidade de programar em variadas linguagens de programação e também em diferentes plataformas operacionais, como por exemplo Android, Linux e Windows (FARIA, 2012, p. 58).

Mas ainda assim como outros sistemas que surgem para revolucionar o meio social e industrial, iniciam-se com altos custos e retorno a longo prazo. Para buscar a melhoria nesse mercado se faz necessário o estudo contínuo e empenho, focado na redução de custos de projetos, materiais e também na confiabilidade do sistema proposto. Pela disponibilidade de acesso aos arquivos de instrução de uso e instalação da biblioteca OpenCV, assim como existem vários livros sobre programação orientada a objetos, com uma visão futurista, se fez necessário aprender mais sobre o assunto e desenvolver um software em linguagem de programação C/C++ que servirá de base para automatizar um processo de inspeção em linha de produção. Assim será possível aprofundar os conhecimentos de processamento digital de imagens e também contribuir com a melhora no processo industrial, sendo possível migrar para outras áreas de aplicação da visão computacional tendo adquirido essa base de conhecimento.

## **1.2 Área de limitação do tema**

Na área de processamento de imagens existem inúmeras técnicas de processamento, identificação e extração de dados das imagens, unindo isso ao uso de máquinas, programação, matemática, estatística e produção seria algo muito extenso e que está sempre em estudo, existindo muitas teorias que talvez na aplicação deste projeto não seja usual. Optou-se então por focar apenas no sistema de visão computacional e as áreas de conhecimento necessárias para o desenvolvimento do software, fazendo-se uso de dois sensores ópticos simples para obtenção das imagens e elaborar em linguagem C/C++, utilizando a biblioteca OpenCV, um sistema de inspeção industrial multipeças em uma esteira transportadora de forma experimental, fornecendo informações relevantes para o processo simulado e também as coordenadas em tempo real da localização das mesmas que estejam em condições diferentes do padrão definido.

### 1.3 Objetivos

Os objetivos estão relacionados a implementação de um sistema de visão computacional completo para inspeção industrial, porém será realizado apenas um protótipo de simulação focando principalmente na parte de desenvolvimento do software, para isso alguns requerimentos são necessários tanto para conhecer como desenvolver o projeto quanto para executá-lo.

#### 1.3.1 Objetivo geral

Implementar um sistema de visão computacional através do processamento de imagens de peças manufaturadas na esteira transportadora simulada para inspeção de qualidade, avaliando as características destas peças e fornecendo as coordenadas das mesmas, podendo assim servir como base de aplicação em uma linha de produção.

#### 1.3.2 Objetivos específicos

- Estudo das técnicas e teorias para o processamento de imagens existentes e mais eficazes.
- Aprimorar os conhecimentos em linguagem de programação em C e C++.
- Estudo da biblioteca OpenCV para o processamento de imagens buscando os métodos mais adequados para o sistema a ser desenvolvido.
- Utilizar softwares de código aberto para desenvolvimento do programa principal e da interface gráfica.
- Utilizar duas câmeras de baixo custo que tenham o desempenho necessário para a obtenção de imagens com qualidade suficiente e taxa de quadros por segundo aceitável.

## 2 REFERENCIAL TEÓRICO

Este capítulo englobará o conjunto de informações necessárias para a elaboração do projeto em estudo citando alguns itens adicionais sobre as áreas de atuação da tecnologia, lembrando que alguns assuntos possuem inúmeras teorias existentes, porém será contemplado apenas o que for relevante para a noção de como o trabalho será realizado, para um maior aprofundamento se necessário é viável a busca de mais informações nas referências.

Inicialmente, na seção secundária 2.1 está descrita uma introdução geral sobre processos industriais e quais os pontos importantes que devem ser levados em consideração no momento da implementação de alguma automação. Já na seção 2.2 é explicado e exemplificado o que é o sistema de visão computacional de forma geral, fornecendo um breve histórico da tecnologia e o que é necessário para compor o sistema em análise. Sequencialmente nas demais seções está o referencial teórico mais aprofundado e segmentado sobre os conteúdos que servirão de base de conhecimento para o desenvolvimento do projeto.

### 2.1 Processos industriais

A industrialização traz consigo a evolução da própria sociedade, países que possuem indústrias altamente desenvolvidas e com forte atuação demonstram melhor renda per capita conforme cita Paranhos Filho (2012), assim consequentemente tem-se condições de sustentar melhores serviços de saúde, educação e infraestrutura para a população.

A manufatura é composta por pessoas, equipamentos e procedimentos organizacionais para que a produção possa ser realizada de forma em que as etapas do processo sejam executadas de maneira correta e eficiente. A busca pela melhoria contínua tem se cultuado dentro das empresas, para se ter um controle efetivo do desempenho da produção são mantidos sob análise indicadores, como a produtividade, a qualidade tanto do produto como do serviço e trabalho realizado, a velocidade do processo visando menores estocagens, a confiabilidade onde a empresa atende os requisitos exigidos com menos falhas e a flexibilidade para se

adequar as variações implantadas pelos clientes ou fornecedores (GROOVER, 2011; COSTA JUNIOR, 2012).

### **2.1.1 Automação industrial**

A automação industrial parte do princípio de sistemas automatizados, que são processos executados automaticamente por máquinas ou equipamentos, previamente programados, são controlados por meio de alguma tecnologia de hardware mais software com instruções que atendam a funcionalidade necessária em determinada atividade, unindo outros tipos de sistemas como mecânico, elétrico, pneumático inclusive a mão de obra humana em sistemas parcialmente automatizados. A automação facilita e padroniza processos, além de vários outros benefícios, mas ainda assim deve-se analisar os pontos fortes do fator humano como pode ser visualizado na tabela 1 (GROOVER, 2011).

**Tabela 1 – Pontos fortes e características entre humanos e máquinas**

<b>Pontos fortes de humanos</b>	<b>Pontos fortes relativos de máquinas</b>
Percebem estímulos inesperados	Executam tarefas repetitivas de forma consistente
Desenvolvem novas soluções para problemas	Armazenam grandes volumes de dados
Lidam com problemas abstratos	Recuperam dados da memória de forma confiável
Adaptam-se às mudanças	Executam diversas tarefas simultaneamente
Generalizam a partir de observações	Aplicam muita força e potência
Aprendem com a experiência	Executam cálculos simples com rapidez
Tomam decisões difíceis com base em dados incompletos	Tomam decisões rotineiras rapidamente

Fonte: GROOVER, 2011, p. 5. Editado pelo autor.

## 2.1.2 Inspeção de qualidade

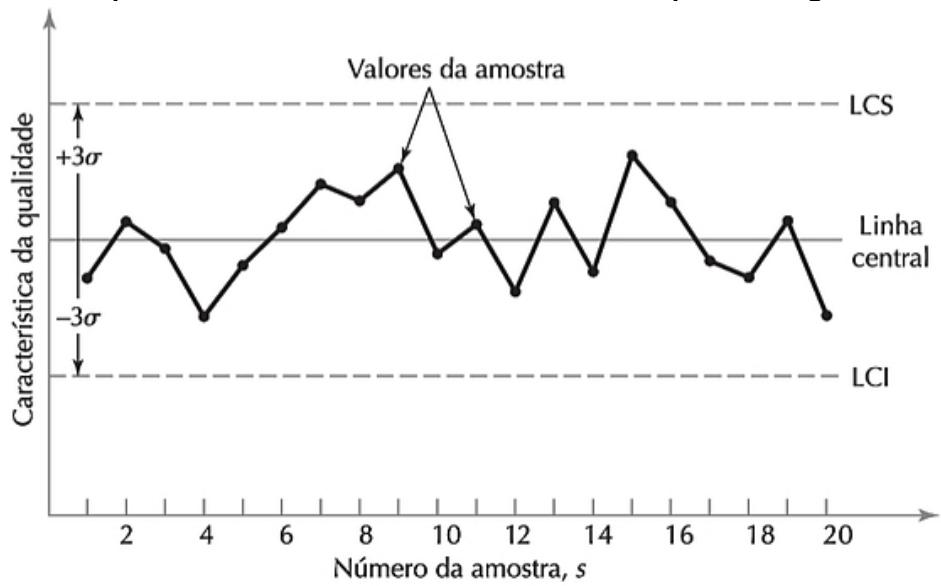
Conforme cita Groover (2011), a qualidade de um produto manufaturado é determinada baseada nas características de projetos e se possui alguma não conformidade ou se é livre de deficiências. As características do produto são atributos relacionados ao projeto construtivo, se atende a forma, peso, cor e detalhes necessários para suprir a função determinada para tal produto, já o caso de ser livre de deficiências tem como análise defeitos ocorridos do processo de fabricação, materiais fora das condições aceitáveis ou também a falta de algum material, geralmente esta avaliação é feita durante a produção em períodos de inspeção, podem ser executadas pelas pessoas responsáveis do departamento de controle de qualidade ou também os próprios operadores das máquinas quando esses são devidamente treinados.

### 2.1.2.1 Controle estatístico do processo (CEP)

Provindo do inglês *statistical process control (SPC)*, o controle estatístico do processo engloba várias maneiras de monitorar o processo, com medições e previsões, possibilita em caso de variação fora dos limites estipulados, uma ação a ser tomada. Segundo Groover (2011, p. 481), os objetivos do CEP são de “melhorar a qualidade de saída do processo, reduzir a variabilidade, conseguir uma estabilidade do processo e solucionar problemas de processamento”.

Groover (2011) ainda indica sete das principais ferramentas utilizadas para este trabalho de controle estatístico, sendo elas “gráficos de controle, histogramas, diagramas de Pareto, folhas de verificação, diagramas de concentração de defeitos, diagramas de dispersão e diagramas de causa e efeito”. Estas ferramentas podem ser aplicadas durante o fluxo do produto manufaturado, em inspeções de qualidade através de amostragens, também podem ser feitas análises de produção em relação a produtividade através da fabricação horária, ou por turno. As informações geradas são bem empregadas na aplicação da melhoria do processo para busca de causas, índices e variações que sugerem algum padrão de erro ou falha que possa ser eliminada.

**Figura 1 – Exemplo de ferramenta de controle estatístico de processo: gráfico de controle**



Fonte: GROOVER, 2011.

### 2.1.2.2 Tecnologias de inspeção

O processo de inspeção de qualidade analisa o produto buscando não conformidades ou a confirmação da boa qualidade, pode ser executada em qualquer ponto da produção, na maioria das empresas o sistema manual de inspeção ainda persiste, porém, a inovação através da automação de processos vem trazendo novos métodos de se monitorar e controlar os detalhes da produção, pois a inspeção manual é elaborada em cima de amostragem, ou seja, periodicamente que pode ocasionar na descoberta de uma não conformidade presente em todo produto muito tempo após já ter passado por todo processo da planta, que acaba gerando em perda de produto, tempo e outros gastos (GROOVER, 2011).

Com a automação deste processo de inspeção o monitoramento é constante, em tempo real, podendo ainda realimentar a operação para realizar ajustes ou ações através da leitura das variáveis, com esta tecnologia e a implementação de conjuntos computacionais para controle estatístico e monitoramento online o gerenciamento de informações pode ficar disponível para acesso remoto e os resultados também se tornam mais precisos para levantamentos de desempenho e qualidade, índices importantes que agregam valor ao produto final (GROOVER, 2011).

Os métodos de inspeção são variados, manualmente ainda existem equipamentos como paquímetros, micrômetros e gabaritos que são utilizados, já no processo automatizado se faz necessário uso de equipamento computacional, existem as máquinas que realizam medição por coordenadas (MMCs) e também o sistema de visão de máquina ou visão computacional (GROOVER, 2011).

### **2.1.3 Interfaces gráficas para supervisório**

Nos sistemas automatizados, os dados sobre variáveis do processo são adquiridos por meio de sensores e estas informações devem ser convertidas em grandezas mensuráveis que representam o estado de determinada etapa, como a temperatura, pressão ou o nível de preenchimento de um produto em um recipiente por exemplo. Para representar estes resultados efetuando um controle e gerenciamento do processo industrial são aplicáveis os sistemas de supervisão e monitoramento, também conhecido por sistema SCADA (*Supervisory Control and Data Acquisition*), sua implementação pode ser dada por meio de uma interface homem-máquina (IHM) onde é possível o operador ou profissional atuante efetuar alguma ação ou modificar algum padrão através dos acionamentos disponíveis no sistema (ROSÁRIO, 2005).

Assim como outros modelos de interface gráfica usuais para o monitoramento do processo o sistema SCADA é um conjunto de softwares que realizam a leitura das informações conectadas por meio de alguma CPU e seus sensores. É possível desenvolver o software por meio de programação via linguagens de programação Visual Basic, Delphi ou C++ baseando-se em funções para a aplicação, ainda é possível utilizar softwares próprios para o desenvolvimento de interface gráfica para supervisório como é o exemplo do programa LabVIEW, dependendo do tipo de aplicação ainda existem os softwares Microsoft Visual Studio e QT Creator, entre outros (ROSÁRIO, 2005).

### **2.1.4 Estratégias para melhoria da qualidade**

Durante anos, teóricos reuniram maneiras de comprovar a efetiva qualidade em um processo para atingir seus objetivos, como Groover (2011) e Paranhos Filho

(2012) ressaltam, uma das maneiras mais importantes de seguir padrões de qualidade e efetivar esta atribuição à empresa é seguir o sistema ISO 9001, que é um conjunto de especificações padronizadas internacionalmente sobre qualidade mantidas pela Organização Internacional para Padronização (ISO do inglês – International Organization for Standardization) com representação em todos países industrializados. Outros itens que surgem apenas para favorecer e valorizar a empresa e seu produto são a implementação de outros sistemas semelhantes como o QS9000 e ISO/TS 16949, assumir uma cultura de melhoria contínua e práticas de sucesso como o Controle Total da Qualidade (TQC) e o Gerenciamento Total da Qualidade (TQM), métodos de redução ou até eliminação de falhas no processo como o modelo Poka Yoke e a administração visual Andon, levantamento de dados específicos aplicando a estatística em meio a análise do processo e produto.

## **2.2 Visão computacional**

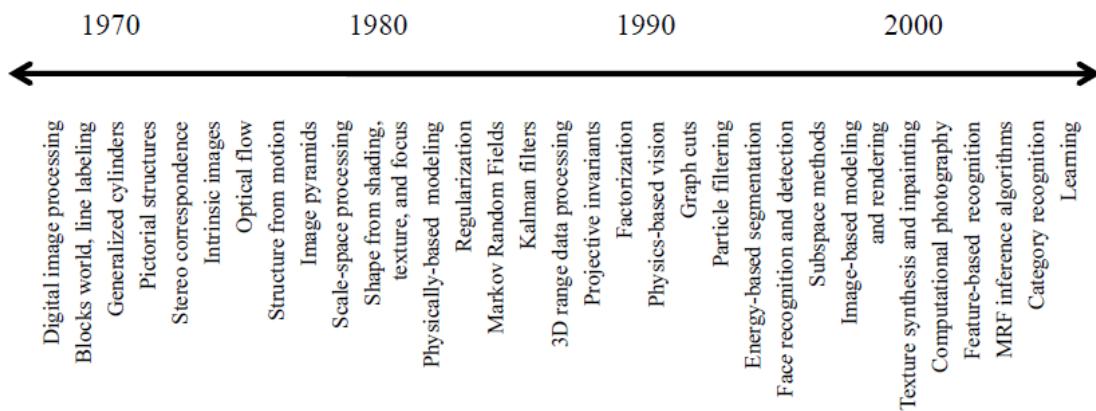
A visão humana vem sendo estudada a muitos anos por cientistas e psicólogos que tentam entender como toda a percepção das imagens é processada e avaliada de forma tão fácil e hábil, assim como na maioria dos animais, automaticamente identificamos cores, bordas, profundidade, transparência entre outras características da visão em três dimensões que captamos. Tudo isso ainda intriga estes estudiosos e a visão computacional surgiu para aplicar esses conceitos e testá-los utilizando processamento de imagem aliado a linguagem de máquina e hardware de implementação, porém, como a compreensão da ciência humana é complexa é mais aplicável o estudo da física para interpretar o que consta em uma imagem (MARR, 1982).

### **2.2.1 Breve histórico de estudos na área**

Como explica Szeliski (2010), quando os pioneiros do ramo de robótica e inteligência artificial dos anos 70 (em institutos e universidades como Stanford, CMU e MIT) tentaram iniciar a visão computacional ambiciosamente para a utilização em robôs com a pretensão de reproduzir a inteligência humana,

acreditavam que era um processo simples e que após isso poderiam evoluir rapidamente para a resolução de problemas com maiores dificuldades como planejamento e controle, mas ao tentar fazer simples leituras de algumas imagens perceberam que este processo era mais complexo do que imaginavam.

**Figura 2 – Linha do tempo de algumas das principais pesquisas em visão computacional**



Fonte: SZELISKI, 2010.

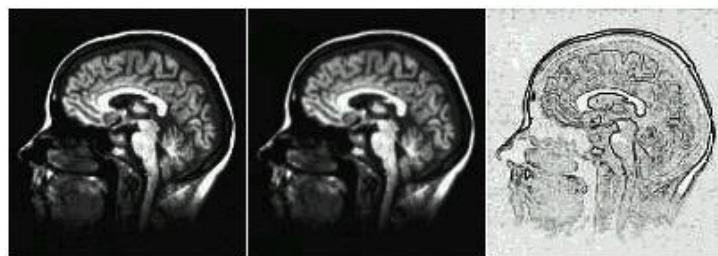
Como mostra a figura 2, com alguns dos principais estudos, a evolução da visão computacional ao longo dessas mais de quatro décadas gerou um aprimoramento sólido à tecnologia chegando aos dias de hoje com conhecimento em técnicas de aprendizagem de máquina. Pode ser notado também que o início e base de todo sistema de visão computacional requere forte conhecimento das técnicas de processamento de imagens e que essa é a área com maior foco de trabalhos.

### 2.2.2 Aplicações diversas com visão computacional

As aplicações no ramo da visão computacional são variadas assim como o processamento de imagens, hoje presente no contexto funcional de inúmeras atividades, auxiliam no trabalho de identificação de informações em imagens de forma automática, com maior velocidade e precisão, algumas das aplicações dessa tecnologia são encontradas em imagens médicas, inspeções de qualidade, segurança no trânsito, reconhecimento de carácteres, reconhecimento de modelos

3D, imagens microscópicas, sensoriamento remoto, leituras espaciais de telescópios, etc. (SHAPIRO; STOCKMAN, 2001; SZELISKI, 2010).

**Figura 3 – Processamento de imagem de uma tomografia do cérebro humano**



Fonte: disponível em: <<http://www.angelfire.com/de/ebenjamim/seminarios/tomo03.jpg>>.

**Figura 4 – Leitura de código QR<sup>1</sup> por visão computacional**



Fonte:disponível em: <[http://www.cognex.com/uploadedImages/Press\\_Releases\\_Items/Press\\_Release\\_PDFs/In-Sight\\_TT-press-landscape.jpg](http://www.cognex.com/uploadedImages/Press_Releases_Items/Press_Release_PDFs/In-Sight_TT-press-landscape.jpg)>.

**Figura 5 – Sistema de identificação de caracteres em placas automotivas**

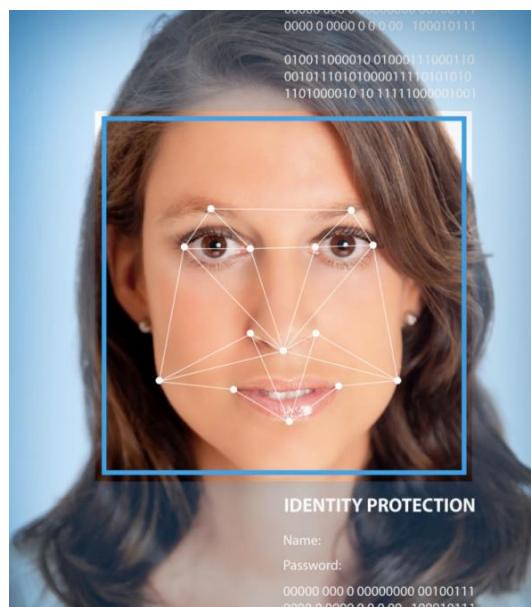


Fonte: disponível em: <<http://fabricioleite.com.br/conteudo/artigo/15/1.jpg>>.

---

<sup>1</sup> Código de barras bidimensional que pode ser facilmente escaneado.

**Figura 6 – Reconhecimento biométrico da face**



Fonte: disponível em: <<https://exagon.files.wordpress.com/2016/01/biometrics.jpg>>.

**Figura 7 – Segmentação de regiões por sensoriamento remoto (geoprocessamento)**



Fonte:disponível em: <[http://www.processamentodigital.com.br/wp-content/uploads/2014/06/Landsat\\_81-696x341.jpg](http://www.processamentodigital.com.br/wp-content/uploads/2014/06/Landsat_81-696x341.jpg)>.

Uma maneira de segmentar estas áreas segundo Gonzales e Woods (2010) é pela divisão do espectro eletromagnético aplicado as imagens obtidas como mostra a figura 8, sendo comumente mais utilizada e presente o espectro visível a olho humano, assim é possível englobar toda forma conhecida de processamento de imagem.

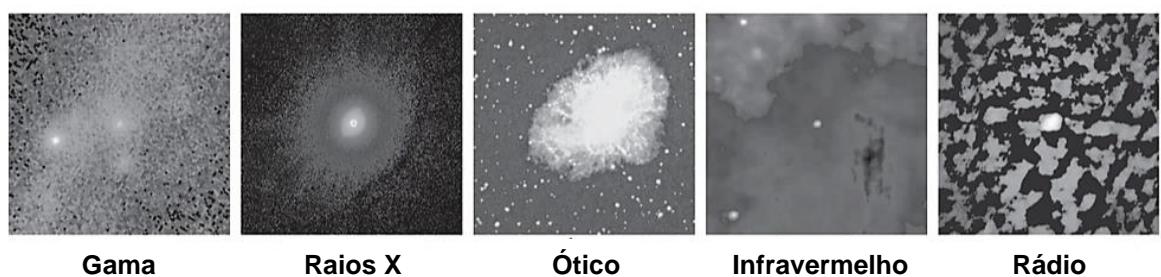
**Figura 8 – Espectro eletromagnético em função da energia por fóton em elétron-volts (eV).**



Fonte: GONZALES; WOODS, 2010, p. 5.

A seguir estão representadas algumas imagens que diferem entre si na fonte utilizada para visualizar a mesma região, dependendo da aplicação é necessária uma melhor definição para poder identificar pontos importantes no estudo, porém requerem uma quantidade de energia maior.

**Figura 9 – Imagens de um pulsar<sup>2</sup> gerado por uma estrela de nêutrons<sup>3</sup> (Crab Pulsar) em seus espectros eletromagnéticos**



Fonte: GONZALES; WOODS, 2010, p. 13, cortesia da NASA.

### 2.2.3 Sistema de inspeção visual em linha de produção

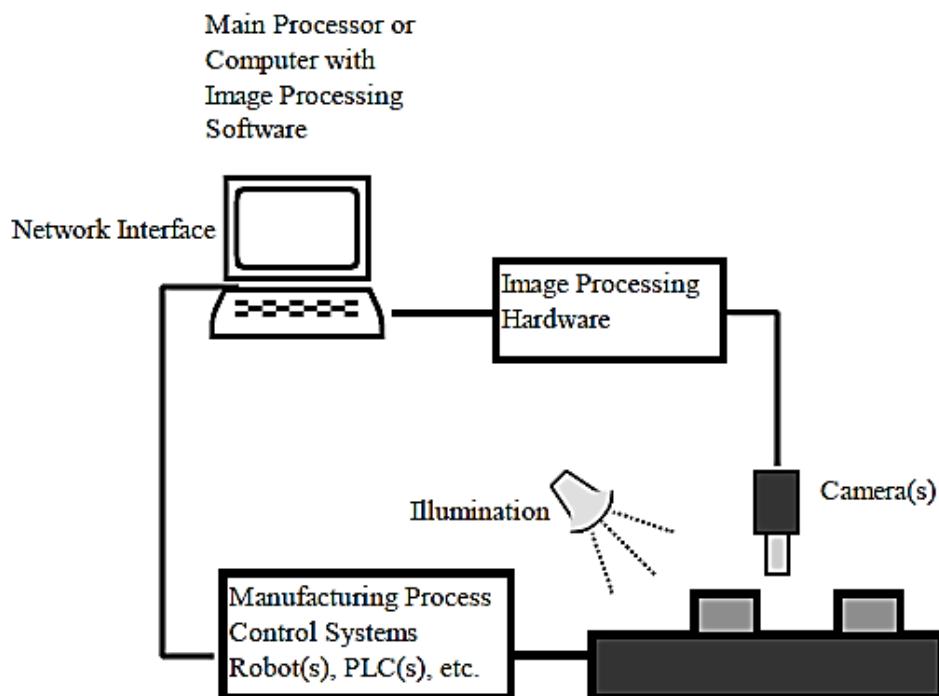
A mão de obra humana é eficaz para inspeções visuais pela facilidade em encontrar anomalias no produto e flexibilidade para mudanças no ambiente, porém esse trabalho é mais lento, suscetível a erros, gera custos com funcionários e treinamentos, entre outros fatores como a fadiga causada pela repetitividade e doenças ocupacionais geradas pela mesma. Isso tudo pode ser melhorado automatizando-se o processo, cujo envolve máquinas que efetuam o serviço mais rápido, podem atingir a máxima confiabilidade, trabalhando sem intervalos e ao mesmo tempo ter controle do processo inspecionando a qualidade do produto (DAVIES, 2012; GROOVER, 2011).

<sup>2</sup> Estrela de nêutrons rotativa que gera pulsos regulares de radiação em sua taxa de rotação.

<sup>3</sup> Núcleo colapsado de uma estrela maciça que permanece após a explosão de uma supernova.

Como explicam Legat, Malamas, Petit, Petrakis e Zervakis (2003) um sistema de inspeção em linha geralmente possui o mesmo tipo de funcionamento. Constituído por algum hardware de processamento, um computador por exemplo, onde este irá realizar o processamento das imagens e também manter o controle do processo. Já para a obtenção das imagens normalmente são utilizadas uma ou mais câmeras, sendo estas na maioria das vezes fixas. Outra parte que auxilia bastante na obtenção das imagens é a utilização de sistema de iluminação artificial, onde com a variação do ambiente podem haver modificações que o software não está preparado para identificar e com uma iluminação corretamente projetada isso pode ser evitado. As imagens captadas e já tratadas pelo hardware de processamento, geralmente incluso com a câmera, então são enviadas para o computador processar e tomar alguma medida se necessário, esse comando pode ser emitido à um CLP ou diretamente acionar um sistema de rejeição ou até mesmo um robô. Esses sistemas na maioria dos casos são projetados para detectar peças ou produtos já conhecidos. A figura 10 demonstra a composição de um sistema básico.

**Figura 10 – Sistema de inspeção em linha básico**



Fonte: LEGAT; MALAMAS; PETIT; PETRAKIS; ZERVAKIS, 2003.

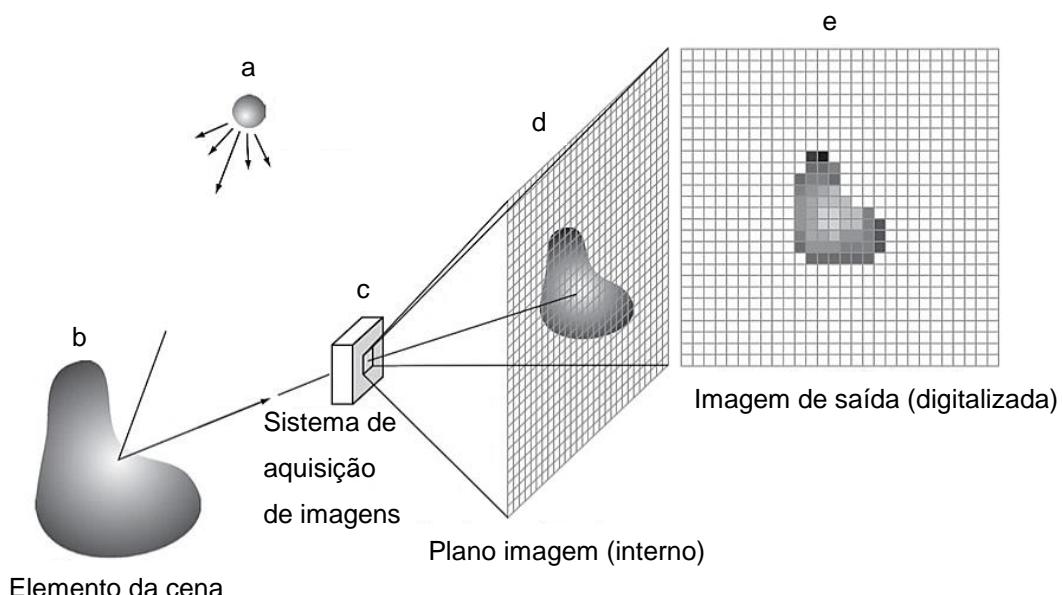
## 2.2.4 Sensores ópticos

Sensores ópticos ou câmeras são os tipos de sensores comumente utilizados para adquirir imagens a partir da energia absorvida ou refletida nos objetos de uma cena a partir de uma fonte de iluminação, para converter essa energia em imagens digitais Gonzales e Woods (2010, p. 30) citam que:

A energia que entra é transformada em tensão pela combinação da energia elétrica de entrada e do material do sensor, sensível a um tipo específico de energia que está sendo detectado. A forma de onda da tensão de saída é a resposta do(s) sensor(es), e uma quantidade digital é obtida de cada sensor por meio da digitalização de sua resposta.

Para a obtenção de imagens digitais em inspeções de linhas de produção e outras aplicações, assim como nas câmeras digitais pessoais o tipo de sensoriamento utilizado é o matricial, onde sensores unitários são arranjados em formato de matriz bidimensional equivalente aos pixels da imagem como demonstra a figura 11. Assim a imagem será formada através da captura da câmera na cena obtendo a integral de energia luminosa captada sobre esses sensores o que gera a função  $f(x, y)$  que é o produto da iluminação<sup>4</sup> com a refletância<sup>5</sup> (GONZALEZ; WOODS, 2010).

**Figura 11 – Leitura de cena formando a imagem na matriz de sensores**



Fonte: GONZALES; WOODS, 2010, p. 32. Editado pelo autor.

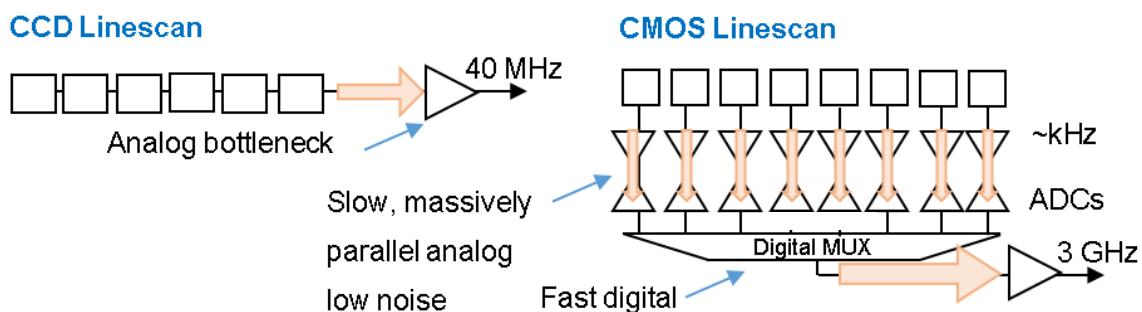
<sup>4</sup> Quantidade de luz que incide sobre o objeto.

<sup>5</sup> Fração de luz incidente que o objeto irá refletir ao ponto (x, y).

Atualmente no mercado de câmeras industriais para visão computacional existem duas arquiteturas baseadas nos conceitos acima citados, são os modelos CCD<sup>6</sup> e CMOS<sup>7</sup>, sistemas inventados nas décadas de 60 e 70 pelo cientista, professor e fundador da empresa DALSA Corporation do ramo de imagem digital, Savvas Chamberlain. Na época os sistemas baseados em semicondutor não eram tão eficientes como hoje por isso a tecnologia CMOS não foi bem-vinda, porém com os avanços tecnológicos os projetos de CMOS foram retomados nos anos 90 e hoje são levados como melhor escolha na maioria das aplicações, pois gera menos gasto em energia e é mais rápido, seu custo em larga escala pode ser reduzido, mas pelo fato do projeto de CMOS ser mais complexo os CCDs têm custo mais baixo (DALSA, ca. 2016).

A rapidez que difere o sistema CMOS de CCD é a forma como estão projetados os sensores, CCDs dependendo da construção podem ter apenas um ou alguns nós de saída de sinal, esse sendo analógico, já nos CMOS cada sensor possui uma saída analógica que ainda possui alguns amplificadores operacionais para filtrar o sinal que logo após é digitalizado paralelamente como mostra a figura 12.

**Figura 12 – CCD x CMOS leitura por varredura em linha**



Fonte:disponível em: <[https://www.teledynedalsa.com/images/imaging/CCDvsCMOSline\\_450w.jpg](https://www.teledynedalsa.com/images/imaging/CCDvsCMOSline_450w.jpg)>. Editado pelo autor.

Em aplicações com fonte geradora de energia visível CMOS se sai melhor, porém para fontes de energia mais baixas como infravermelho o CCD tem

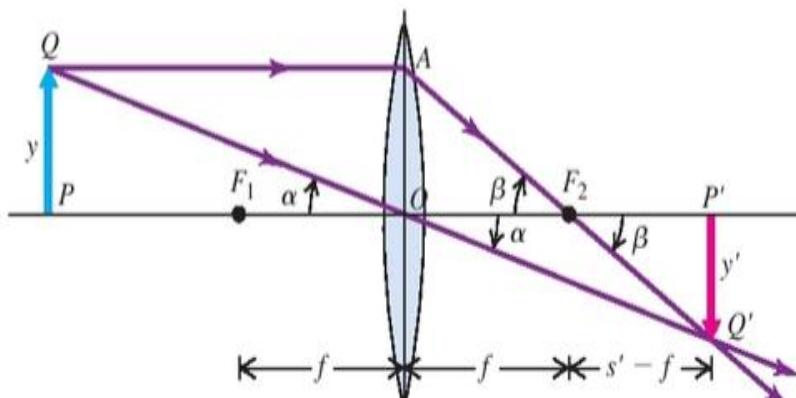
<sup>6</sup> Dispositivo de carga acoplada.

<sup>7</sup> Semicondutor de óxido de metal complementar.

vantagem. Vai depender da análise do tipo de trabalho a ser realizado para definir corretamente qual é a melhor opção (DALSA, ca. 2016).

Para comentar um pouco sobre a parte óptica das câmeras é possível relacionar com o conteúdo de lentes delgadas de física. As lentes são superfícies refratoras e a lente convergente possuí a propriedade de convergir uma imagem real do ponto  $P$  ao outro  $P'$  como mostra a figura 13, onde  $f$  é a distância focal,  $s$  e  $s'$  são as distâncias do objeto e da imagem respectivamente,  $y$  e  $y'$  são as alturas do objeto e da imagem nesta ordem também. Como ilustrado na figura o ângulo  $\alpha$  em ambos os lados são iguais, sendo assim é possível gerar relação pela similaridade dos triângulos dos dois lados entre as alturas e as distâncias, o mesmo é válido para o ângulo  $\beta$  (FREEDMAN; YOUNG, 2016).

**Figura 13 – Relações trigonométricas formadas por lente delgada**



Fonte: FREEDMAN; YOUNG, 2016, p. 57.

$$\frac{y}{s} = -\frac{y'}{s'} \quad (1)$$

$$\frac{y}{f} = -\frac{y'}{s' - f} \quad (2)$$

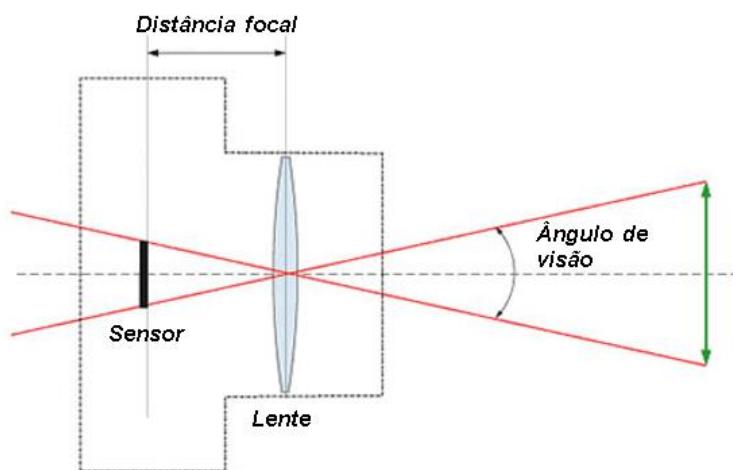
O sinal negativo nas equações representa a imagem projetada que está invertida abaixo do eixo óptico. Com estas equações é possível ainda determinar a ampliação transversal (eq. 3) e a relação objeto-imagem (eq. 4).

$$m = \frac{y'}{y} = -\frac{s'}{s} \quad (3)$$

$$\frac{1}{f} = \frac{1}{s} + \frac{1}{s'} \quad (4)$$

De uma forma simples estes conceitos se aplicam as câmeras que também possuem lentes delgadas, como mostra a ilustração básica da figura abaixo.

**Figura 14 – Distância focal em uma câmera**



Fonte: disponível em:<<http://todo-fotografia.com/wp-content/uploads/2013/02/longitud-focal21.jpg>>.

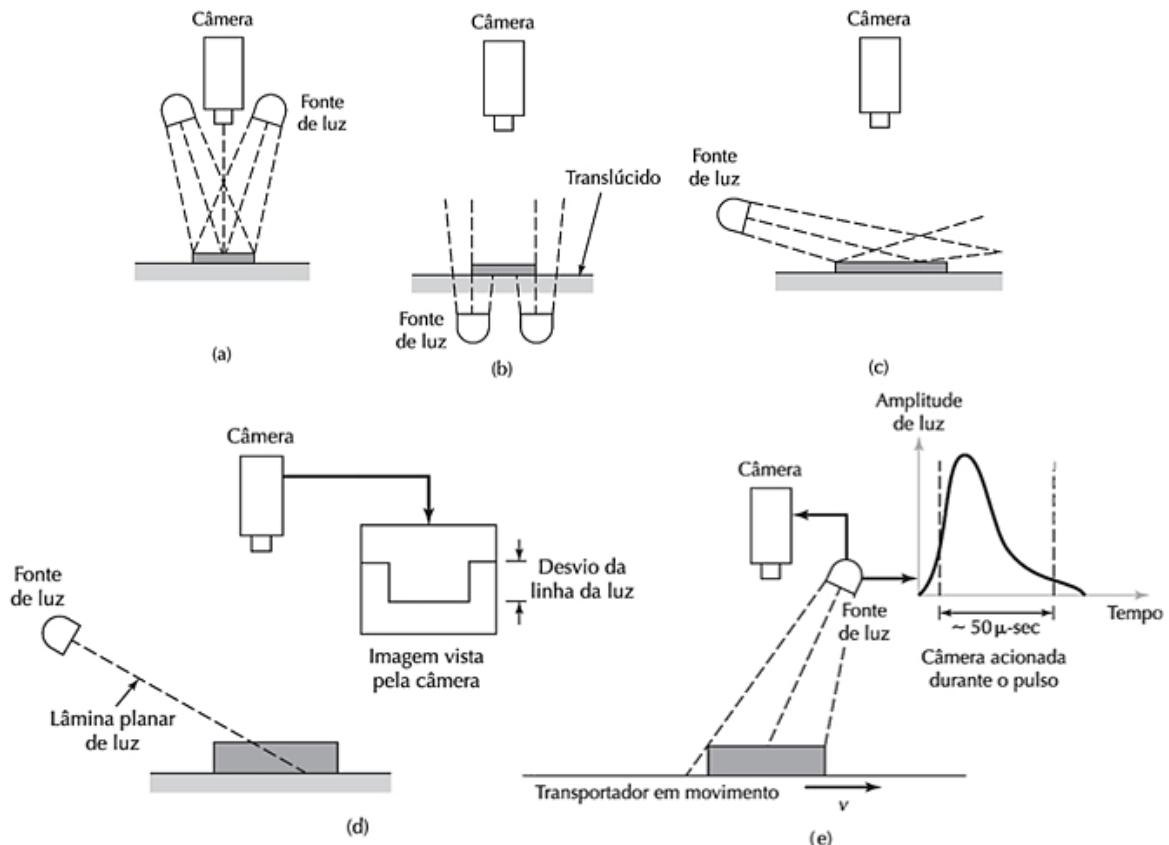
Editado pelo autor.

## 2.2.5 Iluminação

A iluminação é um fator crucial para melhorar a precisão de leitura de imagens no processo de inspeção, o que facilita na interpretação de máquina da cena obtida, assim como em estúdios fotográficos a iluminação serve para causar efeitos sobre algum objeto, cena ou pessoa, realçando características como bordas, irregularidades, cores, etc.

Segundo Groover (2011) a iluminação para visão computacional pode ser separada em cinco modelos de montagem como mostra a figura 15, cada uma direcionada à alguma aplicação específica.

**Figura 15 – Modelos de iluminação para uso em visão computacional: a) Superior; b) De fundo; c) Lateral; d) Estruturada; e) Estroboscópica.**



Fonte: GROOVER, 2011, p. 552.

A iluminação superior, onde a fonte de luz fica paralela a câmera e direciona a luminosidade para a área de interesse, assim sendo refletido quando um objeto está presente, serve para analisar as características da superfície do objeto, como cor e impressão.

A iluminação de fundo (*backlight*) utiliza a fonte de luz abaixo de uma lâmina translúcida por onde a peça irá passar, assim a base da área será iluminada e o objeto ficará mais escurecido assim destacando sua forma, este modelo é geralmente utilizado para adquirir as medidas com maior precisão das peças.

Na iluminação lateral o objetivo é gerar sombras para avaliar se há irregularidades na superfície como rebarbas de metal ou deformações na composição do material de objetos sólidos.

Para formar efeitos geométricos para uma melhor identificação do desenho de uma peça um método é utilizar iluminação estruturada, podendo ter disposições diferentes, assim será possível determinar ângulos e até a altura do objeto através da

sombra gerada, um modelo comumente utilizado é a aplicação de um feixe de luz direcionado a um ângulo pré-estipulado.

O último modelo é bastante utilizado também, normalmente em leitura de códigos e caracteres em produtos para fazer criar um efeito paralização do objeto mesmo que este esteja em movimento, essa iluminação funciona aplicando-se pulsos de alta intensidade de luz em períodos de tempo, esta iluminação é conhecida por estroboscópica.

Outro detalhe importante a ressaltar a respeito de iluminação é sobre aspectos de cores, que conforme a norma ABNT NBR ISO/CIE 8995-1 (2013) podem ser relevantes ao necessitar ajustes nas cores da imagem. A norma também indica que estes aspectos podem ser divididos em duas partes, sendo essas a de aparência de cor e capacidade de reprodução de cor.

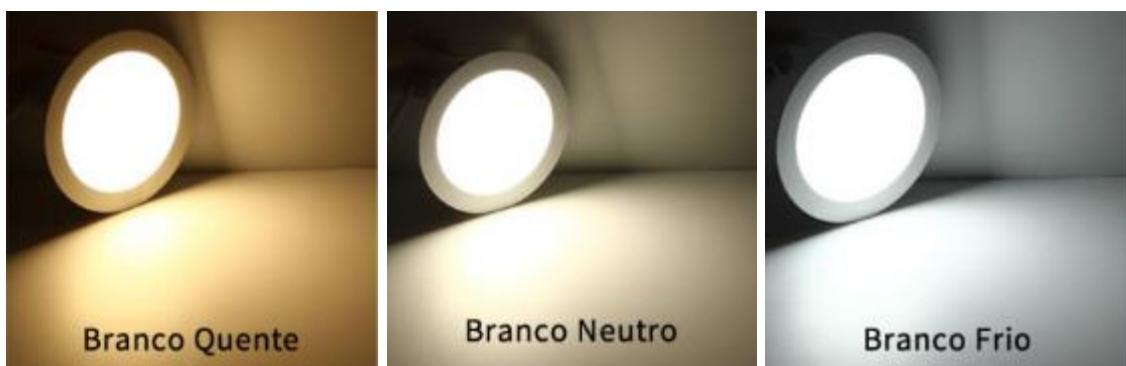
A aparência de cor de uma lâmpada está relacionada a sua cromaticidade como mostra a figura 16, ou seja, à cor aparente que é emitida, seguem as divisões conforme a tabela 2.

**Tabela 2 – Aparência de cor e suas classificações**

Aparência de cor	Temperatura de cor correlata
Quente	Abaixo de 3 300 K
Intermediária	3 300 K a 5 300 K
Fria	Acima de 5 300 K

Fonte: ABNT NBR ISO/CIE 8995-1

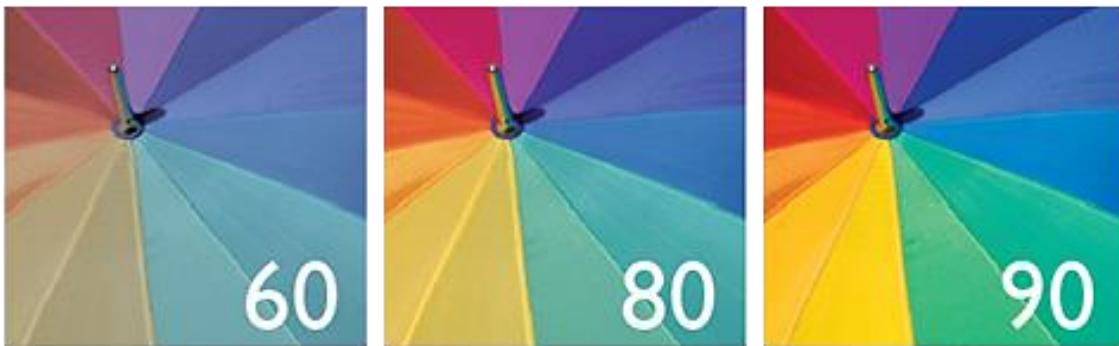
**Figura 16 – Diferença entre a temperatura de cor para lâmpada LED**



Fonte:disponível em <<http://mercadopark.com.br/wp-content/uploads/2016/11/LED-Temperaturas-de-cor-branco-quente-branco-frio-branco-neutro.png>>.

Já a reprodução de cor está ligada diretamente a reprodução fiel das cores do ambiente ou objetos, para quantificar este atributo foi criado o índice geral de reprodução de cor  $R_a$ , onde o valor máximo estipulado é de 100.

**Figura 17 – Diferença no nível do índice de reprodução de cor**



Fonte: disponível em: <<http://www.philips.com.br/c-dam/b2c/category-pages/lighting/generic-lighting-pages/led-pages/led-light-quality/led-cri-2-png2.PNG>>.

## 2.2.6 Hardwares

Para determinadas aplicações apenas a boa escolha do software ou um bom desenvolvimento do mesmo pode não satisfazer os requisitos necessários para o funcionamento correto do processamento dos dados, para isso é preciso analisar e buscar o hardware mais adequado a situação, levando em consideração na maioria das vezes um alto desempenho, assim como custos, flexibilidade de programação, tamanho, questões sobre consumo de potência e aquecimento, confiabilidade, entre outros. Nesse âmbito existem atualmente algumas tecnologias viáveis, são elas os DSPs, GPUs, CPUs, ASICs e FPGAs (LEGAT; MALAMAS; PETIT; PETRAKIS; ZERVAKIS, 2003).

Segundo Fykse (2013) Unidades de Processamento Central (CPUs) são muito utilizados no processamento de tarefas gerais para realizarem multitarefas, presentes em computadores e dispositivos móveis, porém quando se trata de processamento de imagem com o requisito de alto desempenho as CPUs não conseguem obter ótimos resultados, para isso foram desenvolvidas as Unidades de Processamento Gráfico (GPUs) cuja finalidade é voltada para processamento gráfico, trabalhando paralelamente para melhorar o desempenho em exibir imagens. Enquanto a CPU

possui multicores<sup>8</sup> de processamento a GPU possui milhares de cores para processamento voltado a sinais digitais, APUs vem criando solidez no mercado por ser um sistema que une CPU e GPU para uma melhor performance.

Para se ter um processamento rápido em tempo real na visão computacional os Processadores de Sinais Digitais (DSPs) assim como as Matrizes de Portas Lógicas Programáveis (FPGAs) e Circuitos Integrados de Aplicação Específica (ASICs) são projetados exatamente para determinada aplicação o que gera alto custo no design de programação dos arranjos, porém em larga escala esse custo é baixo e o hardware gera confiabilidade. Os DSPs são microprocessadores voltados para o processamento digital de sinais, geralmente utilizados para algoritmos de processamento de áudio e vídeo, envolvendo algumas operações matemáticas, já os ASICs são circuitos integrados projetados e fabricados para determinado uso, assim seu desenvolvimento é caro e sua programação é inalterável, porém como resultado estão o alto desempenho com baixa dissipação de potência sendo assim viável para aplicações em larga escala como na produção de eletrônicos de mesmo padrão. Já os FPGAs têm uma estrutura um pouco diferenciada, sendo constituído por portas lógicas sua programação baseada no próprio hardware também exige maior conhecimento e maior trabalho de desenvolvimento, mas diferente de um ASIC a programação da lógica digital não é fixa podendo ser alterada quando necessário (FYKSE, 2013).

**Tabela 3 – Comparação geral de hardwares**

Hardware	Desempenho	Custo	Consumo de energia	Linguagem de programação
CPU	Regular	Mediano	Alto	C/C++
GPU	Alto	Mediano	Alto	C/C++, Fabricante ex.: CUDA <sup>9</sup>
DSP	Alto	Mediano	Regular	Assembly, C/C++
ASIC	Alto	Baixo (larga escala)	Baixo	HDL/VHDL (fixa)
FPGA	Alto	Alto	Alto	HDL/VHDL

Fonte: FYKSE, 2013; TEXAS INSTRUMENTS, 2002. Editado pelo autor.

<sup>8</sup> Núcleos de processamento que executam instruções de cálculos ou outras ações.

<sup>9</sup> Linguagem de programação baseada em C criada pela empresa NVIDIA.

Na tabela 3 apresentada estão algumas comparações das tecnologias citadas, estimando as variações entre uma e outra e também definindo a linguagem de programação comumente utilizada, lembrando que esta é uma visão geral de cada tecnologia pois depende de cada equipamento individualmente a sua real eficiência e desempenho existindo inúmeras fabricantes e modelos disponíveis no mercado.

### **2.2.7 Interfaces de comunicação**

Assim como a escolha do hardware é essencial para realizar uma aplicação com melhor desempenho e sem gastos desnecessários, o padrão de interface de comunicação a ser adotado também influencia em como o sistema de visão computacional funcionará (FINTEL, 2013).

A interface de comunicação tem a função de ligar a câmera com seu hardware realizando a troca de informações entre um e outro de forma padronizada geralmente seguindo a ordem de encontrar a conexão com a câmera, acessar os registradores e após transmitir os sinais de vídeo e eventos. Atualmente existem algumas interfaces que são mais conhecidas na indústria que serão brevemente explicadas baseando-se nas principais associações de visão computacional do mundo, sendo as interfaces: Camera Link, Camera Link HS, CoaXpress, GigE Vision, USB3 Vision, USB 2.0, IEEE 1394 (AIA; EMVA; JIIA, 2016).

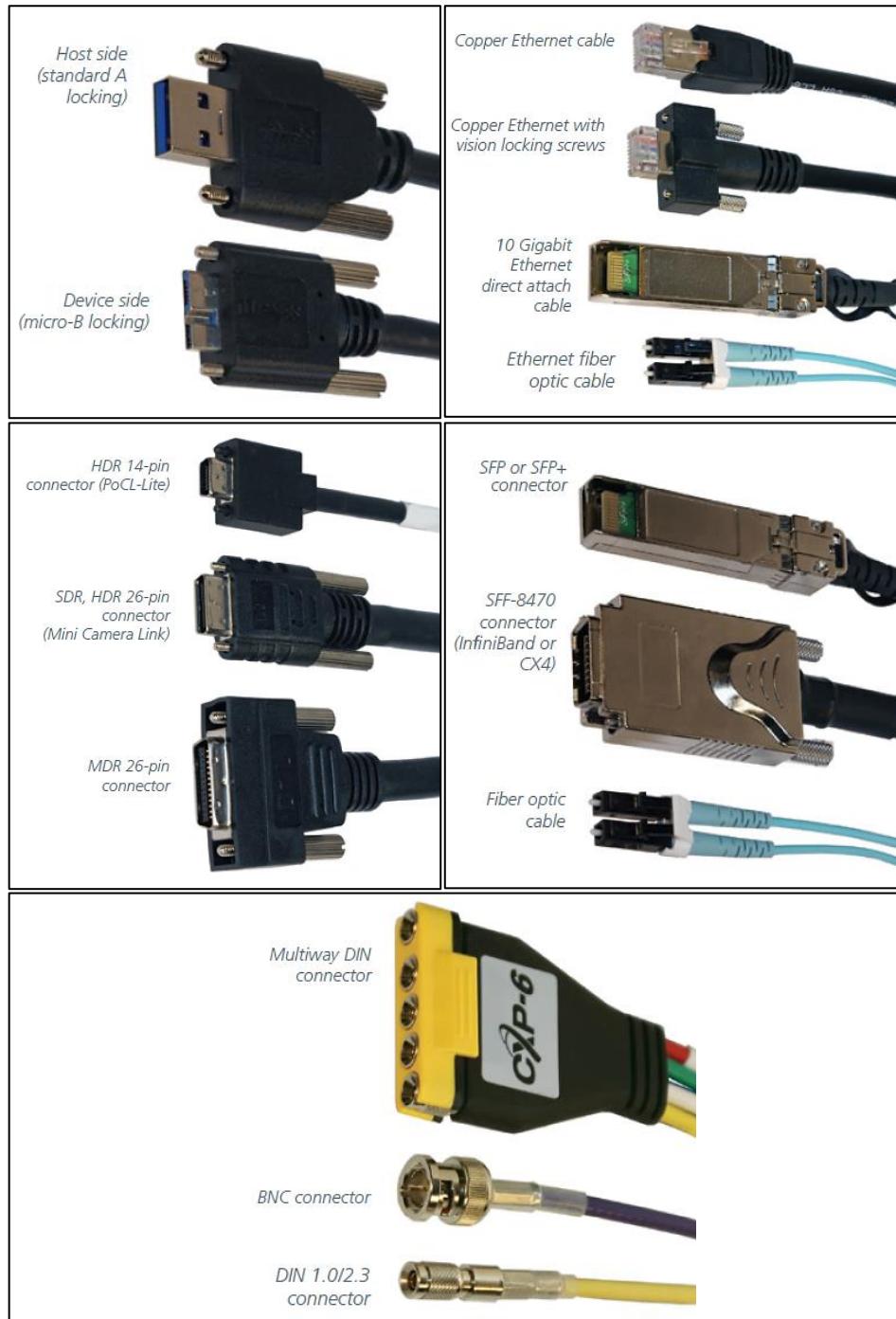
Algumas tecnologias que já estão entrando em desuso pela falta de suporte e suas limitações são a IEEE 1394 ou também conhecida por *FireWire*, que foi desenvolvida pela Apple Inc. em 1987 e a interface USB 2.0 que é popular em vários dispositivos eletrônicos (AIA; EMVA; JIIA, 2016).

**Figura 18 – Conectores FireWire e USB 2.0 respectivamente**



Fonte:disponível em: <[https://http2.mlstatic.com/S\\_14610-MLB183655165\\_9243-O.jpg](https://http2.mlstatic.com/S_14610-MLB183655165_9243-O.jpg)> e <<http://www.digimer.com.br/image/cache/data/produtos/gd/3114-700x700.jpg>>.

**Figura 19 – Conectores USB3 Vision, GigE Vision, Camera Link HS e CoaXpress respectivamente**



Fonte: AIA; EMVA; JIIA, 2016. Editado pelo autor.

O padrão USB3 Vision veio para substituir as interfaces acima citadas sendo mais estável, com maior velocidade na banda de transmissão para um bom desempenho em aplicações em tempo real, baixo custo do sistema, baixo consumo de energia e pouco consumo no carregamento da CPU. Utilizado em vários projetos

o padrão GigE Vision (Gigabit Ethernet) utiliza comunicação Ethernet (IEEE 802.3<sup>10</sup>) e tem a vantagem de ser possível utilizar cabos longos. Outro sistema comumente utilizado em visão computacional, principalmente em aplicações industriais que necessitem inspeção em alta velocidade o padrão Camera Link possui largura de banda de transmissão de alta capacidade, desenvolvida especialmente para processamento de imagem em tempo real, com aprimoramento em alguns aspectos como largura de banda surgiu o padrão Camera Link HS. Outro tipo de interface similar é o CoaXpress com boa eficiência em tempo real, porém um empecilho que pesa na hora de definir o padrão a ser utilizado é que alguns desses sistemas necessitam de placas de captura (*frame grabbers*) para conectar os cabos e ser intermediador da comunicação entre a câmera e o hardware utilizado. Para detalhar um pouco mais essas tecnologias e comparar quais as melhores opções para determinada aplicação foi elaborada a tabela 4 (AIA; EMVA; JIIA, 2016; FINTEL, 2013).

**Tabela 4 – Comparaçao entre padrões de interface de comunicação**

Padrão	Data primeiro lançamento	Data última versão	Comprimento de cabo (m)	Max. largura de banda (MB/s)	Dispositivo receptor	Desempenho (multicâmeras)	Desempenho (tempo real)
 FireWire	Ago 1996	Jul 2008	4,5	100	Plug and Play <sup>11</sup>	Regular	Bom
 USB 2.0	Abr 2000	-	5	40	Plug and Play	Regular	Ruim
 USB VISION	Jan 2013	Mar 2015	3-5, 8+, 100	400	Plug and Play	Regular	Bom
 GiGE VISION	Mai 2006	Abr 2013	10, 100, 500, 5000	125 - 1250	Plug and Play	Bom	Regular
 Camera Link	Out 2000	Fev 2012	4, 5, 10	850	Placa de captura	Ruim	Bom
 Camera Link HS	Mai 2012	Mai 2012	15, 10, 500, 5000	2100	Placa de captura	Bom	Bom
 CoaXPress	Dez 2010	Nov 2015	35, 85, 105	3600	Placa de captura	Bom	Bom

Fonte: AIA; EMVA; JIIA, 2016; FINTEL, 2013. Editado pelo autor.

<sup>10</sup> Conexão padrão que especifica a camada física e a subcamada MAC da camada de ligação dos dados do modelo OSI para o protocolo Ethernet.

<sup>11</sup> Tecnologia ligar e usar onde o computador reconhece e configura automaticamente qualquer dispositivo que seja instalado.

## 2.2.8 Softwares

Os softwares para visão computacional conectam hardware com a interface de comunicação para fazer todo o controle e gerenciamento das informações obtidas e mostra-las de forma clara e voltada para a área da aplicação.

Hoje em dia existem vários softwares específicos para cada necessidade, sendo a parte de interface muitas vezes fornecida pelo próprio fabricante de câmeras de visão computacional, já a parte de *middleware* ou mediador é o ambiente de desenvolvimento que pega os dados adquiridos do hardware e utiliza da programação de alto nível para aplicar métodos de tratamento, entendimento e fornecimento de respostas para a interface ou para a saída do hardware, para conseguir essa implementação se faz uso de bibliotecas e APIs, que serão explicadas mais a frente, onde são voltadas para a necessidade da aplicação (BHATTACHARYYA; KISACANIN; CHAI, 2009).

Na tabela 6 está disposto um resumo de alguns mediadores para desenvolvimento em visão computacional e a área de atuação onde geralmente são empregados, está disponível no (APÊNDICE A).

## 2.3 Processamento digital de imagens

Percebeu-se na década de 20 que as imagens impressas assim como as imagens que captamos a nossa volta tinham grande valor, contendo informações importantes para determinar estudos ou apenas registrar momentos históricos. Para isso se iniciou o aprimoramento no processamento de imagens para que fosse possível interpretar cenas e obter melhores definições e resultados.

Mas foi nos anos 60 com o surgimento de computadores mais robustos que a área de processamento de imagens se ampliou, onde laboratórios de pesquisa como *Jet Propulsion Laboratory*<sup>12</sup> (Pasadena, California – EUA) evoluíram com técnicas de processamento para conseguir melhorar as imagens provindas da lua, captadas pela sonda espacial *Ranger*<sup>13</sup>, cujo haviam avarias e granulações na imagem, assim

<sup>12</sup> Centro tecnológico norte-americano, responsável pelo desenvolvimento e manuseio de sondas espaciais não tripuladas para a NASA.

<sup>13</sup> O programa Ranger foi conduzido pelos Estados Unidos na década de 60 para conduzir missões espaciais não tripuladas com objetivo de captar imagens da superfície lunar.

podendo dar base para outras expedições como do programa Apollo da NASA (GONZALEZ; WOODS, 2010; MARQUES FILHO; VIEIRA NETO, 1999).

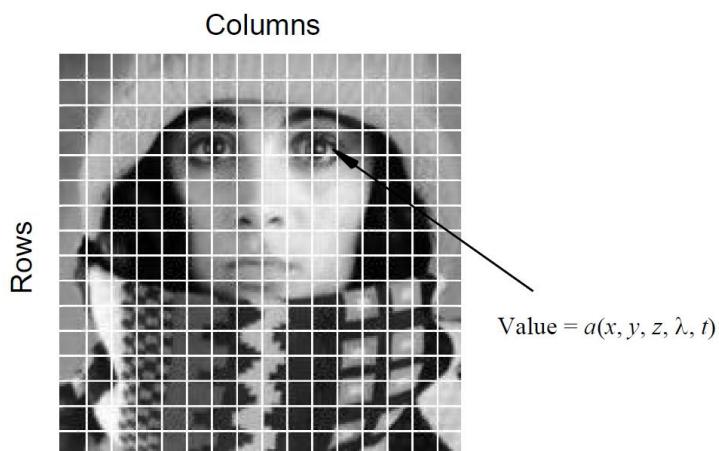
A importância do processamento de imagens segue sendo grandemente valorizada e cada vez mais madura, hoje atuando em diversas áreas como na medicina, meteorologia, indústria, entretenimento, publicidade, espaço, segurança, etc.

Nesta seção serão tratados conteúdos referentes a imagem como sinal digital e alguns métodos de manipulação para melhoria e interpretação da mesma, seguindo a ordem de processamento indicada pela maioria dos especialistas e estudiosos da área.

### 2.3.1 Fundamentos do processamento de imagens

Segundo Gerbrands, Young, Van Vliet (1998) e Pratt (2001) a imagem (a) quando tratada no plano 2D é considerada, nos sistemas digitais, como sendo uma matriz composta por linhas ( $x$ ) e colunas ( $y$ ) que representam uma coordenada espacial, a intersecção da linha com a coluna é denominada pixel, onde podem conter outros atributos como profundidade ( $z$ ), tempo ( $t$ ) e cor em comprimento de onda ( $\lambda$ ) como demonstra a figura 20.

**Figura 20 – Matriz de imagem em escala de cinza**



Fonte: GERBRANDS; YOUNG; VAN VLIET, 1998.

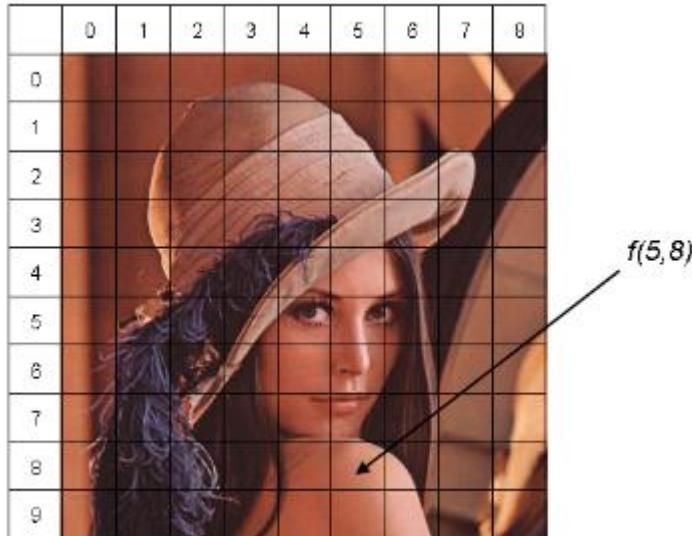
Como foi mencionado na seção sobre sensores ópticos os pixels da imagem são formados pela intensidade luminosa naquele ponto, cujo resultado é  $f(x,y)$  que

corresponde ao nível de cinza ou brilho relativo, o valor é baseado no produto da iluminância  $i(x,y)$  com a refletância  $r(x,y)$  (ALMEIDA; MENESES, 2012; GONZALES; WOODS, 2010; MARQUES FILHO; VIEIRA NETO, 1999).

$$f(x,y) = i(x,y).r(x,y) \quad (5)$$

A matriz pode conter várias linhas e colunas e cada célula será composta pela sua função onde a coordenada geralmente se inicia em 0 como mostra figura 21 (ALMEIDA; MENESES, 2012).

**Figura 21 – Imagem dividida em uma matriz 9X10**



Fonte: imagem disponível em: <<http://shepelavy.com/blog/wp-content/uploads/2009/03/lena.jpg>>.

Editado pelo autor.

Uma imagem monocromática é composta pela mesma matriz de funções podendo variar a quantidade de níveis de cinza, comumente se utilizam valores positivos e inteiros. A diferença de uma imagem monocromática ou em escala de cinza, para uma imagem colorida está na quantidade de camadas, onde para formar uma imagem colorida baseada no sistema RGB existem 3 camadas com matrizes de mesmo tamanho, onde a sobreposição dos níveis de brilho das cores vermelha, verde e azul resultam em variadas cores perceptíveis ao olho humano, sendo considerados, para o sistema RGB, valores de 0 à 255 equivalentes a 1 byte de memória, esta sobreposição é a combinação entre cores básicas para gerar determinada cor. Outros espaços de cores são utilizados em algumas aplicações diferentes, como para

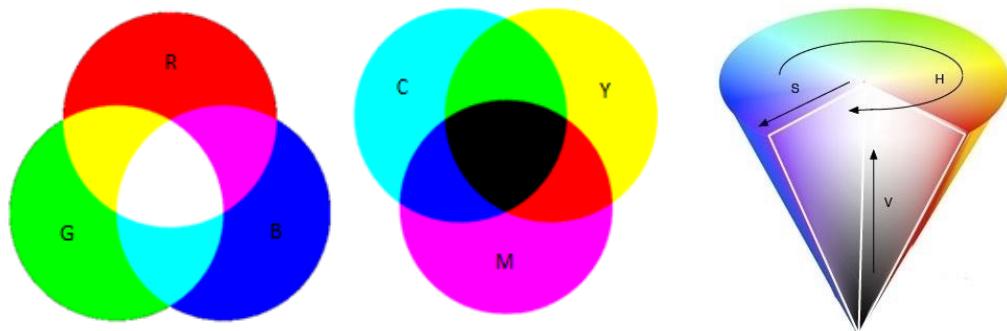
impressão é utilizado o sistema CMY, onde aplicam-se as cores ciano, magenta e amarelo para misturar e formar a imagem desejada, já em pinturas é comum artistas preferirem o sistema HSV, formado pela variação de cor, saturação e brilho (ALMEIDA; MENESSES, 2012; BOYLE; HLAVAC; SONKA, 2008).

**Figura 22 – Comparação imagem colorida (RGB) e monocromática**



Fonte: imagem disponível em: <<https://www.designerd.com.br/wp-content/uploads/2016/02/a-colorida-e-psicodelica-arte-de-nicky-barkla-1.jpg>>. Editado pelo autor.

**Figura 23 – Modelo de cores RGB, CMY e HSV consecutivamente.**



Fonte: disponível em: <<http://www.cardquali.com/imagens/discosdecores.gif>> e <[http://www.medicinalmac.com/img/pasted\\_image2.jpg](http://www.medicinalmac.com/img/pasted_image2.jpg)>.

Se tratando de dados digitais os pixels da imagem são salvos em bits<sup>14</sup> na memória, por isso é mais viável utilizar valores inteiros positivos do que pontos flutuantes como mencionam Almeida e Meneses (2012), um valor de amplitude de

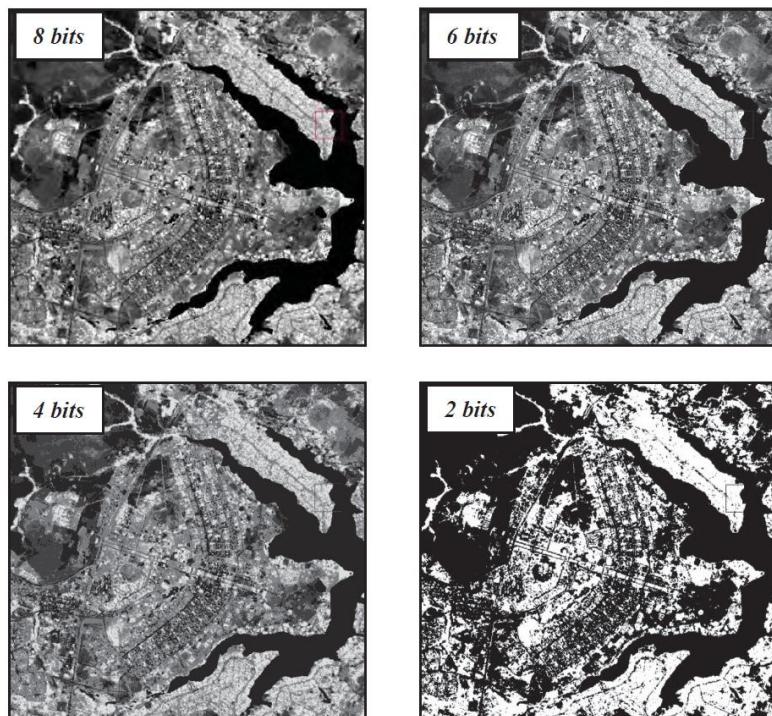
---

<sup>14</sup> Dígito binário, é a menor unidade de informação que pode ser armazenada ou transmitida.

brilho como inteiro ocupa 1 byte já para valores fracionários seriam necessários 2 bytes por pixel.

Como mostra a figura 24, dependendo da necessidade do processamento da imagem o sistema pode ser otimizado para ocupar menos espaço de memória utilizando menos bits para os níveis de cinza da imagem e mesmo assim obter um resultado satisfatório cuja a interpretação seja possível.

**Figura 24 – Mudança dos níveis de cinza no processamento por sensoriamento remoto**

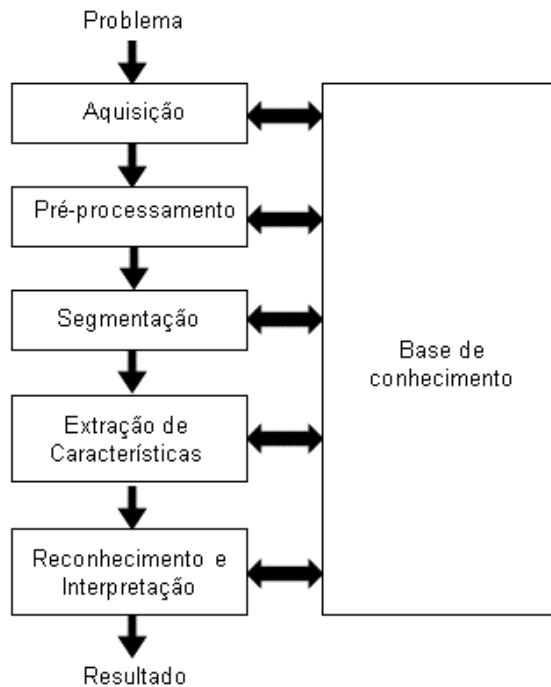


Fonte: ALMEIDA; MENESES, 2012, p. 30.

### 2.3.2 Passos do processamento

Para se realizar um processamento de imagem é viável utilizar alguns métodos de processamento, tanto técnicas para aplicação de filtros como uma metodologia a ser seguida comenta Gonzales e Woods (2010), esse processo pode ser dividido em dois tipos, um quando a entrada é uma imagem e a saída é a imagem tratada e outro quando a entrada é uma imagem e a saída são os atributos obtidos desta imagem. Porém de qualquer maneira é necessário conhecer as técnicas de se processar imagem pois em ambas as situações isto é necessário.

**Figura 25 – Principais passos para o processamento de imagem**



Fonte: MARQUES FILHO; VIEIRA NETO, 1999.

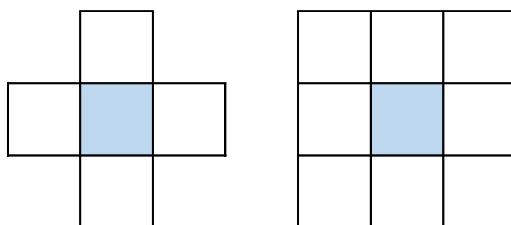
Os passos básicos para se obter resultados satisfatórios em processamento de imagens seguem a sequência ilustrada na figura 25. Assim como está representado, o primeiro item logo após se ter noção do problema ou necessidade é a aquisição da imagem, por meios digitais provindos de locais virtuais ou de hardwares com sensores ópticos. Seguindo, o sistema computadorizado irá realizar as próximas etapas do processo, aplicando filtros passa baixa, passa alta, entre outras técnicas de melhoramento e segmentação de imagens, assim pode-se extrair, a partir de lógicas, atributos de objetos ou de regiões de interesse, com isso é possível armazenar, analisar e exibir os resultados obtidos (GONZALES; WOODS, 2010; MARQUES FILHO; VIEIRA NETO, 1999).

### 2.3.3 Pré-processamento

Esta etapa do processo tem a função de corrigir erros e variações da imagem obtida pelo sensor, também é possível aplicar filtros que possibilitem melhorar ou destacar áreas de interesse na imagem, cujo servirão para interpretação no processo subsequente (MARQUES FILHO; VIEIRA NETO, 1999).

Antes de apresentar alguns tipos de filtros aplicáveis ao processamento digital de imagens é necessário conhecer o conceito de vizinhança. Segundo Pedrini e Schwartz (2008), Gonzales e Woods (2010) vizinhos são, em imagens bidimensionais, pixels com ligação vertical e horizontal em relação ao pixel  $f(x,y)$  em análise, na figura 26 pode ser comparada a relação entre “vizinhança-4” e “vizinhança-8” de um pixel.

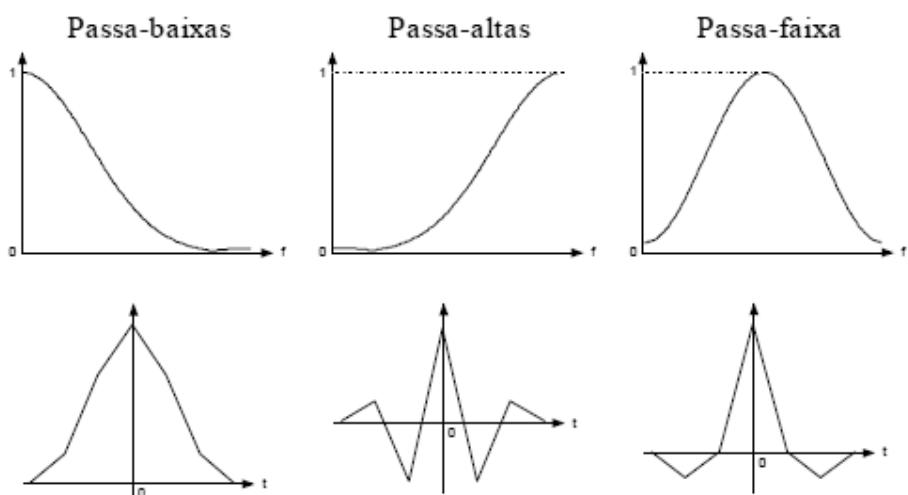
**Figura 26 – Vizinhança-4 e vizinhança-8 de um pixel**



Fonte: PEDRINI; SCHWARTZ, 2008. Editado pelo autor.

Os filtros podem ser tratados no domínio da frequência quanto do espaço, onde geralmente são utilizados filtros passa-baixa, passa-alta e passa-faixa. O domínio espacial está relacionado diretamente à matriz de pixels que forma a imagem, trabalha-se com valores dos níveis de cinza do ponto de interesse ou dos pontos de vizinhança. Já o domínio da frequência se baseia na convolução entre a imagem  $f(x,y)$  e um operador linear  $h(x,y)$ , trabalhando com aplicações da transformada de Fourier<sup>15</sup> (BAXES, 1994; GONZALES; WOODS, 2010; PEDRINI; SCHWARTZ, 2008).

**Figura 27 – Tipos de filtros no domínio da frequência e do espaço respectivamente**



Fonte: MARQUES FILHO; VIEIRA NETO, 1999.

<sup>15</sup> Transformada integral que expressa uma função em termos de funções de base senoidal.

Filtros passa-baixa atenuam sinais em altas frequências, resultando em imagens mais suavizadas, filtros passa-alta atenuam sinais em baixas frequências, que corresponde a um melhoramento na visualização de bordas (BAXES, 1994; GONZALES; WOODS, 2010; PEDRINI; SCHWARTZ, 2008).

Existem vários métodos para este tipo de processamento, porém serão demonstrados apenas alguns elaborados no domínio espacial, que são comumente utilizados e possuem boa eficiência. Para filtros passa-baixa os tipos mais conhecidos são de média, mediana, bilateral, gaussiano. Quando a necessidade for realce de bordas e linhas os filtros passa-alta mais conhecidos são Laplaciano, high-boost e gradientes de Sobel e Roberts.

### 2.3.3.1 Filtros passa-baixa

A maneira de aplicação destes filtros é baseada na utilização de matrizes chamadas de máscaras, onde o resultado desta operação será a substituição do valor do pixel em análise por um novo valor relacionado aos pixels vizinhos e aos pesos das máscaras (PEDRINI; SCHWARTZ, 2008).

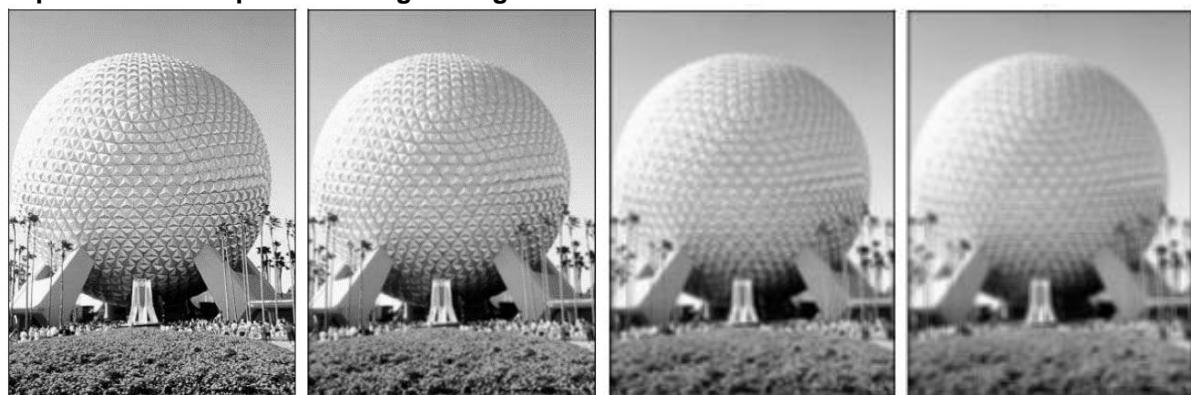
A resposta dos filtros passa-baixa deve necessariamente ser constituídos por uma máscara de coeficientes positivos, uma maneira simples de construir uma máscara para este fim é realizar a convolução por uma matriz de abertura 3x3 constituída por coeficientes com valor igual a 1 e após esta operação dividir por um fator de normalização, assim segue-se o mesmo padrão para máscaras com abertura maior como mostra a figura 28, este método é conhecido por ser um filtro de média (MARQUES FILHO; VIEIRA NETO, 1999).

**Figura 28 – Máscaras para aplicação de filtro de média de tamanhos 3x3, 5x5 e 7x7 respectivamente**

$$\begin{aligned}
 & \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{25} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \quad \frac{1}{49} \begin{bmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{bmatrix}
 \end{aligned}$$

Fonte: MARQUES FILHO; VIEIRA NETO, 1999

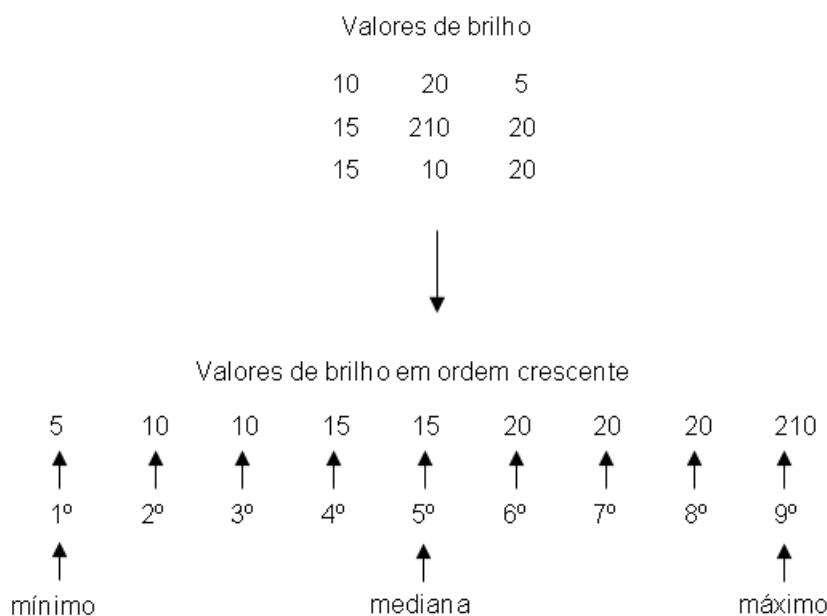
**Figura 29 – Respostas da aplicação de filtro de média de tamanhos 3x3, 5x5 e 7x7 respectivamente a partir da imagem original**



Fonte: MARQUES FILHO; VIEIRA NETO, 1999.

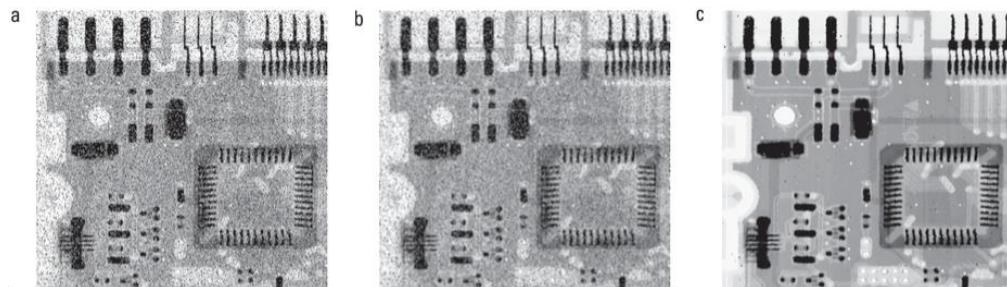
O filtro de mediana é uma das técnicas com melhor resultado no processamento de imagens com ruídos impulsivos (sal-e-pimenta), de forma simples consiste em formar um ranking a partir dos valores da região de interesse em  $f(x,y)$  com vizinhança pré-determinada e definir a resposta naquele ponto, que será a mediana do ranking, ou seja, o valor encontrado na posição do meio conforme ilustra a imagem 28 (BAXES, 1994; GONZALES; WOODS, 2010; MARQUE FILHO; VIEIRA NETO, 1999; SZELISKI, 2010).

**Figura 30 – Obtenção da resposta de um filtro mediana**



Fonte: BAXES, 1994. Editado pelo autor.

**Figura 31 – Exemplo de aplicação de filtro passa-baixa: a) Imagem com ruído; b) Imagem com filtro de média; c) Imagem com filtro mediana**



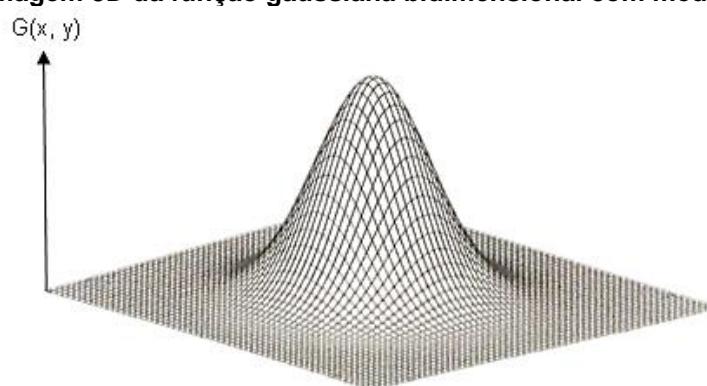
Fonte: GONZALES; WOODS, 2010.

Para redução de ruídos e suavização de imagens que apresentam nível de brilho baixo o filtro Gaussiano é mais indicado, além de ter uma distribuição simétrica para toda a imagem a função Gaussiana pode ser dividida, o que aumenta a velocidade de processamento por necessitar menos cálculos matemáticos. O filtro Gaussiano é baseado em uma máscara criada a partir da função Gaussiana discreta que depende do desvio padrão. O valor do pixel analisado será substituído pela média ponderada de sua vizinhança, sendo que o peso atribuído a determinado vizinho diminui uniformemente conforme se distancia do pixel central (PEDRINI; SCHWARTZ, 2008; PRINCE, 2012).

A função Gaussiana discreta pode ser definida a partir da seguinte equação, com média igual a zero:

$$G(x, y) = \frac{1}{2\pi\sigma^2} \exp\left(\frac{-(x^2 + y^2)}{2\sigma^2}\right) \quad (6)$$

**Figura 32 – Imagem 3D da função gaussiana bidimensional com média (0,0) e  $\sigma = 1$ .**



Fonte: PEDRINI; SCHWARTZ, 2008. Editado pelo autor.

Uma das formas de se chegar aos valores dos coeficientes da expansão binomial é utilizar o triângulo de Pascal, assim extraíndo os valores da n-ésima linha podemos definir a máscara de tamanho  $n$  e desvio padrão pela equação 7.

**Figura 33 – Triângulo de Pascal**

1									
1	1								
1	2	1							
1	3	3	1						
1	4	6	4	1					
1	5	10	10	5	1				
1	6	15	20	15	6	1			
1	7	21	35	35	21	7	1		
1	8	28	56	70	56	28	8	1	
1	9	36	84	126	126	84	36	9	1

Fonte:disponível em: <[http://3.bp.blogspot.com/-mhKIPGnJScY/Uj9eMb\\_Uzjl/AAAAAAAAMFM/p0SGx4AOJ-E/s1600/1.png](http://3.bp.blogspot.com/-mhKIPGnJScY/Uj9eMb_Uzjl/AAAAAAAAMFM/p0SGx4AOJ-E/s1600/1.png)>.

$$\sigma = \frac{\sqrt{n - 1}}{2} \quad (7)$$

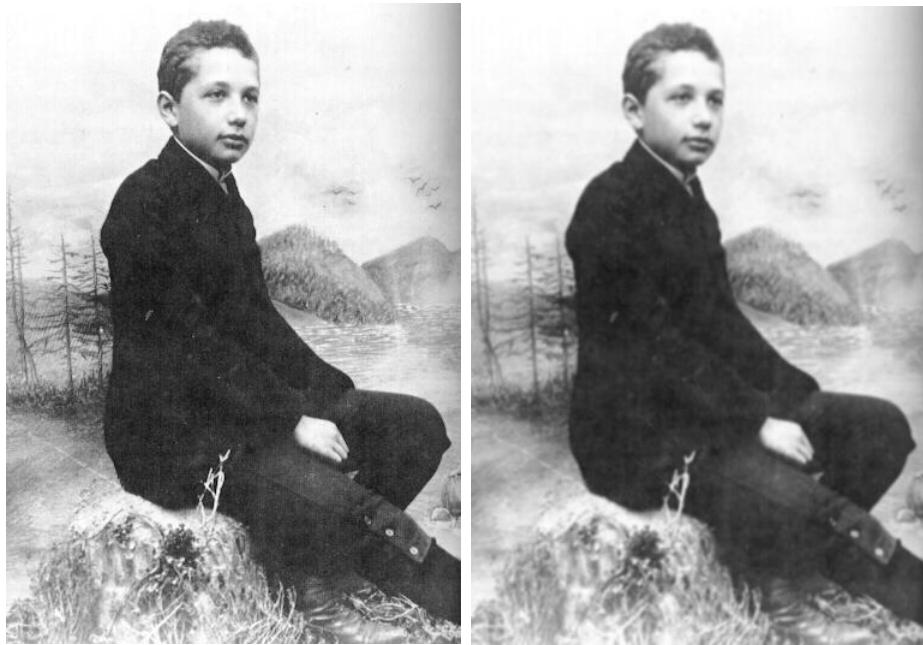
Uma máscara bidimensional para implementar o filtro Gaussiano com  $\sigma = 1.0$  pode ser obtida a partir de duas máscaras unidimensionais horizontal e vertical, conforme mostrado na figura 34 (PEDRINI; SCHWARTZ, 2008).

**Figura 34 – Resposta a duas máscaras bidimensionais**

$$\frac{1}{16} \begin{matrix} 1 \\ 4 \\ 6 \\ 4 \\ 1 \end{matrix} \times \frac{1}{16} \begin{matrix} 1 & 4 & 6 & 4 & 1 \end{matrix} = \frac{1}{256} \begin{matrix} 1 & 4 & 6 & 4 & 1 \\ 4 & 16 & 24 & 16 & 4 \\ 6 & 24 & 36 & 24 & 6 \\ 4 & 16 & 24 & 16 & 4 \\ 1 & 4 & 6 & 4 & 1 \end{matrix}$$

Fonte: PEDRINI; SCHWARTZ, 2008.

**Figura 35 – Aplicação da máscara bidimensional do filtro Gaussiano**



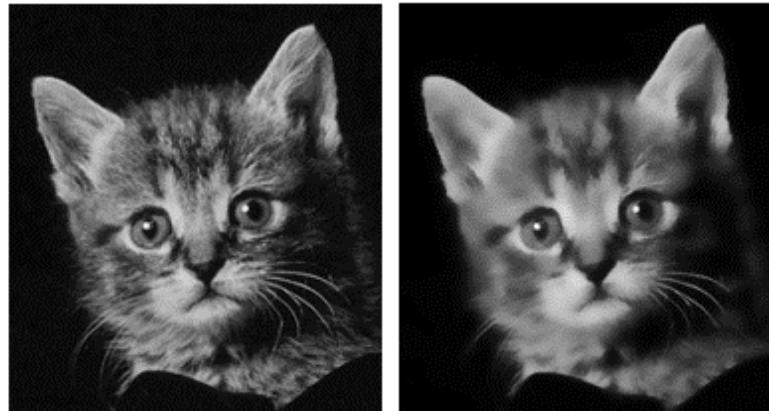
Fonte: imagem disponível em: <[http://scientists2.pbworks.com/f/1273525667/albert\\_einstein\\_as\\_a\\_child.jpg](http://scientists2.pbworks.com/f/1273525667/albert_einstein_as_a_child.jpg)>. Editado pelo autor.

O filtro bilateral foi apresentado a comunidade de visão computacional por Manduchi e Tomasi (1998), após um estudo baseado na função Gaussiana para aproveitar melhor os valores de uma imagem e preservar contornos importantes. Assim como o filtro Gaussiano o bilateral substitui cada pixel pela média ponderada da abertura da máscara e difere na dependência, onde o filtro bilateral fica dependente da variação abrupta de intensidades em relação ao pixel central e leva em consideração a distância geométrica em relação ao mesmo, a resposta desta operação segue definida nas equações a seguir onde fica determinada na primeira o peso dos coeficientes baseado na domínio da máscara  $d(x, y, k, l)$  e na abertura  $r(x, y, k, l)$  assim resultando com a média ponderada em  $g(x, y)$  (Szeliski, 2010).

$$w(x, y, k, l) = \exp\left(-\frac{(x - k)^2 + (y - l)^2}{2\sigma_d^2} - \frac{\|(x, y) - (k, l)\|^2}{2\sigma_r^2}\right) \quad (8)$$

$$g(x, y) = \exp\left(\frac{\sum_{k,l} f(k, l)w(x, y, k, l)}{\sum_{k,l} w(x, y, k, l)}\right) \quad (9)$$

**Figura 36 – Aplicação de filtro bilateral, imagem original e processada respectivamente**



Fonte: MANDUCHI; TOMASI, 1998.

### 2.3.3.2 Filtros passa-alta

Os filtros passa-alta atenuam frequências baixas do espectro de Fourier, as altas frequências representam na maioria das imagens bordas e linhas onde há uma transição abrupta de valores nos pixels. Por característica, uma máscara passa-alta básica consiste na formação por coeficientes positivos perto do centro e negativos quando afastados do mesmo (MARQUES FILHO; VIEIRA NETO, 1999).

Um dos filtros passa-alta mais utilizados e conhecidos é o Laplaciano, que consiste na técnica do emprego de derivadas de segunda ordem como demonstram as equações a seguir, uma para a direção  $x$  e outra para  $y$ . A partir desses cálculos pode-se obter máscaras como da figura 37 que geram realces nas bordas de forma satisfatória, porém pode aguçar ruídos provenientes da imagem de entrada (GONZALES; WOODS, 2010).

$$\nabla^2 = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \quad (10)$$

$$\frac{\partial^2 f}{\partial x^2} = f(x+1, y) + f(x-1, y) - 2f(x, y) \quad (11)$$

$$\frac{\partial^2 f}{\partial y^2} = f(x, y+1) + f(x, y-1) - 2f(x, y) \quad (12)$$

$$\nabla^2 = f(x+1, y) + f(x-1, y) + f(x, y+1) + f(x, y-1) - 4f(x, y) \quad (13)$$

**Figura 37 – Máscaras de filtro Laplaciano**

-1	-1	-1	0	-1	0	1	-2	1
-1	8	-1	-1	4	-1	-2	4	-2
-1	-1	-1	0	-1	0	1	-2	1

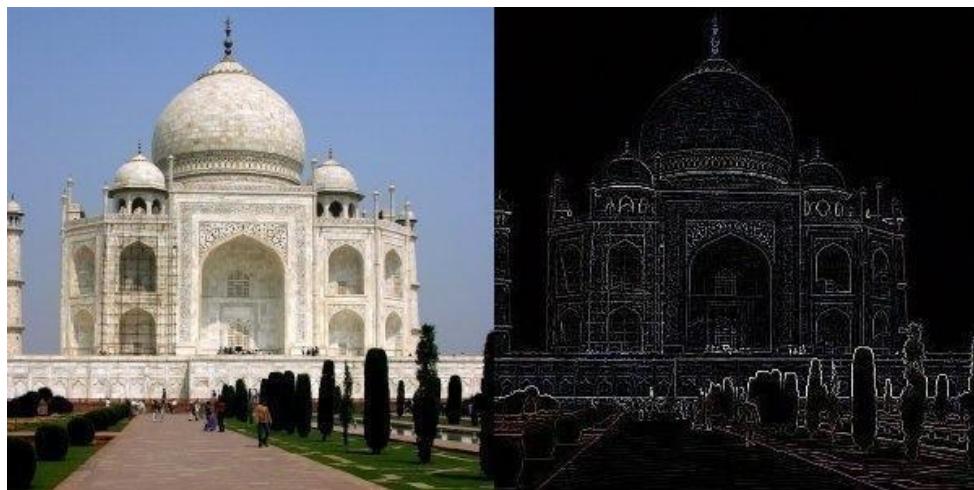
**Mask1**

**Mask2**

**Mask3**

Fonte: BAXES, 1994.

**Figura 38 – Aplicação do filtro Laplaciano**



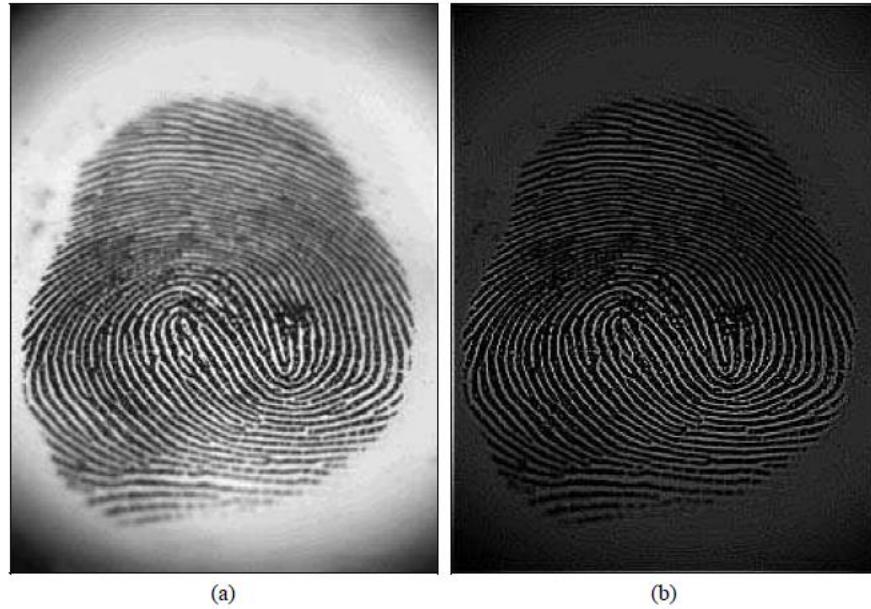
Fonte: disponível em:<[https://upload.wikimedia.org/wikipedia/commons/8/85/Filtro\\_bordes\\_original\\_laplace\\_sobel.jpg](https://upload.wikimedia.org/wikipedia/commons/8/85/Filtro_bordes_original_laplace_sobel.jpg)>.

High-boost ou “alto-reforço” é uma técnica de filtragem formulada em uma extensão da simples ideia de que a resposta a um filtro passa-alta nada mais é do que a subtração da imagem de entrada pela imagem resultante de um filtro passa-baixa, este procedimento também é conhecido como máscara de nitidez (*unsharp mask*). Para então se aplicar um filtro high-boost o primeiro passo é subtrair a imagem original  $f(x,y)$  da imagem suavizada  $s(x,y)$  gerando uma máscara chamada de  $G_{mask}(x,y)$ , após, como demonstrado na equação, a resposta  $G(x,y)$  será a soma da imagem original com a máscara gerada sendo esta multiplicada por um fator  $k$ . O termo  $k$  deve ser maior ou igual a 0, quando  $k < 1$  há uma atenuação da ação da máscara de nitidez, quando  $k = 1$  apenas será adicionada a imagem original a máscara de nitidez sem peso, já quando  $k > 1$  este é definido como sendo o filtro high-boost e irá influenciar na magnitude da máscara de nitidez. Portanto o filtro high-boost é dependente de  $k$  e do filtro passa-baixa aplicado (FILHO; NETO, 1999; GONZALES; WOODS, 2010).

$$G_{mask}(x, y) = f(x, y) - s(x, y) \quad (14)$$

$$G(x, y) = f(x, y) + (k * G_{mask}(x, y)) \quad (15)$$

**Figura 39 – Aplicação de filtro high-boost. a) Original. b) Processada.**



Fonte: MARQUES FILHO; VIEIRA NETO, 1999.

O filtro de Sobel é utilizado para gerar bordas provindas da transição entre pixels com valor mais próximo de 0 e valores próximos de 255, ou o inverso, para isso são aplicadas uma máscara horizontal e uma vertical como mostra a figura 40, onde as respostas dessas máscaras devem ser trabalhadas para se obter a magnitude  $G(x, y)$  e direção  $\theta$  seguindo as duas equações subsequentes. Sobel é menos suscetível a respostas com ruído do que o filtro Laplaciano em relação a imagens de entrada com este tipo de variação além de também gerar bordas mais destacadas (BAXES, 1994; BOYLE; HLAVAC; SONKA, 2008).

$$G(x, y) = \sqrt{x^2 + y^2} \quad (16)$$

$$\theta = \arctan\left(\frac{y}{x}\right) \quad (17)$$

**Figura 40 – Mascaras vertical e horizontal de Sobel**

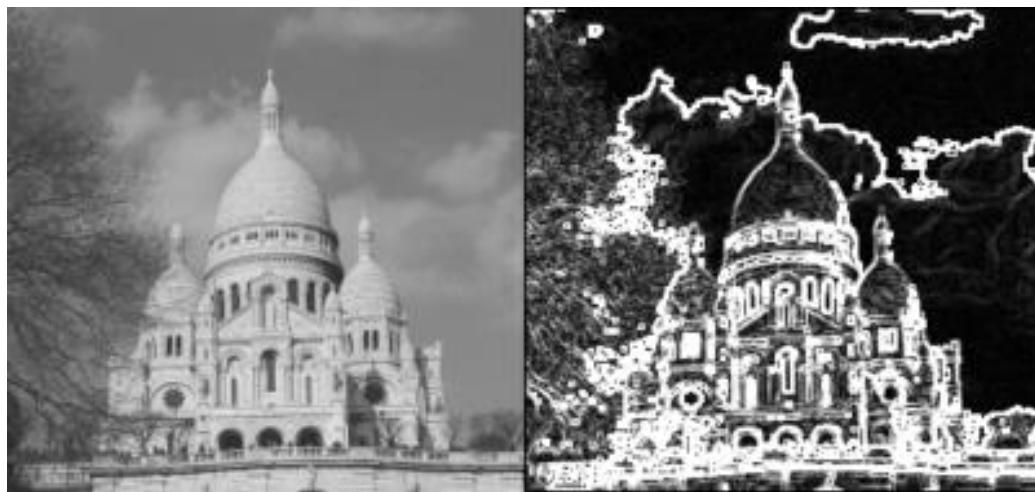
-1 0 1  
-2 0 2  
-1 0 1  
Máscara vertical

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

Máscara horizontal

Fonte: BAXES, 1994. Editado pelo autor.

**Figura 41 – Aplicação do filtro de Sobel**



Fonte: disponível em: <<https://upload.wikimedia.org/wikipedia/commons/thumb/9/95/Sobel-edges-sacrecoeur.png/400px-Sobel-edges-sacrecoeur.png>>.

Verificou-se que a média aplicada como filtro a uma imagem atenua ruídos e suaviza a aparência da mesma, sabendo que este processo pode ser vinculado a integrais e que os filtros passa-alta são o oposto de filtros passa-baixa pode-se dizer então que para altas frequências pode ser utilizada a operação de derivação. Uma das técnicas comumente utilizadas é a do gradiente, cuja definição matemática está declarada na equação seguinte como um vetor de  $f(x,y)$  , na sequência está a apresentação da magnitude resultante deste vetor (FILHO; NETO, 1999; GONZALEZ; WOODS, 2010).

$$\nabla f = \text{grad}(f) = \begin{bmatrix} g_x \\ g_y \end{bmatrix} = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} \quad (18)$$

$$M(x, y) = \text{mag} (\nabla f) = \sqrt{g_x^2 + g_y^2} \quad (19)$$

Para a imagem gradiente uma aproximação matemática pode ser expressa:

$$G[f(x, y)] \cong \{[f(x, y) - f(x + 1, y)]^2 + [f(x, y) - f(x, y + 1)]^2\}^{1/2} \quad (20)$$

ou

$$G[f(x, y)] \cong |f(x, y) - f(x + 1, y)| + |f(x, y) - f(x, y + 1)| \quad (21)$$

Os termos para se implementar a derivada da imagem são conhecidos por operadores de gradientes, assim como a máscara gerada para o filtro de Sobel existem também alguns outros operadores bastante utilizados, um deles é o operador de Roberts que diferente do Sobel se baseia em uma máscara 2x2 como mostra a figura 42, seguindo o cálculo abaixo, porém por ser implementado com uma matriz com poucos pixels para aproximar o gradiente o operador de Roberts é mais sensível a ruídos (MARQUES FILHO; VIEIRA NETO, 1999; GONZALES; WOODS, 2010; BOYLE; HLAVAC; SONKA, 2008).

$$G[f(x, y)] \cong \{[f(x, y) - f(x + 1, y + 1)]^2 + [f(x + 1, y) - f(x, y + 1)]^2\}^{1/2} \quad (22)$$

ou

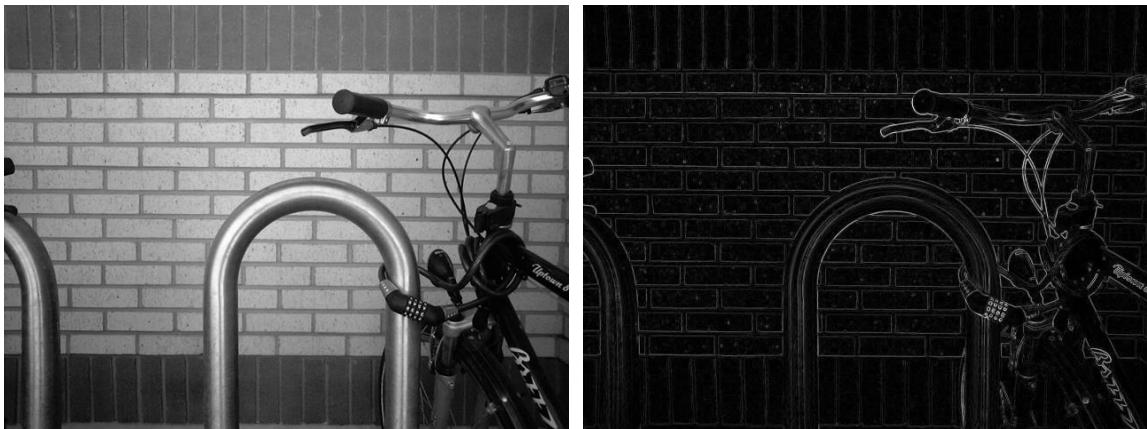
$$G[f(x, y)] \cong |f(x, y) - f(x + 1, y + 1)| + |f(x + 1, y) - f(x, y + 1)| \quad (23)$$

**Figura 42 – Operadores cruzados de Roberts 2x2**

$G_x = \begin{array}{ c c } \hline 1 & 0 \\ \hline 0 & -1 \\ \hline \end{array}$	$G_y = \begin{array}{ c c } \hline 0 & -1 \\ \hline 1 & 0 \\ \hline \end{array}$
--	--

Fonte: PEDRINI; SCHWARTZ, 2008. Editado pelo autor.

**Figura 43 – Aplicação do operador de Roberts 2x2**



Fonte: disponível em: <[https://upload.wikimedia.org/wikipedia/commons/6/62/Bikesgray\\_roberts.JPG](https://upload.wikimedia.org/wikipedia/commons/6/62/Bikesgray_roberts.JPG)>.

### **2.3.4 Segmentação e reconhecimento de objetos**

Segundo Gonzales e Woods (2010) o processo de segmentação é uma das etapas mais difíceis do processamento de imagens, utilizada quando é necessário extrair atributos de determinada aplicação, ou seja, quando a entrada são imagens e a saída são informações sobre os objetos contidos nestas imagens. Para se obter resultados com precisão a imagem deve estar bem tratada após passar pelos filtros que eliminam os ruídos existentes, que poderiam gerar falsos positivos ou variações nas respostas. Nesta seção serão apresentadas algumas técnicas para fins de segmentação e reconhecimento de objetos, como por exemplo a aplicação de limiar, detecção de bordas e separação dos objetos por região.

#### **2.3.4.1 Limiarização**

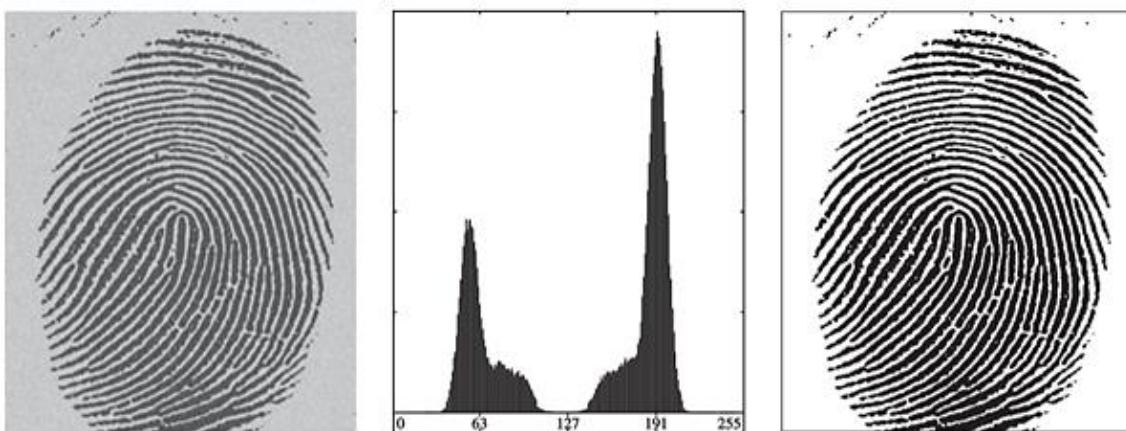
A limiarização de uma imagem é feita pela separação de grupos de pixels, cujos níveis de cinza estão acima ou abaixo de um valor pré-determinado ou calculado, estes grupos então recebem um novo valor de brilho podendo ser considerados como 0 e 1 (0 e 255) correspondendo ao fundo e ao objeto, esse método também é conhecido como binarização por formar uma imagem binária (PEDRINI; SCHWARTZ, 2008).

O método de limiarização com valor inicial fixo é denominado limiar global onde satisfaz o processamento quando a imagem possui intensidade dos pixels entre objeto

e fundo diferentes. Para realizar uma aproximação automática do valor de limiar  $T$  é possível utilizar a fórmula da equação 24, que baseada no histograma dos níveis de cinza da imagem e em um valor inicial de  $T_i$  separa inicialmente em dois grupos de intensidades, sendo  $G_1 > T_i$  e  $G_2 \leq T_i$ , após isso calcula-se os valores médios  $m_1$  e  $m_2$  para  $G_1$  e  $G_2$  respectivamente e então calcula-se o valor de  $T_f$  pela equação. É necessário repetir o processo utilizando  $T_f$  como  $T_i$  a cada ciclo de cálculo até que a diferença entre  $T_f$  e  $T_i$  seja igual ou menor que o valor estipulado de  $\Delta T$  (GONZALES; WOODS, 2010).

$$T_f = \frac{1}{2}(m_1 + m_2) \quad (24)$$

**Figura 44 – Exemplo de limiar global automático com resultado em 125 após 3 iterações**



Fonte: GONZALES; WOODS, 2010.

Outro método de limiarização onde é possível atingir o valor ótimo de um limiar global, ou seja, o melhor valor de limiar para separação dos grupos de pixels, é conhecido como método de Otsu (1979), usual em imagens com maior variância no histograma de intensidades, mas para isso se assume que o histograma deve ter dois picos (bimodal). A técnica se baseia em calcular a variância  $\sigma^2_T$  e a média  $m_T$  globais da imagem, maximizando a razão  $\eta(T)$  da variância entre as classes  $\sigma^2_B$  e a variância total, utilizando todos valores de  $T$  com possibilidade de uso para a imagem (GONZALES; WOODS, 2010; PEDRINI; SCHWARTZ, 2008).

$$\eta(T) = \frac{\sigma_B^2}{\sigma_T^2} \quad (25)$$

$$\sigma_T^2 = \sum_{i=0}^{L-1} (i - \mu_T)^2 p_i \quad (26)$$

$$\mu_T = \sum_{i=0}^{L-1} i p_i \quad (27)$$

$$\sigma_B^2 = \omega_1 - \omega_2 (\mu_1 \mu_2)^2 \quad (28)$$

$$\omega_1 = \sum_{i=0}^T p_i \quad (29)$$

$$\omega_2 = 1 - \omega_1 \quad (30)$$

$$\mu_1 = \frac{\mu_S}{\omega_1} \quad (31)$$

$$\mu_2 = \frac{\mu_T - \mu_S}{\omega_2} \quad (32)$$

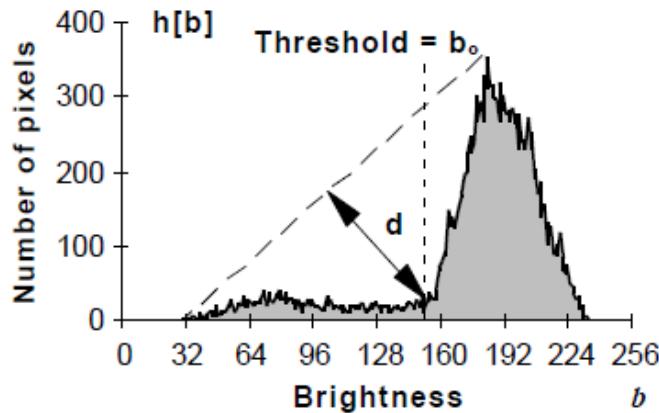
$$\mu_S = \sum_{i=0}^T i p_i \quad (33)$$

$$\sum_{i=0}^{L-1} p_i = 1, \quad p_i = \frac{n_i}{n} \quad (34)$$

Conforme as equações acima relacionadas  $n_i$  é a quantidade de pixels com intensidade  $i$ ,  $n$  é o valor total de pixels contidos na imagem,  $p_i$  está relacionado a probabilidade desta intensidade  $i$  aparecer na imagem, já  $L$  é o número de níveis de cinza encontrados na imagem (PEDRINI; SCHWARTZ, 2008).

Um método geométrico utilizando o histograma para encontrar o valor de limiar global é utilizar o algoritmo triangular, o valor de limiar  $b_o$  é encontrado a partir da determinação de uma linha entre o valor mínimo de intensidade do histograma e o pico máximo, após isso se calcula a maior distância perpendicular  $d$  entre esta linha e as intensidades de brilho do histograma, do valor inicial ao máximo traçado como mostra a figura 45, o nível de cinza correspondente a esta distância será o limiar  $b_o$  (GERBRANDS; YOUNG; VAN VLIET, 1998).

**Figura 45 – Limiarização pelo método triangular**

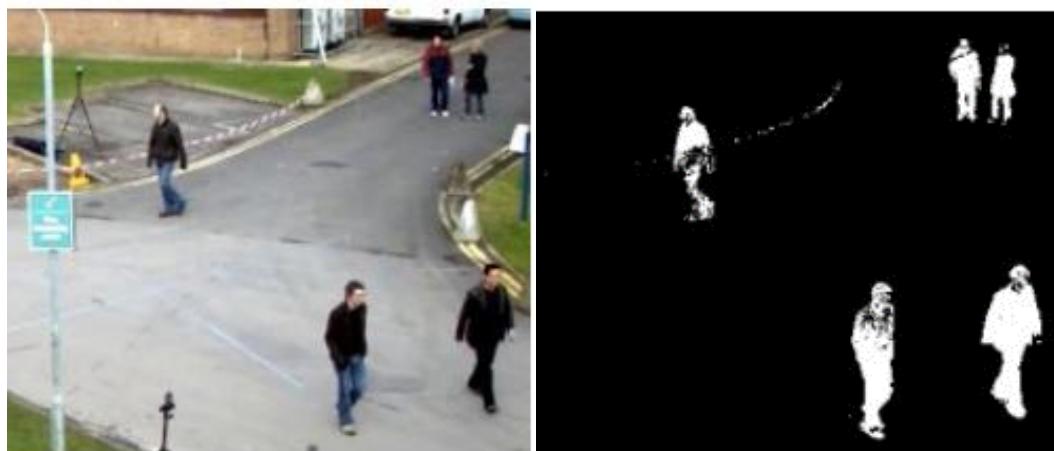


Fonte: GERBRANDS; YOUNG; VAN VLIET, 1998.

### 2.3.4.2 Subtração de fundo

Em diversas aplicações onde se é necessário detectar a presença de objetos em movimento a subtração de fundo, de forma simples, pode fornecer uma resposta correta e eficiente em muitos casos, como em vias para detecção de automóveis, em esteiras de processos industriais e também em áreas de circulação de pessoas (FORSYTH; PONCE, 2003).

**Figura 46 – Segmentação por subtração de fundo**



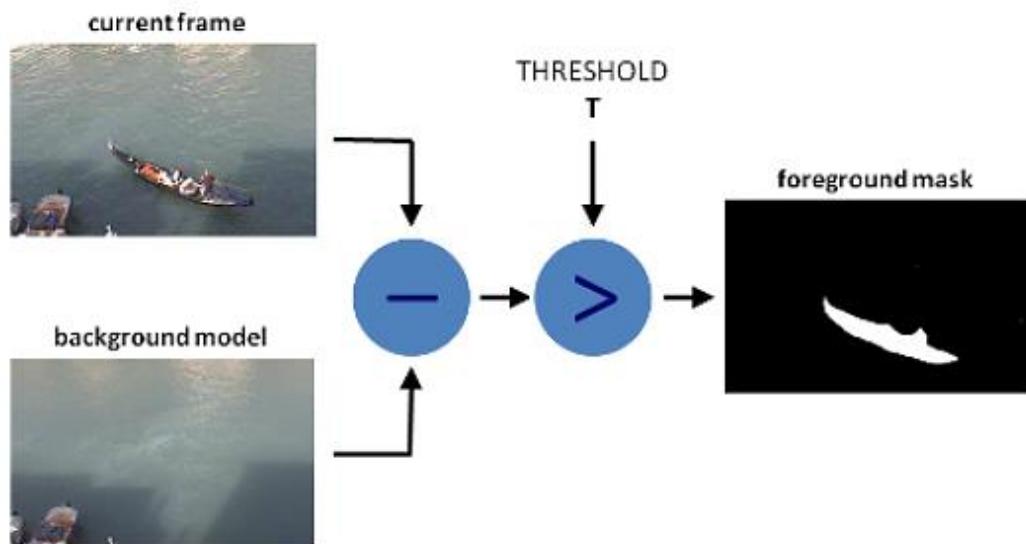
Fonte: disponível em: <[http://docs.opencv.org/3.3.0/db/d5c/tutorial\\_py\\_bg\\_subtraction.html](http://docs.opencv.org/3.3.0/db/d5c/tutorial_py_bg_subtraction.html)>.

Editado pelo autor.

Para efetuar esta operação a primeira ação a se tomar é ter uma imagem do fundo como base de comparação, pode ser capturada da área a ser analisada quando não há presença dos objetos a serem detectados. A subtração deve acontecer ao

longo do tempo, onde a resposta será a imagem atual subtraída do fundo, tudo o que variar em relação a imagem base de fundo irá ser o resultante. Porém como em muitos ambientes o local se altera com o tempo com a interação de outros fatores, como clima, objetos que entram na cena, mas não são de interesse, variações de luminosidade, isso causa ruídos na resposta que podem atrapalhar na hora da interpretação. Para resolver isso deve ser montada alguma lógica que estime estas variações e atualize o plano de fundo ou se forem ruídos pequenos na imagem podem ser removidos através da aplicação de filtros passa-baixa ou limiar (DAVIES, 2012; FORSYTH; PONCE, 2003).

**Figura 47 – Subtração de fundo com uso de limiar**



Fonte: disponível em: <[http://docs.opencv.org/3.2.0/d1/dc5/tutorial\\_background\\_subtraction.html](http://docs.opencv.org/3.2.0/d1/dc5/tutorial_background_subtraction.html)>.

### 2.3.4.3 Reconhecimento por borda

A segmentação de bordas em imagens pode ser feita aplicando os filtros passa-alta como anteriormente comentado, utilizando gradientes e seus operadores, porém neste processo não se leva em consideração as características dessas bordas (GONZALES; WOODS, 2010).

Marr e Hildreth (1980) foram precursores na busca por um método de identificação de bordas que fosse melhor elaborado, defendiam a ideia de que independente da mudança de escala de uma imagem deve ser possível computar os valores de intensidade através da mudança de tamanho dos operadores e que ao

haver uma mudança abrupta de intensidade do nível de cinza um pico será gerado e com a aplicação de uma derivada de segunda ordem acontecerá uma passagem por zero. Para realizar esse processo o método é basicamente explicado pelo uso de um filtro passa-baixa gaussiano que é uma derivada de segunda ordem e após se aplica o filtro passa-alta Laplaciano, assim é formada a expressão resultante (eq. 35) da função LoG (*Laplacian of Gaussian*) (DAVIES, 2012; GONZALES; WOODS, 2010).

$$\nabla^2 G(x, y) = \left[ \frac{x^2 + y^2 - \sigma^2}{\sigma^4} \right] e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (35)$$

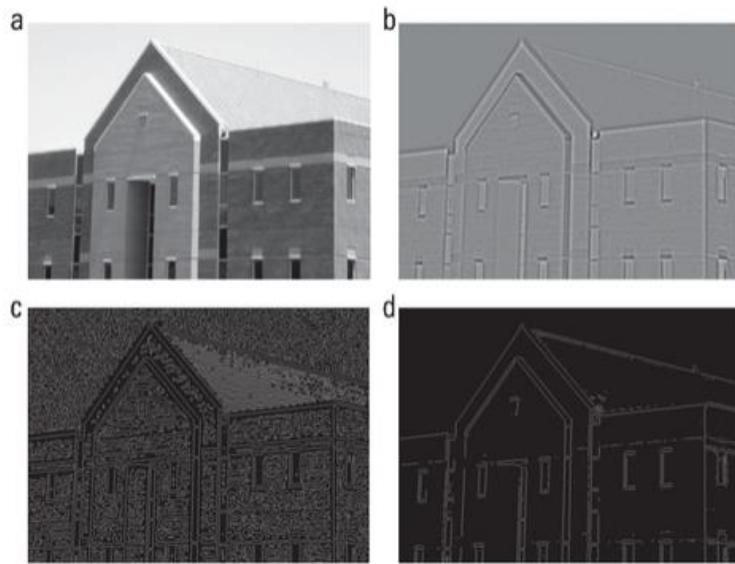
A matriz de operação representada na figura 48 é uma das aproximações possíveis para o método Marr-Hildreth, cujo o somatório deve ser zero, em geral segue o mesmo esquema de formação para outros operadores que se desejar utilizar. Uma exemplificação desse procedimento segue em aplicar-se um filtro gaussiano de tamanho  $n \times n$  e após um filtro Laplaciano de ordem  $3 \times 3$ , feito isso é necessário encontrar os pontos na imagem onde houve o cruzamento por zero (GONZALES; WOODS, 2010).

**Figura 48 – Máscara 5X5 de uma aproximação pelo método Marr-Hildreth**

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Fonte: GONZALES; WOODS, 2010. Editado pelo autor.

**Figura 49 – Segmentação pelo método Marr-Hildreth utilizando  $\sigma=4$  e  $n=25$**



Fonte: GONZALES; WOODS, 2010.

De uma maneira um pouco mais complexa, porém com resultados mais satisfatórios, principalmente em situações onde a imagem possui ruído, o detector de borda Canny (1986) traz uma concepção confiável para seu algoritmo, baseada na baixa taxa de erro, pontos de borda detectados bem localizados e sem duplicação de resultados fornecendo apenas um ponto para cada pixel de borda identificado (BOYLE; HLAVAC; SONKA, 2008; DAVIES, 2012; GONZALES; WOODS, 2010).

O procedimento adotado é iniciar o algoritmo pela convolução da função gaussiana (eq. 36) e a imagem  $f(x,y)$ , após calcula se a magnitude (eq. 37) e direção (eq. 38) da borda cujos arranjos tem mesma dimensão que a imagem, como estes arranjos gerados podem apresentar bordas com variações de pontos não legítimos e cristas deve se eliminar estes problemas através da supressão dos não máximos e a limiarização por histerese, ou seja, limitar a orientação dos pontos de borda por região e após gerar duas imagens resultantes de dois limiares diferentes como mostra a regra 39 e 40 e compará-las para determinar sua ligação (BOYLE; HLAVAC; SONKA, 2008; DAVIES, 2012; GONZALES; WOODS, 2010).

$$G(x,y) = e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (36)$$

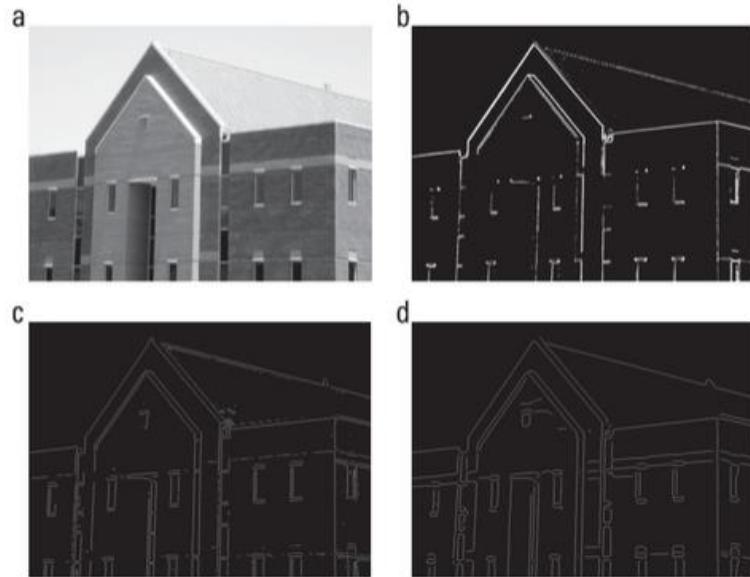
$$M(x,y) = \sqrt{g_x^2 + g_y^2} \quad (37)$$

$$\sigma(x, y) = tg^{-1} \left[ \frac{g_y}{g_x} \right] \quad (38)$$

$$g_{NH}(x, y) = g_N(x, y) \geq T_H \quad (39)$$

$$g_{NL}(x, y) = g_N(x, y) \geq T_L \quad (40)$$

**Figura 50 – Comparação entre detecção por borda de Marr-Hildreth (c) e Canny (d)**



Fonte: GONZALES; WOODS, 2010.

#### 2.3.4.4 Reconhecimento de cantos

A detecção de cantos e junções em uma imagem pode fornecer informações de grande importância para detectar algum objeto ou movimento, para isso existem várias técnicas desenvolvidas ao longo dos anos de amadurecimento do processamento digital de imagens. A maioria das ideias implementadas se baseia na utilização de derivadas de segunda ordem, através do determinante da matriz *Hessian* (eq. 41) utilizado para encontrar a curvatura Gaussiana com  $k_1$  e  $k_2$  (eq. 42) que determina o pixel como sendo hiperbólico caso  $k_1k_2$  seja menor que 0, parabólico caso seja igual a 0 e elíptico se for maior que 0, sendo  $f_x$  e  $f_y$  as derivadas de primeira ordem (BOYLE; HLAVAC; SONKA, 2008; DAVIES, 2012; PEDRINI; SCHWARTZ, 2008).

$$H = \begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{bmatrix} = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} \quad (41)$$

$$k_1 k_2 = \frac{\begin{bmatrix} f_{xx} & f_{xy} \\ f_{yx} & f_{yy} \end{bmatrix}}{(1 + f_x^2 + f_y^2)^2} = \frac{f_{xx} f_{yy} - f_{xy}^2}{(1 + f_x^2 + f_y^2)^2} \quad (42)$$

Uma junção pode ser definida como um ponto de intersecção entre dois ou mais segmentos da borda de objetos. As quatro formas mais comuns de junção em imagens digitais são mostradas na figura 51, representadas pelas letras L, T, X e Y, respectivamente (PEDRINI; SCHWARTZ, 2008).

**Figura 51 – Formas comuns de cantos em imagens digitais**



Fonte: PEDRINI; SCHWARTZ, 2008. Editado pelo autor.

O detector de Harris proposto em 1987 foi bastante popular por exigir pouco processamento no algoritmo e não ser sensível as pequenas variações de iluminação. Este detector é baseado no cálculo de derivadas de primeira ordem através de um filtro de suavização Gaussiano representado por  $\langle \rangle$  e calculado para as direções  $x$  e  $y$ , conforme a definição na equação 43. Com a matriz  $G$  é possível então encontrar o determinante e o traço<sup>16</sup>, juntamente com o valor de  $k$  pré estimado por Harris entre 0,04 a 0,15, assim se conclui o operador de Harris  $R(x,y)$  (BOYLE; HLAVAC; SONKA, 2008; DAVIES , 2012; PEDRINI; SCHWARTZ, 2008; PRINCE, 2012).

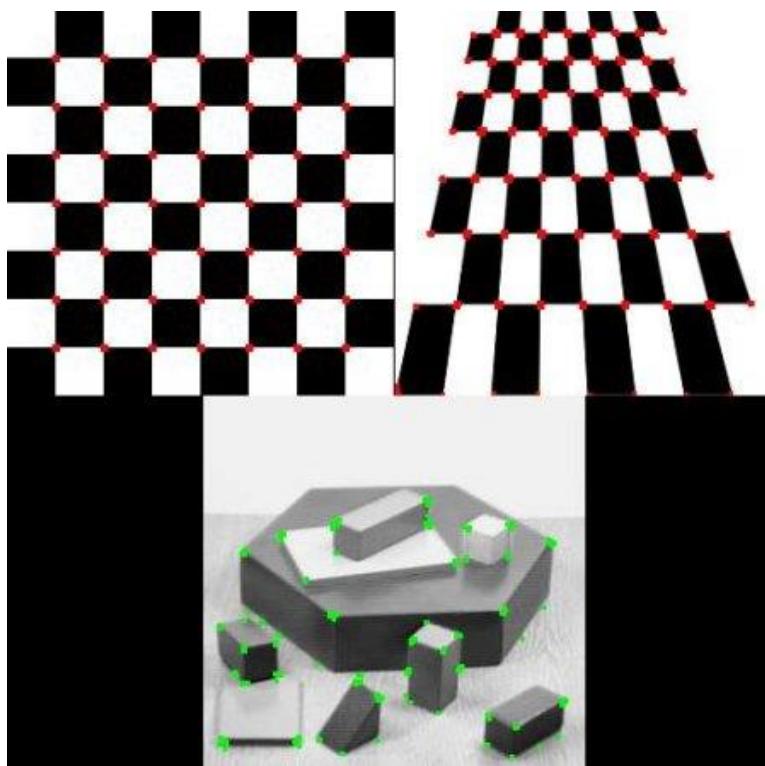
$$G = \begin{vmatrix} \langle f_x^2 \rangle & \langle f_x f_y \rangle \\ \langle f_x f_y \rangle & \langle f_y^2 \rangle \end{vmatrix} \quad (43)$$

$$R(x,y) = \det(G) - k \text{ traço}^2(G) \quad (44)$$

---

<sup>16</sup> Somatório dos valores da diagonal principal da matriz.

**Figura 52 – Resultados da aplicação do detector de Harris**



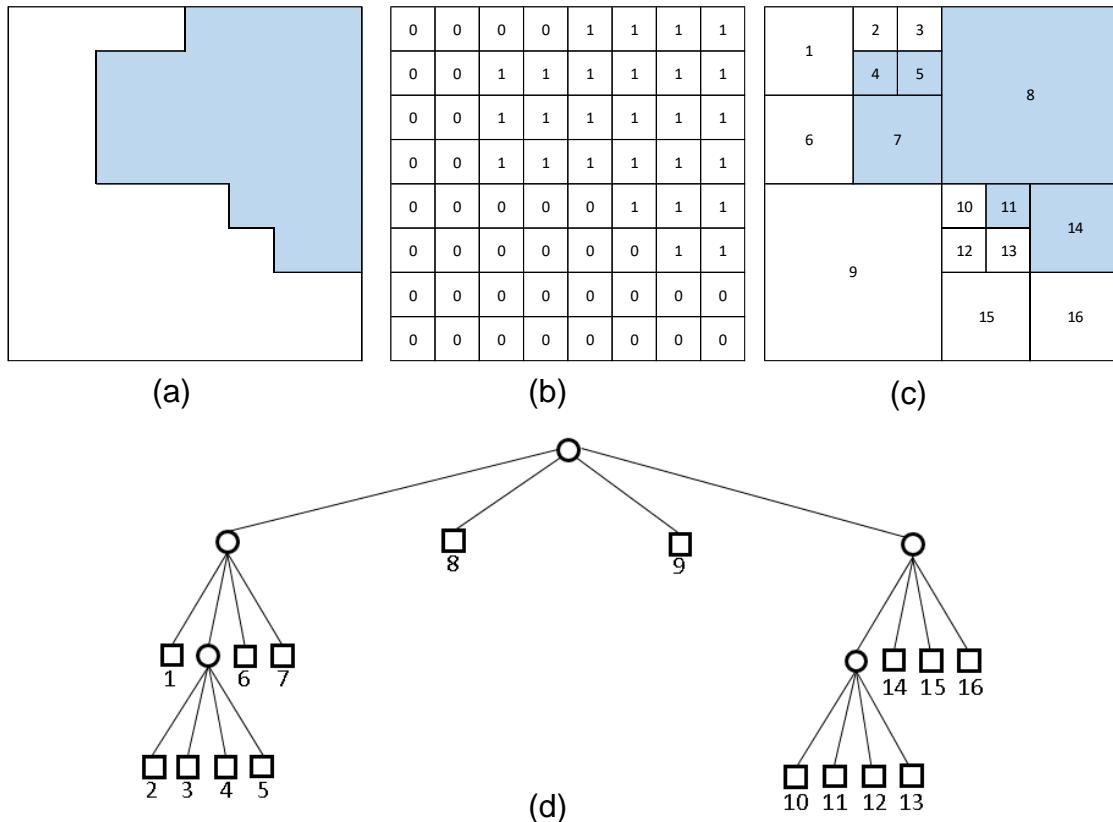
Fonte: disponível em: <[http://docs.opencv.org/trunk/harris\\_result.jpg](http://docs.opencv.org/trunk/harris_result.jpg)>.

#### 2.3.4.5 Reconhecimento por região

A detecção de objetos por região se torna viável quando se tem conhecimento parcial prévio da localização e tipos de objetos a serem analisados na imagem, assim é possível dividir a imagem em regiões de interesse facilitando a análise das bordas e a conexão entre os pixels para chegar a uma aproximação das fronteiras dos objetos (GONZALES; WOODS, 2010).

O processo de divisão de regiões pode ser iniciado através da própria imagem como sendo a região de partida e subdividir até que todas as regiões possuam satisfaçam os critérios de similaridade. Baseado em sistemas hierárquicos utilizados em algoritmos computacionais, também conhecido como árvore de quadrantes ou “quadtree”, caso a primeira região definida na imagem atenda aos critérios de conectividade e similaridade dos pixels então esta área já satisfaz a segmentação por divisão de regiões, caso não atenda aos requisitos então a imagem será dívida em quadrantes e ainda se estes quadrantes não satisfizerem pela similaridade entre os pixels da região então serão novamente divididos em subquadrantes como demonstra a figura 53 (JÄHNE, 2002; PEDRINI; SCHWARTZ, 2008).

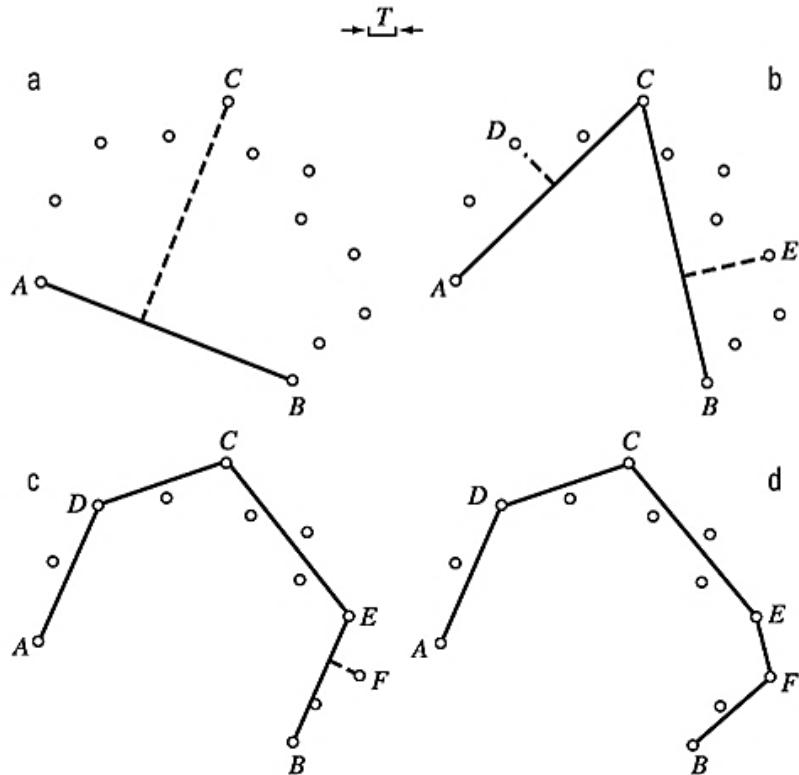
**Figura 53 – Divisão de regiões por decomposição regular.** a) Imagem original. b) Imagem em matriz binária. c) Imagem particionada em quadrantes. d) Quadtree correspondente.



Fonte: PEDRINI; SCHWARTZ, 2008. Editado pelo autor.

Segundo Gonzales e Woods (2010) pode-se também utilizar outra técnica baseada no reconhecimento de formas poligonais em função dos pontos de uma curva 2D, como mostra a figura 54, os vértices do polígono são definidos como sendo os pontos *A* e *B*, assim a reta que faz ligação a estes pixels serve para o cálculo das distâncias perpendiculares até os demais pontos da curva, a reta perpendicular que possuir a maior distância e que este valor ultrapasse a referência pré estipulada *T* será definida como novo vértice e então se refaz o cálculo agora ligando os pontos *A* até *C* e *B* até *C*, seguindo esta mesma lógica para novos vértices até que todos os vértices estejam definidos e nenhuma distância perpendicular ultrapasse a referência, ao final deste processo o polígono estará formado e poderá servir de aproximação na segmentação dos objetos por região.

**Figura 54 – Reconhecimento de objetos por aproximação poligonal**



Fonte: GONZALES; WOODS, 2010.

#### 2.3.4.6 Detecção pela transformada de Hough

A detecção por região é usual quando se tem noção dos objetos contidos na imagem, com critérios pré-determinados para certa aplicação, porém quando a imagem oferece variações em relação aos objetos sem previamente ter informações que auxiliem na segmentação das bordas para determinar onde estão localizadas as fronteiras, Hough (1962) propõe uma forma diferente de analisar as ligações entre os pixels de borda (GONZALES; WOODS, 2010).

A partir da equação de uma reta no modelo inclinação-intersecção (eq. 45) onde  $m$  é a declividade da reta e  $b$  é o ponto onde a reta passa pelo eixo  $y$ , é possível realizar a detecção de retas onde os pixels são transformados em retas no plano  $mb$ , também conhecido por espaço de parâmetros, onde também os pontos de intersecção das inúmeras retas são identificados e reunidos para formar segmentos de uma reta. Com esta mesma percepção a transformada de Hough permite detectar circunferências com a equação 46 referente a função em coordenadas no plano

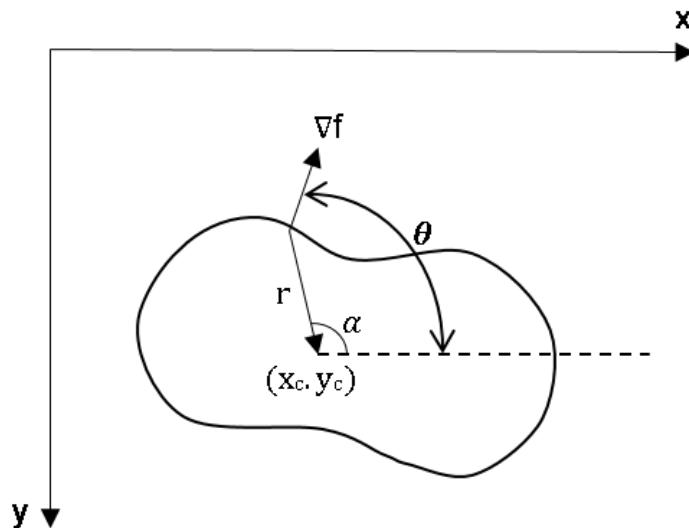
cartesiano, utilizando as informações dos pontos centrais ( $a, b$ ) e raio ( $r$ ) (PEDRINI; SCHWARTZ, 2008).

$$y = mx + b \quad (45)$$

$$(x - a)^2 + (y - b)^2 = r^2 \quad (46)$$

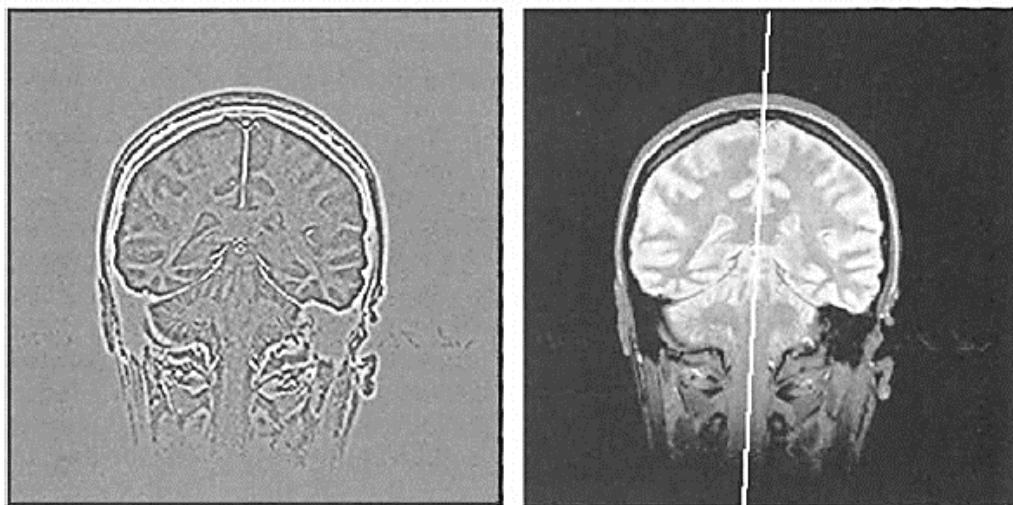
Já para formas que não exercem algum padrão de rastreamento de pontos existe a generalização da transformada de Hough expressa por Ballard (1981), cujo processo consiste em definir inicialmente o centroide do objeto, após, traçar um segmento de reta em direção a algum ponto da borda a partir deste ponto central como demonstra a figura 55. Assim será possível construir a tabela de referências das direções da borda para efetuar uma busca nessa tabela pelo ângulo que se aproxima do gradiente no determinado ponto  $(x, y)$ . Um algoritmo explicativo deste procedimento pode ser consultado no ANEXO A (PEDRINI; SCHWARTZ, 2008).

**Figura 55 – Ponto de referência em um objeto**



Fonte: PEDRINI; SCHWARTZ, 2008. Editado pelo autor.

**Figura 56 – Aplicação da transformada de Hough para detecção de reta na separação de hemisférios esquerdo e direto do cérebro.**



Fonte: BOYLE; HLAVAC; SONKA, 2008.

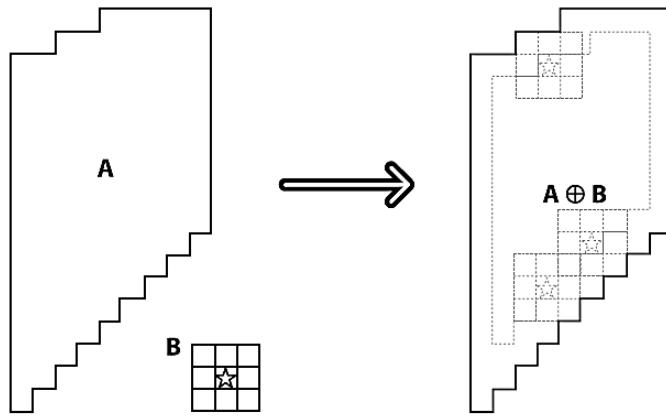
### 2.3.5 Processamento morfológico

A morfologia matemática, definida pelo estudo de estruturas geométrica existentes em uma imagem, foi iniciada por volta dos anos 1960 por Georges Matheron (1967) e Jean Serra (1982), onde as técnicas são baseadas em álgebra de operadores não lineares, que muitas vezes substitui o sistema algébrico linear das convoluções. O processamento com morfologia matemática é muito usual em diversas aplicações como no pré-processamento, segmentação, realce e detecção de bordas, afinamento, entre outros (BOYLE; HLAVAC; SONKA, 2008; GONZALES; WOODS, 2010; MARQUES FILHO; VIEIRA NETO, 1999).

Em imagens binárias os conceitos básicos das técnicas para o processo morfológico consistem em dilatação e erosão, onde a partir desses métodos é possível elaborar meios de se trabalhar uma imagem e aplicar demais operações (BOYLE; HLAVAC; SONKA, 2008; JÄHNE; HAUBECKER, 2000).

A dilatação é formada através da operação de uma matriz ou conjunto de valores em relação a uma área de interesse em uma imagem, a dilatação de uma área A pelo conjunto B é o deslocamento de B pelo entorno de A, sendo assim todo ponto que sobressair como mostra a figura 57 será preenchido formando a dilatação da imagem (GONZALES; WOODS, 2010; MARQUES FILHO; VIEIRA NETO, 1999).

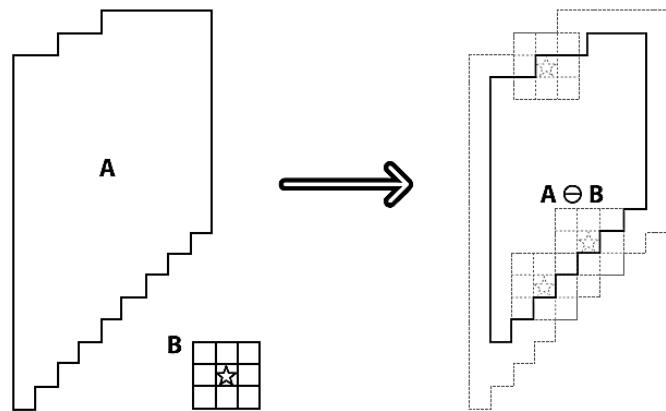
**Figura 57 – Dilatação de uma região de interesse**



Fonte: BRADSKI; KAEHLER, 2008.

A erosão é o inverso da dilatação, onde esse processo também acontece pela aplicação de um conjunto de valores em que a erosão de A por B são todos pontos centrais do conjunto B, que deslocado deve estar contido em A como ilustra a figura 58 (GONZALES; WOODS, 2010; MARQUES FILHO; VIEIRA NETO, 1999).

**Figura 58 – Erosão de uma região de interesse**



Fonte: BRADSKI; KAEHLER, 2008.

Sabendo-se estes dois conceitos básicos é possível aplicar vários processos que auxiliam no tratamento de imagens como, por exemplo, suavizar bordas, remover ruídos e preencher buracos, estes são resultados da aplicação das operações de abertura (eq.47) e fechamento (eq.48) (GONZALES; WOODS, 2010; SZELISKI, 2010).

$$\text{abertura}(a, b) = \text{dilatação}(\text{erosão}(a, b), b) \quad (47)$$

$$\text{fechamento}(a, b) = \text{erosão}(\text{dilatação}(a, b), b) \quad (48)$$

**Figura 59 – Exemplo de transformação morfológica por abertura e fechamento respectivamente.**



Fonte: disponível em: <[http://docs.opencv.org/trunk/d9/d61/tutorial\\_py\\_morphological\\_ops.html](http://docs.opencv.org/trunk/d9/d61/tutorial_py_morphological_ops.html)>.

## 2.4 Linguagens de programação

Na linguagem de máquina ou código objeto o tipo de informação que o chip de um microcontrolador ou microprocessador comprehende é disposta na forma binária, porém criar um código apenas baseado neste tipo de informação seria muito trabalhoso exigindo do programador conhecimento sólido sobre o funcionamento do dispositivo e detalhamento a cada função a ser executada. Para facilitar tanto o trabalho de programar quanto o entendimento do funcionamento do código através de seus comandos foram criadas linguagens de programação, que se aproximam da linguagem humana, divididas em níveis para sua complexidade e proximidade entre linguagem de máquina e humana. Mas para o chip interpretar os comandos gerados é necessário que o código escrito em alguma linguagem de programação seja convertido para linguagem de máquina, assim cada instrução será executada conforme a forma binária pré-instalada pelo fabricante (AMARAL; DIAGO, 2011).

### 2.4.1 Assembly

Uma das primeiras linguagens de programação a surgir foi o assembly, utilizado na programação de DSP's, microcontroladores e algumas CPU's, a linguagem de programação assembly é usual em aplicações onde os dispositivos apresentam certas

limitações de hardware e se quer obter o melhor desempenho dos mesmos, mas para isso é exigido do programador um conhecimento mais aprofundados do funcionamento elétrico e computacional do dispositivo a ser utilizado, pois a programação de baixo nível é baseada no envio de instruções em bytes para que o dispositivo possa executar o comando diretamente sem necessitar interpretação e conversão anterior (GIMENEZ, 2002).

Conforme cita Blum (2005), chips de computadores e microcontroladores rodam os códigos pré-instalados pelos fabricantes na memória para interpretar instruções geradas na camada mais acima do sistema entre software e hardware, portanto muitas das linguagens de programação de médio e alto nível necessitam que após o código estar pronto, seja feita a conversão para linguagem de máquina, ou seja, em código binário, também representado em hexadecimal. Assim o assembly também necessita conversão para linguagem de máquinas, um dos compiladores mais conhecidos é o assembler, comumente confundido com a linguagem de programação pela nomenclatura similar.

**Figura 60 – Parte de um código em assembly para microcontrolador Motorola MC6800**

```

MONITOR FOR 6802 1.4          9-14-80  TSC ASSEMBLER PAGE  2

C000          ORG    ROM+$0000 BEGIN MONITOR
C000 SE 00 70  START LDS #STACK

*****
* FUNCTION: INITA - Initialize ACIA
* INPUT: none
* OUTPUT: none
* CALLS: none
* DESTROYS: acc A

0013      RESETA EQU    %00010011
0011      CTLREG EQU    %00010001

C003 86 13  INITA   LDA A  #RESETA  RESET ACIA
C005 B7 80 04          STA A  ACIA
C008 86 11          LDA A  #CTLREG  SET 8 BITS AND 2 STOP
C00A B7 80 04          STA A  ACIA

C00D 7E C0 F1      JMP    SIGNON GO TO START OF MONITOR

```

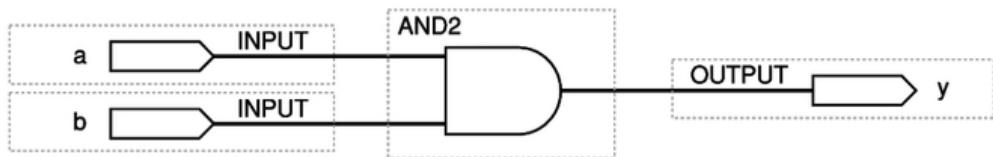
Fonte:disponível em: <[https://upload.wikimedia.org/wikipedia/commons/thumb/f/f3/Motorola\\_6800\\_Assembly\\_Language.png/800px-Motorola\\_6800\\_Assembly\\_Language.png](https://upload.wikimedia.org/wikipedia/commons/thumb/f/f3/Motorola_6800_Assembly_Language.png/800px-Motorola_6800_Assembly_Language.png)>.

## 2.4.2 VHDL

Assim como assembly esta é uma linguagem de baixo nível antiga, desenvolvida nos anos 80 pelo Departamento de Defesa dos Estados Unidos com o intuito de

projetar circuitos integrados de velocidade muito alta (VHSIC), entre estes estão os ASICs e FPGAs. A linguagem foi padronizada pela IEEE sendo utilizada até hoje por engenheiros e programadores da área. Seu método é descrito na forma textual de portas lógicas, formando o sistema digital, este irá transferir quais serão as ligações necessárias para o funcionamento, por isto este tipo de sistema é complexo, porém vantajoso em relação ao desempenho. A Altera Corporation foi uma das empresas pioneiras na fabricação destes circuitos reprogramáveis utilizando também uma linguagem de descrição de hardware própria, a AHDL (Altera hardware description language), a Altera permanece como uma das favoritas no mercado de dispositivos FPGAs, assim como a Xilinx (MOSS; TOCCI; WIDMER, 2011).

**Figura 61 – Descrição em diagrama esquemático**



Fonte: MOSS; TOCCI; WIDMER, 2011.

**Figura 56 – Alguns elementos da programação VHDL**

```

ENTITY and_gate IS
PORT ( a, b :IN BIT;
       y      :OUT BIT);
END and_gate;
ARCHITECTURE ckt OF and_gate IS
BEGIN
       y <= a AND b;
END ckt;

```

Fonte: MOSS; TOCCI; WIDMER, 2011.

### 2.4.3 C/C++

A linguagem C é derivada de outras duas linguagens que foram criadas para desenvolvimento de sistemas operacionais e compiladores, a BCPL criada por Martin Richards em 1967 e a B baseada nessa mesma linguagem, porém com modificações para desenvolver o sistema operacional UNIX na Bell Laboratories em 1970. Mas foi em 1972 que Dennis Ritchie, também na Bell Laboratories, desenvolveu a linguagem

de programação C deduzida da B e aplicada a um computador DEC-PDP-11. A linguagem C está presente em praticamente todos computadores no sistema operacional e é usual por estar em um nível intermediário de programação, pois é de fácil entendimento, útil para acesso de registradores e eficaz para a maioria das aplicações envolvendo software e máquina. A linguagem C++ é uma extensão de C com mesma sintaxe, mas com mais instruções voltadas para programação orientada a objetos (DEITEL, H.; DEITEL, P., 2011; ASCENCIO; CAMPOS, 2012).

**Figura 62 – Computador DEC-PDP-11**



Fonte: disponível em <<http://www.cosam.org/images/pdp11-35/pdp11-35.jpg>>.

#### **2.4.4 Python**

Para a programação de altíssimo nível orientada a objetos, onde a sintaxe é muito intuitiva, Python é uma linguagem constituída de várias estruturas de alto nível como listas, dicionários, data/hora, bibliotecas prontas para uso, entre outros, também é possível integrar a outras linguagens de programação como o próprio C por exemplo (BORGES, 2010).

Segundo Borges (2010) essa linguagem de programação foi desenvolvida por Guido van Rossum em 1990 no Instituto Nacional de Pesquisa para Matemática e Ciência da Computação da Holanda. Criada especialmente para profissionais da ciência e engenharia, hoje atende as necessidades de aplicações em grandes empresas como Google, Yahoo, Microsoft, Nokia e Disney.

**Figura 63 – Exemplo de código em Python**

```

def add5(x):
    return x+5

def dotwrite(ast):
    nodename = getNodeName()
    label=symbol.sym_name.get(int(ast[0]),ast[0])
    print '%s [%s=%s]' % (nodename, label),
    if isinstance(ast[1], str):
        if ast[1].strip():
            print ' = %s'; ' % ast[1]
        else:
            print ''
    else:
        print '';
        children = []
        for i, n, child in enumerate(ast[1:]):
            children.append(dotwrite(child))
        print ',    %s -> {' % nodename
        for i, name in enumerate(children):
            print '    %s' % name,

```

Fonte: disponível em: <[https://upload.wikimedia.org/wikipedia/commons/thumb/e/e1/Python\\_add5\\_syntax.svg/300px-Python\\_add5\\_syntax.svg.png](https://upload.wikimedia.org/wikipedia/commons/thumb/e/e1/Python_add5_syntax.svg/300px-Python_add5_syntax.svg.png)>. Editado pelo autor.

## 2.4.5 Java

A linguagem de programação de alto nível Java é dependente da plataforma de desenvolvimento Java, que serve para criar programas, aplicativos e animações em web de forma simples com boa performance e portabilidade. O Java foi criado através de um projeto financiado pela empresa Sun Microsystems em 1991, baseada em C++ voltada a orientação de objetos, pois a empresa percebeu o impacto que a evolução dos computadores estava causando no cotidiano das pessoas, assim como nos negócios pois o potencial do uso de programação em Java na Web para animações e personalização de páginas era iminente (DEITEL, H.; DEITEL, P., 2010; ASCENCIO; CAMPOS, 2012).

**Figura 64 – Exemplo de código em linguagem de programação Java**

```

1 //pacote "estudo";
2 package estudo;
3
4 //a classe "OláMundo" pertence está no pacote "estudo"
5 public class OláMundo {
6     //método principal - inicia a execução do aplicativo java
7     public static void main(String[] args){
8
9         //Exibe a frase "Olá Mundo!" no console.
10        System.out.println("Olá Mundo!");
11
12    }//fim do método principal
13
14 } // fim da classe "OláMundo"
15

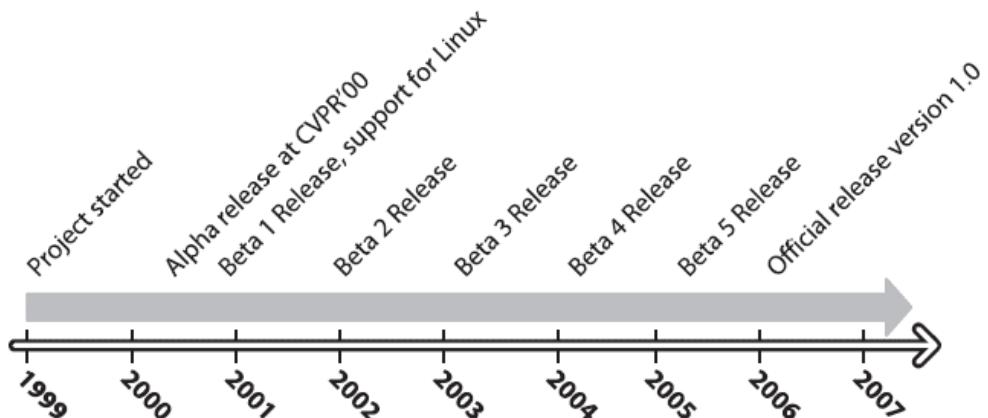
```

Fonte: disponível em: <<https://donnialmeida.files.wordpress.com/2011/08/prog1.jpg>>.

## 2.5 OpenCV

OpenCV é uma biblioteca de código aberto na área de visão computacional (Open Source Computer Vision), focada principalmente em aplicações em tempo real, baseada em C/C++ esta biblioteca foi desenvolvida por Gary Bradsky e alguns colegas na empresa Intel com início em 1999. O suporte e portabilidade oferecida pelo OpenCV é de grande valia, pois é possível programar tanto em C quanto em Python, Ruby, Matlab, Visual Basic entre outras, assim como nas plataformas Linux, Windows, Mac OS X, Android e BlackBerry (BRADSKI; KAEHLER, 2008; SZELISKI, 2010).

**Figura 65 – Linha do tempo fase inicial de desenvolvimento da biblioteca OpenCV**



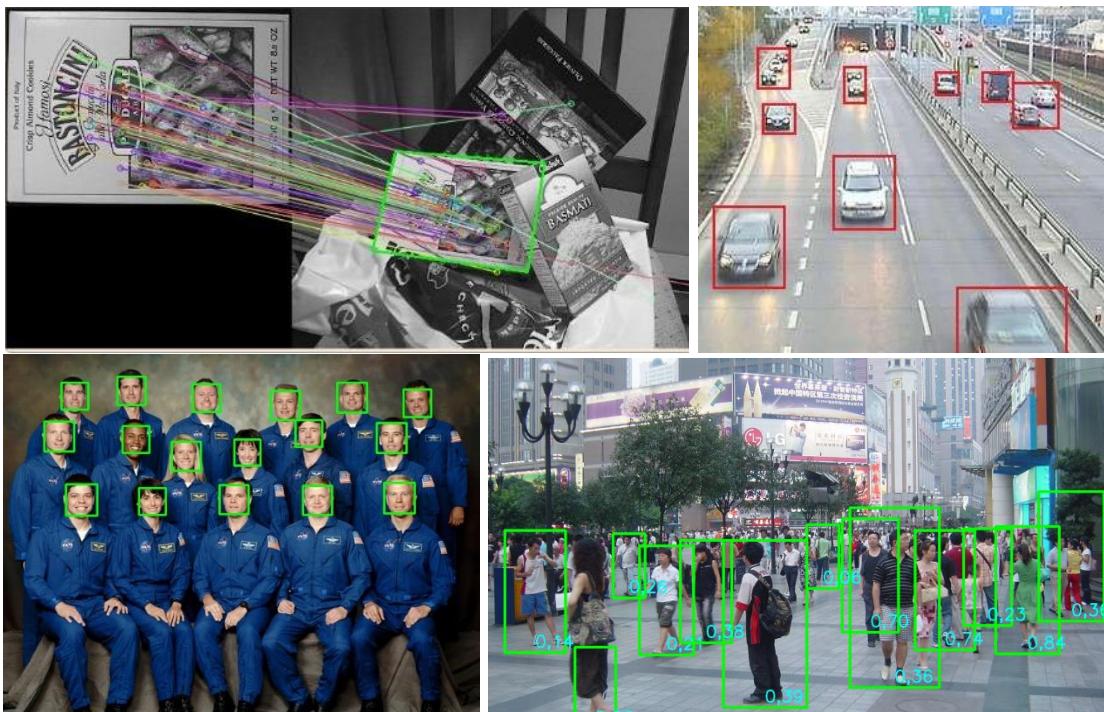
Fonte: BRADSKI; KAEHLER, 2008.

A biblioteca OpenCV é conhecida mundialmente tanto no meio acadêmico como industrial, onde empresas e instituições como IBM, Microsoft, Intel, Sony, Siemens, Google, Stanford, MIT e Cambridge fazem uso da biblioteca porquê além de proporcionar a possibilidade de aplicar conceitos de processamento de imagens digitais também é possível desenvolver produtos comerciais pois sua licença é livre, seguindo o modelo de licença BSD como informa “O Manual de Referência OpenCV” (2014), que desde 1999 é possível integrar com outros softwares de licença GPL, comumente utilizada em softwares livres (BRADSKI; KAEHLER, 2008).

Segundo Szeliski (2010) alguns dos trabalhos possíveis de serem realizados com OpenCV baseados nos pacotes de sua biblioteca são de processamento de imagens como aplicação de filtros, transformação da geometria de imagens como rotação e mudança de escala, aplicação de transformadas diversas em imagens, quantização de histogramas, segmentação, detecção de características, análise de

movimento e rastreamento de objetos, calibração de câmera e recriação de imagens em 3D e por fim de modo geral a aplicação no aprendizado de máquina.

**Figura 66 – Aplicações com OpenCV para detecção de objeto conhecido utilizando SURF detector, rastreamento de tráfego, detecção facial e rastreamento de pessoas, respectivamente**



Fonte: disponível em: <[http://docs.opencv.org/3.0-rc1/Feature\\_Homography\\_Result.jpg](http://docs.opencv.org/3.0-rc1/Feature_Homography_Result.jpg)>, <<http://www.potucek.net/img/praha.jpg>>, <<http://rpihome.blogspot.com.br/2015/03/face-detection-with-raspberry-pi.html>> e <[https://cdn-images-1.medium.com/max/2000/1\\*uEWwpKm1-MA41S0QDmcjw.png](https://cdn-images-1.medium.com/max/2000/1*uEWwpKm1-MA41S0QDmcjw.png)>. Editado pelo autor.

### 3 METODOLOGIA

Neste capítulo será explicada a necessidade da aplicação juntamente com a descrição do funcionamento do sistema proposto, incluindo também quais as ferramentas necessárias para o desenvolvimento do projeto. Sendo este um trabalho de pesquisa experimental, devem ser realizadas simulações referentes a inspeção em um processo industrial manufaturado, que possibilitem a aplicação e entendimento dos conceitos explicados no referencial teórico.

#### 3.1 Estudo da necessidade de aplicação

Buscando um cenário onde a aplicação da visão computacional pudesse agregar conhecimento e melhorias na automação de processos industriais, viu-se a possibilidade de implementar o sistema para reconhecimento de objetos e características dos mesmos em uma linha de produção, onde geralmente a automação da inspeção de qualidade ainda possui custo elevado.

Dentre várias aplicações possíveis dentro de uma indústria optou-se por efetivar o estudo baseado em uma linha de produção onde houvessem peças separadas que necessitem uma avaliação dos fatores de qualidade. Através de pesquisas realizadas em busca de um produto viável para servir de aplicação, a popularização do café expresso feito a partir de vapor pressurizado em cápsulas chamou a atenção pelas características construtivas, como o formato padrão e a separação dos sabores, juntamente com o valor agregado ao produto.

A empresa ARNO se tornou uma das maiores fabricantes de máquinas de café expresso, trazendo uma tecnologia inovadora ao desenvolver e patenteiar a máquina Nescafé Dolce Gusto em 2006 na Europa com a parceria da Nestlé. As cápsulas para a Nescafé Dolce Gusto possuem vários sabores além dos cafés, como chás, chocolates e cappuccinos, cada modelo possuí uma característica tanto no produto em pó quanto a parte física da cápsula, geralmente diferenciadas pela cor e descrição no selo da mesma como mostra a figura 67 (NESCAFÉ DOLCE GUSTO, 2017).

**Figura 67 – Cápsulas Nescafé Dolce Gusto**



Fonte: disponível em: <<http://www.embalagemmarca.com.br/wp-content/uploads/2016/10/dolcegusto.jpg>>.

No processo manufaturado deste produto geralmente são processados os mesmos sabores em conjunto, pois nota-se que atualmente no mercado brasileiro a caixa com cápsulas Nescafé Dolce Gusto possui 16 peças de mesmo sabor ou metade de um sabor e outra de leite. Porém quando a empresa necessita elaborar produtos promocionais onde há uma mixagem destas cápsulas o processo de separação das peças se torna trabalhoso e complexo ao tentar-se a automação do mesmo. Porém com a visão computacional a aplicação se torna possível e por este motivo foi determinado o estudo deste trabalho, baseado na aplicação de um sistema automatizado por processamento de imagem identificando as diferenças entre as cápsulas Nescafé Dolce Gusto, classificando-as nos dados de produção e demarcando peças não classificadas ou fora do padrão, durante o processo na linha de produção simulada para o trabalho.

**Figura 68 – Produto promocional de cápsulas**



Fonte: disponível em: <[https://www.nescafe-dolcegusto.com.br/media/catalog/product/cache/15/image/9df78eab33525d08d6e5fb8d27136e95/p/o/portacapsreciclavel\\_4.png](https://www.nescafe-dolcegusto.com.br/media/catalog/product/cache/15/image/9df78eab33525d08d6e5fb8d27136e95/p/o/portacapsreciclavel_4.png)>.

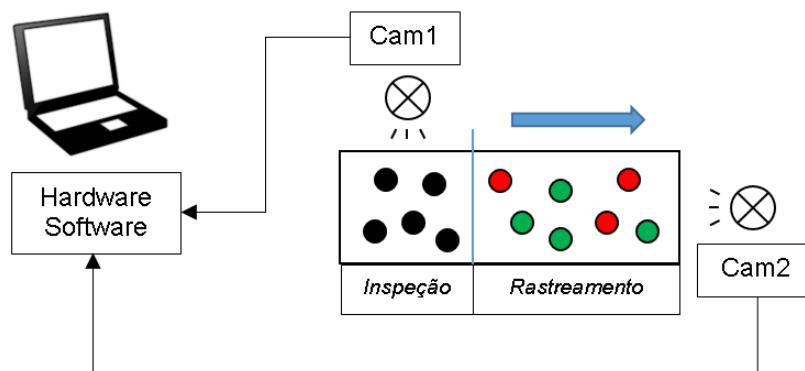
### 3.2 Descrição do funcionamento

Seguindo a ideia de que a inspeção por visão computacional seja executada ao final do processo de encapsulamento do pó do produto, ou seja, após as capsulas receberem o produto e serem seladas, deverá ser realizada leitura das cápsulas que passam pelo sistema na esteira transportadora, as peças poderão estar dispostas paralelamente sem ordem padronizada.

O sistema deverá ser capaz de inspecionar a qualidade de cada objeto, identificando na imagem, através de software e hardware compatíveis, o tamanho e cor de cada cápsula. A base de dados dos padrões para serem utilizados na comparação deverá ser previamente definida.

Após a passagem das peças pela área de inspeção, na área de rastreamento o sistema deverá demarcar e rastrear as mesmas, diferenciando quando identificadas como não conformes ou de classes de produto diferentes, onde a partir destas peças demarcadas deverá estar disponível a coordenada para servir de informação caso aplicado algum sistema de rejeição ou de coleta das cápsulas de forma automatizada ou robotizada (fora do projeto), assim como está ilustrado na figura 69.

**Figura 69 – Funcionamento do sistema de inspeção e rastreamento**



Fonte: Editado pelo autor.

### 3.3 Equipamentos e ferramentas para o desenvolvimento

Para desenvolver o sistema de visão computacional seguindo a proposta elaborada na descrição do funcionamento serão necessárias duas câmeras, uma voltada para inspeção e outra para o rastreamento das cápsulas, com resolução e desempenho que atendam às necessidades. Também será preciso um ambiente de

programação que sirva de base para o desenvolvimento dos algoritmos para o processamento digital de imagens, a escolha irá depender da linguagem de programação a ser utilizada e se suporta a biblioteca OpenCV. O hardware para executar tais funções será determinado conforme a disponibilidade e custos.

Em relação as câmeras o tipo de hardware selecionado para o protótipo foram webcams com resolução e desempenho razoáveis. De fácil aquisição no mercado o modelo Logitech HD C270 com 720 pixels de resolução proporciona confiabilidade e ótimo custo benefício, sendo sua interface de comunicação por USB 2.0, populares no uso em computadores. Esta opção foi a mais viável por ter custo acessível (em torno de R\$100,00 no mercado brasileiro) e bom desempenho em determinadas aplicações, pois câmeras industriais para visão computacional possuem em sua grande maioria valores elevados de aquisição, o que consequentemente aumenta o custo do sistema como um todo. As interfaces de comunicação mais cogitadas seguindo as indicações passadas no referencial teórico são a GigE e USB3 Vision, porém pelos sensores ópticos escolhidos essa tecnologia não está disponível.

**Figura 70 – Câmera Logitech C270**



Fonte:disponível em: <<http://www.logitech.com/assets/32825/5/hd-webcam-home-banner.jpg>> e <<http://images.highspeedbackbone.net/SKUimages/gallery/large/L23-0071-call04-cba.jpg>>.

Para o processamento computacional será utilizado um notebook com entradas USB e especificações conforme a tabela 5. O sistema operacional a ser utilizado será Windows por dispor todas ferramentas necessárias para a aplicação e pela familiaridade com o sistema. A linguagem de programação para o desenvolvimento dos algoritmos será em C/C++, então se faz necessário utilizar um ambiente de programação compatível a este fim e que proporcione suporte a biblioteca OpenCV, o software escolhido então foi o CodeBlocks. Tanto este software quanto a biblioteca OpenCV também estão disponíveis para sistema operacional Linux o que flexibiliza dependendo da base utilizada em determinado computador.

**Tabela 5 – Especificações técnicas do hardware utilizado**

Modelo	Notebook ASUS X550C
CPU	Intel® Core™ i3-3217U 1,80 GHz
Memória RAM	4GB
Sistema Operacional	Windows 10 (x64)

Fonte: Editado pelo autor.

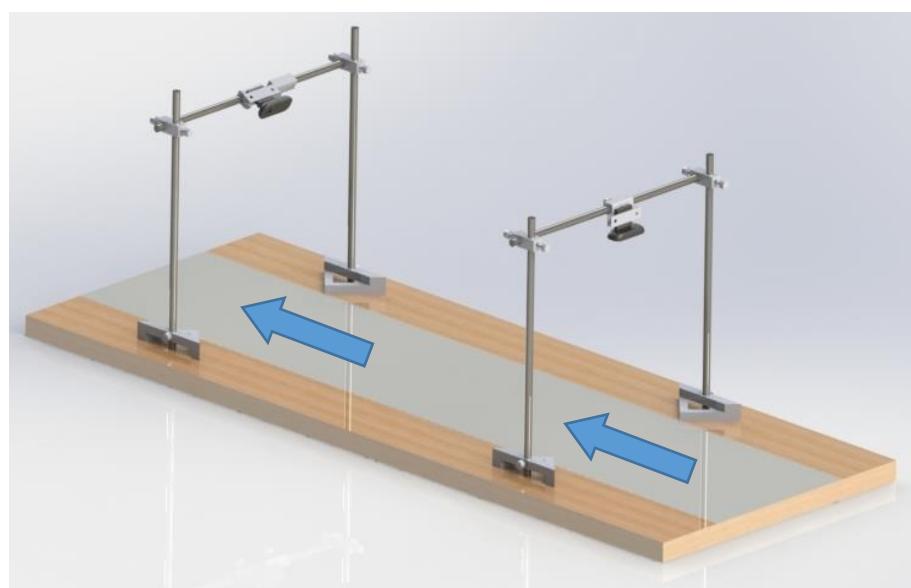
## 4 DESENVOLVIMENTO E APLICAÇÃO

Neste capítulo serão detalhados os métodos escolhidos para elaboração dos testes no protótipo de inspeção em linha de produção, as variações encontradas e a estrutura do sistema em si. A lógica foi executada através de algoritmos para obtenção dos melhores resultados, estes sendo verificados posteriormente na comparação entre as demais técnicas. Lembrando que para a realização desta etapa do trabalho a aplicação foi adequada para obter a melhor resposta com os equipamentos dispostos e assim atender os objetivos do funcionamento do sistema, explicado na seção 3.2, com isto foi possível esboçar alguns resultados e prover ligação entre a teoria já explicada com a prática.

### 4.1 Estrutura do protótipo

Para fins de simulação do sistema de visão computacional em um processo produtivo, foi elaborada uma estrutura para a instalação dos equipamentos necessários. Composta por barras de aço 1020 zincado para prevenir a corrosão, possuem a função de serem ajustáveis para posicionar as duas câmeras como mostra a figura 71, abaixo simulando a passagem da esteira transportadora foram colocadas folhas brancas de largura 297 mm.

**Figura 71 – Estrutura do protótipo inicial**



Fonte: Autor deste trabalho.

#### 4.1.1 Correção da iluminação

Como a simulação deve ser aproximada a um processo industrial, o qual não pode ser afetado pela iluminação natural ou artificial gerada pelo ambiente em volta, o protótipo foi testado em um ambiente sem iluminação, o qual se equipara a um enclausuramento da área onde a estrutura está instalada. Com isso para se obter as imagens e ser possível a visualização dos objetos se fez necessária a implementação de iluminação artificial focada nos ambientes de captura de imagem das câmeras.

O tipo de sistema de iluminação empregado foi superior com direção apontada para o foco da câmera em uso. Para a área de inspeção das peças foi utilizado um anel de LEDs em torno da câmera, este iluminando a área das imagens capturadas, deste modo a iluminação não gera sombras excessivas nas peças, já que as mesmas possuem certa altura que faz com que a parte do objeto a ser analisada fique acima do plano da esteira.

**Figura 72 – Anel de LEDs para iluminação**



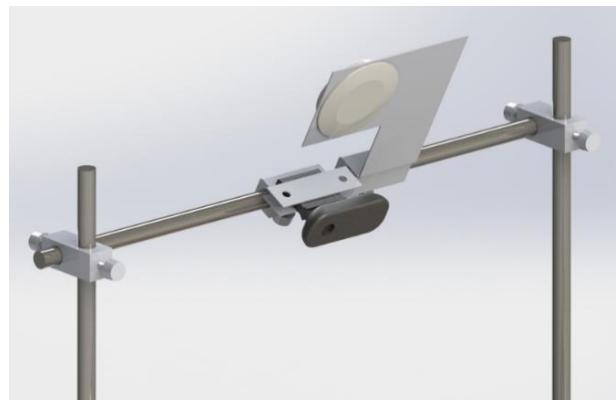
Fonte: Autor deste trabalho.

Com essa aplicação da iluminação foi possível obter uma captura de imagem com melhor definição dos objetos contidos para posteriormente efetuar o processamento de imagem, porém uma variação foi acrescentada, como o selo das cápsulas de café são feitas de plástico com uma textura reflexiva a mesma acabou gerando um pouco de reflexo da iluminação, o que pode dificultar na hora de processar a imagem.

Já na parte de rastreamento das peças foi utilizada uma luminária embutida de LEDs que possui uma face leitosa para iluminação mais homogênea, esta,

positionada a frente das peças que vem pela esteira, assim como o foco da câmera, para não gerar sombras.

**Figura 73 – Luminária de LED embutida**



Fonte: Autor deste trabalho.

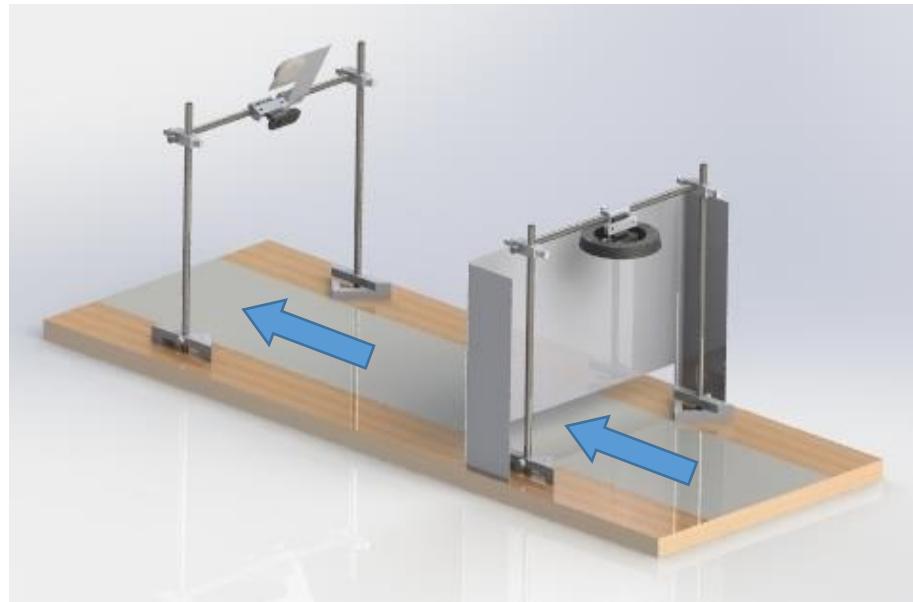
A iluminação pelo anel de LEDs na área de inspeção acabou iluminando o início da área de rastreamento que desta maneira gera sombras pelas cápsulas que passam pelo local, e estas variações capturadas pela segunda câmera também podem gerar falsos positivos. Para corrigir isto foi implementada uma barreira na linha divisória entre a área de inspeção e rastreamento, como mostra a figura 74, assim a iluminação excessiva é contida.

**Figura 74 – Barreira de luz**



Fonte: Autor deste trabalho.

**Figura 75 – Estrutura do protótipo final**



Fonte: Autor deste trabalho.

## 4.2 Implementação das técnicas e algoritmos

Nesta seção será explicada a lógica utilizada para executar o processamento das imagens adquiridas pelos sensores ópticos, seguindo os passos do processamento, citados na seção 2.3.2, e alguns dos métodos demonstrados no referencial teórico utilizando funções dispostas na biblioteca OpenCV. Serão comparadas algumas das aplicações destas técnicas a fim de definir qual a melhor opção para o funcionamento correto do sistema de inspeção multipeças, onde estas serão analisadas através das imagens obtidas pela câmera 1, referente a inspeção das peças e a câmera 2, referente ao rastreamento das peças corretas e erradas.

Como observação, o algoritmo e comentários contidos no mesmo foram desenvolvidos em inglês, pois a programação em C está baseada na língua inglesa, e desta forma se mantém uma padronização da escrita e leitura do código para o desenvolvedor.

### 4.2.1 Módulos utilizados

OpenCV é constituído por módulos que servem de base para o desenvolvimento de cada tipo de aplicação, como mostra a tabela 6, variadas funções podem ser

desenvolvidas com a biblioteca, estes módulos devem ser incluídos ao algoritmo no cabeçalho.

Porém neste trabalho será utilizado o módulo “OpenCV world”, que reúne todos módulos OpenCV em um só, o que gera melhora na disposição do programa para o usuário final, pois reduz o tamanho do arquivo tanto em MB quanto em quantidade de DLLs<sup>17</sup>. Estes devem ser selecionados ao se construir o módulo através do programa CMake que serve para gerar e testar pacotes de bibliotecas.

**Tabela 6 – Relação de alguns módulos presentes na biblioteca OpenCV**

<b>Core</b>	Define estruturas de dados básicos, incluindo Mat multidimensional.
<b>Imgproc</b>	Processamento, filtragem e transformação geométrica de imagens.
<b>Video</b>	Análise de vídeo com estudo de movimento, subtração de fundo e restreamento de objetos.
<b>Calib3d</b>	Calibração de câmera e elementos de reconstrução 3D.
<b>Features2d</b>	Detector de recursos salientes e descritores correspondentes.
<b>Objdetect</b>	Detecção de objetos de classes definidas, como rostos, canetas, carros, etc.
<b>Highgui</b>	Recursos simples de interface.
<b>Videoio</b>	Captura e codecs de vídeo.
<b>Gpu</b>	Aceleração de algoritmo por GPU.

Fonte: disponível em: <<http://docs.opencv.org/3.1.0/d1/dfb/intro.html>>. Editado pelo autor.

#### **4.2.2 Obtenção e conversão das imagens**

Utilizando as câmeras para obtenção das imagens, para se obter as capturas em forma de imagem digital é necessário utilizar funções no algoritmo. Para isto foi empregada a função *VideoCapture*, disponível na biblioteca OpenCV, assim designando também uma matriz para receber a imagem referenciada a determinada porta ou classe, como mostram as parcelas do algoritmo na figura 76.

---

<sup>17</sup> Bibliotecas compartilhadas no sistema operacional Windows.

**Figura 76 – Algoritmo de captura de imagens das câmeras**

```
//Define arrays to receive images
Mat img1, img2;
//Capture image from web camera
VideoCapture cap1(0); //Camera 1
VideoCapture cap2(2); //Camera 2
//Pass capture to arrays
cap1 >> img1;
cap2 >> img2;
```

Fonte: Autor deste trabalho.

Deste modo adquiriu-se a imagem fornecida pelo hardware das câmeras na forma colorida com resolução de 640x480 pixels. Na necessidade de se trabalhar com imagem em escala de cinza para facilitar o processamento na filtragem ou demais etapas, se utilizou a função descrita na figura 77.

**Figura 77 – Descrição da função cvtColor**

**C++:** void cvtColor(InputArray **src**, OutputArray **dst**, int **code**, int **dstCn=0** )

- **src** – input image: 8-bit unsigned, 16-bit unsigned ( CV\_16UC... ), or single-precision floating-point.
- **dst** – output image of the same size and depth as **src**.
- **code** – color space conversion code (see the description below).
- **dstCn** – number of channels in the destination image; if the parameter is 0, the number of the channels is derived automatically from **src** and **code** .

Fonte: disponível em: <[http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous\\_transformations.html](http://docs.opencv.org/2.4/modules/imgproc/doc/miscellaneous_transformations.html)>. Editado pelo autor.

#### **4.2.3 Câmera 1 (Inspeção) - Processamento digital de imagens**

Para atingir os objetivos do processamento de imagem da parte da câmera 1 é necessário encontrar a melhor lógica que possibilite identificar os objetos contidos na cena sem causar ruídos, que geram falsos positivos, assim será possível extrair informações que podem servir de base para a qualificação do produto. Deste modo foram realizados testes e comparações como seguem as descrições a seguir.

#### 4.2.3.1 Câmera 1 - Pré-processamento do sistema

Após obter as imagens, foram aplicados alguns métodos seguindo orientações de processamento de imagens digitais que estão citadas nas referências, na sequência estão os modelos dos algoritmos aplicados.

**Figura 78 – Função de conversão de imagem RGB em escala de cinza e funções de filtragem com máscaras 7x7**

```
///Convert image type
cvtColor(img1, img1Gray, COLOR_RGB2GRAY, 1);

///Low-pass filters
blur(img1Gray, img1Proc, Size(7,7));
medianBlur(img1Gray, img1Proc, 7);
GaussianBlur(img1Gray, img1Proc, Size(7,7), 0, 0, BORDER_DEFAULT);
bilateralFilter(img1Gray, img1Proc, 7, 20, 20);
```

Fonte: Autor deste trabalho.

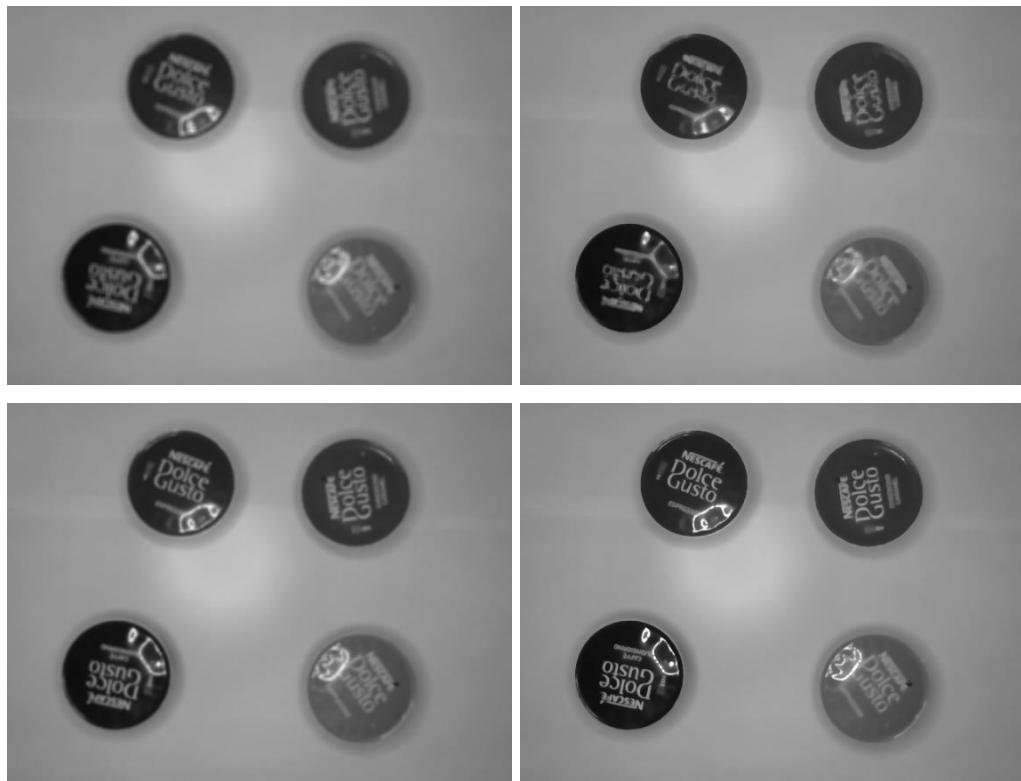
Para realizar um pré-processamento adequado é necessário aplicar um filtro passa-baixa que disponibilize resposta eficaz para o processo seguinte, ou seja, que ao se aplicar um filtro passa-alta ou outro tipo de análise em relação as bordas, é necessário que estas não contenham tanta variação, realizando um leve borramento com filtro passa-baixa a resposta é melhorada. A imagem a ser tratada está em escala de cinza como mostra a figura 79, determinada como *img1Gray* no algoritmo, sendo o primeiro algoritmo aplicado de um filtro de média, após o filtro de mediana, Gaussiano e bilateral (*blur*, *medianBlur*, *GaussianBlur* e *bilateralFilter*) como mostram os resultados na figura 80 onde *img1Proc* é a imagem processada.

**Figura 79 – Amostra de imagem capturada colorida (RGB) e em escala de cinza**



Fonte: Autor deste trabalho.

**Figura 80 – Comparação dos resultados da aplicação de filtro passa-baixa de média, mediana, Gaussiano e bilateral respectivamente, com máscaras 7x7**



Fonte: Autor deste trabalho.

Com os resultados obtidos dos filtros passa baixa notou-se que para a função de suavizar bordas todos atenderam, alguns com mais variações do que os outros, mas pela qualidade da resposta comparada à capacidade de processamento necessária os filtros de mediana e Gaussiano se saíram melhor. Se tomará como base o filtro Gaussiano para as seguintes etapas do processamento.

Após realizar a suavização da imagem foram aplicados alguns filtros passa-alta para evidenciar a presença das bordas. Inicialmente foi aplicado o filtro Laplaciano, na sequência está descrito o algoritmo deste processo.

**Figura 81 – Algoritmo do filtro Laplaciano**

```
//LAPLACIAN high-pass filter
Laplacian(img1Proc, img1Proc, img1Proc.depth(), 3);
```

Fonte: Autor deste trabalho.

Outro método recomendado por sua simplicidade e em muitas aplicações se torna satisfatório o resultado, é o filtro High-boost onde a resposta deste filtro passa-

alta nada mais é do que a diferença entre a imagem original e a imagem suavizada com filtro passa-baixa, essa lógica foi elaborada no algoritmo como mostra a figura 82, pois a mesma não está disponível na forma de função dentro da biblioteca OpenCV.

A resposta ao se aumentar o valor de  $k$ , ressalta as regiões de borda dos objetos, porém aumenta o nível de ruído e granulações presentes na imagem, sendo assim o valor de  $k=3$  ficou em uma resposta intermediária de forma satisfatória.

**Figura 82 – Algoritmo filtro passa-alta High-boost**

```
//HIGH-BOOST high-pass filter
Mat mask;
int k=3;
img1Proc = k*(img1Gray-img1Proc);
```

Fonte: Autor deste trabalho.

Após os testes com filtro Laplaciano e High-boost foram realizados testes com os gradientes de Sobel e Roberts. Para o gradiente de Sobel é necessário criar as máscaras vertical ( $X$ ) e horizontal ( $Y$ ), que aplicadas a imagem a ser tratada, utilizando máscaras de tamanho  $3x3$ , fornece a resposta representada na figura 85.

**Figura 83 – Algoritmo do gradiente de Sobel**

```
//SOBEL high-pass filter
//Generate grad_x and grad_y
Mat grad_x, grad_y;
Mat abs_grad_x, abs_grad_y;
//Gradient X
Sobel(img1Proc, grad_x, CV_16S, 1, 0, 3);
convertScaleAbs( grad_x, abs_grad_x );
//Gradient Y
Sobel(img1Proc, grad_y, CV_16S, 0, 1, 3);
convertScaleAbs( grad_y, abs_grad_y );
//Total Gradient (approximate)
addWeighted(abs_grad_x, 0.5, abs_grad_y, 0.5, 0, img1Proc);
```

Fonte: Autor deste trabalho.

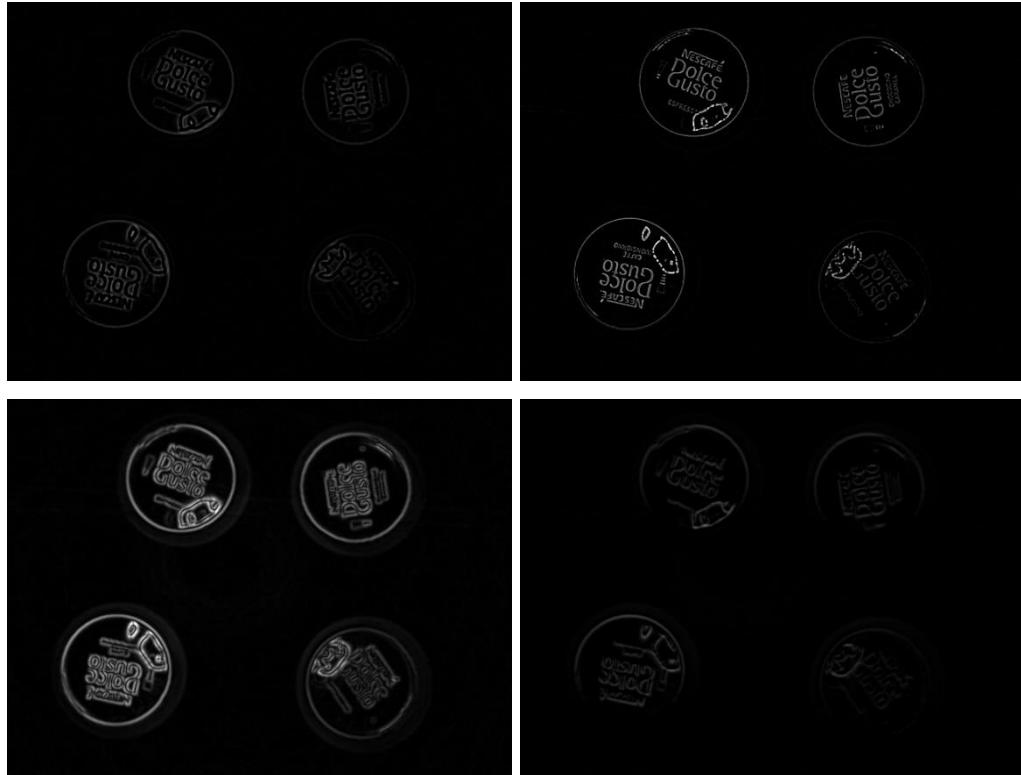
Para aplicar o gradiente de Roberts o mesmo processo é utilizado, porém é necessário fornecer as máscaras  $2x2$  e aplica-las através de um filtro 2D disponível na biblioteca, pois esta função não está disponível no OpenCV diretamente.

**Figura 84 – Algoritmo do gradiente de Roberts**

```
//ROBERTS high-pass filter
//Generate grad_x and grad_y
Mat grad_x, grad_y;
Mat abs_grad_x, abs_grad_y;
//Gradient X
grad_x = (Mat<char>(2,2) << 1, 0, 0, -1);
filter2D(img1Proc, abs_grad_x, img1Proc.depth(), grad_x);
//Gradient Y
grad_y = (Mat<char>(2,2) << 0, 1, -1, 0);
filter2D(img1Proc, abs_grad_y, img1Proc.depth(), grad_y);
//Total Gradient (approximate)
addWeighted(abs_grad_x, 1, abs_grad_y, 1, 0, img1Proc);
```

Fonte: Autor deste trabalho.

**Figura 85 – Respostas da aplicação dos filtros Laplaciano, High-Boost, Sobel e Roberts após a suavização por filtro Gaussiano respectivamente**



Fonte: Autor deste trabalho.

No processo subsequente a parte de filtros passa-alta que melhor se ajusta a necessidade irá depender do método utilizado, mas analisando de forma superficial os filtros Laplaciano e High-boost geraram bordas mais precisas, sem tanta variação ou borramento como os gradientes de Sobel e Roberts.

#### 4.2.3.2 Câmera 1 - Segmentação e detecção dos objetos do sistema

Para conseguir identificar os objetos presentes na imagem é necessário efetuar o processo de segmentação, que consiste em métodos de separação e detecção de bordas ou variações nos níveis de cinza existentes na imagem.

Após aplicado os filtros de pré-processamento a primeira técnica utilizada foi de aproximação poligonal, juntamente com a limiarização da imagem tratada em escala de cinza.

Com a aplicação de filtro passa-baixa, onde após foi aplicado o filtro de Sobel, sendo este o único que não gerou erros na execução do programa, e posteriormente foi elaborada uma limiarização desta imagem resultante do pré-processamento com valor de limiar definido pelo usuário, no teste realizado o valor utilizado foi de 27, assim na sequência foi aplicada a detecção de bordas e o método de aproximação poligonal, como mostra o algoritmo da figura 86, que a partir deste se gerou a representação de uma circunferência em volta do objeto identificado para fins de comprovação.

**Figura 86 – Algoritmo de segmentação e detecção de objetos por limiarização e aproximação poligonal com uso do filtro de Sobel**

```
//THRESHOLD
threshold(img1Proc, img1Segm, 27, 255, CV_THRESH_BINARY);

//POLYGONAL APPROXIMATION
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
vector<vector<Point>> contours_poly( num_max_parts );
vector<Point2f> center( num_max_parts );
vector<float> radius( num_max_parts );

//Find contours
findContours(img1Segm, contours, hierarchy, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

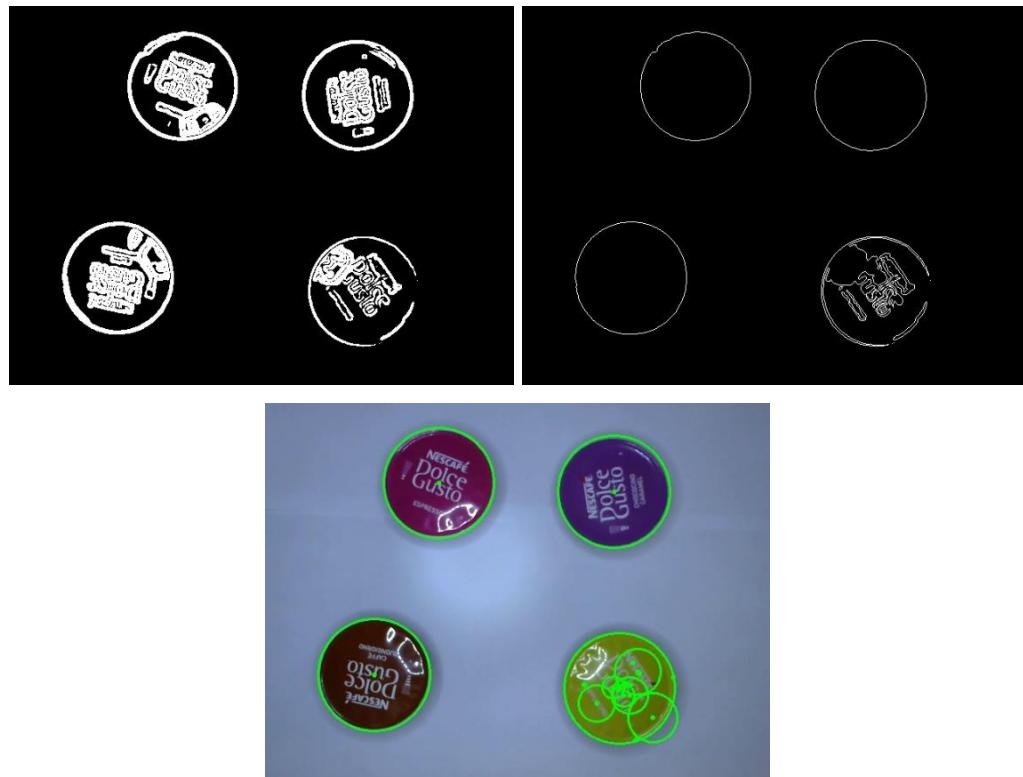
for(int i=0; i<contours.size(); i++){

    //Approximate contours to polygons + center + radius
    approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
    minEnclosingCircle((Mat)contours_poly[i], center[i], radius[i]);
    //Draw circle center
    circle(img1, center[i], 3, Scalar(0,255,0), -1, 8, 0);
    //Draw circle outline
    circle(img1, center[i], radius[i], Scalar(0,255,0), 2, 8, 0);

}
```

Fonte: Autor deste trabalho.

**Figura 87 – Respostas do algoritmo de limiarização e aproximação poligonal com uso do filtro de Sobel. Imagens da limiarização, detecção de bordas e resultados finais respectivamente**



Fonte: Autor deste trabalho.

Como mostra o algoritmo da figura 88, na mesma concepção da lógica utilizada anteriormente foi aplicado ao sistema inicialmente o filtro passa-baixa Gaussiano, após, sem o uso de filtro passa-alta foi feita diretamente a limiarização da imagem, onde tanto o caso do valor de limiar sendo definido pelo usuário quanto estimado pelo método de Otsu funcionam corretamente, porém vale lembrar que o método de Otsu não será aplicável quando o sistema estiver funcionando em tempo real, pois caso a imagem capturada mude de uma cena com peças para uma sem o limiar irá variar e fazer com que a própria cena proporcione falsos positivos dos objetos, na sequência então foi aplicada a aproximação poligonal, que apresenta as respostas da figura 89.

**Figura 88 – Algoritmo de segmentação e detecção de objetos por limiarização e aproximação poligonal sem uso do filtro de Sobel**

```

//THRESHOLD
threshold(img1Proc, img1Segm, 125, 255, CV_THRESH_OTSU);
img1Segm = Scalar::all(255) - img1Segm; //Invert binary image

//POLYGONAL APPROXIMATION
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
vector<vector<Point>> contours_poly( num_max_parts );
vector<Point2f> center( num_max_parts );
vector<float> radius( num_max_parts );

//Find contours
findContours(img1Segm, contours, hierarchy, CV_RETR_EXTERNAL,
            CV_CHAIN_APPROX_SIMPLE, Point(0, 0));

for(int i=0; i<contours.size(); i++){

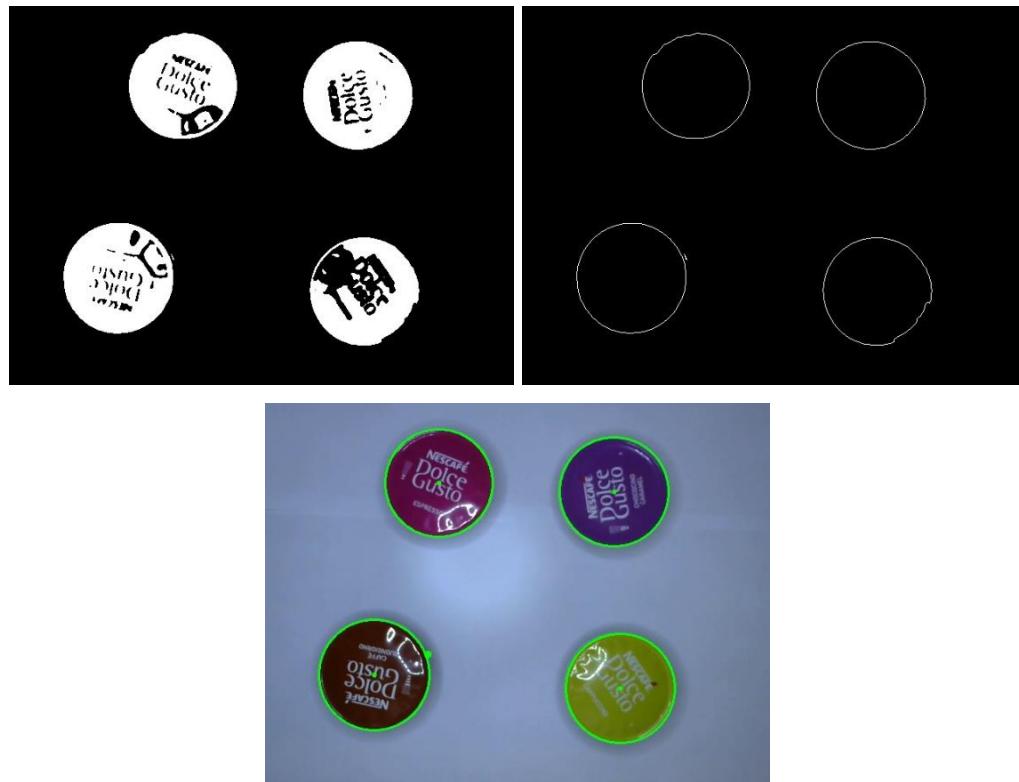
    //Approximate contours to polygons + center + radius
    approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
    minEnclosingCircle((Mat)contours_poly[i], center[i], radius[i]);
    //Draw circle center
    circle(img1, center[i], 3, Scalar(0,255,0), -1, 8, 0);
    //Draw circle outline
    circle(img1, center[i], radius[i], Scalar(0,255,0), 2, 8, 0);

}

```

Fonte: Autor deste trabalho.

**Figura 89 – Respostas do algoritmo de limiarização e aproximação poligonal sem uso do filtro de Sobel. Imagens da limiarização, detecção de bordas e resultados finais respectivamente**



Fonte: Autor deste trabalho.

Mudando a concepção do processo outra técnica testada foi da Transformada de Hough, conforme explicado na seção 2.3.4.6, para implementar o algoritmo é necessário suavizar a imagem de entrada, sendo assim os métodos de filtragem para suavização de imagem anteriormente relatados foram aplicados para remover ruídos presentes da imagem original, os quatro modelos de filtro passa-baixa funcionaram com o algoritmo da Transformada de Hough para detecção de círculos, onde este, disponível na biblioteca OpenCV, executa o processo de detecção de círculos na imagem como demonstra o algoritmo na figura 90.

**Figura 90 – Algoritmo de segmentação e detecção de objetos pela transformada de Hough**

```
//HOUGH TRANSFORM
vector<Vec3f> circles; //Vector for circles
//Apply Hough Transform to find circles
HoughCircles(img1Proc, circles, CV_HOUGH_GRADIENT, 1, 10);
//Draw the circles detected
for( size_t i = 0; i < circles.size(); i++ ){
    Point center(cvRound(circles[i][0]), cvRound(circles[i][1]));
    int radius = cvRound(circles[i][2]);
    //Draw circle center
    circle(img1, center, 3, Scalar(0,255,0), -1, 8, 0);
    //Draw circle outline
    circle(img1, center, radius, Scalar(0,255,0), 2, 8, 0);
}
```

Fonte: Autor deste trabalho.

**Figura 91 – Amostras das respostas do algoritmo de segmentação e detecção de objetos pela transformada de Hough**



Fonte: Autor deste trabalho.

Buscando outras formas de realizar o processamento e detecção dos objetos onde a lógica forneça um resultado mais preciso em relação a inspeção quando ocorre em tempo real, como foi explicado no referencial teórico uma forma simples e indicada para essas situações é utilizar a subtração de fundo, eliminando o fundo que é

praticamente estático e deixando apenas os objetos em movimento. Aplicou-se então o algoritmo da figura 92.

**Figura 92 – Algoritmo de subtração de fundo absoluto**

```
//Load background image
img1Backg = imread("Img1Background.jpg");

//Background subtraction
absdiff(img1, img1Backg, img1Foreg);
```

Fonte: Autor deste trabalho.

**Figura 93 – Resultado da operação de subtração de fundo. Imagem original, de fundo e resultante respectivamente**



Fonte: Autor deste trabalho.

Com a imagem resultante dos objetos foi aplicado um filtro Gaussiano para suavizar e assim distinguir as cores da sobra do fundo que não tenha sido eliminada, é possível trabalhar tanto em escala de cinza quanto nos planos RGB de cores, porém dependendo do caso se houverem variações no fundo que após a subtração e conversão para escala de cinza se assemelhe aos níveis das peças, quando for aplicar o limiar esta peça pode não aparecer ou parte do fundo ser detectada como objeto,

gerando uma segmentação errônea. Como é possível visualizar no algoritmo da figura 94 e os resultados na sequência, as duas lógicas podem vir a atender sendo que o limiar por cores é chamado pela função *inRange* e é necessário encontrar as faixas dos níveis de cores de cada cápsula, na simulação dois limiares por cor foram aplicados e suas resultantes somadas.

**Figura 94 – Algoritmos de pré-processamento e segmentação dos objetos em escala de cinza e RGB**

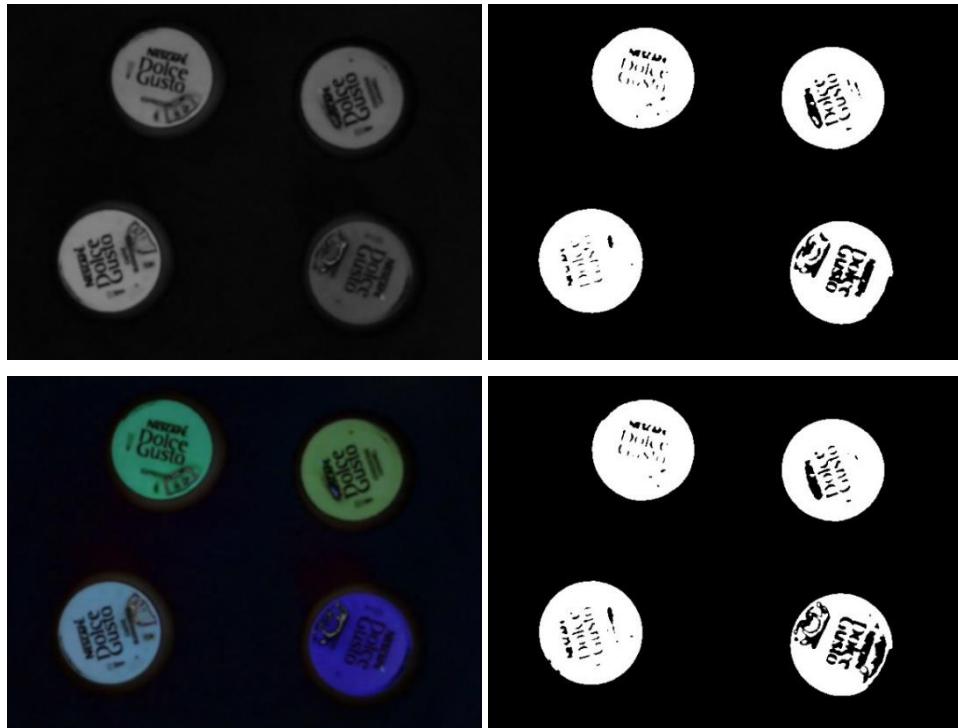
```
//Convert image type
cvtColor(img1Foreg, img1Gray, COLOR_RGB2GRAY, 1);
//LOW-PASS FILTER
GaussianBlur(img1Gray, img1Proc, Size(7,7), 0, 0, BORDER_DEFAULT);
//THRESHOLD
threshold(img1Proc, img1Segm, 40, 255, CV_THRESH_BINARY);

//OR//

//LOW-PASS FILTER
GaussianBlur(img1Foreg, img1Proc, Size(7,7), 0, 0, BORDER_DEFAULT);
//THRESHOLD BY COLOR
inRange(img1Proc, Scalar(0, 45, 0), Scalar(255, 255, 255), img1Segm);
inRange(img1Proc, Scalar(89, 0, 0), Scalar(218, 35, 52), img1Proc);
//Add first image thresholded to second
add(img1Segm, img1Proc, img1Segm);
```

Fonte: Autor deste trabalho.

**Figura 95 – Respostas de limiar para imagem em escala de cinza e colorida**



Fonte: Autor deste trabalho.

A ideia desta lógica é aplicar um dos métodos já testados para detecção de objetos, que é a aproximação poligonal, porém a mesma pode vir a resultar em falsos positivos se houverem deformações na imagem limiarizada, como pontos fora do objeto ou buracos e ilhas de imagem dentro do mesmo. Uma solução encontrada para eliminar estes problemas é a utilização de processamento morfológico.

Aplicando-se as morfologias de dilatação e erosão voltadas para abertura (eliminação de ruídos) e fechamento (preenchimento de buracos), é possível efetuar correções nas imagens adquiridas, pois as peças como verificado anteriormente resultam em reflexo ocasionado pela iluminação artificial, entre outros detalhes que o pré-processamento e limiarização não conseguem filtrar. A biblioteca OpenCV possui estas técnicas já disponíveis para utilização, podendo chama-las através das funções *erode*, *dilate* ou *morphologyEx*.

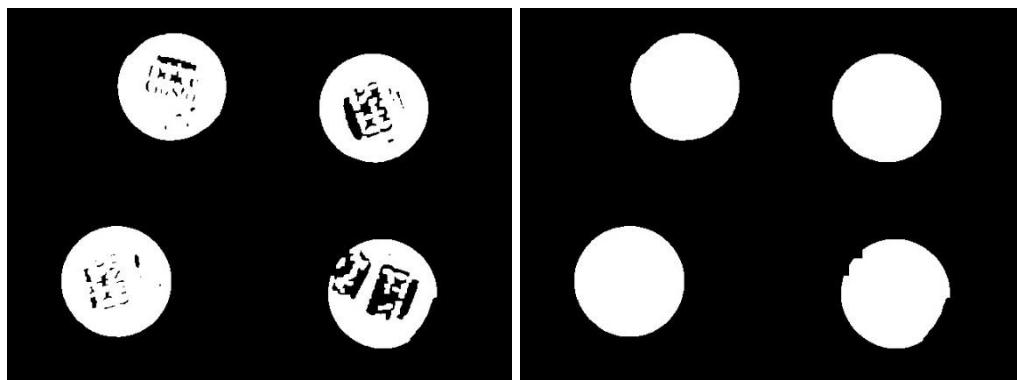
Na simulação foi utilizada uma matriz (*kernel*) de tamanho 3x3 preenchida pelo valor 1, para realizar as operações, sendo então a aplicação da abertura com 2 iterações seguida do processo de fechamento com 9 iterações conforme está ilustrado no algoritmo da figura 96.

**Figura 96 – Algoritmo de processamento morfológico para abertura e fechamento**

```
//DILATION AND EROSION
Mat kernel = Mat::ones(Size (3,3), CV_8U);
//Apply opening that is erosion followed by dilation |to remove noise
morphologyEx(img1Segm, img1Segm, MORPH_OPEN,
    kernel, Point(-1,-1), 2, BORDER_DEFAULT);
//Apply closing that is dilation followed by erosion to fill the holes
morphologyEx(img1Segm, img1Segm, MORPH_CLOSE,
    kernel, Point(-1,-1), 9, BORDER_DEFAULT);
```

Fonte: Autor deste trabalho.

**Figura 97 – Resposta da aplicação de abertura e fechamento morfológico respectivamente**



Fonte: Autor deste trabalho.

Após este tratamento das imagens ser realizado é possível aplicar então a aproximação poligonal e obter os resultados como mostra a figura 98.

**Figura 98 – Respostas da aplicação do algoritmo de subtração de fundo e processamento morfológico**



Fonte: Autor deste trabalho.

#### 4.2.3.3 Câmera 1 - Extração de características dos objetos

Após conseguir detectar os objetos corretamente passou-se para a lógica de extração dos dados informativos do sistema, na parte de inspeção as características a se verificar foram de tamanho, cor média e posição das cápsulas.

Os dados extraídos se dão em função dos pixels contidos na matriz da imagem analisada. O tamanho e posição são adquiridos ao se aplicar a lógica de aproximação poligonal e envolvimento circular onde estes dados ficam contidos nos vetores de cada objeto em *center* que informa o ponto central (*x,y*) e *radius* que retêm a informação do raio daquela peça.

**Figura 99 – Aquisição do ponto central e raio de cada peça**

```
//Approximate contours to polygons + center + radius
approxPolyDP(Mat(contours[i]), contours_poly[i], 3, true);
minEnclosingCircle((Mat)contours_poly[i], center[i], radius[i])
```

Fonte: Autor deste trabalho.

A lógica para extração da média de cores de cada cápsula foi elaborada ao se detectar o objeto contido na imagem, e retirado um recorte da região de interesse, nessa região é feita a média dos níveis de cores existentes, porém como a estampa da peça também contém escrita o cálculo foi elaborado em cima de pontos

específicos. Já que a escrita fica em sua maioria na parte central do selo a técnica foi extrair os níveis RGB dos pixels que formam um losango na parte superior e efetuar média aritmética dos mesmos. Não foi utilizada a geometria circular por haver reflexo na forma circular onde caso essa linha de referência usasse estes valores para cálculo iria tender o resultado da média à cor branca (255, 255, 255).

**Figura 100 – Geometria de referência para cálculo de média aritmética das cores**



Fonte: Autor deste trabalho.

#### 4.2.3.4 Câmera 1 - Reconhecimento e interpretação das informações

O sistema tem a função de realizar inspeções e para isto deve conter uma base de dados para fins de comparação. Para criar esta base de padrões foi desenvolvido um algoritmo para salvar as imagens dos objetos detectados quando o usuário quiser efetuar o registro dos padrões, onde também é possível determinar a qual classe o produto pertence, as cápsulas utilizadas para esse registro devem ser os modelos padrões a serem adotados. Informando o diâmetro real do selo da peça é possível então ter o tamanho, cor média e classe padrão para realizar a inspeção.

Se o diâmetro da peça real é  $d$  (mm) e na detecção o objeto possuí um raio em  $r$  (pixels) é possível definir uma relação entre pixels e milímetros como mostra a equação 49. Assim tanto o tamanho quanto a posição das peças extraídas em pixels pode ser multiplicada pela relação encontrada, a resposta será a medida em mm real como se encontra.

$$\text{relação} = \frac{d \text{ (mm)}}{r. 2 \text{ (pixels)}} \quad (49)$$

Caso o tamanho detectado no objeto contido na imagem for maior ou menor que os limites superior e inferior definidos esta peça será nomeada como defeituosa ou fora do padrão e o problema demarcado.

O mesmo ocorre na comparação por cor, que é feita pela subtração entre os níveis RGB das cores médias extraídas tanto das peças padrões do banco de dados quanto das cápsulas em análise, assim comparando o objeto de interesse a todos os padrões. As diferenças resultantes de cada plano de cor serão somadas e um valor limite definido pelo usuário, se a soma destas diferenças for maior que o limite então a peça é classificada como fora do padrão e demarcada com o problema encontrado, caso contrário, se na comparação com os padrões a soma resultante estiver dentro do limite e for o menor resultado encontrado, a cápsula será classificada como “ok” e nomeada com o tipo do produto cadastrado.

**Figura 101 – Inspeção de tamanho e cor ok**



Fonte: Autor deste trabalho.

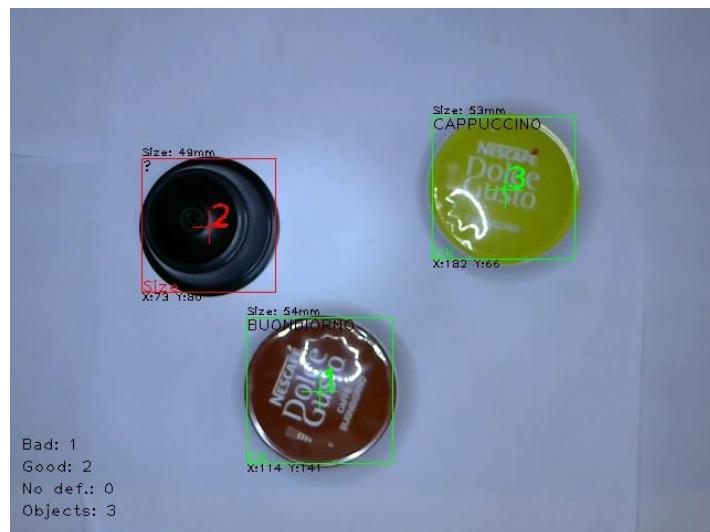
**Figura 102 – Inspeção com peças fora do tamanho e não registrada como padrão**



Fonte: Autor deste trabalho.

Obtendo a informação da posição de cada peça é possível numerar a quantidade de cápsulas inspecionadas e qual o número correto atribuído a cada uma. A leitura dos objetos com os algoritmos utilizados é feita da maior linha para a menor, com as peças vindo no sentido contrário da leitura é possível numerar corretamente cada uma, porém a cada peça que chegar na linha limite da matriz da imagem é necessário deslocar o endereço dos vetores para não sobrescrever os dados adquiridos em relação à aquela cápsula. Por exemplo, na leitura de 4 peças estas são numeradas como (1,2,3,4), quando a cápsula 1 chegar ao limite e sair da cena os dados identificados serão endereçados à (2,3,4) e não (1,2,3).

**Figura 103 – Numeração das peças iniciais**



Fonte: Autor deste trabalho.

**Figura 104 – Numeração das peças seguintes**



Fonte: Autor deste trabalho.

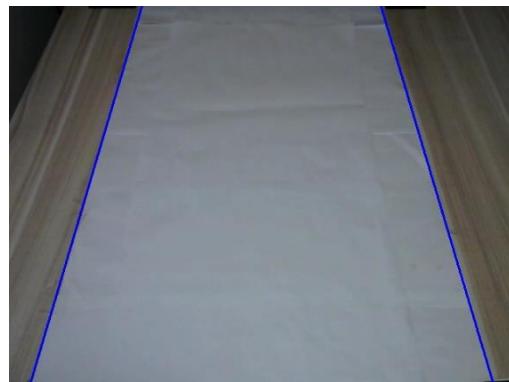
#### **4.2.4 Câmera 2 (Rastreamento) - Processamento digital de imagens**

Assim como na parte de inspeção, para identificar os objetos contidos nas imagens da segunda câmera é necessário realizar um processamento digital, esta parte que tem o objetivo de rastrear as cápsulas com a definição previamente avaliada na inspeção e fornecer sua coordenada. A lógica de processamento foi elaborada baseada nas técnicas identificadas como sendo mais eficientes na parte de inspeção, assim não sendo necessário comparar e avaliar todas as metodologias anteriormente citadas.

##### **4.2.4.1 Câmera 2 - Calibração**

A calibração da câmera 2 está relacionada a direcionar e posicionar a câmera conforme a angulação pré-determinada pelo usuário, esta requere as informações de distância de recuo e ângulo dos limites das bordas (em pixels), que seriam os limites da esteira. Definido isto automaticamente são geradas duas linhas de referência como ilustra a figura 105.

**Figura 105 – Linhas limites da esteira com recuo de 26 pixels e ângulo de 73,5º**



Fonte: Autor deste trabalho.

#### **4.2.4.2 Câmera 2 - Pré-processamento do sistema**

Utilizando as operações que ofereceram melhores resultados na parte de inspeção o pré-processamento no rastreamento das peças foi elaborado baseado na subtração de fundo inicialmente como mostra a figura 106.

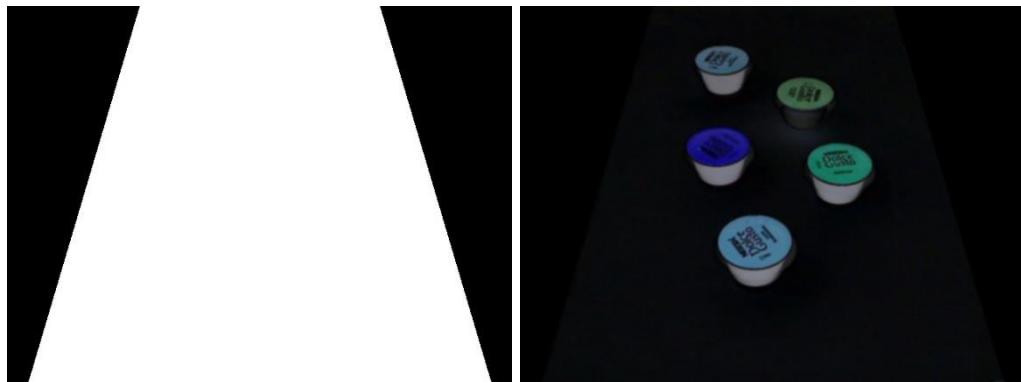
**Figura 106 – Resultado da operação de subtração de fundo da câmera 2. Imagem original, de fundo e resultante respectivamente**



Fonte: Autor deste trabalho.

Os limites definidos anteriormente não ajudam só a posicionar a câmera no ângulo correto, mas também, os dados informados servem para criar uma máscara a partir das linhas que irão eliminar tudo o que ficar fora dos limites, assim, essa região não será analisada, o que poderia gerar falsas detecções. Após a implementação da máscara à imagem resultante da subtração de fundo foi aplicado um filtro passa baixa Gaussiano de máscara  $5 \times 5$ .

**Figura 107 – Máscara aplicada e resposta ao filtro Gaussiano respectivamente**



Fonte: Autor deste trabalho.

#### **4.2.4.3 Câmera 2 - Segmentação e detecção dos objetos do sistema**

A imagem convertida em escala de cinza acabou gerando falsos positivos pela semelhança com as cores a segmentar, então optou-se por utilizar a limiarização por cor novamente, com duas faixas de valores ajustadas conforme o algoritmo mostrado na figura 108. Logo após, os resultados são unidos a uma imagem apenas para dar sequência ao processamento.

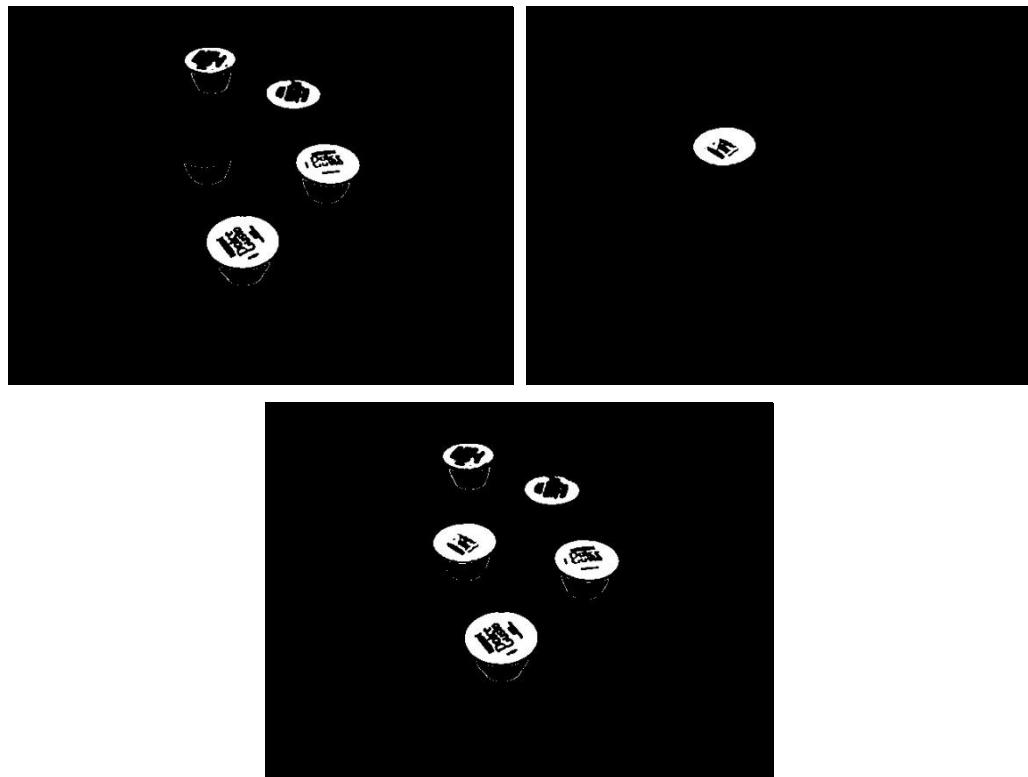
**Figura 108 – Algoritmo de limiarização por cor**

```
//THRESHOLD BY COLOR
inRange(img2Proc, Scalar(0, 90, 0), Scalar(220, 178, 95), img2Segm);
inRange(img2Proc, Scalar(83, 0, 0), Scalar(255, 46, 75), img2Proc);

//Add first image thresholded to second
add(img2Segm, img2Proc, img2Segm);
```

Fonte: Autor deste trabalho.

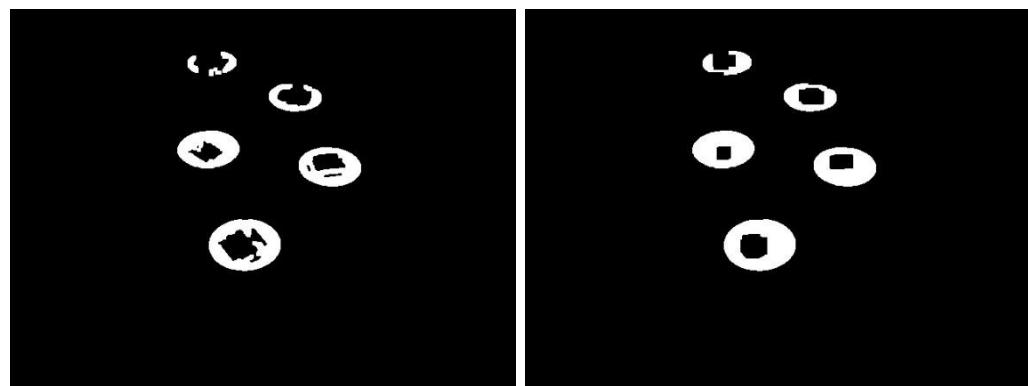
**Figura 109 – Soma das imagens limiarizadas**



Fonte: Autor deste trabalho.

O processo seguinte utiliza a morfologia matemática para melhorar a imagem limiarizada eliminando ruídos indevidos e fechando buracos gerados por não entrar na faixa do limiar, como mostra a resposta na imagem a seguir. As técnicas aplicadas foram de abertura e fechamento morfológico.

**Figura 110 – Respostas do processamento morfológico de abertura (2 iterações) e fechamento (7 iterações)**



Fonte: Autor deste trabalho.

Aplicando a aproximação poligonal então é possível detectar os objetos contidos na imagem como mostra a figura 111.

**Figura 111 – Respostas da detecção dos objetos de rastreamento**



Fonte: Autor deste trabalho.

#### **4.2.4.4 Câmera 2 - Extração de características dos objetos**

Nesta parte de rastreamento das peças a informação necessária a ser adquirida é da coordenada em relação a imagem, esta é adquirida ao se aplicar a aproximação poligonal e obter o ponto central de cada objeto, a partir disto, cálculos devem ser realizados para fornecer a real coordenada do objeto em milímetros. As demais informações sobre determinada peça já provem da inspeção realizada.

#### **4.2.4.5 Câmera 2 - Reconhecimento e interpretação das informações**

Como as coordenadas adquiridas em pixels da imagem 2D não correspondem a coordenada real, pois as peças estão em um plano inclinado em relação a câmera, não é possível determinar uma simples relação dos pixels para milímetros em ambos sentidos ( $x,y$ ), pois os valores variam conforme muda se a linha em que o objeto se encontra na imagem.

A distância em  $x$  foi determinada pela regra de três onde, sabendo que a largura da esteira ou entre as extremidades das linhas de referência é  $Lmm$  ( $mm$ ) e que esta mesma corresponde na linha em que o ponto central do objeto se encontra à  $Lp$  ( $pixels$ ), pode-se determinar pela diferença entre o ponto central ( $Point(x,y)$ ) e o ponto limite da linha de referência ou o recuo da mesma uma relação, conforme mostram as equações 50 e 51.

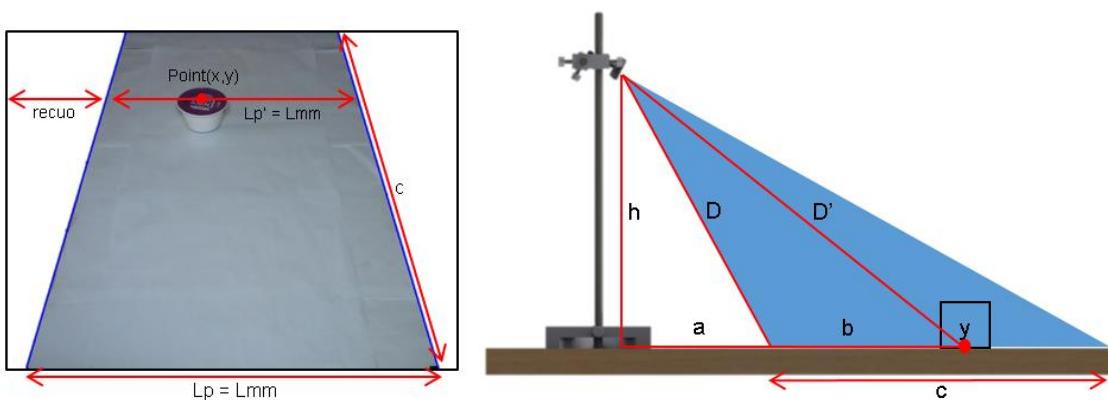
$$\frac{Point.x - recuo}{L_p} \text{ (pixels)} = \frac{x}{L_{mm}} \text{ (mm)} \quad (50)$$

$$x = \frac{(Point.x - recuo).L_{mm}}{L_p} \text{ (mm)} \quad (51)$$

Para estimar a coordenada correta em  $y$  foram elaborados cálculos diferentes, baseados na distância focal da câmera e o uso de geometria com similaridade de triângulos. Para a realização dos cálculos foi necessário requisitar a distância em milímetros da câmera em relação a última linha da imagem capturada, também a altura  $h$  da câmera em relação a esteira e a distância real  $c$  que a câmera consegue capturar da esteira, estes dados devem ser informados pelo usuário.

A distância focal  $F$  pode ser definida pelas regras de óptica sobre lentes delgadas, onde a altura do objeto será substituída pela largura de determinada referência, esta largura será a medida da esteira, em pixels  $L_p$  e em milímetros  $L_{mm}$ , na última linha da imagem, juntamente com a informação da distância  $D$  informada pelo usuário. Desta forma será possível calcular a ampliação transversal para a nova distância entre a câmera e o objeto  $D'$ , agora com a largura  $L_p'$  da esteira em pixels na linha onde o ponto central do objeto se encontra. A coordenada real será a distância  $b$  encontrada subtraída da distância  $c$ .

**Figura 112 – Referências para cálculo da coordenada real em milímetros**



Fonte: Autor deste trabalho.

$$\frac{F}{L_p} = \frac{D}{L_{mm}} \quad (52)$$

$$F = \frac{L_p \cdot D}{L_{mm}} \quad (53)$$

$$D' = \frac{L_{mm} \cdot F}{L_p'} \quad (54)$$

$$a = \left( \sqrt{D^2 - h^2} \right) \quad (55)$$

$$b = \left( \sqrt{D'^2 - h^2} \right) - a \quad (56)$$

$$y = c - b \quad (57)$$

Como as cápsulas possuem certa altura é necessário efetuar uma correção na distância conforme o objeto se desloca para isso foi criado um multiplicador que pode ser ajustado, e que assim desloca a linha em que o ponto central do objeto está, desta forma é feita a correção da distância percorrida pela coordenada em  $y$ .

Partindo do mesmo princípio para numeração das peças da parte de inspeção no rastreamento as cápsulas são numeradas da mesma forma, porém estas devem seguir a ordem conforme a inspeção realizada e assim demarcar as mesmas com o resultado já obtido. Com isso é possível obter os resultados visualizados na figura 113.

**Figura 113 – Detecção e fornecimento das coordenadas das cápsulas com as marcas da correção em azul**



Fonte: Autor deste trabalho.

#### 4.2.5 Visualização dos resultados

Implementando o sistema com as duas câmeras em funcionamento é possível visualizar alguns resultados. Para avaliar não apenas se foi possível detectar os objetos e atender os objetivos, mas também como está o desempenho do programa de processamento de imagem desenvolvido, foi elaborada uma lógica para a contagem de quadros por segundo (*FPS*).

O algoritmo se baseia na contagem de tempo decorrido entre uma faixa de frames passados, para isso foram utilizadas a função *getTickCount()* que retorna o número de ciclos de clock entre as chamadas desta função, juntamente com a função *getTickFrequency()* que retorna a frequência dos ciclos, ou seja, o número de ciclos por segundo. Para o sistema foi determinada a contagem de 20 quadros como base para o cálculo, ao atingir este valor o mesmo é dividido pelo tempo decorrido.

**Figura 114 – Algoritmo do cálculo de FPS**

```
//Get time to FPS
double time_fps = (double) getTickCount();
...
//PROCESS
...
//Calculate time to FPS
time_fps = ((double) getTickCount() - time_fps)
           /getTickFrequency(); //Elapsed time in ms in a frame
cont_time_fps = cont_time_fps + time_fps; //Counter of total elapsed time
cont_frames++;
if(cont_frames == 20){
    avg_fps = cont_frames/cont_time_fps; //n frames/second
    cont_frames = 0;
    cont_time_fps = 0;
}
```

Fonte: Autor deste trabalho.

Nesta simulação foram registradas 3 cápsulas de café como padrões, são elas: *Chococino Caramel*, *Espresso* e *Caffè Buondiorno*. Estas serviram como base de comparação na parte de inspeção das cápsulas, verificando a cor das peças padrões com as que estão sendo avaliadas, juntamente com o tamanho especificado.

**Figura 115 – Cápsulas padrões registradas para simulação**



Fonte: Autor deste trabalho.

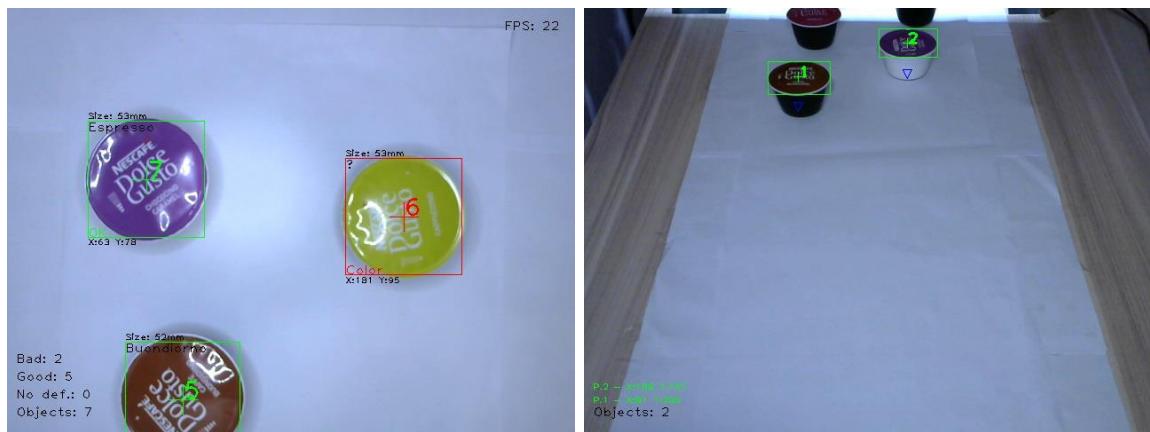
Com isso foram obtidos os resultados das imagens 116, 117 e 118 demonstrando a taxa de quadros por segundo, a contagem de peças, quais estão dentro dos padrões e suas coordenadas.

**Figura 116 – Captura 1 da simulação**



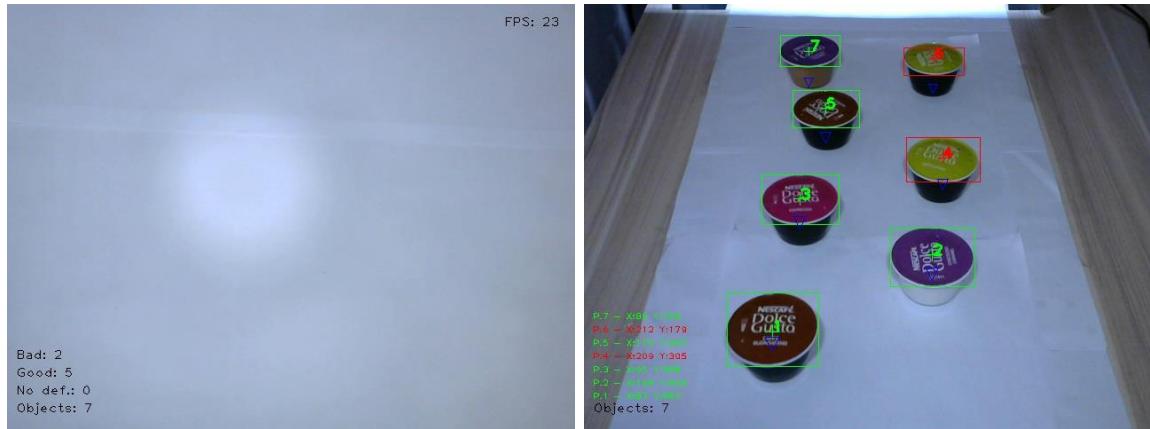
Fonte: Autor deste trabalho.

**Figura 117 – Captura 2 da simulação**



Fonte: Autor deste trabalho.

**Figura 118 – Captura 3 da simulação**



Fonte: Autor deste trabalho.

#### 4.3 Desenvolvimento da interface gráfica

O programa desenvolvido no software CodeBlocks com as lógicas implementadas funcionou corretamente, mas para se obter um ambiente visual mais intuitivo e prático para fins de regulagens do processamento, uma interface gráfica (*GUI* – Interface Gráfica do Usuário) foi criada. O software utilizado para a elaboração da interface foi o Qt Creator na modalidade *open source* através da licença<sup>18</sup> *LGPL* de utilização. Nesta versão de código-aberto o desenvolvedor deve disponibilizar o código para acesso livre, entre outros requisitos da licença. Como no caso deste trabalho o mesmo não será disponibilizado como produto final e sim servindo à estudos acadêmicos então o programa nesta licença é a melhor opção.

O Qt Creator é um ambiente de desenvolvimento com várias funções para criação de softwares e aplicativos de maneira fácil e intuitiva para diversas plataformas como computadores, dispositivos móveis e sistemas embarcados. Possui tanto a parte para desenvolvimento do código quanto a elaboração dos objetos de uma interface de usuário.

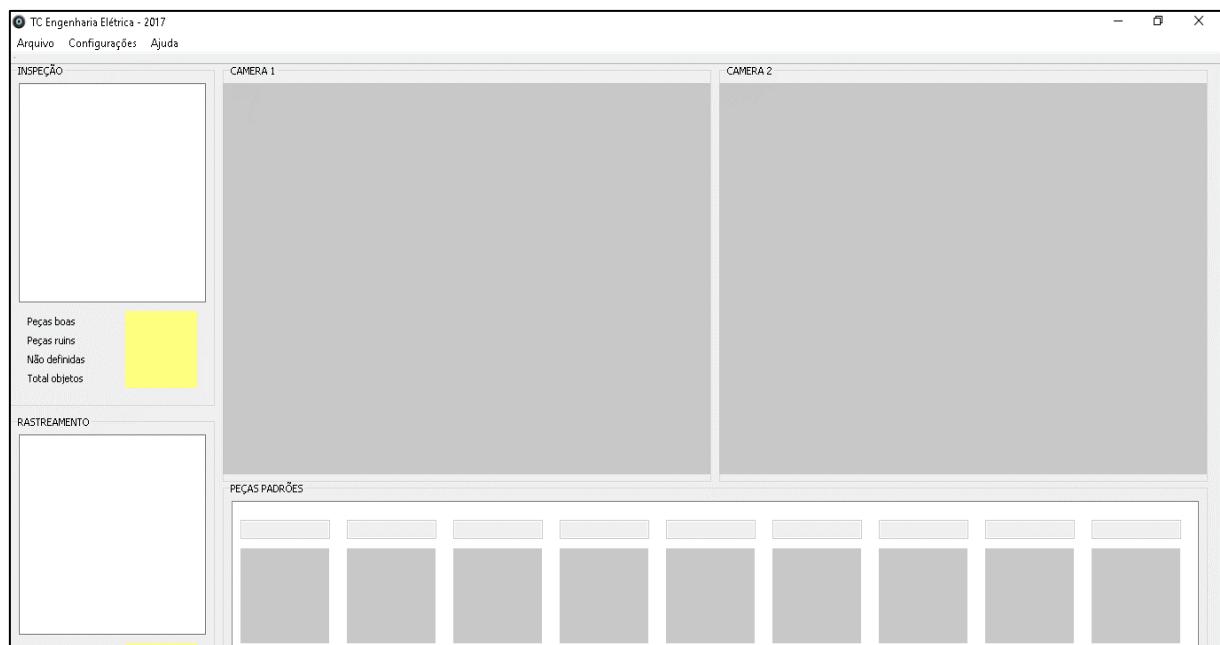
Para unir o código já desenvolvido à interface criada foi necessário fazer algumas adaptações e conversão de alguns tipos de variáveis para incorporar nas telas desenvolvidas no *Qt Creator*. Na tela principal é possível visualizar todo o processamento, os resultados e acessar as demais opções, como é possível visualizar

---

<sup>18</sup> GNU Lesser General Public License

na figura 119, ao se iniciar, sem ter feito nenhuma configuração e não ter nenhuma base de padrões para comparação os quadros permanecem sem informação.

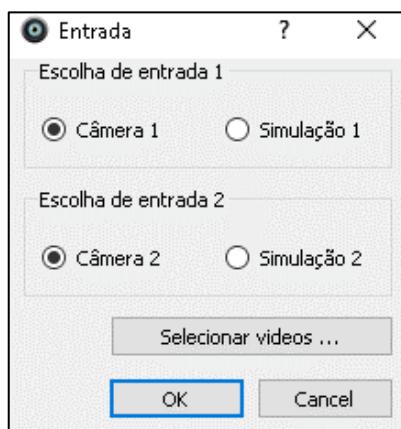
**Figura 119 – Tela inicial da GUI**



Fonte: Autor deste trabalho.

Para dar início ao processamento basta clicar na aba “Arquivo” e após em “Iniciar” que a janela seguinte irá abrir, esta serve para escolher o método de entrada, através das duas câmeras ou para fins de simulação as outras opções que utilizam vídeo pré-gravado para simular o funcionamento.

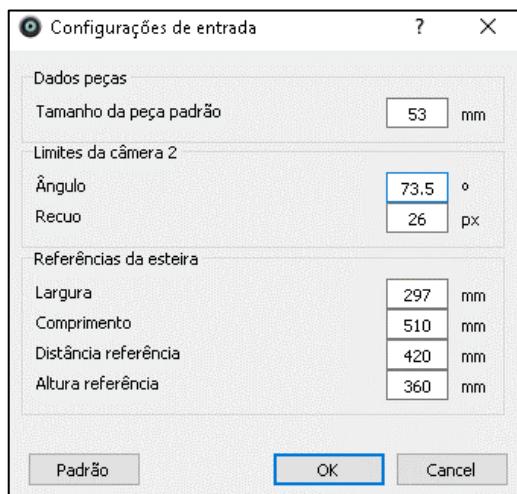
**Figura 120 – Janela da seleção do método de entrada**



Fonte: Autor deste trabalho.

Uma das configurações necessárias para que os resultados tanto da inspeção quanto do rastreamento são as configurações de entrada como mostra a figura 121 da janela desenvolvida, assim requisitando alguns valores como tamanho padrão das cápsulas, informações para calibração dos limites e angulação da câmera 2 e medidas de referência entre a câmera 2 e a esteira.

**Figura 121 – Janela das configurações de entrada**

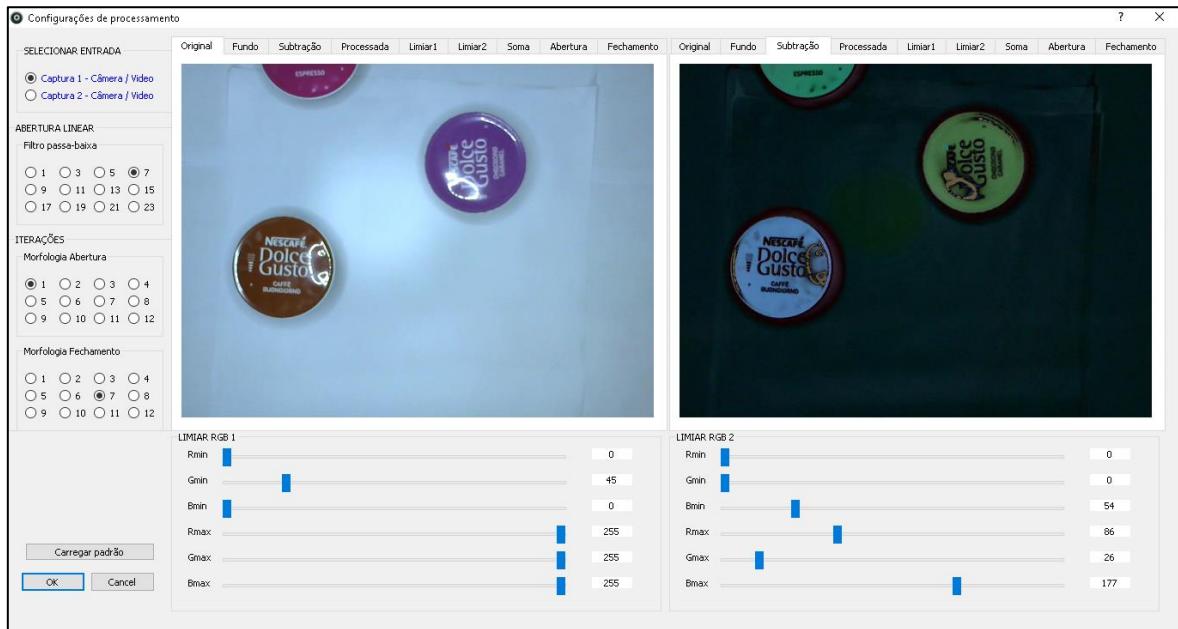


Fonte: Autor deste trabalho.

Outras configurações muito importantes são dos valores do processamento de imagem de cada entrada, para isso foi desenvolvida uma tela para regular e ao mesmo tempo simular as mudanças para desta forma ajustar as configurações para obter uma melhor segmentação dos objetos.

Como mostra a figura 122, a esquerda estão o meio de entrada a ser configurado e os valores para seleção que serão atribuídas as funções do filtro passa-baixa e dos processos morfológicos. Na parte de baixo da tela estão disponíveis duas áreas para regulagem dos níveis RGB utilizados na aplicação das duas etapas de limiarização por cor. Para uma melhor visualização foram dispostas duas áreas onde são exibidas as imagens selecionadas conforme o processo, assim é possível comparar uma imagem com a outra. As abas destes processos estão organizadas na ordem do processamento o que resume qual a lógica utilizada.

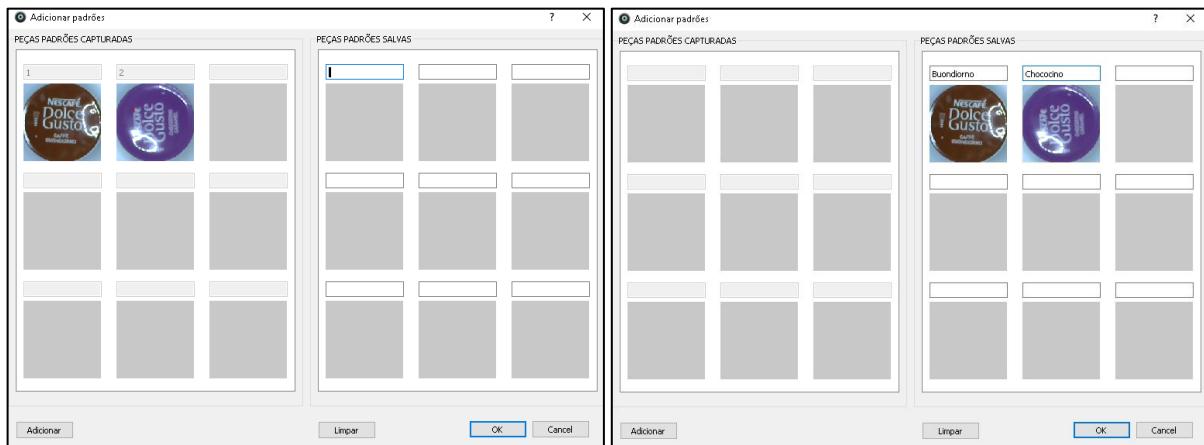
**Figura 122 – Tela das configurações de processamento**



Fonte: Autor deste trabalho.

Após configuradas estas informações importantes já é possível através da detecção correta dos objetos adicionar os padrões à base de dados e salvar. Para elaborar esta ação de maneira intuitiva a tela da figura 123 foi desenvolvida, onde do lado esquerdo estão dispostas as peças capturadas da imagem atual exibida e do lado direito ficam as peças padrões salvias ou adicionadas.

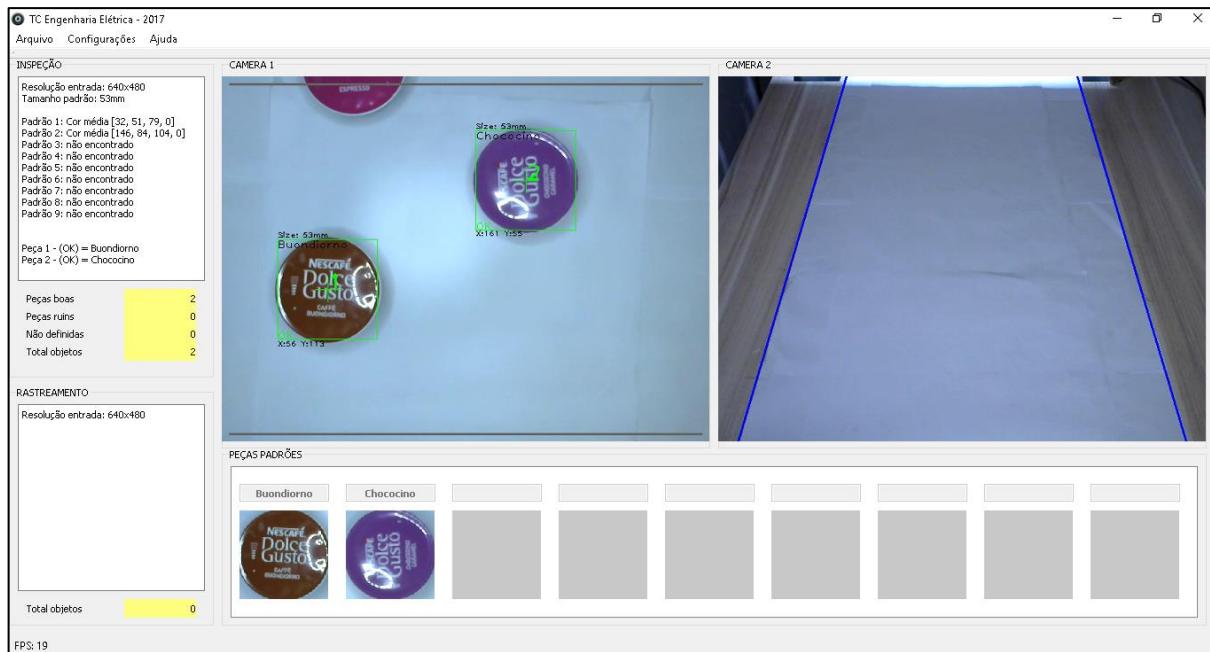
**Figura 123 – Janela para adicionar padrões**



Fonte: Autor deste trabalho.

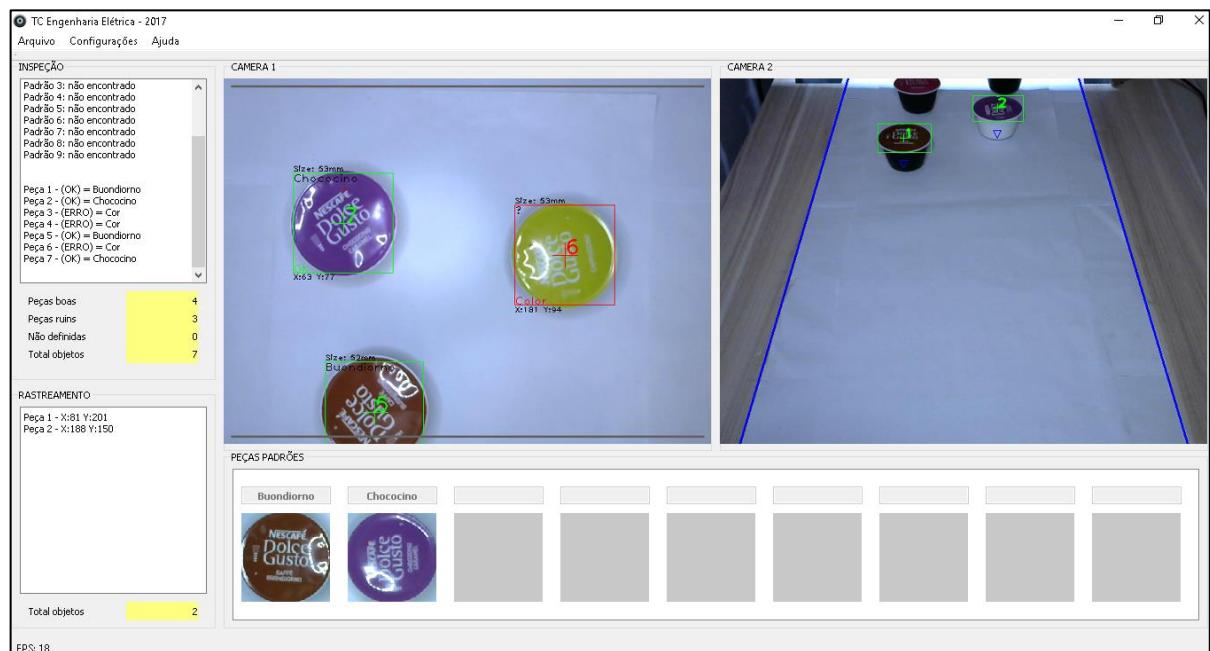
Com isso o processamento das imagens pode ser executado e desta forma obter os resultados da inspeção, como mostra a figura abaixo e também as coordenadas das peças da câmera 2 como mostra a figura 125 no canto inferior.

**Figura 124 – Tela da simulação do processamento**



Fonte: Autor deste trabalho.

**Figura 125 – Tela da simulação do processamento seguinte**



Fonte: Autor deste trabalho.

#### 4.4 Resultados obtidos e conclusões

Através dos algoritmos e técnicas empregadas obteve-se resultado satisfatório para algumas das situações, sendo analisado apenas se atendia os objetivos iniciais da aplicação.

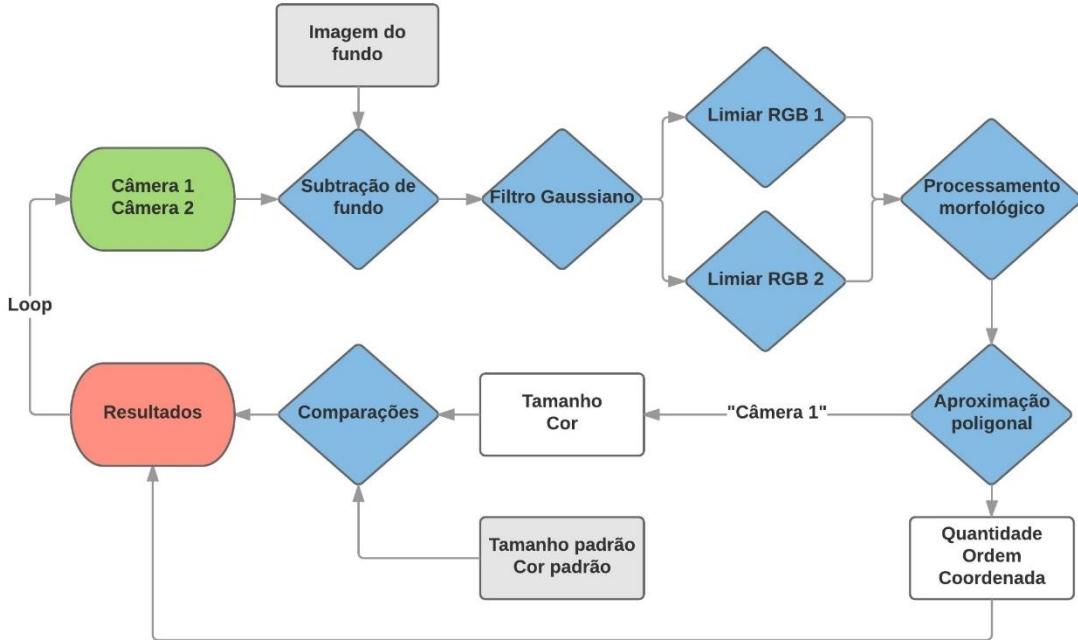
Os filtros passa-baixa aplicados operaram de forma correta, sendo que para a necessidade de remover ruídos presentes não foi preciso suavizar com máscaras de grande vizinhança, portanto em sua maioria os filtros se comportaram de forma semelhante. No primeiro algoritmo executado, com a segmentação de filtro passa-baixa e passa-alta de Sobel aplicou-se um limiar de valor determinado onde este conseguiu segmentar bem os objetos do fundo, facilitando a detecção por aproximação poligonal, porém com a ação do filtro passa-alta geraram-se alguns falsos positivos pelas bordas excedentes referentes da impressão no selo das cápsulas. Outro problema foi a instabilidade e algumas falhas na execução do programa quando se variava o valor de limiar.

No segundo algoritmo removendo então este processo de aplicação de filtro passa-alta para realce de bordas e testando também a limiarização pelo método de Otsu notou-se uma melhora no sistema, sem falsos positivos e sem erros na execução do programa. Assim foi possível identificar a presença das quatro cápsulas na imagem e representar o resultado.

O terceiro método aplicado se deu pela utilização da Transformada de Hough sob a imagem original suavizada, este, porém apresentou lentidão no processamento e variações na detecção dos objetos, onde em certos momentos captava quantidades diferentes de objetos detectados. Em relação a lentidão, com um filtro Gaussiano aplicado houve uma resposta mais rápida do que os outros.

Após testes realizados com o algoritmo de subtração de fundo notou-se que este juntamente com o processamento morfológico apresentou melhorias na segmentação dos objetos e desta forma se concluiu qual o melhor método a ser aplicado, seguindo o fluxograma 126 da lógica do programa desenvolvido foi possível obter resultados precisos que permitiram obter as informações de forma correta.

**Fluxograma 126 – Lógica do processamento das imagens do sistema**



Fonte: Autor deste trabalho.

No entanto durante as simulações foram notadas variações na numeração das peças e na inspeção de algumas, notou-se que para prever tantas variações e garantir a confiabilidade do sistema é necessário efetuar melhorias que aprimorem a precisão na detecção dos objetos. O desempenho apresentado em uma média de 20 quadros por segundo é aceitável sendo possível captar as informações sem distorções na imagem, porém irá depender da velocidade real em que a esteira estiver funcionando, em baixa rotação o sistema funciona corretamente.

#### 4.5 Oportunidades de melhoria

O sistema como protótipo para comprovação dos objetivos iniciais realizou as tarefas conforme a lógica aplicada com sucesso, porém se os objetivos futuros forem de realmente aplicar em meio industrial, deverá fornecer mais confiabilidade e desempenho, portanto algumas melhorias futuras são sugeridas.

Uma análise mais aprofundada sobre outros métodos de processamento de imagem, para melhorar a parte de segmentação dos objetos é uma necessidade de melhoria, assim como aperfeiçoar o código da programação elaborado para obter

melhor desempenho. Algumas destas melhorias no programa podem ser buscadas em bibliografias e estudos sobre a biblioteca CUDA da Nvidia e uso do TBB<sup>19</sup> da Intel, para deixar o programa mais eficiente e com melhor desempenho gráfico, trabalhando com multiprocessadores. Outra parte relacionada a programação seria acrescentar funções ao código para gerenciamento e controle estatístico do processo, assim como salvar relatórios dos resultados obtidos, estes são itens que podem valorizar a necessidade do uso do sistema desenvolvido.

A utilização de hardware não específico para visão computacional pode ter limitado também o desempenho e qualidade dos resultados finais, para isso seriam necessárias câmeras industriais que possuam interface de comunicação como USB3 Vision ou GigE, assim como uma CPU com GPU para trabalhos gráficos.

---

<sup>19</sup> Intel® Threading Building Blocks

## REFERÊNCIAS

- ALMEIDA, T.; MENESSES, P. R. *Introdução ao processamento de imagens de sensoriamento remoto*. Brasília: Universidade de Brasília, 2012.
- AMARAL, V. M.; DIAGO, R. *Eletrônica: eletrônica digital*. São Paulo: Fundação Padre Anchieta, 2011.
- ASCENCIO, A. F.; CAMPOS, E. A. *Fundamentos da programação de computadores: algoritmos, Pascal, C/C++ (padrão ANSI) e Java*. 3. ed. São Paulo: Pearson, 2012. 569 p.
- ASSOCIAÇÃO BRASILEIRA DE NORMAS TÉCNICAS. *NBR ISSO/CIE 8995-1: iluminação de ambientes de trabalho: parte 1: interior*. Rio de Janeiro, 2013.
- BAXES, Gregory A. *Digital image processing: principles and applications*. Nova Iorque: John Wiley & Sons, 1994. 452 p.
- BHATTACHARYYA, S. S.; CHAI, S.; KISACANIN, B. *Embedded computer vision: advances in pattern recognition*. Londres: Springer, 2009.
- BLUM, Richard. *Professional Assembly language*. Indianápolis: Wiley Publishing, 2005.
- BORGES, Luiz E. *Python para desenvolvedores*. 3. ed. Rio de Janeiro: Edição do Autor, 2010. Disponível em: <<https://ark4n.wordpress.com/python>>. Acesso em: 15 out. 2017.
- BOYLE, R.; HLAVAC, V.; SONKA, M. *Image processing, analysis and machine vision*. 3. ed. Toronto: Thomson Learning, 2008.
- BRADSKI, G.; KAEHLER, A. *Learning OpenCV*. Sebastopol, CA, EUA: O'Reilly Media, 2008. Disponível em: <<http://www-cs.ccny.cuny.edu/~wolberg/capstone/opencv/LearningOpenCV.pdf>>. Acesso em: 01 nov. 2017.
- CONFERÊNCIA SOBRE VISÃO COMPUTACIONAL, 1998, Bombaim. MANDUCHI, R.; TOMASI, C. *Bilateral filtering for gray and color images*. Bombaim, Índia: 1998.
- COSTA JUNIOR, Eudes L. *Administração da produção: gestão em processos produtivos*. Curitiba: InterSaberes, 2012. 159 p.
- DAVIES, E. Roy. *Computer & machine vision: Theory, algorithms and practicalities*. 4. ed. Waltham: Elsevier, 2012. 912 p.
- DEITEL, H.; DEITEL, P. C: como programar. 6. ed. São Paulo: Pearson, 2011. 818 p.
- DEITEL, H.; DEITEL, P. Java: como programar. 8. ed. São Paulo: Pearson, 2010. 1144 p.

EUROPEAN MACHINE VISION ASSOCIATION (EMVA). *Global machine vision: interface standards*. [S.I.]: AIA, EMVA, JIIA, 2016. Disponível em: <[http://www.emva.org/wp-content/uploads/FSF\\_Vision\\_Standards\\_Brochure\\_A4\\_screen.pdf](http://www.emva.org/wp-content/uploads/FSF_Vision_Standards_Brochure_A4_screen.pdf)>. Acesso em: 15 out. 2017.

FARIA, Alessandro O. *Visão computacional para todos*. *Linux Magazine*, [S.I.], [ca. 2013]. 58-64 p. Disponível em:<[http://www.linuxnewmedia.com.br/images/uploads/pdf\\_aberto/LM\\_89\\_58\\_65\\_08\\_ana-cabelo.pdf](http://www.linuxnewmedia.com.br/images/uploads/pdf_aberto/LM_89_58_65_08_ana-cabelo.pdf)>. Acesso em: 15 out. 2017.

FINTEL, René von. *Comparison of the most common digital interface technologies in vision technology*. Ahrensburg, Alemanha: Basler AG, 2013. Disponível em: <[http://www.i2s-vision.fr/upload/BAS1303\\_White\\_Paper\\_Interface\\_Comparsion\\_e.pdf](http://www.i2s-vision.fr/upload/BAS1303_White_Paper_Interface_Comparsion_e.pdf)>. Acesso em: 28 out. 2017.

FORSYTH, D. A.; PONCE, J. *Computer vision: A modern approach*. [S.I.]: Pearson, 2003.

FREEDMAN, R. A.; YOUNG, H. D. *Física IV: ótica e física moderna*. 14. ed. São Paulo: Pearson, 2016. 534 p.

FYKSE, Egil. *Performance comparison of GPU, DSP and FPGA implementations of image processing and computer vision algorithms in embedded systems*. 2013. Tese (Graduação em Engenharia Elétrica) – Universidade de Ciência e Tecnologia da Noruega, Trondheim, 2013.

GERBRANDS, J. J.; YOUNG, I. T.; VAN VLIET, L. J. *Fundamentals of image processing*. Delft, Holanda: Universidade Técnica de Delft, 1998.

GIMENEZ, Salvador P. *Microcontroladores 8081: Teoria do hardware e do software, aplicações em controle digital, laboratório e simulação*. São Paulo: Pearson, 2002.

GONZALEZ, R. C.; WOODS, R. E. *Processamento digital de imagens*. 3. ed. São Paulo: Pearson, 2010. 624 p.

GROOVER, Mikell P. *Automação Industrial e sistemas de manufatura*. 3. ed. São Paulo: Pearson, 2011. 582 p.

JÄHNE, Bernd. *Digital image processing*. 5. ed. Berlim; Heidelberg; Nova Iorque: Springer, 2002.

JÄHNE, B.; HAUBECKER, H. *Computer vision and applications: A guide for students and practitioners*. San Diego, USA: Academic Press; London, UK: Academic Press, 2000.

LEGAT, J.; MALAMAS, E. N.; PETIT, L.; PETRAKIS, E. G.; ZERVAKIS, M. *A survey on industrial vision systems, applications and tools*. [S.I.]: Elsevier, 2003. 38 p.

MARQUES FILHO, O.; VIEIRA NETO, H. *Processamento digital de imagens*. Rio de Janeiro: Brasport, 1999.

MARR, David. *Vision: a computacional investigation into human representation and processing of visual information*. San Francisco: W. H. Freeman and Company, 1982.

MOSS, G. L.; TOCCI, R. J.; WIDMER, N. S. *Sistemas digitais: princípios e aplicações*. 11. ed. São Paulo: Pearson, 2011. 819 p.

NESCAFÉ DOLCE GUSTO. *Sobre Nescafé Dolce Gusto*. Disponível em: <<https://www.nescafe-dolcegusto.com.br/m/about-nescafe-dolce-gusto>>. Acesso em: 23 nov. 2017.

OPENCV. *The OpenCV reference manual: release 3.0.0-dev*. [S.I.: s.n.], 2014. 899 p. Disponível em: <<http://docs.opencv.org/3.0-beta/opencv2refman.pdf>>. Acesso em: 01 nov. 2017.

PARANHOS FILHO, Moacyr. *Administração da produção: gestão da produção industrial*. Curitiba: InterSaberes, 2012. 340 p.

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

PRATT, William K. *Digital image processing*. 3. ed. Nova Iorque: John Wiley & Sons, 2001.

PRINCE, Simon J. D. *Computer vision: models, learning and inference*. [S.I.]: Cambridge University, 2012.

ROSÁRIO, Maurício J. *Princípios de mecatrônica*. São Paulo: Pearson, 2005.

SELEME, R.; SELEME, R. B. *Automação da produção: uma abordagem gerencial*. 2. ed. Curitiba: Ibpea, 2012.

SHAPIRO, L. G.; STOCKMAN, G. C. *Computer vision*. [S.I.]: Pearson, 2001.

SZELISKI, Richard. *Computer vision: algorithms and applications*. Londres: Springer, 2010. Disponível em: <<http://szeliski.org/Book>>. Acesso em: 01 nov. 2017.

TELEDYNE DALSA. *Application note: Line scan imaging basics*. [S.I.]: Dalsa, [ca. 2016]. 11 p. Disponível em: <[https://www.teledynedalsa.com/public/mv/appnotes/00541-00\\_03-32\\_Linescan\\_Imaging\\_Basics.pdf](https://www.teledynedalsa.com/public/mv/appnotes/00541-00_03-32_Linescan_Imaging_Basics.pdf)>. Acesso em: 03 out. 2017.

TEXAS INSTRUMENTS. *Choosing the right architecture for real-time signal processing designs*. Dallas: Texas Instruments Incorporated, 2002. 11 p. Disponível em: <<http://www.tij.co.jp/jp/lit/wp/spra879/spra879.pdf>>. Acesso em: 03 out. 2017.

## APÊNDICE A – Tabela de softwares

**Tabela 7 – Lista de alguns softwares, bibliotecas, APIs e suas aplicações**

Softwares	Aplicação
3DSlicer	Visualização científica e análise de imagens médicas.
Accord.NET	Algebra linear, estatística, redes neurais artificiais e aprendizagem de máquina.
AForge.NET	Redes neurais artificiais, genética, lógica Fuzzy, aprendizagem de máquina e robótica.
Amira (software)	Visualização científica de dados 3D e 4D, análise de dados.
ANIMAL (image processing)	Prototipagem rápida, testes e modificação de algoritmos.
Aphelion (software)	Processamento e análise de imagem em aplicações diversas.
AutoCollage 2008	Detecção de rosto e reconhecimento de objetos.
Avizo (software)	Visualização e análise de dados científicos e industriais.
AVM Navigator	Reconhecimento de objetos e robôs autônomos.
CellCognition	Análise quantitativa de imagens biomédicas e sistemas de microscopia.
Dlib	Interfaces gráficas, estruturas de dados, álgebra linear, aprendizagem de máquina e processamento de imagens.
Fiji (software)	Análise de imagem científica em geral ou ciências da vida.
GemIdent	Processamento de imagens e reconhecimento de objetos.
GIMIAS	Simulação e processamento de imagens biomédicas.
Ginkgo CADx	Visualização e processamento de imagens científicas.
Ilastik	Classificação e segmentação de imagem, aprendizagem de máquina.
ILNumerics	Implementação de algoritmos e interface gráfica para matemática.
ImageNets	Prototipagem rápida para uso em visão de máquina.
Integrating Vision Toolkit	Processamento de imagens e calibração de câmeras.
Intel RealSense	Reconhecimento de gestos humanos, realidade aumentada, processamento de som.
MATLAB	Manipulação de dados matemáticos, implementação de algoritmos, criação de interface gráfica.
MeVisLab	Visualização científica e processamento de imagens médicas.
Mocolo	Monitora e analisa fluxo em vídeos.
OpenCV	Processamento de imagem e visão computacional em tempo real.
Pfinder	Reconhecimento de gestos e figuras humanas em tempo real.
RapidMiner	Aprendizagem de máquina, extração de dados e textos, análise de negócios.
Recursive Cortical Network	Aprendizagem de máquina baseada no cérebro humano.
Scilab Image Processing	Processamento de imagens.
SigmaScan	Análise de imagem para extração de medidas.
VIGRA	Processamento de imagens, aprendizagem de máquina e ferramentas matemáticas.
VTK	Visualização científica e computação gráfica 3D.
VXL	Processamento de imagens, matemática, computação científica.
YaDICs	Correlação de imagens tomográficas digitais 2D e 3D de imagens

Fonte: disponível em: <[https://en.wikipedia.org/wiki/Category:Computer\\_vision\\_software](https://en.wikipedia.org/wiki/Category:Computer_vision_software)>. Editado pelo autor.

## ANEXO A – Algoritmo da transformada de Hough generalizada

```

1 // Criação da tabela-R
2 Criar uma tabela com k entradas, cada uma delas indexada por um
ângulo  $\theta_i$ ,  $i = 1, 2, \dots, k$ , que varia de 0 a 180 graus com
incremento de  $180/k$ , em que k é a resolução da orientação do
gradiente.
3
4 Escolher um ponto de referência  $(x_c, y_c)$  no interior do objeto
5
6 Para cada ponto  $(x, y)$  pertencente à borda do objeto, calcular
7  $r \leftarrow \sqrt{(x - x_c)^2 + (y - y_c)^2}$ 
8  $\alpha \leftarrow \arctan((y - y_c)/(x - x_c))$ 
9  $\theta$ : direção do gradiente
10
11 Adicionar o par  $(r, \alpha)$  na entrada da tabela-R com  $\theta_i$  mais próximo de
 $\theta$ .
12
13 // Detecção da forma do objeto
14 Quantizar o espaço de parâmetros com uma matriz de acumulação,
 $M(x_{c_{\min}} \dots x_{c_{\max}}, y_{c_{\min}} \dots y_{c_{\max}}, S_{\min} \dots S_{\max}, \phi_{\min} \dots \phi_{\max})$ ,
inicializada com zero.
15
16 Para cada ponto  $(x, y)$  da borda
17 Recuperar todos os pares  $(r, \alpha)$  da tabela-R cuja entrada  $\theta_i$  esteja
próxima do valor do gradiente no ponto  $(x, y)$ 
18 Para cada par  $(r, \alpha)$ , calcular os pontos de referência
candidatos
19 para  $\phi = \phi_{\min}$  até  $\phi = \phi_{\max}$  faça
20     para  $S = S_{\min}$  até  $S_{\max}$  faça
21          $x_c \leftarrow x + r \cos(\alpha + \phi)$ 
22          $y_c \leftarrow y + r \sin(\alpha + \phi)$ 
23          $M(x_c, y_c, S, \phi) \leftarrow M(x_c, y_c, S, \phi) + 1$ 
24
25 // Identificação das possíveis localizações da forma do objeto
26 Se  $M(x_c, y_c, S, \phi) > T$ , o contorno do objeto é localizado no ponto
 $(x_c, y_c)$ , com respeito ao fator de escala S e ao ângulo de
rotação  $\phi$ .

```

Fonte: PEDRINI; SCHWARTZ, 2008, p. 182. Editado pelo autor.