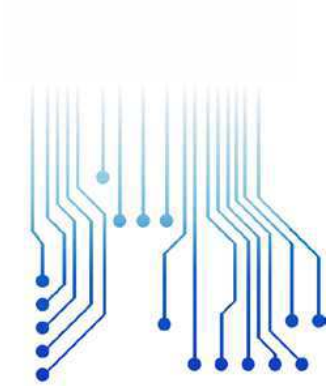


CURSO DE GRADUAÇÃO EM ENGENHARIA ELÉTRICA

Universidade Federal
de Campina Grande

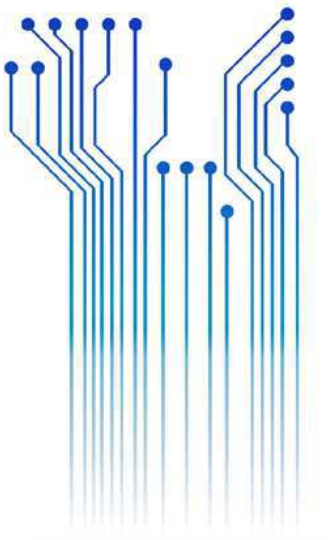
HIGOR ITALO DOS SANTOS

Centro de Engenharia
Elétrica e Informática

Departamento de
Engenharia Elétrica

Trabalho de Conclusão de Curso

Sistema de detecção de parafusos em placas de circuitos eletrônicos utilizando técnicas de visão computacional



CAMPINA GRANDE
2017

HIGOR ITALO DOS SANTOS

SISTEMA DE DETECÇÃO DE PARAFUSOS EM PLACAS DE CIRCUITOS ELETRÔNICOS UTILIZANDO TÉCNICAS DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso submetido à Coordenação do curso de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Orientador:
Professor Edmar Candeia Gurjão, D. Sc.

Campina Grande, Paraíba
Setembro de 2017

HIGOR ITALO DOS SANTOS

SISTEMA DE DETECÇÃO DE PARAFUSOS EM PLACAS DE CIRCUITOS ELETRÔNICOS UTILIZANDO TÉCNICAS DE VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso submetido à Coordenação do curso de Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Aprovado em / /

Professor Avaliador
Universidade Federal de Campina Grande
Avaliador

Professor Edmar Candeia Gurjão, D. Sc.
Universidade Federal de Campina Grande
Orientador, UFCG

Dedico este trabalho a Deus e todos aqueles que de alguma forma estiveram ao meu lado durante toda a jornada deste curso, me apoiando e torcendo pela minha formação.

AGRADECIMENTOS

Agradeço a essa Instituição, em primeiro lugar, pela minha acolhida e pelas condições oferecidas, que me permitiram concluir este trabalho.

Agradeço também à minha mãe, Inácia, a meu pai, Adilson, por ter se esforçado tanto para me proporcionar uma boa educação, por ter me alimentado com saúde, força e coragem, as quais que foram essenciais para superação de tantas adversidades ao longo desta caminhada.

Agradeço também a toda minha família e amigos, que com todo carinho e apoio, não mediram esforços para eu chegar a esta etapa da minha vida.

À minha irmã, Amanda, que compartilhou comigo os momentos mais difíceis de nossas vidas e teve garra e coragem para supera-los, podendo assim me dar suporte para chegar até aqui.

Gabriela Chagas, minha amada, que soube me dar forças nos momentos mais difíceis e compartilhar comigo grandes momentos desta jornada.

Agradeço ao meu orientador, Prof. Dr. Edmar Candeia, por aceitar me orientar e por toda ajuda e atenção prestada, bem como por todo apoio e bom humor oferecido durante todo o decorrer deste trabalho.

Agradeço a professora Luciana Veloso, que com sua experiência e conhecimento na área de processamento digital de imagens, concedeu-me valiosas sugestões sobre pontos importantes deste trabalho.

Aos colegas que compartilharam a convivência e utilização do laboratório, pelos conhecimentos compartilhados nas dificuldades surgidas durante a elaboração dos trabalhos desenvolvidos e por bons momentos compartilhados.

Enfim, agradeço a todos que de alguma forma, passaram pela minha vida e contribuíram para a construção de quem sou hoje.

"Olhos que olham são comuns.
Olhos que veem, são raros."

J. Oswald Sanders

RESUMO

Visão computacional é o estudo da extração de informações de uma imagem, a construção de descrições explicitas e claras dos objetos focando na obtenção e manipulação de dados dela oriundos para fins diversos. Também permite o estudo de formas de extração de informações provenientes da imagem, tais como formas de objetos nela contidos, por exemplo. Algoritmos que buscam simular o processo de visão da mesma forma como o sistema humano, a visão computacional surge como ferramenta auxiliar à ações realizadas pelo homem, até mesmo tarefas repetitivas, como o desmonte de equipamentos pela retirada de parafusos. Neste trabalho, propõe-se a utilização de visão computacional para a implementação de um sistema que faça a detecção e localização de parafusos em placas de circuitos eletrônicos.

Palavras-chave: visão computacional, sistema, localização, detecção, parafusos.

ABSTRACT

Computational vision is the study of the extraction of information from an image, the construction of explicit and clear descriptions of the objects focusing on obtaining and manipulating data from it for different purposes. It also allows the study of ways of extracting information from the image, such as forms of objects contained therein, for example. Algorithms that seek to simulate the vision process in the same way as the human system, the computational vision emerges as an auxiliary tool to the actions performed by the man, even repetitive tasks, such as the dismantling of equipment by the removal of screws. In this work, it is proposed the use of computer vision for the implementation of a system that detects and locates screws on electronic circuit boards.

Keywords: computer vision, system, location, detection, screws.

LISTA DE ILUSTRAÇÕES

Figura 1. Matriz de níveis de intensidade associada a uma imagem em níveis de cinza. 18
Figura 2. Representação das matrizes que compõe uma imagem no padrão RGB. 18
Figura 3. Representação de uma imagem colorida utilizando padrão RGB. 19
Figura 4. Exemplo de uma imagem e histograma associado. 21
Figura 5. Resultado da aplicação de equalização de histograma. 22
Figura 6. Efeito da Binarização sobre uma imagem. 23
Figura 7. Representação ilustrativa de um kernel. 23
Figura 8. Efeito de suavização sobre uma imagem. 24
Figura 9. Efeito de segmentação sobre uma imagem. 25
Figura 10. Resultado da detecção de círculos numa imagem através da Transformada Circular de Hough. 29
Figura 11. Aplicação de Template Matching: imagem original (A); template (B); resultado de busca do template na imagem original (C). 29
Figura 12. Fonte de bancada utilizada para criação das imagens utilizadas no trabalho. 32
Figura 13. Imagem da face frontal da fonte de bancada. 33
Figura 14. Imagem da face lateral da fonte de bancada. 33
Figura 15. Trecho do código que realiza adição de bibliotecas e aquisição da imagem. 34
Figura 16. Trecho do código que realiza cópias redimensionadas da imagem original 35
Figura 17. Trecho do código que realiza aplicação de mascara de pixels em preto a imagem da face frontal da fonte. 36
Figura 18. Trecho do código que realiza aplicação de mascara de pixels em preto a imagem da face lateral da fonte. 36
Figura 19. Trecho do código que realiza conversão do arquivo de imagem de RGB para níveis de cinza. 37
Figura 20. Trecho do código que realiza equalização de histograma de uma imagem associada. 38
Figura 21. Trecho do código que realiza suavização sobre uma imagem associada. 38
Figura 22. Trecho do código que realiza detecção de bordas via detector Canny. 39
Figura 23. Trecho do código da versão 1 do programa que realiza detecção de bordas via detector Canny. 40
Figura 24. Trecho do código que realiza a contabilização e sinalização dos parafusos na imagem. 41
Figura 25. Template utilizado para detecção de parafusos da face frontal da fonte de bancada. 42
Figura 26. Templates utilizados para detecção de parafusos da face lateral da fonte de bancada. 42
Figura 27. Carregamento e conversão para níveis de cinza da imagem modelo. 42
Figura 28. Trecho do código que realiza a busca da imagem template numa imagem de interesse. 43
Figura 29. Trecho do programa que armazena e sinaliza locais reconhecidos com presença de parafusos. 44
Figura 30. Remoção de dados redundantes do template matching pelo programa. 45
Figura 31. Trecho do programa que sinaliza e exibe parafusos detectados via uso de template matching. 46
Figura 32. Trecho do código para exibição de resultados intermediários e encerramento do programa. ... 46
Figura 33. Imagem resultante da detecção via detecção de círculos na face frontal da fonte de bancada. . 47
Figura 34. Imagem resultante da detecção de parafusos via template matching na face frontal da fonte de bancada. 48
Figura 35. Visualização da janela gráfica que apresenta as imagens intermediárias formadas durante a execução do programa. 48
Figura 36. Visualização das informações no prompt do programa relacionadas à posição dos parafusos detectados. 49
Figura 37. Imagem resultante da detecção de parafusos via detecção de círculos na face lateral da fonte de bancada. 50
Figura 38. Imagem resultante da detecção de parafusos via template matching na face lateral da fonte de bancada. 50
Figura 39. Visualização da janela gráfica que apresenta as imagens intermediárias formadas durante a execução do programa. 51
Figura 40. Visualização das informações no prompt do programa relacionadas à posição dos parafusos detectados. 51

LISTA DE ABREVIATURAS E SIGLAS

DPI	*Dots Per Inch* (pontos por polegada)
RGB	Sistema de representação de cores para vermelho *(RED)*, verde *(GREEN)* e azul *(BLUE)*
HSV	Sistema de representação de cores para matiz (H - hue), saturação (S - saturation) e valor (V - value)
CMYK	Sistema de representação de cores subtrativas formado por Ciano (C - cyan), Magenta (M- magenta), Amarelo(- yellow) e Preto (K - Black **K**ey)
Lab	Sistema de representação de cores para luminosidade (L), coordenada vermelho/verde (a) e coordenada amarelo/azul (b)
OpenCV	Open Source Computer Vision Library
IPP	Integrated Performance Primitives
MLL	Machine Learning Library
IDE	Integrated Development Enviroment
VCSes,	Version Control Systems
jpg/JPEG	Joint Photographics Experts Group
CLAHE	Contrast Limited Adaptive Histogram Equalization

SUMÁRIO

1 Introdução 14
1.1 Motivação para o trabalho 14
1.2 Objetivos do trabalho 15
1.3 Estrutura do trabalho 15
2 Fundamentação teórica 17
2.1 Imagem digital 17
2.2 Processamento digital de imagens 19
2.2.1 Equalização de Histograma 20
2.2.2 Binarização (Thresholding) 22
2.2.3 Suavização (Smoothing) 23
2.2.4 Segmentação e detecção de bordas 24
2.3 Visão computacional 25
2.3.1 Reconhecimento de padrões 26
2.3.2 Transformada de Hough 27
2.3.2.1 Transformada de Hough Circular 28
2.3.3 Template matching 29
2.4 OpenCV 30
3 Desenvolvimento 31
3.1 Materiais utilizados 31
3.2 Métodos utilizados 32
3.2.1 Aquisição da imagem das placas de circuito pelo sistema 34
3.2.2 Redimensionamento e criação de cópias da imagem original 35
3.2.3 Reconhecimento de parafusos por meio de detecção de círculos 36
3.2.3.1 Conversão da imagem para níveis de cinza 37
3.2.3.2 Equalização histogrâmica 37
3.2.3.3 Aplicação de Suavização 38
3.2.3.4 Detecção de bordas 39
3.2.3.5 Detecção de círculos sobre bordas 39
3.2.3.6 Contabilização e sinalização de parafusos na imagem original 40
3.2.4 Reconhecimento de parafusos por meio do uso de templates 41
3.2.4.1 Fornecimento de templates ao programa 42
3.2.4.2 Busca do template na imagem 43
3.2.4.3 Adoção de limiar de semelhança aos template 43
3.2.4.4 Sinalização dos parafusos identificados por template matching 45
3.2.5 Exibição de resultados intermediários e enceramento do programa 46
4 Resultados 47
4.1 Detecção de parafusos na face frontal da fonte de bancada 47

4.2 Detecção de parafusos na face lateral da fonte de bancada 49
5 Conclusão 52
Bibliografia 54

1 INTRODUÇÃO

O desenvolvimento dos sistemas computacionais na atualidade ocorre de forma continua e intensa. Neste sentido, a manipulação de imagens digitais através de aplicações que envolvem técnicas de visão computacional tem ganhado destaque como ramo promissor em diversas áreas, possibilitando obtenção de informações contidas em uma imagem que possam ser direcionadas para algum fim.

Visão computacional trata da construção de descrições de objetos em uma imagem, com o intuito de se obter e manipular dados e informações nela contidos para uso posterior em finalidades diversas.

O processo de extração de dados a partir de imagens não é uma ação trivial, necessitando por vezes da utilização de técnicas complexas e dados que apresentem bons atributos para se obter resultados satisfatórios em determinação aplicação.

Diante disso, este trabalho tem por objetivo desenvolver um sistema que analise imagens associadas a circuitos eletrônicos e seja capaz de detectar a presença de parafusos utilizando técnicas de visão computacional, indicando a localização de cada um deles.

1.1 MOTIVAÇÃO PARA O TRABALHO

Atualmente verifica-se o uso de visão computacional em sistemas que utilizam algoritmos que buscam simular o processo de visão da mesma forma como o sistema visual humano. Devido a isso é possível utilizar tais sistemas para realização de diversas tarefas comuns ao ser humano, que por vezes podem ser insalubres ou monótonas, seja pela existência de condições adversas ou a sucessiva repetição de ações.

Um exemplo destas atividades inadequadas é o desmonte de equipamentos pela retirada de parafusos, considerando-se a necessidade de esforço físico e repetições de movimentos para esse tipo de atividade. Com isso, temos como motivação para este trabalho a associação do sistema de detecção de parafusos a ser desenvolvido com um aparelho que realize a remoção dos parafusos de formar automática, de tal modo que o

sistema forneça a localização do parafuso ao aparelho para que este possa realizar o desparafusagem.

1.2 OBJETIVOS DO TRABALHO

O objetivo principal deste trabalho é o desenvolvimento de um sistema de detecção de parafusos em placas de circuitos eletrônicos utilizando técnicas de visão computacional, tendo como objetivos específicos:

- Realizar a aquisição da imagem das placas de circuitos;
- Realizar o tratamento da imagem pra posterior manipulação dos dados.
- Efetuar a identificação dos parafusos na placa do circuito eletrônico por meio de técnicas de visão computacional

1.3 ESTRUTURA DO TRABALHO

O primeiro capítulo do trabalho é introdutório e faz a apresentação das motivações e objetivos, deixando claro o objetivo principal de criação de um sistema de detecção de parafusos em placas de circuitos eletrônicos utilizando técnicas de visão computacional, tendo como motivação sua aplicação em equipamentos para desaparafusamento automático.

No Capitulo 2 é feita uma revisão bibliográfica com os principais assuntos que serviram de base para o desenvolvimento do trabalho, a saber, a definição e constituição de imagens digitais, os principais métodos de processamento digital de imagens e de visão computacional utilizados neste trabalho e uma explanação acerca da biblioteca de visão computacional OpenCV.

No Capitulo 3 é feita uma descrição detalhada do desenvolvimento das atividades realizadas neste trabalho, que corresponderam a delimitação de materiais e métodos utilizados, apresentando cada etapa dos processo de aquisição, tratamento de dados e identificação de parafusos através de duas formas de detecção implementadas utilizando técnicas de visão computacional.

No Capitulo 4 são apresentados os resultados gráficos e numéricos obtidos ao final do desenvolvimento do sistema utilizando as duas formas de detecção de parafusos elaboradas.

No Capitulo 5 são apresentadas as principais conclusões das tarefas realizadas, uma análise dos resultados obtidos e os aprendizados teóricos e práticos adquiridos ao final de todas as ações realizadas.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo são apresentados alguns fundamentos teóricos e técnicas fundamentais de processamento de imagens, além de abordar conceitos de visão computacional que serviram de base para produção deste trabalho.

2.1 IMAGEM DIGITAL

Pode-se definir de forma simplificada que uma imagem corresponde a um suporte bi ou tridimensional do qual pode ser efetuar uma troca de informação. Tal informação consta de variáveis, como cores e intensidades de luz, e pode provir de diversas origens: óticas, magnéticas, infravermelhas entre outras.

Para a representação de uma imagem na forma digital faz-se uso de valores numéricos agrupados, com o intuito de mapeá-la de sua essência física com relação às cores – sensações visuais geradas em resposta à luz que incide sobre um sistema visual e diversos tipos de matérias – para um formato que corresponda a um determinado tipo de informação interpretável e manipulável por uma máquina, tal como um computador, transformando pontos elementares da imagem em *pixels*. Um pixel corresponde a menor parte componente de uma imagem digital. O número de pixels que compõem tal imagem depende da resolução da mesma, ou seja, quanto maior a resolução, maior a quantidade de pixels que a compõe. A medida dessa resolução é dada em *Dots Per Inch* (DPI) ou pontos por polegada.

Ao se criar uma imagem digital, gera-se uma variável que fornece acesso ao objeto dessa imagem, que nada mais é que uma matriz de valores que variam entre um limites mínimo e outro máximo (que digitalmente são representados por números inteiros de 8 bits sem sinal, correspondendo a 256 valores possíveis, de 0 a 255) e cuja dimensão é determinada pela quantidade de *canais de cor* da imagem. Para imagens em nível de cinza ou imagens em preto-e-branco, faz-se uso de uma matriz de valores com apenas uma dimensão, visto que tais imagens apresentam apenas um canal de cor, referente ao nível de intensidade dos pixels entre um limiar mínimo igual a 0 (associado ao preto) e o limiar máximo igual a 255 (associado ao branco). Na Figura 1 é mostrada

uma planilha que corresponde a uma matriz representativa para os valores dos níveis de intensidade dos pixels de uma imagem em níveis de cinza.

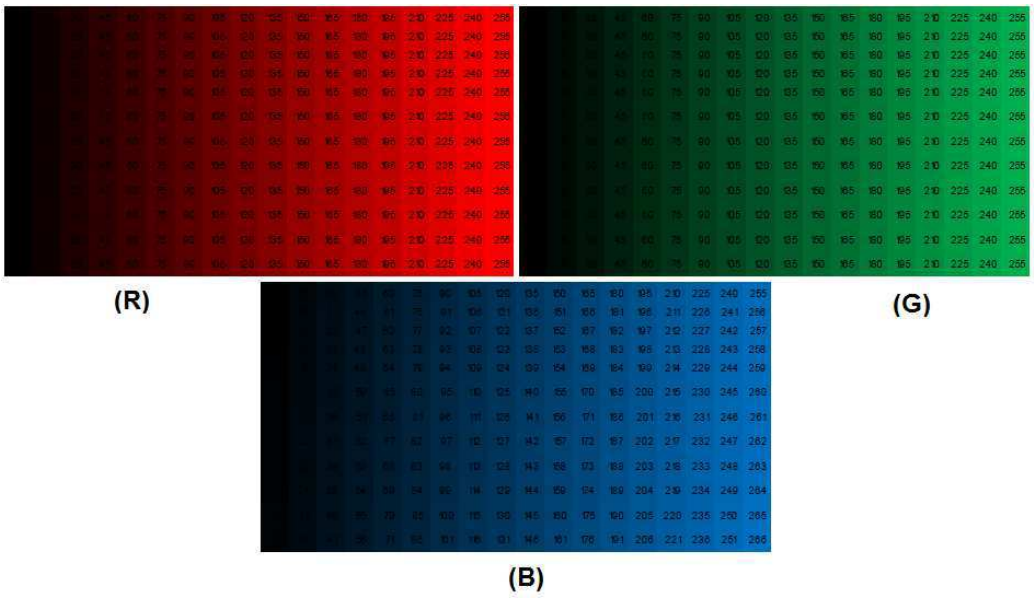
Figura 1. Matriz de níveis de intensidade associada a uma imagem em níveis de cinza.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R
1		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
2		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
3		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
4		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
5		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
6		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
7		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
8		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
9		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255
10		15	30	45	60	75	90	105	120	135	150	165	180	195	210	225	240	255

Fonte: ANTONELLO. Introdução a Visão Computacional com Python e OpenCV, p. 7.

Já para a representação de uma imagem digital colorida deve-se adotar um sistema que padronize a forma como a composição de cores é feita. Por exemplo, no sistema de cores RGB *(R* – vermelho, do inglês *red; G* – verde, do inglês *green, B* – azul, do inglês *blue),* valores entre 0 e 255 são associados a cada uma dessas três cores primarias e armazenados em matrizes distintas, conforme se vê na Figura 2. Assim, imagens no padrão RGB apresentam 3 dimensões (ou 3 canais) contendo em cada dimensão os valores de intensidade correspondentes a cada uma das cores no referido padrão.

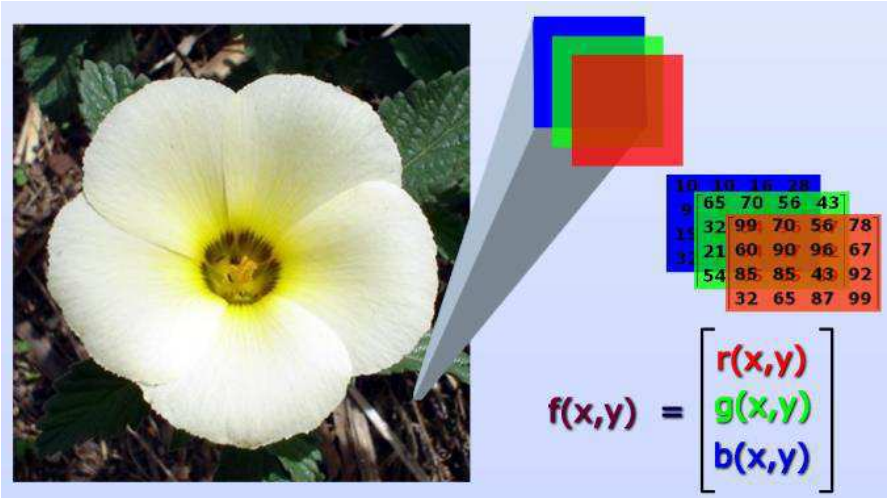
Figura 2. Representação das matrizes que compõe uma imagem no padrão RGB.



Fonte: ANTONELLO. Introdução a Visão Computacional com Python e OpenCV, p. 7.

Uma vez conhecida a existência dos canais de cores, pode-se entender a formação de uma imagem digital colorida como a adição desses canais, ou seja, a associação do valores dos pixels equivalentes em cada canal resulta em determinada coloração, de tal forma que a cor dos pixels na imagem é uma função dos valores contidos em cada canal da imagem, como vemos simbolizado na Figura 3, e assim, torna-se possível a geração de uma imensa gama de cores.

Figura 3. Representação de uma imagem colorida utilizando padrão RGB.



Fonte: Imagem obtida da internet.

Existem outros padrões de cores (HSV, CMYK, Lab entre outros), que não são abordados neste trabalho, porém apresentam grande utilização em diversas aplicações envolvendo imagens.

2.2 PROCESSAMENTO DIGITAL DE IMAGENS

Dá-se o nome de processamento digital de imagens às técnicas voltadas para a análise de dados multidimensionais provenientes de tipos diversos de sensores. Por meio dele, a manipulação de uma imagem por computador é tal que a entrada e a saída do processo são imagens. Sua maior utilidade volta-se para a melhoria do aspecto visual de natureza estrutural de uma imagem para o analista humano, além de fornecer outros tipos de informações para a sua interpretação, permitindo inclusive a criação de resultados que possam ser posteriormente submetidos a outros processamentos.

Por meio de técnicas de processamento digital de imagens, além de ser possível analisar uma cena nas várias regiões do espectro eletromagnético, também se possibilita a integração de diversos tipos de dados de alguma forma referenciados à imagem. Este processamento pode ser basicamente dividido em três pontos: Pré-processamento (processamento inicial de dados brutos para calibração radiométrica da imagem, correção de distorções geométricas e remoção de ruído), Realce (melhoria da qualidade da imagem, permitindo uma melhor discriminação dos objetos presentes na mesma) e Classificação (atribuição de classes aos objetos presentes na imagem).

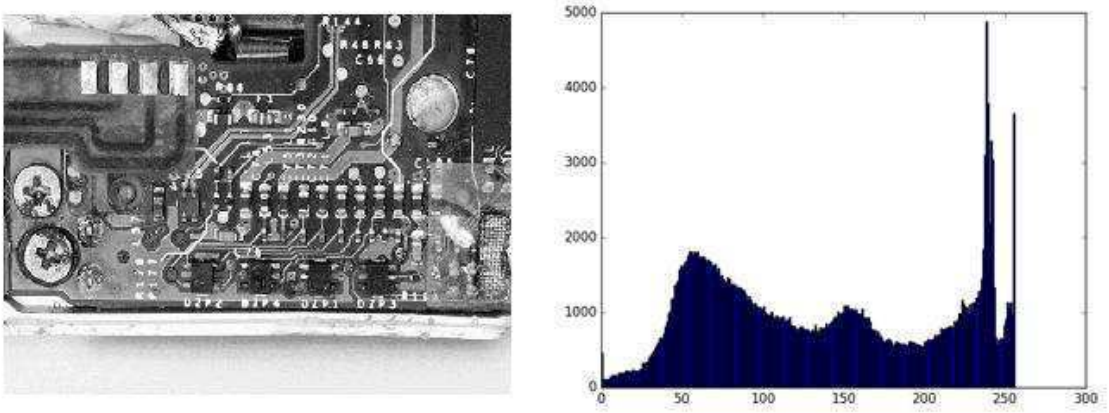
É comum a necessidade de se realizar um tratamento inicial nos dados dos arquivos de imagem a serem manipulados computacionalmente. Tais arquivos dos quais se deseja extrair informações precisam em alguns casos ser convertidos para um determinado formato ou tamanho, ou mesmo ser realizado algum método para remoção de *ruídos*, que podem ser entendidos como interferências nos dados da imagem que possam ter sido agregados durante seu processo de aquisição e que podem causar perda de informação ou ainda dados errôneos. Ruídos podem provir de fontes variadas, tais como o tipo de sensor utilizado na captação, a iluminação e/ou condições climáticas do ambiente no momento da aquisição da imagem, ou até mesmo a posição relativa entre o objeto de interesse e a câmera. Dessa forma, o ruído numa imagem não é apenas interferência no sinal de captura, mas também interferências que podem acarretar em má interpretação ou reconhecimento de objetos na imagem.

A seguir, abordam-se os principais tipos de processamento de imagens adotados neste trabalho.

2.2.1 EQUALIZAÇÃO DE HISTOGRAMA

Um histograma de uma imagem digital trata-se de um gráfico composto por linhas ou colunas que representa a distribuição dos valores dos pixels na mesma. O eixo das abscissas (eixo X) do gráfico apresenta normalmente uma distribuição referente à intensidade do pixel, e no eixo das ordenas (eixo Y) apresenta-se o número de pixels associados a determinado valor de intensidade. A Figura 4 consta de uma imagem e seu histograma.

Figura 4. Exemplo de uma imagem e histograma associado.



Fonte: próprio autor.

A analise do histograma de uma imagem permite obter uma imediata noção sobre as características da mesma, dentre elas o grau de contraste: quanto maior o espalhamento de pontos ao longo do eixo X, maior o contraste da imagem.

Cada valor no gráfico é obtido por meio da Equação 1:

$$p_r(r_q) = \frac{n_q}{n} \tag{1}$$

onde:

$$0 \leq r_q \leq 1$$

q = 0, 1, ... , L-1, sendo L o numero de níveis de cinza da imagem;

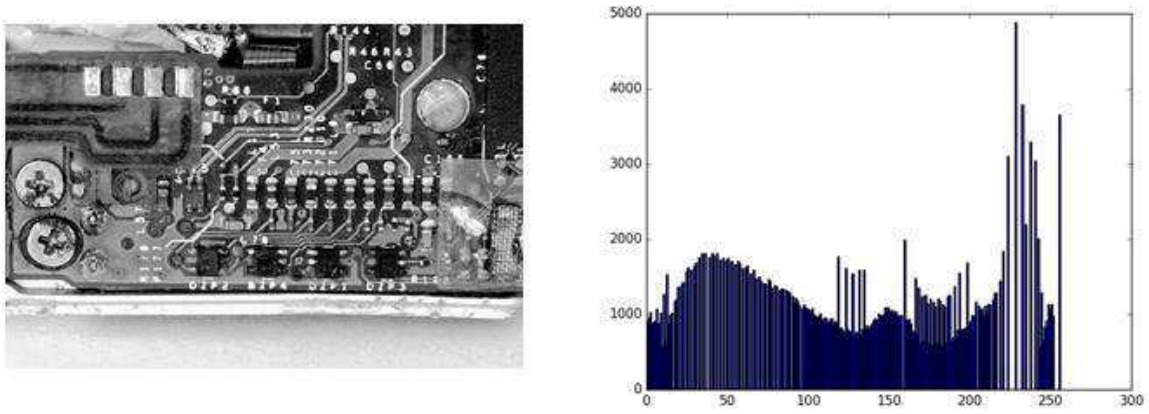
n é o número total de pixels na imagem;

n_k é número de pixels cujo nível de cinza corresponde à k;

$p_r(r_q)$ é a probabilidade do K-ésimo nível de cinza.

É possível realizar um cálculo matemático sobre a distribuição de pixels para aumentar o contraste de uma imagem com o objetivo de distribuir de forma mais uniforme as intensidades dos pixels sobre a imagem, fazendo com que o acumulo de pixels próximos a alguns valores seja suavizado. A esse processo chamamos de Equalização de histograma. Na Figura 05 mostra-se o resultado da aplicação equalização na imagem da Figura 4.

Figura 5. Resultado da aplicação de equalização de histograma.



Fonte: Próprio autor.

2.2.2 BINARIZAÇÃO (THRESHOLDING)

A binarização é um método utilizado geralmente quando se faz necessária a separação entre objetos de interesse numa imagem. Este processamento, também denominado limiarização (do inglês, *thresholding*) se baseia na adoção de um ponto limiar (ou mais de um ponto, quando necessário) para separação das regiões de interesse e de não interesse na imagem. Os valores desses limiares são escolhidos de acordo com o histograma dos pixels da imagem em escala de cinza da seguinte forma: pixels com valores de intensidade L menores que o limiar de corte tem sua intensidade alterada para o nível de branco (valor 255), e aqueles com valor L acima, para o nível preto (valor 0), conforme se representa na Equação 2.

$$P(x,y) = \begin{cases} 255, & f(x,y) \leq L \\ 0, & f(x,y) > L \end{cases} \quad (2)$$

onde:

$P(x,y)$ é o valor a ser atribuído ao pixel pela binarização;

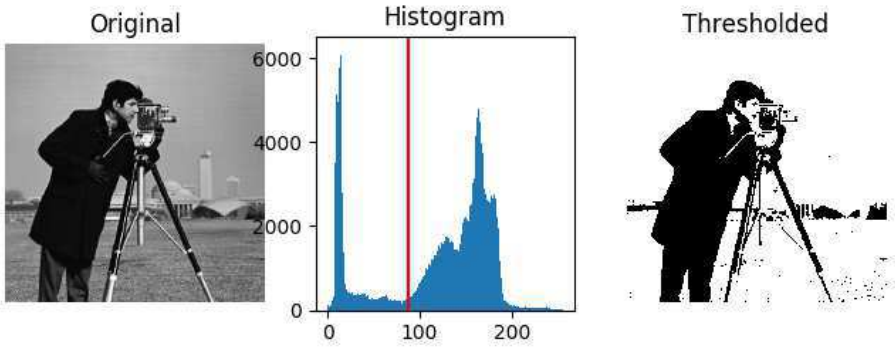
$f(x,y)$ é o valor do pixel analisado;

L é o valor de limiar estabelecido.

Na Figura 6 observa-se o efeito da Binarização sobre uma imagem para um determinado valor de limiar. A escolha de um limiar conveniente apresenta grande

importância para que se garanta o melhor resultado possível no que diz respeito a melhor separação do conteúdo da imagem e/ou de possíveis ruídos.

Figura 6. Efeito da Binarização sobre uma imagem.



Fonte: imagem obtida da internet.

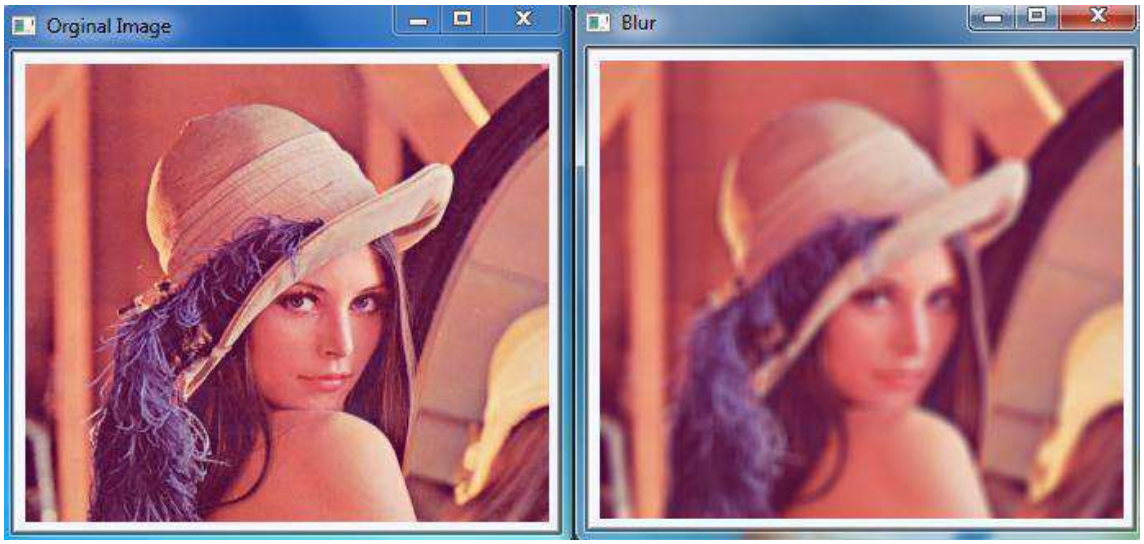
2.2.3 SUAVIZAÇÃO (SMOOTHING)

O processo de suavização (do inglês *Smoothing*) de uma imagem digital consiste em efetuar uma alteração no valor de intensidade da cor de cada pixel da mesma, realizando-se uma espécie de "borrão" (do inglês *Blurring*), onde a intensidade de cada pixel é dada por uma relação com a intensidade dos pixels vizinhos. Essa relação pode ser estabelecida através do calculo de uma média simples dos valores dos pixels imediatamente vizinhos que "envolvem" o pixel analisado, formando uma "janela", também denominada de *kernel* (do inglês núcleo). Nas Figuras 7 e 8 observa-se um exemplo de kernel e o efeito de suavização sobre uma imagem, respectivamente.

Figura 7. Representação ilustrativa de um kernel.

Fonte: ANTONELLO. Introdução a Visão Computacional com Python e OpenCV, p. 30.

Figura 8. Efeito de suavização sobre uma imagem.



Fonte: disponível em http://answers.opencv.org/question/66681/difference-between-blur-and-dilate/. Acessado em 20/07/2017.

Esse efeito apresenta grande utilidade em ações como identificação de objetos em imagens, visto que alguns processos, tal como detecção de bordas, por exemplo, apresentam melhores resultados após a aplicação de *blurring* sobre a imagem.

2.2.4 SEGMENTAÇÃO E DETECÇÃO DE BORDAS

A análise digital de uma imagem comumente utiliza uma classificação estatística, ou seja, um processo de verificação de pixels isoladamente, que apresenta a limitação de se basear apenas em características espectrais da mesma. Devido a isso, o uso de segmentação de imagem antes de sua etapa de classificação apresenta-se como opção de contorno a tal problema, para com isso ser possível extrair os dados de interesse para a uma determinada aplicação.

Neste processo, efetua-se a divisão da imagem em locais, a fim de se obter regiões úteis à aplicação, sendo essas regiões conjuntos de "pixels" adjacentes que apresentam uniformidade e se espalham bidireccionalmente. Essa divisão de regiões pode ser feita fundamentando-se em diferentes tipos de processos, dentre os quais cita-se o de detecção de bordas.

O processo para extração de bordas de uma imagem é implementado por um algoritmo de detecção de bordas que faça uso de um filtro (elemento capaz de remover partes não desejadas ou extrair partes úteis de um sinal, como ruídos ou determinadas

componentes de frequência que estão dentro de uma faixa). Tal algoritmo leva em conta os gradientes de nível de cinza da imagem original para assim gerar o que se chama “imagem gradiente” ou “imagem de intensidade de borda”, calculando um limiar para detecção das bordas. Ao identificar um pixel cujo valor associado é superior ao limiar estabelecido, inicia-se o processo de “perseguição” de uma borda, na qual as vizinhanças desse pixel são monitoradas no intuito de identificar o próximo que possui maior valor de nível digital, seguindo-se nesse sentido até que se encontre uma borda distinta ou a fronteira da imagem.

Deste processo chega-se a uma imagem binária, composta por valores iguais a 1 referentes às bordas e 0 às regiões que não representa bordas. Por fim, a imagem binária produzida será rotulada de modo que as porções com valores 0 estarão limitadas pelos valores 1, que correspondem às bordas detectas na imagem.

Na Figura 9 tem-se o resultado de uma imagem resultante do processo de segmentação e detecção de bordas.

FIGURA 9. Efeito de segmentação sobre uma imagem.

Fonte: disponível em http://opencvexamples.blogspot.com/2013/10/void-canny-inputarray-image-outputarray.html. Acessado em 27/07/2017.

2.3 VISÃO COMPUTACIONAL

Por visão computacional considera-se o processo de modelagem e replicação da visão humana usando software e hardware. Por meio dele, desenvolve-se teoria e

tecnologia para construção de sistemas artificiais que obtém informação de imagens ou quaisquer dados multidimensionais. Exemplos de aplicações que envolvem visão computacional incluem o controle de processos (como robôs industriais ou veículos autônomos), detecção de eventos, organização de informação, modelagem de objetos ou ambientes e interação (atrelado à interação homem-máquina). Subcampos de pesquisa incluem reconstrução de cena, detecção de eventos, reconhecimento de objetos, aprendizagem de máquina e restauração de imagens.

Visão computacional e reconhecimento de imagem são termos frequentemente usados como sinônimos, porém o primeiro abrange mais do que apenas analisar imagens. O reconhecimento de imagens em si – a análise de pixels e padrões de imagens – são uma parte integrante do processo de visão computacional, que envolve tudo, desde reconhecimento de objetos e caracteres até análise de texto e até mesmo expressões faciais humanas. O reconhecimento de imagem de hoje, ainda na maior parte, identifica apenas objetos básicos. Mas o potencial da visão computacional é sobre-humano, com a capacidade de realizar atividades tais como interpretação visual na ausência de luz e à longas distâncias e processar dados associados a essas ações rapidamente e em volume maciço.

A visão computacional, em seu sentido mais pleno, está sendo usada na vida cotidiana e nos negócios para conduzir diversos tipos de tarefas, incluindo identificar doenças médicas em raios-x, identificar produtos e onde comprá-los, anúncios dentro de imagens editoriais, entre outros. A visão computacional pode ser usada para digitalizar plataformas de mídia social a fim de encontrar imagens relevantes que não podem ser descobertas por meio de pesquisas tradicionais. A tecnologia é complexa e, assim como todas as tarefas acima mencionadas, requer mais do que apenas reconhecimento de imagem, mas também análise semântica de grandes conjuntos de dados.

Com relação a processos mais relacionados com a visão computacional, podemos citar o reconhecimento de padrões e o rastreamento de um objeto numa imagem.

2.3.1 RECONHECIMENTO DE PADRÕES

Um processo que envolva reconhecimento de padrões requer inicialmente algum conhecimento prévio e forma de armazenamento da informação sobre um objeto a ser reconhecido. Sendo assim, um sistema de visão necessita de uma base de conhecimento

dos objetos que serão foco do reconhecimento. Essa base pode ser implementada diretamente num código, através, por exemplo, de um sistema baseado em regras, ou pode ser "aprendida" a partir de um conjunto de amostras daqueles objetos utilizando técnicas de aprendizado de máquina.

Um objeto pode ser definido por mais de um tipo de padrão, como forma, dimensões, cor, textura etc., e o reconhecimento individual de cada um deles tende a facilitar o recognição do objeto como um todo. As técnicas de reconhecimento de padrões podem ser divididas em dois grupos: estruturais, na qual os padrões são descritos de forma simbólica e a estrutura é a forma como se relacionam; e técnicas que utilizam teoria de decisão, onde os padrões são descritos por propriedades quantitativas, fazendo-se uma decisão acerca da detenção ou não dessas propriedades pelo objeto. Os processos de reconhecimento de padrões também podem integrar técnicas utilizadas nestes dois grupos.

Contudo, não é comum a existência de técnicas estruturais pré-desenvolvidas em bibliotecas, uma vez que essas variam de acordo com o objeto analisado. Para alguns casos específicos, porém, podem-se encontrar pacotes que as abranjam de forma mais genérica. Técnicas baseadas em teoria possuem caráter mais geral e podem ser adaptadas a diferentes tipos de objetos.

2.3.2 TRANSFORMADA DE HOUGH

A Transformada de Hough é um método padrão para extração de recursos comumente utilizada em análise de imagens, processamento de digital de imagens e visão computacional. Elaborada e patenteada originalmente por Paul Hough em 1962, estava voltada à identificação de retas em imagens. Posteriormente, foi estendida para identificar formas geométricas que possam ser parametrizada, tais como círculos ou elipses. A transformada, tal como é usada hoje, é atribuída a Richard Duda e Peter Hart, que em 1972 a chamaram de "Transformada de Hough Generalizada" após a patente Hough. A transformada foi popularizada na comunidade de Visão Computacional por Dana H. Ballard através de um artigo publicado em 1981.

O objetivo do método é encontrar instâncias de objetos dentro de uma determinada classe de formas por um procedimento de votação em um espaço de parâmetros, a partir do qual os candidatos a objeto são obtidos como um máximo local num chamado espaço de acumulação, que é explicitamente construído pelo algoritmo

para o calculo da transformada. A ideia é que, ao ser aplicada uma transformação na imagem, todos os pontos pertencentes a uma mesma curva sejam mapeados num único ponto de um novo espaço de parametrização de uma curva procurada.

2.3.2.1 TRANSFORMADA DE HOUGH CIRCULAR

Constituindo uma técnica básica usada no processamento de imagens digitais para detecção de objetos circulares, a Transformada Circular de Hough é uma especialização da transformada básica. Seu objetivo é encontrar círculos em entradas de imagens imperfeitas. Os candidatos a circulo são eleitos num espaço de parâmetros e, em seguida, é feita uma seleção dos máximos locais numa chamada matriz de acumulação.

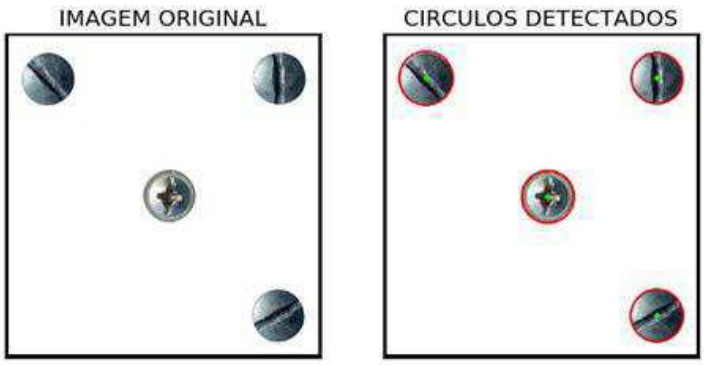
De forma analítica, sendo um circulo descrito pela Equação (3)

$$(x-a)^2 + (y-b)^2 = r^2 \tag{3}$$

Em que (a, b) corresponde as coordenadas do centro do circulo e r o tamanho de seu raio, tomando-se um ponto fixo (x, y), os parâmetros podem ser encontrados de acordo com a equação (3), permitindo com isso a construção de um espaço de parâmetros tridimensional (a, b, r), onde todos os parâmetros que satisfaçam (x, y) ficariam na superfície de um cone reto invertido, cujo ápice situa-se em $(x, y, 0)$. No espaço tridimensional, os parâmetros do círculo podem ser identificados pela interseção de varias superfícies cônicas que são definidas por pontos no círculo naquele espaço bidimensional. Este processo pode ser dividido em duas etapas. O primeiro estágio corresponde a fixação do valor do raio para em seguida encontrar-se o centro ótimo dos círculos em um espaço de parâmetros bidimensional, enquanto o segundo estágio volta-se para encontrar o raio ótimo em um espaço de parâmetros unidimensional.

Na Figura 9 exibe-se o resultado da detecção de círculos numa imagem realizada computacionalmente utilizando a transformada em questão.

Figura 10. Resultado da detecção de círculos numa imagem através da Transformada Circular de Hough.



Fonte: Próprio autor.

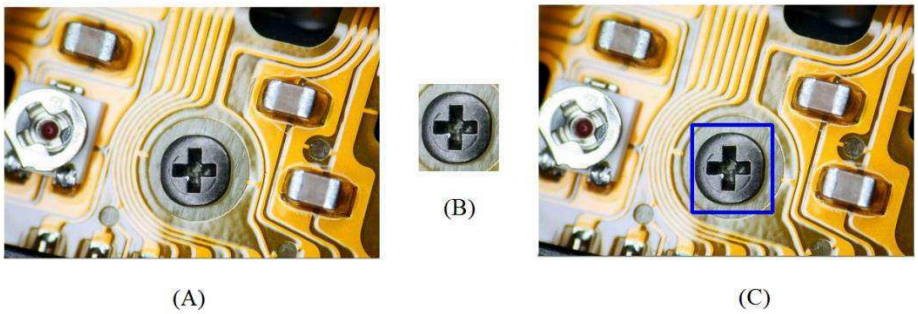
2.3.3 TEMPLATE MATCHING

Template matching (do inglês comparação de modelo) e uma técnica associada à imagens digitais utilizada para localizar partes de uma imagem que correspondam ou apresentem semelhanças com uma imagem modelo ou *template*.

O processamento de correspondência ocorre com o "deslizamento" da imagem modelo ao longo de todas as posições possíveis na imagem onde se faz a busca, ou seja, cada pixel da imagem original é comparado aos pixels do template, e a partir disso é calculado um índice numérico que indica quão próximo o modelo se ajusta à imagem principal em determinada fração da mesma.

Na Figura 11 mostra-se uma imagem, um template a ela associado, e o resultado computacional da busca da segunda na primeira proveniente do uso da técnica em questão.

Figura 11. Aplicação de Template Matching: imagem original (A); template (B); resultado de busca do template na imagem original (C).



Fonte: Próprio autor.

2.4 OPENCV

OpenCV (Open Source Computer Vision Library) é uma biblioteca de visão computacional de código aberto disponível em http://SourceForge.net/projects/opencvlibrary. A biblioteca é escrita em linguagem de programação C e C ++ e é executada em sistemas operacionais Linux, Windows e Mac OS X, existindo ainda desenvolvimento ativo em interfaces para Python, Ruby, Matlab e outras linguagens.

Essa biblioteca, oficialmente lançada em 1999 como uma proposta da Intel Research de melhorar aplicações de uso intensivo de processamento, é projetada com foco em eficiência computacional e para aplicações em tempo real, fazendo proveito do uso de processadores multicore. Caso se deseje maior otimização automática nas arquiteturas da Intel, é possível comprar as bibliotecas IPP (Integrated Performance Primitives) da Intel, que consistem em rotinas otimizadas de baixo nível em muitas áreas algorítmicas diferentes. O OpenCV usa automaticamente a biblioteca IPP apropriada em tempo de execução se essa biblioteca estiver instalada.

Um dos objetivos da OpenCV é fornecer uma infra-estrutura de visão computacional simples de usar que ajude seus usuários a criar aplicativos bastante sofisticados e de forma rápida, contendo mais de 500 funções que abrangem muitas áreas, incluindo inspeção de produtos de fábrica, imagens médicas, segurança, interface de usuário, calibração da câmera, visão estéreo e robótica.

Como a visão computacional e o aprendizado automático da máquina estão fortemente presentes, o OpenCV também contém uma Biblioteca de Aprendizado de Máquinas (MLL, do inglês *Machine Learning Library*) completa e de uso geral. Esta sub-biblioteca é focada no reconhecimento e no agrupamento de padrões estatísticos. A MLL é altamente útil para as tarefas de visão computacional que estão no cerne da missão do OpenCV, sendo suficientemente generalista para ser usado para qualquer problema de aprendizado de máquina.

3 DESENVOLVIMENTO

Neste capitulo são explanados os materiais e métodos utilizados neste trabalho para a obtenção do resultado planejado, a saber, a detecção de parafusos em imagens associadas à placas de circuitos eletrônicos, através dos *scripts* desenvolvidos nos programas computacionais elaborados pelo próprio autor e que implementam as técnicas de visão computacional apresentadas anteriormente.

3.1 MATERIAIS UTILIZADOS

Para desenvolvimento dos programas que implementam os conceitos de visão computacional neste trabalho foi adota a linguagem de programação Python. Criada por Guido van Rossum em 1991, os objetivos de projeto da linguagem focam em produtividade e legibilidade. Destacam-se entre suas principais características:

- Baixo uso de caracteres especiais, tornando-a muito semelhante a um pseudocódigo executável;
- Uso de identação para delimitação blocos executáveis no programa;
- Uso mínimo de palavras-chave voltadas à compilação;
- Gerenciamento automático de uso de memória etc.

Assim, associando a essas características outras vantagens da linguagem, tais como o uso de programação procedimental para geração de programas simples e rápidos, biblioteca padrão vasta contendo classes, métodos e funções que permitem realizar diversas tarefas, desde acesso a bancos de dados a interfaces gráficas com o usuário, além de ferramentas para lidar com dados científicos, a utilização de Python associada à biblioteca OpenCV apresentou-se como escolha conveniente à execução de cada etapa desse trabalho. A versão Python utilizada foi a 2.7.5, e a da biblioteca OpenCV 3.0.0

Por conseguinte, sendo Python é uma linguagem livre e multiplataforma, isto é, programas escritos em uma plataforma serão executados sem empecilhos em plataformas diferentes, optou-se por adotar a plataforma de edição de programas PyCharm desenvolvido pela empresa JetBrains. Essa plataforma é um ambiente de desenvolvimento integrado (IDE, do inglês Integrated Development Enviroment) usado na programação de computadores, especificamente para a linguagem Python. Sendo uma plataforma cruzada com versões para Windows, MacOS e Linux, fornece vantagens como análise de código, depurador gráfico, testador de unidade integrado e integração com sistemas de controle de versão (VCSes, do inglês Version Control Systems). A versão utilizada neste trabalho foi a PyCharm Edu 3.5.1 Community Edition.

O *hardware* utilizado no desenvolvimento e experimentos dos *scripts* foi um *notebook* com 1 processador Core 2 Duo, 4 GB de memoria RAM e placa de vídeo *on-board Mobile* Intel® 4 Series Express Chipset Family, operando com sistema operacional Windows® 7 Home Basic - 64 bits.

3.2 MÉTODOS UTILIZADOS

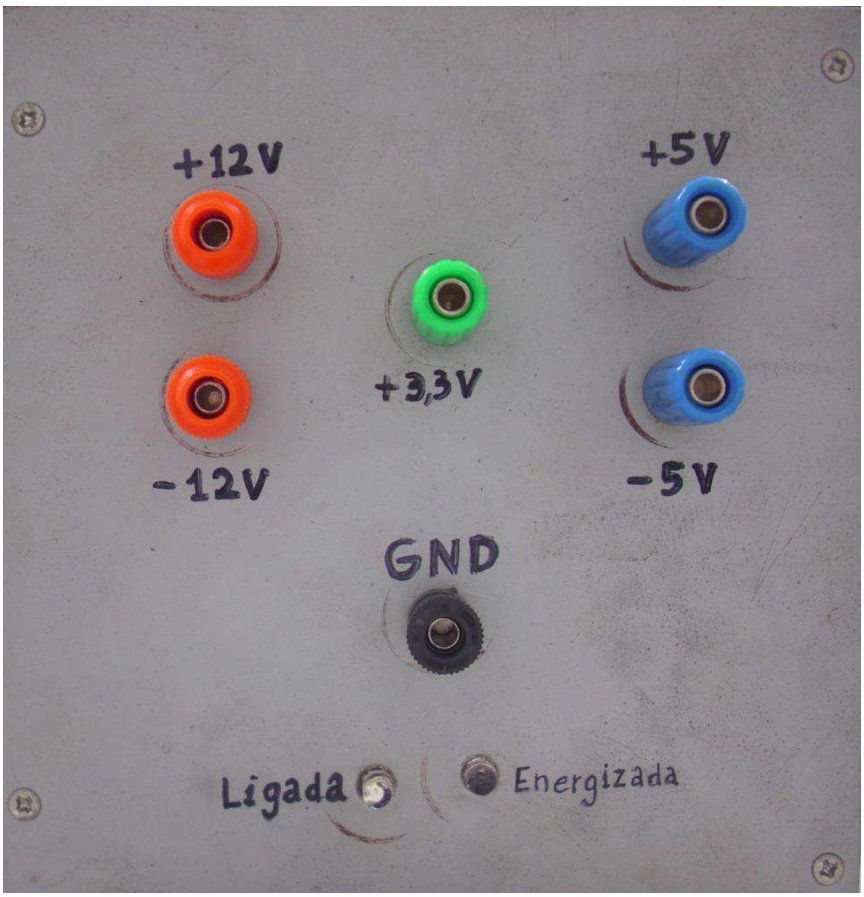
Inicialmente foi escolhido o objeto de estudo para analise visual do programa. Escolheu-se então uma fonte de bancada, que é apresentada na Figura 12.

Figura 12. Fonte de bancada utilizada para criação das imagens utilizadas no trabalho.

Fonte: Amanda Thaisa, 2017.

De posse da fonte, foram obtidas duas imagens referentes às faces que apresentam parafusos nela fixados. Estas imagens encontram-se exibidas nas Figuras 13 e 14.

Figura 13. Imagem da face frontal da fonte de bancada.



Fonte: Amanda Thaisa, 2017.

Figura 14. Imagem da face lateral da fonte de bancada.

Fonte: Amanda Thaisa, 2017.

Para a análise destas imagens, decidiu-se criar duas versões de código para o programa desenvolvido, onde cada versão opera sobre a imagem de uma das faces da fonte, apenas variando-se valores de parâmetros e/ou acrescentando comandos necessários à verificação dos parafusos em cada caso, reduzindo com isso o tamanho do programa e o esforço de processamento do mesmo. As diferenças entre os códigos das duas versões serão expostas neste trabalho conforme conveniente. A versão 1 recebeu o arquivo de imagem associado à face frontal da fonte, e versão 2, o arquivo da face lateral.

3.2.1 AQUISIÇÃO DA IMAGEM DAS PLACAS DE CIRCUITO PELO SISTEMA

Para que o sistema desenvolvido pudesse armazenar e manipular os dados referentes aos arquivos das imagens da fonte, deu-se inicio ao código do programa acrescentando-se as bibliotecas padrões necessárias, a saber, a biblioteca gráfica OpenCV, a biblioteca matemática Numpy e a biblioteca de manipulação de gráficos Matplotlib, e em seguida, foi acrescentada a função "cv2.imread" do OpenCV, responsável pela criação das variáveis de armazenamento das imagens digitais. O trecho do código que contem as ações descritas acima é visto na Figura 15.

Figura 15. Trecho do código que realiza adição de bibliotecas e aquisição da imagem.

```
#INCLUSAO DE BIBLIOTECAS UTILIZADAS
import cv2
import numpy as np
from matplotlib import pyplot as plt

#CARREGAMENTO DO ARQUIVO DE IMAGEM A SER UTILIZADO PELO PROGRAMA
original = cv2.imread('IMAGEM_Face-Frontal.jpg')
```

Fonte: próprio autor.

Como visto na Figura 15, o nome do arquivo que contem a imagem da face frontal da fonte, "IMAGEM_Face-Frontal" (no caso da segunda versão do programa, foi atribuído o nome do arquivo relativo a face lateral, "IMAGEM_Face-Lateral") é levado à função do OpenCV juntamente com o formato do arquivo, no caso uma imagem do tipo jpg ou JPEG (acrônimo em inglês para Joint Photographics Experts Group). É de suma importância que esse arquivo seja contido no diretório em que o

programa é executado, caso contrário não é possível o carregamento da imagem pelo programa.

3.2.2 REDIMENSIONAMENTO E CRIAÇÃO DE CÓPIAS DA IMAGEM ORIGINAL

A fim de melhorar as condições de manipulação dos dados da imagem, o passo seguinte no trabalho constituiu-se em redimensionar a imagem original quanto à quantidade de pixels, reduzindo-a a um tamanho físico para favorecer não só a visualização através da tela do computador utilizado, como para reduzir processamento, uma vez que reduzir o número de pixels que a representa acarreta um menor número de dados para ser processado.

Assim, criou-se uma variável em OpenCv que recebe o arquivo contendo a nova imagem reduzida, "img", gerada pela função "cv2.resize", que contem parâmetros internos à função e o valor percentual a que se deseja transformar a imagem. Vê-se na Figura 16 o trecho do código (versão 1 do programa) responsável pelo escalonamento, onde foi escolhida uma redução igual a 0,25% tanto na largura como no comprimento original da imagem.

Ainda nessa figura observa-se a existência de três variáveis criadas com o objetivo de armazenar copias da imagem redimensionada, a saber, "img0", "img1" e "img2". A criação dessas variáveis foi necessária, pois algumas manipulações feitas posteriormente aos arquivos de imagem alteram os dados nelas contidos de forma irreversível. Desse modo, as copias sofrem as alterações ao invés da original, mantendo sua integridade ao longo da execução do código.

FIGURA 16. Trecho do código que realiza cópias redimensionadas da imagem original

```
#REDIMENSIONAMENTO DO IMAGEM PARA MANIPULACAO DA MESMA PELO PROGRAMA
img = cv2.resize(original,None,fx=0.25, fy=0.25, interpolation = cv2.INTER_CUBIC)

#CRIACAO DE COPIAS DO ARQUIVO ORIGINAL DA IMAGEM PARA MANIPULACAO DAS MESMAS PELO PROGRAMA
img0 = img.copy()
img1 = img.copy()
img2 = img.copy()

#INDICACAO NO PROMPT DO PROGRAMA DAS DIMENSOES DA IMAGEM ORIGINAL E APOS REDIMENSIONAMENTO
print 'Dimensoes da imagem original (y, x, canais): ',original.shape
print '\nImagem redimensionada. Novas Dimensoes (y, x, canais): ',img.shape
```

Fonte: próprio autor.

As duas últimas linhas de código vistas na Figura 16 foram acrescentadas ao programa para apresentar em seu prompt de comando as propriedades da imagem original e das copias, que corresponde terno de valores (y, x, canais), sendo “y” o numero de linhas de pixels (largura da imagem), “x” equivale ao numero de colunas de pixels (comprimento da imagem) e “canais” indica o numero de canais de cores da imagem.

3.2.3 RECONHECIMENTO DE PARAFUSOS POR MEIO DE DETECÇÃO DE CÍRCULOS

Como primeira forma proposta de detecção dos parafusos na imagem, optou-se pela identificação dos mesmos por meio de identificação de formas circulares que correspondam aos locais onde se visualiza parafusos. Nessa etapa, leva-se em consideração o conhecimento prévio da imagem a ser averiguada, e com isso, supõe-se já serem conhecidas as áreas da placa em que não há presença de parafusos. Sendo assim, foram atribuídas a estas máscaras de pixels com nível de intensidade igual a 0 (correspondente a pixels em preto) para auxílio as etapas seguintes de tratamento dos dados. Fornecendo-se a largura e o comprimento da região a ser coberta por uma máscara às variáveis de armazenamento, a adição das máscaras na imagem é feita através dos trechos de código exibidos nas Figuras 17 (versão 1) e 18 (versão 2).

FIGURA 17. Trecho do código que realiza aplicação de mascara de pixels em preto a imagem da face frontal da fonte.

```
#ATRIBUICAO DE PIXELS DE VALOR 0 NA REGIAO SEM PRESENCA DE PARAFUSOS, COMPPRENDIDA ENTRE xi E xf E ENTRE yi E yf
xi_1 = 60
xf_1 = 423
yi_1 = 60
yf_1 = 480

img0[yi_1:yf_1,xi_1:xf_1] = [0,0,0] # EXCLUSAO DE REGIAO COM AUSENCIA DE PARAFUSOS NA IMAGEM
```

Fonte: próprio autor.

FIGURA 18. Trecho do código que realiza aplicação de mascara de pixels em preto a imagem da face lateral da fonte.

```
#ATRIBUICAO DE PIXELS DE VALOR 0 NA REGIAO SEM PRESENCA DE PARAFUSOS, COMPPRENDIDA ENTRE xi E xf E ENTRE yi E yf
xi_1 = 96;      xi_2 = 13;      xi_3 = 392
xf_1 = 325;     xf_2 = 372;     xf_3 = 575
yi_1 = 9;       yi_2 = 54;      yi_3 = 178
yf_1 = 322;     yf_2 = 278;     yf_3 = 295

img0[yi_1:yf_1,xi_1:xf_1] = [0,0,0] # EXCLUSAO DE REGIAO COM AUSENCIA DE PARAFUSOS NA IMAGEM
img0[yi_2:yf_2,xi_2:xf_2] = [0,0,0] # EXCLUSAO DE REGIAO COM AUSENCIA DE PARAFUSOS NA IMAGEM
img0[yi_3:yf_3,xi_3:xf_3] = [0,0,0] # EXCLUSAO DE REGIAO COM AUSENCIA DE PARAFUSOS NA IMAGEM
```

Fonte: próprio autor.

De posse da imagem associada a máscara, passou-se ao tratamento da mesma para detecção de círculos.

3.2.3.1 CONVERSÃO DA IMAGEM PARA NÍVEIS DE CINZA

Devido a fato das funções em OpenCV adotadas neste trabalho que são responsáveis por efetuar ações de limiarização, filtragem e detecção de bordas exigirem que os arquivos de imagem contenham apenas um canal de cores (ou seja, imagens em preto e branco), foi necessária a conversão da variável da imagem a ser passada para tais funções para níveis de cinza. Essa ação pode ser facilmente implementada através da função “cv2.cvtColor”, que recebe a imagem a ser convertida (neste caso, a imagem “img0”) e um parâmetro que informa o tipo de conversão entre padrões de cores (no caso deste trabalho, do padrão de cores RGB para níveis de cinza ou GRAY) e a armazena na variável denominada “img_gray”.

Na Figura 19 encontra-se trecho do programa presente em ambas as versões do programa em que se faz essa conversão.

FIGURA 19. Trecho do código que realiza conversão do arquivo de imagem de RGB para níveis de cinza.

```
#CONVERSAO DA IMAGEM EM RGB PARA NIVEIS DE CINZA PARA TRATAMENTO DE DADOS
img_gray = cv2.cvtColor(img0, cv2.COLOR_BGR2GRAY)
```

Fonte: próprio autor.

3.2.3.2 EQUALIZAÇÃO HISTOGRÂMICA

A fim de buscar um nível de contraste satisfatório para a imagem produzida após a conversão em nível de cinza, foi utilizada uma função de equalização otimizada de histogramas do OpenCv, denominada “cv2.createCLAHE”. O termo CLAHE (do inglês Contrast Limited Adaptive Histogram Equalization) refere-se a uma aplicação de equalizador de histograma adaptativo no qual a imagem é dividida em pequenos blocos denominados "tiles" (de dimensões 8x8 pixels, por padrão do OpenCV) e em seguida cada um desses blocos é utilizado para construção do histograma como de costume. Há ainda o uso de limitação de contraste pelo CLAHE para impedir amplificação de ruídos na imagem devidos a este processo de equalização, de modo que, se qualquer lote de histograma estiver acima do limite de contraste especificado (por padrão 40 no

OpenCV), esses pixels são separados e distribuídos uniformemente para outros compartimentos da imagem antes da equalização ser aplicada. Após a equalização, para remover artefatos em bordas, a função faz uso interno de interpolação bilinear.

Mostra-se na Figura 20 o trecho do programa em ambas as versões do programa que abriga a criação e aplicação do equalizador de histogramas.

FIGURA 20. Trecho do código que realiza equalização de histograma de uma imagem associada.

```
#APLICACAO DE EQUALIZACAO DE HISTOGRAMA A IMAGEM EM NIVEIS DE CINZA
clahe = cv2.createCLAHE(clipLimit=2.0, tileGridSize=(8,8)) #CRIACAO DE EQUALIDOR DE HISTOGRAMA ADAPTAVEL LIMITADO EM CONTRASTE
img_clahe = clahe.apply(img_gray)    #APLICACAO DO OBJETO CLAHE A IMAGEM EM NIVEIS DE CINZA
```

Fonte: próprio autor.

3.2.3.3 APLICAÇÃO DE SUAVIZAÇÃO

O passo seguinte no programa foi a suavização da imagem após a equalização de seu histograma. O objetivo desta ação foi melhorar os dados a serem passados às funções posteriores do código responsáveis por detecções de bordas. Para isso, testou-se o uso de funções da biblioteca gráfica que produzissem um nível se suavização ótima às necessidades das etapas seguintes do sistema. Assim, foi escolhida a função “cv2.blur”, que basicamente convolve a imagem com um filtro utilizando um kernel normalizado, gerando simplesmente a média de todos os pixels na área do kernel e substituindo o elemento central pelo resultado. Nos parâmetros da função deve-se especificar a largura e a altura dessa janela. Com isso, após algumas verificações empíricas, chegou-se a um tamanho ótimo para esta última de 3x3 para a aplicação.

Na Figura 21, exibe-se o trecho do programa presente em ambas as versões do programa onde se encontra o código para a suavização.

FIGURA 21. Trecho do código que realiza suavização sobre uma imagem associada.

```
#APLICACAO DE BLURRING A IMAGEM EM NIVEIS DE CINZA PARA UNIFORMIZACAO DE PIXELS
b = 3
img_blur = cv2.blur(img_clahe,(b,b))
```

Fonte: próprio autor.

3.2.3.4 DETECÇÃO DE BORDAS

Nessa etapa do trabalho efetuou-se a detecção de bordas na imagem gerada após a suavização. Dentre diversos métodos disponíveis avaliados no OpenCV, aquele que apresentou melhores resultados a essa tarefa foi o método de detecção de borda de Canny, pois, além de favorecer a delimitação de bordas bastante nítidas, também auxilia na redução de ruídos existentes na imagem analisada. A função associada, denominada "cv2.Canny", recebe como parâmetros os limiares inferior e superior os quais utiliza, atuando como um filtro, para avalizar o gradiente de intensidade dos pixels que são candidatos à composição de bordas.

Com isso, após varios teste, chegou-se aos valores otimizados para os limiares inferior e superior iguais a 55 e 60, respectivamente, desse modo gerando-se o trecho de código que foi utilizado em ambas as versões do programa, visualizado na Figura 22.

FIGURA 22. Trecho do código que realiza detecção de bordas via detector Canny.

```
#APLICACAO DE FILTRO CANNY PARA DETECCAO DE BORDAS NA IMAGEM COM BLUR
e_inf = 55  # VALOR DE LIMIAR INFEIROR UTILIZADO PELA FUNCAO CANNY
e_sup = 60 # VALOR DE LIMIAR SUPERIOR UTILIZADO PELA FUNCAO CANNY
img_edges = cv2.Canny(img_blur, e_inf, e_sup)
```

Fonte: próprio autor.

3.2.3.5 DETECÇÃO DE CÍRCULOS SOBRE BORDAS

Esta é a etapa crucial na identificação de parafusos na imagem, pois se espera que o programa seja capaz nesse momento de identificar bordas circulares que correspondam efetivamente às bordas dos parafusos visíveis na imagem. Para isso, adotou-se a função que implementa a Transformada Circular de Hough em OpenCV, a função "cv2.HoughCircles". A ela são passados os seguintes parâmetros: a imagem na qual se deseja identificar círculos (neste caso, a imagem resultante da detecção de bordas por Canny), a relação inversa da resolução do acumulador para resolução da imagem "dp" (utilizado pela função para definir redução na resolução das imagens em que são identificados os círculos, que tem valor 1 por padrão na Opencv), distância mínima entre os centros de círculos detectados "minDist" (que auxilia na exclusão de falsos positivos detectados), primeiro parâmetro específico do método "parâmetro_1", segundo parâmetro específico do método "parâmetro_2" (este último também sendo de

grande peso para exclusão de falsos positivos), valor mínimo do raio dos círculos detectados “raioMinimo” e valor máximo do raio dos círculos detectados “raioMaximo”, sendo esses dois últimos úteis para delimitação de uma faixa de valores de raios de círculos aceitáveis para a aplicação.

Com isso, foram feitos diversos teste até se chegar aos valores vistos no trecho de código apresentado na Figura 23 para a versão 1 do programa. Na versão 2, o código utilizado é idêntico, apenas modificando-se o valor da variável associada ao raio mínimo para 6 para que houvesse a exclusão de falsos positivos presentes neste caso, fato esse que demonstra a importância dos valores adotados para os parâmetros nas análises das imagens.

O resultado produzido pela função aqui abordada é armazenado na variável “circulos”, um vetor em que cada elemento corresponde a um vetor contendo três valores de ponto flutuante (x, y, r), sendo x e y os valores da abscissa e ordenada do ponto central do círculo, respectivamente, e r o valor do seu raio.

FIGURA 23. Trecho do código da versão 1 do programa que realiza detecção de bordas via detector Canny.

```
#DETECCAO DE CIRCULOS SOBRE AS BORDAS DETECTADAS - PARAMETROS AJUSTADOS ESPECIFICAMENTE PARA A IMAGEM ANALISADA

dp = 1              #RELACAO INVERSA DA RESOLUCAO DO ACUMULADOR ADOTADA PARA A RESOLUCAO DA IMAGEM
minDist = 50        #DISTANCIA MINIMA ENTRE OS CENRO DOS CIRCULOS DETECTADOS ADOTADA
parametro_1 = 200   #PRIMEIRO PARAMETRO ESPECIFICO DO METODO ADOTADO
parametro_2 = 18    #SEGUNDO PARAMETRO ESPECIFCO DO METODO ADOTADO
raioMinimo = 6      #VALOR MININO DO RAIO DOS CIRCULOS DETECTADOS ADOTADO
raioMaximo = 15     #VALOR MAXIMO DO RAIO DOS CIRCULOS DEDTECTADOS ADOTADO

circulos = cv2.HoughCircles(img_edges,cv2.HOUGH_GRADIENT,dp,minDist,
                            param1=parametro_1,param2=parametro_2,minRadius=raioMinimo, maxRadius=raioMaximo)
```

Fonte: próprio autor.

3.2.3.6 CONTABILIZAÇÃO E SINALIZAÇÃO DE PARAFUSOS NA IMAGEM ORIGINAL

Chegamos aqui à etapa final da detecção de parafusos por meio dos círculos na imagem. Uma vez tendo-se gerados os dados referentes a cada círculo visualizado pelo programa, fez-se uso deste para a contabilização de elementos circulares obtidos anteriormente. Isso foi possível fazendo-se uso da função “len” da biblioteca matemática Numpy, que permite retornar a quantidade de elementos contida em um vetor. Assim, sendo a variável “circulos” propriamente um vetor onde se encontram

armazenados cada um dos vetores associados a cada circulo detectado, o uso da função em questão permitiu a exibição do número desses vetores no prompt do programa.

Feito isso, foi elaborada uma rotina que permitiu representar sobre a imagem original a visualização dos círculos encontrados, tanto suas circunferências como seus pontos centrais. Para tal, foi efetuado inicialmente um arredondamento dos valores das coordenadas e do raio de cada círculo com o auxilio da biblioteca matemática, e logo após, pode-se utilizar a função “cv2.circle” da biblioteca gráfica, que é especifica para criar representações gráficas circulares num arquivo de imagem, recebendo como parâmetros os dados do ponto central e raio do círculo a ser exibido, além de informações sobre cor e tipo do traço desejados para o contorno circular.

Por fim, exibe-se no prompt do programa um texto indicando o posicionamento dos parafusos (em termos de pixels), bem como a exibição da imagem original da face da fonte de bancada analisada com os círculos sobre os parafusos visualizados.

O trecho de código existente nas duas versões do programa e que compreende todas essas ações pode ser visto na Figura 24.

FIGURA 24. Trecho do código que realiza a contabilização e sinalização dos parafusos na imagem.

```
#UTILIZACAO DOS DADOS OBTIDOS NA GERACAO DE CIRCULOS PARA CONTABILIZACAO DE CIRCULOS IDENTIFICADOS NA IMAGEM
j1=len(circulos[0])
print '\nNumero de parafusos detectados na imagem: ', j1

#---REPRESENTACAO DE CIRCULOS SOBRE OS PARAFUROSOS IDENTIFICADOS---

circles = np.uint16(np.around(circulos))
conjunto = []    #CRIACAO DO ARRAY 'arr' PARA ARMAZENAMENZAR IMAGENS
c=1         #VARIAVEL AUXILIAR PARA CONTAGEM
for i in circles[0,:]:
    cv2.circle(img1,(i[0],i[1]),i[2],(0,255,0),2)    #   DESENHO DA CIRCUNFERENCIA DE UM CIRCULO
    cv2.circle(img1,(i[0],i[1]),2,(255,0,),3)        #   DESENHO DO PONTO CENTRAL DE UM CIRCULO
    print '   Parafuso {} centrado em ({}, {})'\
        .format(       c,            i[0],i[1])
    c=c+1

#APRESENTACAO DOS PARAFUSOS LOCALIZADOS CIRCULADOS NO IMAGEM ORIGINAL
cv2.imshow('Resultado - Identificacao de parafusos via circulos',img1)
```

Fonte: próprio autor.

3.2.4 RECONHECIMENTO DE PARAFUSOS POR MEIO DO USO DE TEMPLATES

Como segunda forma de detecção dos parafusos nas imagens das fontes, foi adotada a técnica de Template Matching, sendo uma opção ao caso em que não se

consideram conhecidos os locais da face da fonte onde não há parafusos, como foi pressuposto no método de reconhecimento feito anteriormente.

3.2.4.1 FORNECIMENTO DE TEMPLATES AO PROGRAMA

Inicialmente ao processo de template matching, foi passado à ambas versões do programa os arquivos de imagem a serem usados como template nesta etapa do trabalho, que são exibas nas Figuras 25 e 26.

Figura 25. Template utilizado para detecção de parafusos da face frontal da fonte de bancada.

Fonte: próprio autor.

Figura 26. Templates utilizados para detecção de parafusos da face lateral da fonte de bancada.

Fonte: próprio autor.

A imagens modelo foram salvas em variáveis com denominações especificas para cada versão do programa e, para fins de posterior manipulação de dados, foram convertidas para níveis de cinza de forma direta pela função ”cv2.imread” do OpenCV acrescentando-se o parâmetro 0 logo após o nome do arquivo à função, como se exemplifica com o trecho de código retirado da versão 1 do programa, visto na Figura 27.

Figura 27. Carregamento e conversão para níveis de cinza da imagem modelo.

```
#ADICAO DE TEMPLATE DA IMAGEM DO PARAFUSO A SER IDENTIFICADO NA IMAGEM ORIGINAL
template = cv2.imread('template_fonte_Face-Lateral.jpg',0)

#ARAMAZENAMENTO DAS DIMENSOES DA IMAGEM DO TEMPLATE PARA TRATAMENTO DE DADOS
larg_tpl, comp_tpl = template.shape[::-1]
```

Fonte: próprio autor.

Ainda na Figura 27 pode-se ver o armazenamento da largura e comprimento do template nas variáveis “larg_tpl” e “comp_tpl”, respectivamente, que servem como referencia para localização do centro do template e posterior indicação da localização dos parafusos identificados.

3.2.4.2 BUSCA DO TEMPLATE NA IMAGEM

A biblioteca OpenCV dispõe de uma função especifica para execução de template matching, a função “cv2. matchTemplate”. Nela, são fornecidas as variáveis do template e da imagem a qual se deseja buscar a correspondência, além de um dos métodos de comparação implementados em OpenCV, dos quais foi escolhido para execução nesse trabalho o método “cv2.TM_CCOEFF_NORMED” por atender satisfatoriamente o fornecimento de resultados na comparação das imagens.

Logo, no trecho de código presente na FIGURA 28, a variável “semelhanca” armazena um conjunto de valores proporcionais ao nível de identidade entre o modelo e a imagem principal em ambas as versões do programa.

Figura 28. Trecho do código que realiza a busca da imagem template numa imagem de interesse.

```
#BUSCA DO TEMPLATE NA IMAGEM ORIGINAL
semelhanca = cv2.matchTemplate(img_gray,template,cv2.TM_CCOEFF_NORMED)
```

Fonte: próprio autor.

3.2.4.3 ADOÇÃO DE LIMIAR DE SEMELHANÇA AOS TEMPLATE

Aqui se inicia a parte principal do processo de identificação de parafusos por meio de comparação com uma imagem modelo. A partir dos valores que indicam o grau de semelhança entre determinada área da imagem analisada e o template, foi estabelecido um valor limiar com o intuito de, por meio de uma comparação matemática entre esse valor e os resultados do template matching, serem armazenados apenas aqueles que apresentarem valor igual ou maior ao do limiar estabelecido, determinando-se com isso as regiões com maiores possibilidades de apresentarem parafusos nelas inseridos.

Conforme pode ser visto na Figura 29, que apresenta o trecho de código utilizado nas versões do programa para esta fase do trabalho, os valores resultantes

dessa comparação numérica foram armazenados pelo programa na variável "locais_Tplmatch", e a partir deles, juntamente com as dimensões físicas do template e o auxilio de um laço de execução, foram guardadas cada uma das coordenadas do ponto central de regiões que apresentam as mesmas dimensões do template e que correspondem a um local com presença de parafuso. O objetivo desta ação é se chegar a um único ponto que corresponda à localização de um parafuso na imagem.

De posse das coordenadas das regiões candidatas a conter parafusos, foi utilizada a função "cv2.rectangle" da biblioteca gráfica para delimita-las com marcações retangulares. Esta função recebe como parâmetros: a imagem onde se deseja fazer as marcações, as coordenadas de dois vértices da região de forma retangular que se deseja representar (que aqui correspondem às coordenadas dos vértices das imagens template utilizadas) e informações sobre cor e tipo de traço utilizado para delimitar o contorno geométrico.

Figura 29. Trecho do programa que armazena e sinaliza locais reconhecidos com presença de parafusos.

```
#VALOR LIMIAR DE SEMELHANCA ENTRE REGIOES DA IMAGEM E TEMPLATE - VALOR ATRIBUIDO EMPIRICAMENTE
limiar = 0.44

#GERACAO DE VALORES ASSOCIADOS A AREAS NA IMAGEM ORIGINAL SEMELHANTES AO TEMPLATE
locais_Tplmatch = np.where( semelhanca >= limiar )

#VETORES AUXILIARES PARA ARMAZENAMENTO DE COORDENADAS DOS PARAFUSOS A SEREM LOCALIZADOS
positionY = []
positionX = []

#--------DETERMINACAO DA POSICAO DOS PARAFUSOS NA IMAGEM ORIGINAL--------
for pt in zip(*locais_Tplmatch[::-1]):
    positionY.append((pt[0] + larg_tpl/2))      #ARMAZENAMENTO DA COORDENADA Y DO CENTRO DE REGIAO ASSOCIADA A UM PARAFUSO
    positionX.append((pt[1] + comp_tpl/2))      #ARMAZENAMENTO DA COORDENADA X DO CENTRO DE REGIAO ASSOCIADA A UM PARAFUSO

    #DELIMITACAO DE REGIOES CANDITADAS A CONTER UM PARAFUSO POR UM POLIGONO RETANGULAR
    cv2.rectangle(img2, pt, (pt[0] + larg_tpl, pt[1] + comp_tpl), (255,0,0), 2)
```

Fonte: próprio autor.

O tratamento de dados que se seguiu consistiu em remover áreas detectadas contendo parafusos que apresentam redundâncias. Por áreas redundantes se entendem aquelas com coordenadas iguais ou muito próximas umas das outras, dado que se sabe não haver parafusos muito próximos uns dos outros nas faces da fonte. As coordenadas restantes são unidas em pares (y,x) de coordenadas, que representam, portanto, a posição real dos parafusos.

Na Figura 30 mostra-se a fração do código pra o tratamento dos dados redundantes na versão 1 do programa. Na versão 2, devido ao uso de três imagens modelo, este mesmo código foi repetido mais duas vezes no programa, a fim de que em

cada repetição fosse realizada a remoção das redundâncias relacionadas ao template nela especificado, apenas ajustando-se para isso os valores dentro dos laços implementados, conforme os valores tratados em cada busca de template .

Figura 30. Remoção de dados redundantes do template matching pelo programa.

```
#CRIACAO DE VETORES AUXILIARES PARA MANIPULACAO DOS DADOS REFERENTES AS COORDENADAS DO CENTRO DOS PARAFUSOS
pontosY = []
pontosX = []
centro = []

pontosY.append(positionY[0])    #   ADICAO DE PRIMEIRA COORDENADA EM Y AO VETOR DE COORDENADAS-Y
pontosX.append(positionX[0])    #   ADICAO DE PRIMEIRA COORDENADA EM X AO VETOR DE COORDENADAS-X

#                   VERIFICACAO E EXCLUSAO DE REDUNDANCIAS
#   NAS COORDENADAS X E Y DOS PONTOS CANDIDADOS A POSICAO CENTRAL
#                           DE UM PARAFUSO

i=0
while i < (len(positionY)-1):
    if ((positionY[i] != positionY[i+1]) & (abs(positionY[i] - positionY[i+1])>2)):
        pontosY.append(positionY[i+1])
    i = i + 1

j=0
while j < (len(positionX)-1):
    if ( (positionX[j]!= positionX[j+1]) & (positionX[j+1]!= positionX[j-1]) & (abs(positionX[j] - positionX[j+1])>2)):
        pontosX.append(positionX[j+1])
    j = j + 1

k=0
while k < (len(pontosY)):
    if (pontosY[k] != pontosX[k]):
        centro.append([pontosY[k],pontosX[k]])
    k = k + 1
```

Fonte: próprio autor.

3.2.4.4 SINALIZAÇÃO DOS PARAFUSOS IDENTIFICADOS POR TEMPLATE MATCHING

Concluindo o processo de detecção de parafusos por template, é feita a marcação do ponto central de um parafuso localizado na imagem da fonte, acompanho de uma indicação textual (a inscrição da letra “P” junto ao retângulo que delimita o parafuso detectado) com o uso da função “cv2.putText” e seus devidos parâmetros (o local da imagem onde deve ser colocado o texto e o tipo, tamanho e cor da fonte textual), além da exibição no prompt do programa das coordenadas desta posição central e da imagem da face da fonte com as sinalizações indicativas em cada parafuso identificado. Todos esses procedimentos são vistos no fragmento de código mostrado na Figura 31.

FIGURA 31. Trecho do programa que sinaliza e exibe parafusos detectados via uso de template matching.

```
# DESENHA O CENTRO DO PARAFUSO IDENTIFICADO APOS O PROCESSAMENTO DE DADOS ANTEIROR
for z in range(len(centro)):
    cv2.circle(img2,(centro[z][0],centro[z][1]),2,(0,0,255),3)
    print '\nparafuso localizado em \n',centro[z]
    #IMPRESSAO DE UMA LETRA "P" JUNTO AO PARAFUSO LOCALIZADO
    font = cv2.FONT_HERSHEY_SIMPLEX
    cv2.putText(img2,'P',(centro[z][0]-(larg_tpl/2), centro[z][1]-(comp_tpl/2)), font, 0.8, (0,0,0), 2, cv2.LINE_AA)

#IMPRESSAO DA IMAGEM ORIGINAL COM OS PARAFUSOS IDENTIFICADOS
cv2.imshow('Resultado - Identificacao de parafusos via template',img2)
```

Fonte: próprio autor.

3.2.5 EXIBIÇÃO DE RESULTADOS INTERMEDIÁRIOS E ENCERAMENTO DO PROGRAMA

Considerando-se interessante a averiguação das imagens intermediárias geradas ao longo do programa, foi feito uso das funcionalidades da biblioteca Matplotlib para reuni-las em uma única janela gráfica, onde cada uma delas recebe um titulo indicativo na janela.

O programa é encerrado pelo comando “cv2.destroyAllWindows”, cuja função é fechar todas as janelas abertas geradas pelo programa e encerrar sua execução.

Vemos a implementação desta janela gráfica pelo trecho de código, utilizado em ambas as versões do programa, na Figura 32.

Figura 32. Trecho do código para exibição de resultados intermediários e encerramento do programa.

```
#ORGANIZACAO DAS PRINCIPAIS IMAGENS INTERMEDIARIAS DO PROCESSAMENTO E DOS RESULTADOS OBTIDOS NUMA UNICA JANELA GRAFICA

titles = ['IMAGEM REDIMENSIONADA','IMAGEM COM MASCARA','IMAGEM GRAY',\
          'IMAGEM POS-CLAHE','IMAGEM SUAVIZADA', 'BORDAS']

images = [img,img0,img_gray,\
          img_clahe,img_blur,img_bordas]

for i in xrange(6):
    plt.subplot(2,3,i+1)
    plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()

#COMANDO QUE ENCERRA O PROGRAMA FECHANDO TODOS AS JANELAS ABERTAS
cv2.destroyAllWindows()
```

Fonte: próprio autor.

4 RESULTADOS

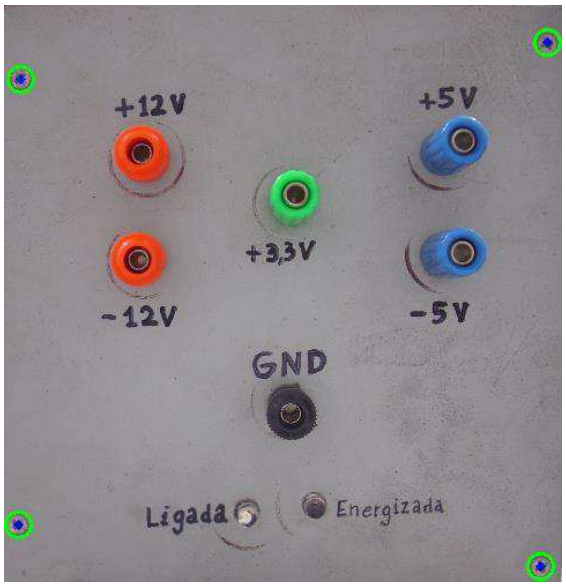
Com a execução conjunta de cada parte do código do programa apresentada neste trabalho, foi possível chegar aos resultados apresentados a seguir.

As versões finais neste trabalho estão disponíveis no endereço http://laps.dee.ufcg.edu.br.

4.1 DETECÇÃO DE PARAFUSOS NA FACE FRONTAL DA FONTE DE BANCADA

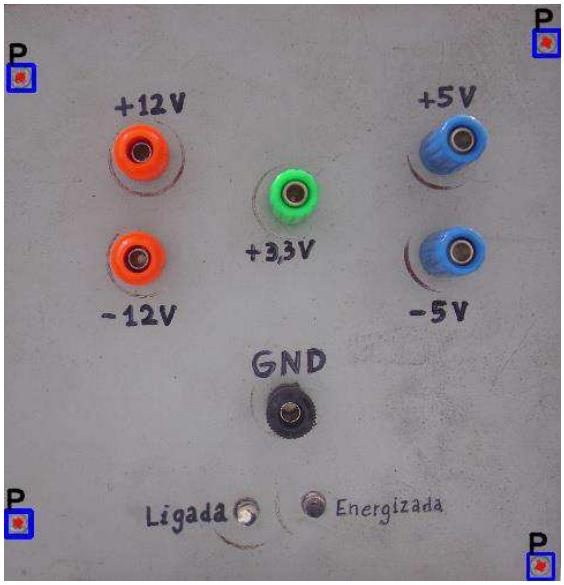
A execução da versão 1 do programa finalizado produziu os resultados exibidos nas Figuras 33, 34, 35 e 36, onde se veem, respetivamente, a imagem de saída gerada via detecção de círculos, a imagem de saída gerada via *template matching*, uma visualização da janela gráfica que apresenta as imagens intermediárias geradas ao longo da execução do programa e as informações relacionadas com o posicionamento dos parafusos nas imagens de saída.

Figura 33. Imagem resultante da detecção via detecção de círculos na face frontal da fonte de bancada.



Fonte: próprio autor.

Figura 34. Imagem resultante da detecção de parafusos via *template matching* na face frontal da fonte de bancada.

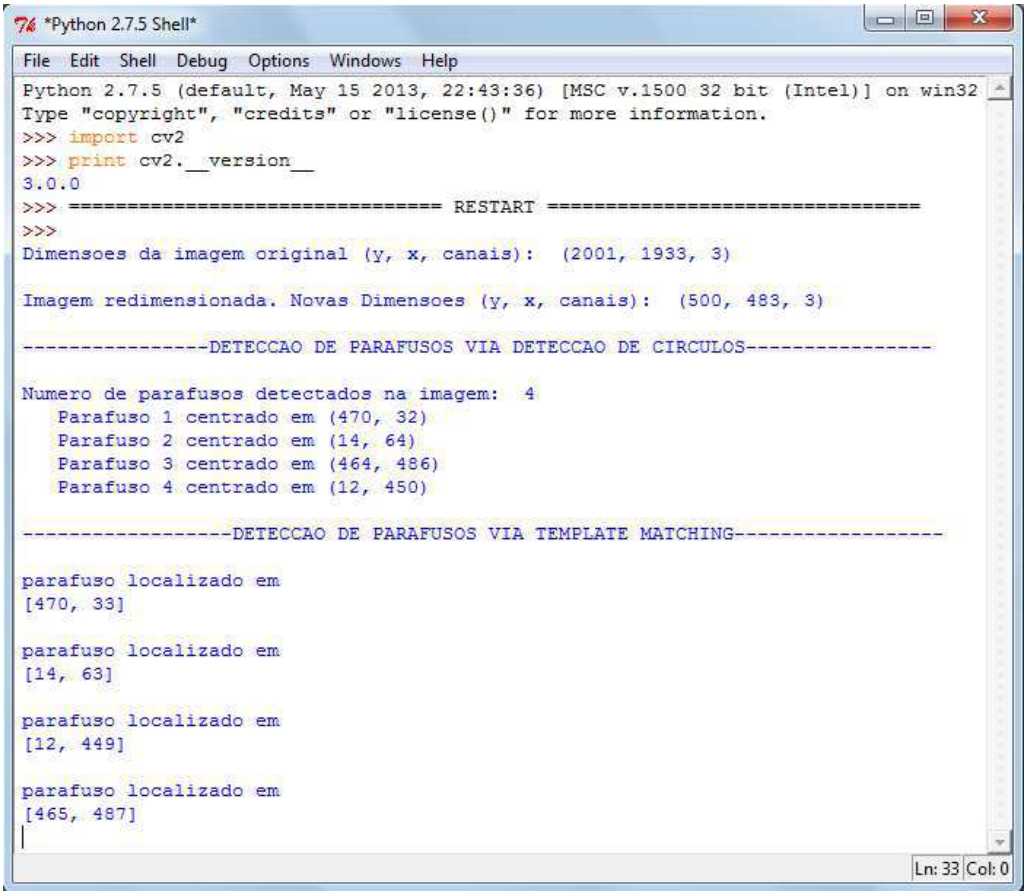


Fonte: próprio autor.

Figura 35. Visualização da janela gráfica que apresenta as imagens intermediárias formadas durante a execução do programa.

Fonte: próprio autor.

Figura 36. Visualização das informações no prompt do programa relacionadas à posição dos parafusos detectados.

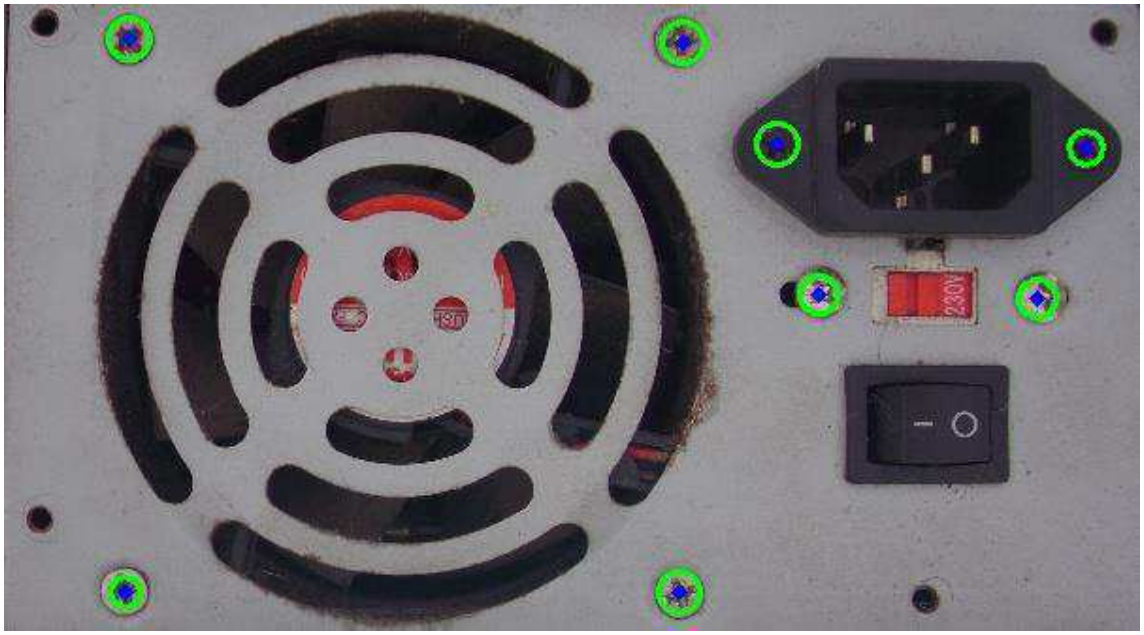


Fonte: próprio autor.

4.2 DETECÇÃO DE PARAFUSOS NA FACE LATERAL DA FONTE DE BANCADA

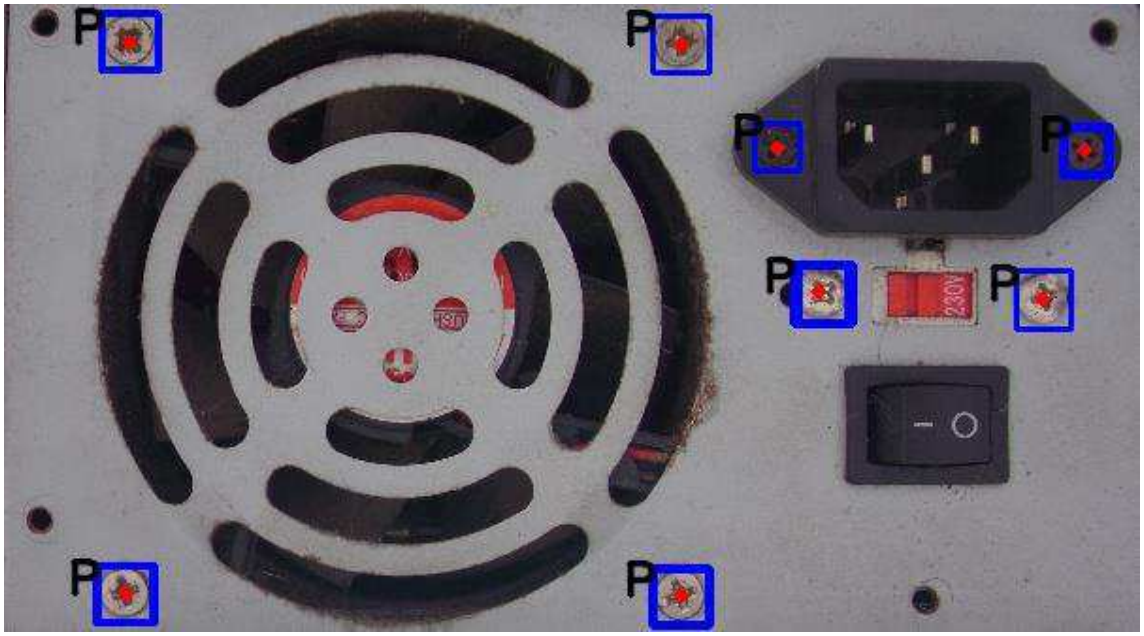
A execução da versão 2 do programa finalizado produziu os resultados exibidos nas Figuras 37, 38, 39 e 40, que constam, respetivamente, da imagem de saída gerada via detecção de círculos, da imagem de saída gerada via template matching, de uma visualização da janela gráfica que apresenta as imagens intermediárias geradas ao longo da execução do programa e das informações relacionadas com o posicionamento dos parafusos nas imagens de saída.

Figura 37. Imagem resultante da detecção de parafusos via detecção de círculos na face lateral da fonte de bancada.



Fonte: próprio autor.

Figura 38. Imagem resultante da detecção de parafusos via *template matching* na face lateral da fonte de bancada.

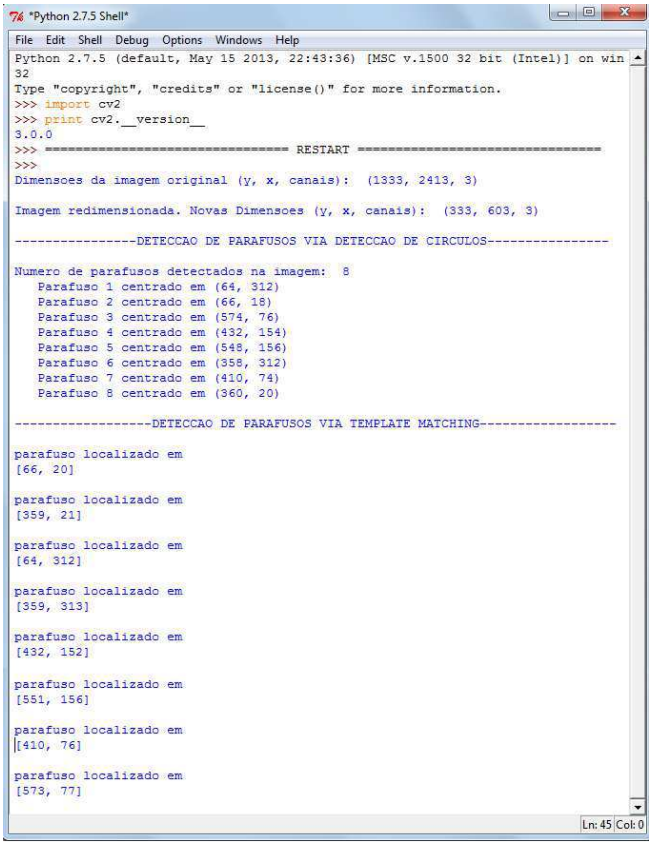


Fonte: próprio autor.

Figura 39. Visualização da janela gráfica que apresenta as imagens intermediárias formadas durante a execução do programa.

Fonte: próprio autor.

Figura 40. Visualização das informações no prompt do programa relacionadas à posição dos parafusos detectados.



Fonte: próprio autor.

5 CONCLUSÃO

Este relatório de trabalho teve como objetivo registrar as etapas do desenvolvimento de um sistema para detecção de parafusos em placas de circuitos eletrônicos através da análise de imagens utilizando técnicas de visão computacional.

O trabalho mostrou-se bastante produtivo para o graduando, permitindo ao mesmo não só adquirir conhecimentos sobre a linguagem de programação Python, antes não familiar a ele, como também adquirir conhecimentos das etapas que compõem processos de concepção, desenvolvimento e criação de sistemas de visão computacional por meio de um vasto aprendizado teórico e prático na área de processamento digital de imagens. A orientação do professor Edmar Candeia foi de grande importância, fornecendo embasamento através de seus conhecimentos próprios para superar as principais dificuldades encontradas.

Desejava-se inicialmente que o sistema projetado fosse genérico, permitindo a identificação de parafusos em qualquer imagem de circuitos eletrônicos, porem, ao decorrer da execução do trabalho, esta tarefa se mostrou ser de um nível de complexidade elevado, exigindo um grau de conhecimento de técnicas de visão computacional maior do que o possível de ser assimilado pelo projetista dentro do prazo estabelecido para conclusão do trabalho. Devido a isso, desenvolveu-se um sistema específico, capaz de detectar parafusos numa imagem conhecida previamente, com resultados bastante satisfatórios quanto ao esperado.

A aquisição das imagens a serem analisadas pelo sistema foi realizada com sucesso, bastando remete-las ao diretório em que o programa que implementa o sistema de visão computacional é executado.

O tratamento de dados oriundos das imagens fornecidas ao sistema se mostrou complexo e minucioso, exigindo conhecimentos de processamento digital de imagens associados aos valores dos parâmetros aplicados às funções das bibliotecas computacionais utilizadas, determinados após diversos e exaustivos testes empíricos. Contudo, a obtenção de resultados coerentes ao longo da execução do projeto testificou o êxito na manipulação desses dados.

A identificação de parafusos através do uso de técnicas de visão computacional no sistema desenvolvido ocorreu de forma satisfatória. Apesar de haverem existido discrepâncias entre os resultados obtidos em relação aos esperados, como por exemplo, a delimitação das bordas e indicação do posicionamento central de alguns parafusos, todos estes foram identificados nas imagens analisadas dentro de uma margem de erro tolerável.

Com isso, conclui-se que os objetivos específicos propostos neste trabalho foram alcançados, uma vez que foi possível detectar cada um dos parafusos presentes nas imagens fornecidas ao sistema com as técnicas de visão computacional adotadas.

Portanto, pode-se afirmar que, a partir dos resultados conseguidos neste trabalho, é plenamente possível serem desenvolvidos sistemas de detecção mais genéricos, capazes inclusive de determinar o tipo de chanfro dos parafusos detectados, fazendo-se uso de técnicas de visão computacional mais robustas, e permitindo com isso que esses sistemas possam futuramente ser agregados à aplicações práticas, tal como a construção de uma máquina de desparafusagem automática.

BIBLIOGRAFIA

BRADSKI, G., KAEHLER,. **Learning OpenCV Computer Vision with the OpenCV Library**. O'REILLY, 2008.

GONZALEZ, R.C.; WOODS, R.E. "**Digital Image Processing"**. 3th ed. Person Prentice Hall. New Jersey, 2008.

ANTONELLO, R. **Introdução a Visão Computacional com Python e OpenCV**. v0.1a, s.d.

MORAIS DOS SANTOS,M. **Um Estudo de Processamento de Imagens com OpenCV**. 2011. 99f. Trabalho de Graduação (Curso de Tecnologia em Sistemas de Computação) - Universidade Federal Fluminense, UFF, Niterói, 2000.

Wikipedia, a enciclopedia livre, em português. <https://pt.wikipedia.org/wiki/Equaliza%C3%A7%C3%A3o_de_histograma>. Acesso em 30 de Junho de 2017.

OpenCV - Smoothing Images. <http://docs.opencv.org/3.1.0/d4/d13/tutorial_py_filtering.html>. Acesso em 5 de Julho de 2017.

Opencv - Contour Features. <http://docs.opencv.org/trunk/dd/d49/tutorial_py_contour_features.html>. Acesso em 8 de Julho de 2017.

ROSEBROCK, A. Detecting Circles in Images using OpenCV and Hough Circles. <http://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/>. Acesso em 13 de Julho de 2017.

OpenCV - Hough Circle Transform. <http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghcircles/py_houghcircles.html>. Acesso em 15 de Julho de 2017.

OpenCV - Template Matching. <http://docs.opencv.org/3.2.0/d4/dc6/tutorial_py_template_matching.html>. Acesso em 20 de Julho de 2017.

OpenCV - Histogram Equalization <http://docs.opencv.org/3.1.0/d5/daf/tutorial_py_histogram_equalization.html>. Acesso em 1 de Agosto de 2017