UNIVERSIDADE DO ESTADO DE SANTA CATARINA – UDESC
CENTRO DE CIÊNCIAS TECNOLÓGICAS – CCT
DEPARTAMENTO DE ENGENHARIA ELÉTRICA – DEE
MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA

RAMON CASCAES SEMIM

SISTEMA DE VISÃO PARA GUIAR UM ROBÔ DE MANIPULAÇÃO DE CABEÇOTES FUNDIDOS

JOINVILLE / SC
2012

RAMON CASCAES SEMIM

SISTEMA DE VISÃO PARA GUIAR UM ROBÔ DE MANIPULAÇÃO DE CABEÇOTES FUNDIDOS

Dissertação apresentada para a obtenção do título de mestre em Engenharia Elétrica da Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas – CCT.

Orientador: Prof. Dr. Roberto Silvio Ubertino Rosso Junior.
Coorientador: Prof. Dr. Alexandre Gonçalves Silva.

JOINVILLE / SC
2012

FICHA CATALOGRÁFICA

S471s

Semim, Ramon Cascaes.

Sistema de Visão para guiar um Robô de Manipulação de Cabeçotes Fundidos / Ramon Cascaes Semim;

Orientador: Roberto Silvio Ubertino Rosso Junior; Coorientador: Alexandre Gonçalves Silva – Joinville, 2012.

125 f. : il ; 30 cm.

Incluem referências.

Dissertação (mestrado) – Universidade do Estado de Santa Catarina, Centro de Ciências Tecnológicas, Mestrado Profissional em Engenharia Elétrica, Joinville, 2012.

1. Robô Industrial. 2. Sistema de Visão.

I. Rosso Jr., Roberto Silvio Ubertino.

II. Silva, Alexandre Gonçalves.

CDD 629.8

"SISTEMA DE VISÃO PARA GUIAR UM ROBÔ DE MANIPULAÇÃO DE CABEÇOTES FUNDIDOS"

por

RAMON CASCAES SEMIM

Esta dissertação foi julgada adequada para a obtenção do título de

MESTRE EM ENGENHARIA ELÉTRICA

área de concentração em "Automação de Sistemas",
e aprovada em sua forma final pelo

CURSO DE MESTRADO PROFISSIONAL EM ENGENHARIA ELÉTRICA
CENTRO DE CIÊNCIAS TECNOLÓGICAS DA
UNIVERSIDADE DO ESTADO DE SANTA CATARINA.

Dr. Roberto Silvio Ubertino Rosso Junior
CCT/UDESC (orientador/presidente)

Banca Examinadora:

Dr. Marcelo da Silva Hounsell
CCT/UDESC

Joinville, 13 de agosto de 2012.

Dr. Roberto de Alencar Lotufo
FEEC/UNICAMP

Dr. Silas do Amaral
CCT/UDESC

Dr. Alexandre Gonçalves Silva
CCT/UDESC (coorientador/suplente)

Aos meus pais, Ruben e Romilda.

À minha esposa, Cristina.

Ao meu filho, Lucas.

AGRADECIMENTOS

Agradeço a Deus por me guiar e me ajudar a vencer as dificuldades encontradas no decorrer desta jornada.

À minha esposa, Cristina, pela ajuda, motivação e paciência...

Ao meu orientador, Prof. Roberto Silvio Ubertino Rosso Jr., pela capacidade em gerenciar projetos, pela inteligência e amizade.

Ao meu coorientador, Prof. Alexandre Gonçalves Silva, pelo companheirismo e pela imensa contribuição técnica ao projeto.

Aos bolsistas que participaram deste projeto, Anderson Resendes Silva e Camila Tormena, pelo auxílio no desenvolvimento.

À Tupy S.A., pela disponibilidade de tempo e equipamentos.

A Ricardo Cardoso e à equipe de Engenharia de Automação da Tupy S.A., pela ajuda e pelo apoio em todos os momentos.

Ao meu grande amigo Daniel Spricigo, pela ajuda e motivação.

Ao Prof. Silas do Amaral, pelo grande incentivo e pela ajuda no início do mestrado.

À UDESC, pela oportunidade.

A minha família, por tudo...

RESUMO

SEMIM, Ramon Cascaes. **Sistema de visão para guiar um robô de manipulação de cabeçotes fundidos**. 2012. 125 f. Dissertação (Mestrado Profissional em Engenharia Elétrica – Área: Automação de Sistemas) – Universidade do Estado de Santa Catarina. Programa de Pós-graduação em Engenharia Elétrica, Joinville, 2012.

Este trabalho apresenta o desenvolvimento de um sistema de visão de baixo custo que será utilizado em uma célula robotizada, implantada em uma das linhas de acabamento de cabeçotes da Tupy S.A. Tal célula será responsável pelas operações de oleação e paletização de cabeçotes fundidos, e a função do robô será manipular as peças entre as estações de trabalho. Os cabeçotes entrarão na célula por meio de um transportador com roletes, onde o robô fará a pega das peças para iniciar a sua manipulação. A célula processará 26 tipos diferentes de cabeçotes e será abastecida manualmente, por isso as peças são disponibilizadas ao robô de forma aleatória. Foram empregados uma *webcam* e módulos de iluminação de baixo custo no projeto. O sistema de visão deve classificar os cabeçotes, de modo a identificar o modelo de cada peça analisada. Outra função do sistema de visão é encontrar a posição e a orientação de cada cabeçote que entrar na célula. Com essas informações o robô será capaz de efetuar a pega de forma correta, independentemente da orientação com a qual o cabeçote entrar na linha. As principais ferramentas de processamento de imagens utilizadas são a Correlação de Pearson e a Transformada de Hough. A calibração do sistema de visão define a posição e a orientação do sistema de coordenadas da câmera, bem como a dimensão de um *pixel*. Esses parâmetros são fundamentais para que a posição de pega do robô seja encontrada corretamente. Para que seja possível efetuar a pega dos cabeçotes o robô terá de reposicionar e reorientar o seu sistema de coordenadas. O robô usado é o modelo IRB6640, fabricado pela empresa ABB. Ele possui seis graus de liberdade e utiliza quatérnios para definir a orientação do seu efetuador final.

Palavras-chave: **Robô Industrial. Sistema de Visão. Correlação de Pearson. Transformada de Hough**.

ABSTRACT

SEMIM, Ramon Cascaes. **Industrial robot guided by vision system to handling foundry head blocks.** 2012. 125 f. Dissertação (Mestrado Profissional em Engenharia Elétrica – Área: Automação de Sistemas) – Universidade do Estado de Santa Catarina. Programa de Pós-graduação em Engenharia Elétrica, Joinville, 2012.

This work presents the development of low cost vision system to use on the robotized work cell installed in a foundry finishing line at Tupy S.A. The work cell will use an industrial robot to make the Oil bath and Palletizing of head blocks. The robot job will be moving the parts inside work station. The head blocks input at work cell through a conveyor that moves the parts to robot grasp area. The robotized cell will process 26 different kinds of parts and will be feeding by a human operator, so this is a problem because the head blocks position/orientation on the conveyor will be random. To development was used both low cost webcam and lightning modules. The vision system should find the head block model as well as find its position/orientation. These information are enough to robot grasp the head blocks on the input conveyor. The main image processing tools used on the project are the Pearson Correlation and the Hough Transform. System calibration defines the coordinate system position and orientation, as well as the *pixel* size. These parameters are important to find out the head block position/orientation. To grasp the parts the robot will reposition and reorient its coordinate system. The industrial robot used in this work is IRB6640, supplied by ABB. It has six degrees of freedom and uses quaternion to define the position/orientation of its end effector.

Keywords: **Industrial Robot. Vision System. Pearson Correlation. Hough Transform**.

LISTA DE FIGURAS

SUMÁRIO

INTRODUÇÃO

O processo de acabamento mecânico em peças fundidas exige uma alta resistência física do operador que exerce tal função. Isso se dá pelo fato de que o ambiente de trabalho é insalubre, não é climatizado, as peças são pesadas e de difícil manuseio. Por conta dessas dificuldades, há cada vez menos pessoas dispostas a trabalhar sob tais condições. Seguindo esse raciocínio, acredita-se que em alguns anos poucas pessoas se proporão a executar a função.

Em linhas de produção onde são fabricados vários modelos diferentes de peças, é importante não haver perdas de tempo em preparação de máquina[1]. Linhas onde existem muitas preparações em um único dia podem perder produtividade e, por consequência, faturamento. Automatizar linhas de acabamento mecânico de peças fundidas é uma das soluções para contornar o problema da falta de mão de obra e incrementar a produtividade. Os robôs industriais são uma alternativa para a implementação desse tipo de automação.

É importante lembrar que peças fundidas sem acabamento não possuem precisão em suas medidas, pois suas superfícies ainda não foram usinadas. Isso dificulta a utilização de robôs, pois as peças podem não estar na posição correta de pega em um transportador.

Tarefas de montagem e manipulação envolvendo robôs industriais têm aumentado em número e complexidade nos últimos anos, em virtude das crescentes exigências por produtos de maior qualidade e menor tempo de fabricação (CHEN *et al*., 2008). O abastecimento de peças a uma célula robotizada é um processo crítico nas aplicações, haja vista que o robô precisa pegar e colocar objetos em locais específicos (CHENG; DENMAN, 2005). O processo de acabamento mecânico em peças fundidas é um exemplo típico dessa situação, em que muitas operações manuais são realizadas em diversas peças diferentes. Nesse tipo de fabricação, a qualidade dos produtos fica inteira e exclusivamente nas mãos do operador e, por conseguinte, sofre fortes variações. Portanto, o emprego de robôs industriais mostra-se vantajoso, uma vez que eles mantêm de forma constante a qualidade e a produtividade nas

[1] Preparação de máquina – alterações necessárias para tornar uma máquina, ou linha de produção, disponível à fabricação de um tipo de produto diferente do que estava produzindo anteriormente.

linhas. Em muitos casos, nesse tipo de produção, as peças são transportadas por linhas de roletes e não são guiadas, o que dificulta o emprego de robôs industriais. Porém o abastecimento randômico a células robotizadas reduz o custo de dispositivos auxiliares que posicionam as peças e aumenta a flexibilidade de um robô (CHENG; DENMAN, 2005). Nesse contexto, os processos exigem que o robô seja capaz de identificar a localização e a orientação de peças em uma célula de trabalho. Para contornar tais dificuldades, os sistemas de visão computacional podem ser úteis, auxiliando na pega das peças.

A visão computacional utiliza recursos de aprendizado e toma decisões baseadas nas entradas visuais do sistema para emular a visão humana (GONZALES; WOODS, 2002). Usar sistemas de visão em conjunto com um robô industrial aumenta a robustez e a eficiência do processo (JAMALUDDIN *et al*., 2006).

Guiar robôs industriais constitui uma tarefa que requer rapidez e precisão de um sistema de visão computacional. No caso de um robô utilizado para manipular peças que podem estar acondicionadas em locais previamente conhecidos ou não, um sistema de visão é capaz de auxiliar a definir a melhor posição de pega ou determinar a melhor trajetória para evitar obstáculos (FACON, 2002).

Este trabalho foi desenvolvido em parceria com a empresa Tupy S.A., que está implementando uma Célula de Oleação e Paletização de Cabeçotes Fundidos. Tal célula será instalada na última etapa do processo de acabamento de cabeçotes de motores a combustão da linha de Fundição E-II na unidade Joinville/SC.

Há três famílias distintas de cabeçotes manufaturados nessa linha, cada uma possui modelos para motores de seis ou quatro cilindros. No total são 26 modelos diferentes de cabeçotes, todavia são apenas seis geometrias diferentes, ou seja, as famílias possuem geometrias similares entre si. Os cabeçotes entrarão na linha com a região denominada de *fire deck*[2] virada para cima. Nessa região o cabeçote possui furos que servirão para determinar a sua posição e a sua orientação na linha de entrada da célula.

Um robô industrial modelo IRB6640 fabricado pela empresa ABB será utilizado para olear e paletizar todos os modelos de cabeçotes acabados nessa linha. Existem várias dificuldades no desenvolvimento dessa célula, uma delas é garantir que não haverá a mistura de peças diferentes em um mesmo *pallet*[3]. Outra dificuldade é executar a pega dos cabeçotes

---

[2] *Fire deck* – região de um motor a combustão onde são montadas as válvulas que liberam o combustível para a queima.
[3] *Pallet* – base de madeira utilizada para transporte de peças.

com o robô, uma vez que os cabeçotes são brutos e estão livres sobre a linha de roletes, assim, não há como garantir que os cabeçotes estarão sempre na mesma posição e orientação para o robô efetuar sua pega. Em uma abordagem tradicional, o problema seria resolvido mediante um dispositivo indexador mecânico para garantir a posição e a orientação da peça que o robô vai pegar. Esse tipo de dispositivo fixa a peça em um local conhecido pelo robô e o informa se ela está pronta ou não para executar a sua pega. Eles são pouco flexíveis e precisam ser trocados nas preparações de máquina, pois, na grande maioria das vezes, cada modelo de peça possui o seu indexador.

Este trabalho propõe uma solução para o robô industrial efetuar a pega dos cabeçotes, posicionados de forma aleatória, na entrada da célula de oleação e paletização. Para isso, recorreu-se a um sistema de visão computacional. O abastecimento do transportador que alimenta a célula é feito manualmente, por isso, a posição e a orientação das peças variam a cada cabeçote que entra na área de trabalho do robô. O problema da posição/orientação ocorreria de toda forma, porque nos roletes os cabeçotes estão livres. As funções do robô serão pegar uma peça na linha de entrada, mergulhá-la em um tanque com óleo, depositá-la em uma mesa rotativa para escorrer o excesso do óleo, retirar um cabeçote da mesa de escorrimento e por último posicioná-lo em um *pallet* de madeira. O sistema de visão será instalado na linha de entrada. Um bom sistema de visão permite ao robô industrial extrair informações de peças como posição, orientação e dimensão para manipulá-las corretamente em um tempo de ciclo pequeno (CHENG; DENMAN, 2005).

Existem algumas restrições impostas ao sistema desenvolvido, a saber:

- O tempo de processamento do sistema de visão deverá ser inferior a 10 segundos;
- A diferença entre os modelos de cabeçotes produzidos na célula não poderá interferir no resultado do processamento de imagem do sistema;
- O processamento de imagens deverá superar a falta de contraste entre os cabeçotes e o transportador de entrada da célula de trabalho;
- A solução deverá utilizar equipamentos de baixo custo.

1.1 OBJETIVOS

O objetivo geral deste trabalho é desenvolver um sistema de visão para guiar um robô industrial na pega de diferentes tipos de peças, dispostas livremente no transportador, de forma eficiente.

Alguns desafios devem ser superados pelo presente trabalho e, com base neles, foram definidos os objetivos específicos a serem alcançados, a destacar:

- Identificar o modelo da peça utilizando reconhecimento de padrões;
- Calcular a posição/orientação da peça a ser manipulada pelo robô;
- Utilizar programação paramétrica no robô;
- Implementar um protótipo (*software* e *hardware*) para testar o sistema de visão desenvolvido.

O que se espera da solução proposta é que ela resolva um problema real, no ambiente de uma indústria metalúrgica. Ela precisa ser vantajosa ante as soluções comerciais existentes.

1.2 CONTRIBUIÇÕES

A seguir são apresentadas as principais contribuições do trabalho.

- Identificação de peças utilizando um método de análise global de imagens. Uso da correlação de Pearson (MIRANDA NETO *et al*., 2009; MIRANDA NETO, 2007; YEN; JOHNSTON, 2005) para identificar o modelo dos cabeçotes que estarão na região de pega do robô;
- Aplicação da transformada de Hough para detectar furos nos cabeçotes que auxiliarão nos cálculos de posição/orientação;
- Detecção da posição/orientação de peças por meio de descritores: as coordenadas do centro dos furos segmentados servem para calcular a posição/orientação de cada cabeçote;
- Apresentação do artigo “Engine head blocks handling robot guided by vision system” (SEMIM *et al*., 2012) no congresso INCOM 2012[4], realizado na Romênia.

[4] INCOM 2012 – mais informações sobre o congresso em <http//:www.incom2012.ro>.

1.3 ORGANIZAÇÃO DO TRABALHO

A dissertação está dividida em sete capítulos. Após a introdução (Capítulo 1), conceitos de processamento de imagens utilizados na solução proposta e trabalhos correlatos são apresentados (Capítulo 2). O Capítulo 3 introduz a representação matemática e o emprego dos sistemas de coordenadas, bem como fundamentos de programação do robô usado no projeto. O problema abordado por este trabalho e a solução proposta estão expostos no Capítulo 4. A implementação da solução é mostrada no Capítulo 5. A análise e os resultados dos testes constam do Capítulo 6. O Capítulo 7 traz as considerações finais e propostas de trabalhos futuros.

2 VISÃO COMPUTACIONAL

Este capítulo introduz sistemas de visão e reconhecimento de padrões, bem como apresenta fundamentos de processamento de imagens importantes no entendimento da solução proposta abordada nos capítulos seguintes. Os conceitos utilizados na implementação do projeto estão descritos e agrupados de forma a facilitar a compreensão. No final do capítulo há alguns trabalhos correlatos que utilizam visão computacional para guiar robôs industriais.

2.1 SISTEMAS DE VISÃO

Os sistemas de visão propõem métodos de resolução de problemas inspirados na visão humana, utilizando recursos de aprendizado e também tomando decisões baseadas nas entradas visuais do sistema (GONZALES; WOODS, 2002). Os sistemas de visão, com o auxílio do conhecimento de diversas áreas, visam obter um conjunto de técnicas e metodologias que possam dar suporte ao desenvolvimento de aplicações práticas baseadas em imagens. Como exemplos, há a automação de processos de controle de qualidade, a identificação e a classificação de produtos e exploração de ambientes diversos (FACON, 2002).

Um sistema de visão é, basicamente, composto por: módulos de iluminação, sensores de visão, *hardware* de digitalização de imagens e um computador (STIVANELLO; GOMES, 2006; FACON, 2002). A escolha adequada da iluminação pode evidenciar algumas características que facilitam o emprego das ferramentas de processamento de imagens (ARAÚJO, 2010; DAVIES, 2005). Sensores de visão capturam imagens do objeto desejado; podem ser câmeras, alguns tipos de sensores ópticos etc. O *hardware* de digitalização de imagens coloca a imagem capturada pelo sensor de visão na memória, transforma as imagens contínuas em imagens digitais e cria uma imagem digitalizada, representada por uma matriz de números, cujos elementos são chamados *pixels* (*picture elements*). O computador executa os algoritmos de processamento de imagens, permitindo flexibilidade e custos de processamento e de memória relativamente baixos. Os algoritmos são desenvolvidos para possibilitar a escolha da informação adequada nas imagens, a ser interpretada e com base na qual serão tomadas as decisões.

2.1.1 Sistemas de Visão Aplicados à Indústria

Na indústria, os sistemas de visão requerem a utilização de técnicas de processamento de imagens associadas a uma base de conhecimento do produto (normas de concepção, regras

de produção etc.) e exigem técnicas para atender aos processos dentro de um determinado tempo de resposta (DAVIES, 2005; FACON, 2002).

Uma tarefa industrial muito comum é a inspeção visual. O processo de inspeção implica a medição de determinadas propriedades de um produto, como dimensões geométricas, superfícies, posição, orientação etc. Um dos alvos dos sistemas de visão consiste em analisar os defeitos cuja detecção seria impossível ou muito difícil por outros métodos (DAVIES, 2005), como, por exemplo, em linhas de envase de cerveja, onde milhares de garrafas são abastecidas por hora e é necessário inspecionar o nível do líquido em 100% da produção. A inspeção automatizada possibilita a quantificação de propriedades e coleta de dados de um produto que um inspetor humano não é capaz de realizar e permite uma realimentação constante e imediata do processo de manufatura (STIVANELLO; GOMES, 2006; FACON, 2002).

Outra tarefa industrial importante é guiar robôs; ela requer rapidez e precisão. Um robô serve para manipular peças que podem ser ordenadas ou montadas com outros tipos de peças. Estas podem estar acondicionadas em locais previamente conhecidos ou não. Os sistemas de visão computacional podem auxiliar o robô a escolher uma peça entre várias outras, ajudar a definir a melhor posição de pega ou determinar a melhor trajetória para evitar obstáculos (DAVIES, 2005; FACON, 2002).

2.2 RECONHECIMENTO DE PADRÕES

O reconhecimento de padrões visa determinar um mapeamento que relacione as propriedades extraídas de um conjunto de rótulos, apresentando a restrição de que amostras com características semelhantes devem ser mapeadas ao mesmo rótulo. Amostras distintas e com um mesmo rótulo pertencem a uma mesma classe (PEDRINI; SCHWARTZ, 2008).

Uma classe é uma família de padrões que compartilha propriedades em comum. Reconhecimento de padrões em sistemas de visão requer técnicas para atribuir padrões a suas respectivas classes, sem o envolvimento do ser humano (GONZALES; WOODS, 2002).

A exigência crescente de informatização e automatização de tarefas humanas repetitivas e cansativas levou os pesquisadores a desenvolverem ferramentas específicas para atender a aplicações cujo conhecimento inicial é pobre ou incompleto e onde certo grau de inteligência é importante (FACON, 2002).

Dentro de reconhecimento de padrões, podem-se citar o reconhecimento de caracteres (OCR *Optical Character Recognition*), o reconhecimento de códigos gráficos (barras ou *datamatrix*), aplicações biométricas (reconhecimento de pessoas ou impressão digital), além

da segregação e contagem de peças individuais ou em lotes.

2.3 PROCESSAMENTO DE IMAGENS

O processamento de imagens permite modificar, analisar e manipular imagens digitais por intermédio de recursos computacionais (STIVANELLO; GOMES, 2006; FACON, 2002). Para Gonzáles e Woods (2002), o processamento de imagens pode ser definido como uma disciplina em que entrada e saída do processamento são imagens. A essa definição adiciona-se a extração de atributos de imagem, incluindo o reconhecimento de objetos. Ainda para Gonzáles e Woods (2002), consideram-se três os tipos de processos computadorizados entre o processamento de imagens e a visão computacional: baixo, médio e alto nível de processamento. Baixo nível de processamento envolve operações primitivas, como redução de ruídos, melhoramento de contraste e nitidez. No baixo nível, ambos, entrada e saída, são imagens. Médio nível de processamento refere-se a tarefas como segmentação (partição de imagens em regiões ou objetos), descrição daqueles objetos para reduzi-los a uma forma adequada para processamento computacional, e classificação (reconhecimento) de objetos individuais. Um médio nível é caracterizado pelo fato de que suas entradas são imagens, mas suas saídas são atributos extraídos dessas imagens (cantos, bordas, contornos e objetos). Alto nível de processamento envolve a interpretação de um conjunto de objetos reconhecidos, como em análise de imagem, realizando funções cognitivas.

Um sistema de processamento digital de imagens é normalmente constituído por cinco etapas: aquisição, pré-processamento, segmentação, representação e descrição, reconhecimento e interpretação (PEDRINI; SCHWARTZ, 2008). A etapa de aquisição captura a imagem por meio de um dispositivo e converte-a em uma representação adequada para o processamento posterior. A imagem digital resultante do processo de aquisição pode apresentar imperfeições. A etapa do pré-processamento visa melhorar a qualidade da imagem mediante a aplicação de técnicas que atenuam o ruído, corrigem o contraste ou brilho e suavizam determinadas propriedades da imagem. A etapa da segmentação realiza a extração e identificação de áreas de interesse contidas na imagem. Essa etapa é geralmente baseada na detecção de bordas ou regiões na imagem. Estruturas adequadas de representação devem ser utilizadas para armazenar e manipular os objetos de interesse extraídos da imagem. O processo de descrição visa à extração de características ou propriedades que possam ser úteis na discriminação entre classes de objetos. Essas características são, em geral, descritas por atributos numéricos que formam um vetor de características. A última etapa envolve o reconhecimento e a interpretação dos componentes de uma imagem. Reconhecimento ou

classificação é o processo que atribui um identificador ou rótulo aos objetos da imagem, baseado nas características providas pelos seus descritores. O processo de interpretação consiste em atribuir um significado ao conjunto de objetos reconhecidos (PEDRINI; SCHWARTZ, 2008).

O objetivo do processamento de imagens é tornar uma imagem mais adequada para uma aplicação específica, e isso implica necessidades diferentes para aplicações distintas, ou seja, o que pode ser muito utilizável para uma aplicação pode não ser tão aplicável em outra (PEDRINI; SCHWARTZ, 2008). A seguir, serão apresentadas algumas ferramentas usadas em processamento de imagens digitais.

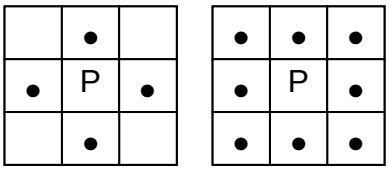
Neste trabalho não serão abordadas ferramentas de processamento de imagem para imagens coloridas.

2.3.1 Fundamentos de Imagens Digitais

Uma imagem em níveis de cinza pode ser definida como uma função *f(x,y)*, em que *x* e *y* são coordenadas espaciais e a amplitude de *f* é chamada de intensidade daquele ponto (GONZALES; WOODS, 2002). A imagem digital é composta por um número finito de elementos, cada um dos quais possui uma localização e um valor particular. Esses elementos são conhecidos como *pixels*. Dessa forma, uma imagem pode ser definida como uma matriz de *pixels*, em que o valor de cada *pixel* é proporcional ao brilho correspondente a cada ponto que forma uma cena capturada (NIXON; AGUADO, 2009).

Vizinhança de um *pixel* é definida como o grupo de *pixels* ao redor de um ponto *P* pertencente à imagem. Tal agrupamento pode ser do tipo Vizinhança-4 ou Vizinhança-8. Vizinhança-4 abrange os 4 *pixels* ao redor do ponto *P* sem contar com os *pixels* das diagonais que passam por *P*. Vizinhança-8 inclui os 4 *pixels* ao redor do ponto *P* e também os 4 *pixels* das diagonais que passam por *P* (FACON, 2002). A Fig. 2.1(a) mostra a Vizinhança-4, e a Fig. 2.1(b) traz um exemplo de Vizinhança-8.

Fig. 2.1: Exemplos típicos de vizinhança. (a) Vizinhança-4 e (b) Vizinhança-8



(a) (b)

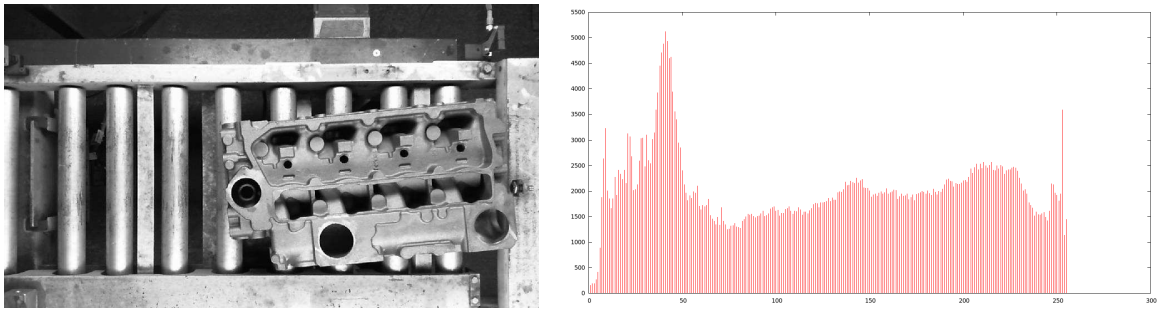
Fonte: produção próprio autor

A distribuição dos níveis de intensidades de uma imagem pode ser chamada de histograma. Ele é representado por um gráfico que indica a quantidade de *pixels* na imagem para cada nível de cinza (PEDRINI; SCHWARTZ, 2008), como evidencia a Fig. 2.2. A análise do conteúdo de um histograma fornece informações sobre o brilho e o contraste de uma imagem (SILVA; RIBEIRO, 2009). Ele pode ser representado pela função discreta mostrada na equação 2.1 (GONZALES; WOODS, 2002):

$$h(r_k) = n_k \tag{2.1}$$

Em que n_k é a quantidade de *pixel*s para cada valor de intensidade r_k (GONZALES; WOODS, 2002).

Fig. 2.2: Imagem (a) e seu histograma (b)



(a) (b)

Fonte: produção próprio autor

A transformação de brilho/contraste é uma função *T* para mapeamento de intensidades, descrita como:

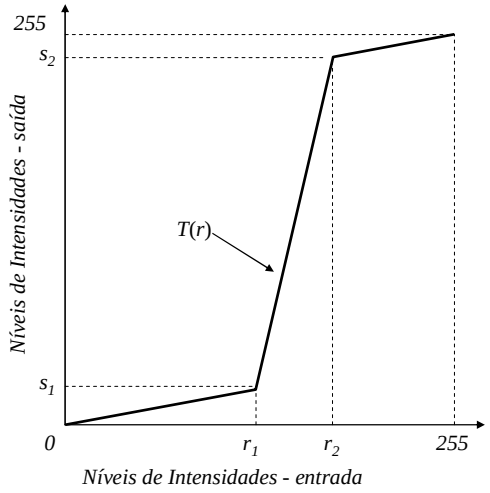
$$g = T(f) \tag{2.2}$$

Em que *g* representa o nível de cinza dos *pixels* da imagem modificada e *f* o nível de cinza dos *pixels* da imagem inicial.

A função *T* determina qual será o efeito visual obtido e deve ser escolhida de acordo com o desejado (PEDRINI; SCHWARTZ, 2008; PRATT, 2007). A Fig. 2.3 apresenta a forma de uma

função de transformação *T* que atenua os *pixels* com intensidades menores a r_1 e amplifica aqueles com intensidades maiores que r_2.

Fig. 2.3: forma da função de transformação *T*



Fonte: produção próprio autor

A transformação de brilho/contraste foi utilizada neste projeto para aumentar o contraste na região dos furos dos cabeçotes. O objetivo é tornar mais robusta a segmentação dessas circunferências.

2.3.2 Filtro Linear no Domínio Espacial

A operação de filtragem produz uma nova imagem a partir da imagem de entrada. Filtragem no domínio espacial refere-se a uma operação de vizinhança aplicada no plano da imagem (PEDRINI; SCHWARTZ, 2008; GONZALES; WOODS, 2002). Alguns filtros necessitam do auxílio de uma máscara (como pode ser visto na Fig. 2.4) aplicada sobre a imagem original, para gerar o efeito desejado. A filtragem ajuda a eliminar alguns ruídos, a intensificar as transições de intensidades ou suavizar contornos (PEDRINI; SCHWARTZ, 2008; GONZALES; WOODS, 2002).

Filtros lineares[5] são dados por um somatório dos produtos entre os pesos da máscara e os correspondentes *pixels* da imagem, resultando um novo valor ao *pixel* central na imagem resultante. O filtro é aplicado *pixel* a *pixel* de modo a efetuar uma varredura por toda a

[5] Outro tipo de filtro é o não linear, que utiliza operações matemáticas não lineares, como, por exemplo, mínimo, máximo e mediana da vizinhança de cada *pixel*. Neste trabalho não serão aplicados esses tipos de filtros.

imagem. Em geral, um filtro linear aplicado em uma imagem *f*, de dimensão *M* x *N* com máscara *w* de tamanho *m* x *n* (*m* e *n* ímpares), é dado por (GONZALES; WOODS, 2002):

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) = f * w \tag{2.3}$$

Em que $a = (m\text{-}1)/2$, $b = (n\text{-}1)/2$, $x = [0, M\text{-}1]$, $y = [0, N\text{-}1]$ e "*" denota a operação de convolução.

Filtros lineares também podem ser chamados de Convolução, que é dada pela seguinte fórmula:

$$R = w_1 z_1 + w_2 z_2 + \ldots + w_{mn} z_{mn} = \sum_{i=1}^{mn} w_i z_i \tag{2.4}$$

Em que w_i são os pesos da máscara, z_i são as intensidades da imagem na posição da máscara e *mn* o tamanho da máscara.

Um exemplo de filtro linear é o filtro de Sobel (O'GORMAN; SAMMON; SEUL, 2009; GONZALES; WOODS, 2002), que realça cantos e bordas (alta frequência), porém atenua pequenas variações de intensidades (baixa frequência) na imagem. Suas máscaras podem ser vistas na Fig. 2.4.

Fig. 2.4: Máscaras para o Filtro de Sobel, (a) vertical e (b) horizontal

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

(a)

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

(b)

Fonte: Gonzalez e Woods (2002)

Se for aplicada a convolução utilizando apenas uma máscara, como na equação 2.9, o resultado do filtro de Sobel é uma imagem com realce de bordas em uma única direção, entretanto se a equação 2.5 for usada, o resultado será a intensificação dos contornos na imagem.

$$g = \sqrt{(f * m_1)^2 + (f * m_2)^2} \tag{2.5}$$

Em que *g* é a imagem filtrada, *f* é a imagem original, m_1 e m_2 são as máscaras de Sobel.

2.4 SEGMENTAÇÃO

A essência da segmentação é a identificação de objetos por meio de um processo de classificação de diferentes características de acordo com os *pixels* de uma imagem (GUO; JU;

YAO, 2009). Ela tem como objetivo dividir a imagem em elementos significativos buscando isolar os objetos de interesse (STIVANELLO; GOMES, 2006).

A segmentação separa as regiões de interesse, ou objetos, que constituem uma imagem. O nível dessa separação depende do problema a ser resolvido. Segmentação de imagens é a tarefa mais difícil no processamento de imagens (GONZALES; WOODS, 2002).

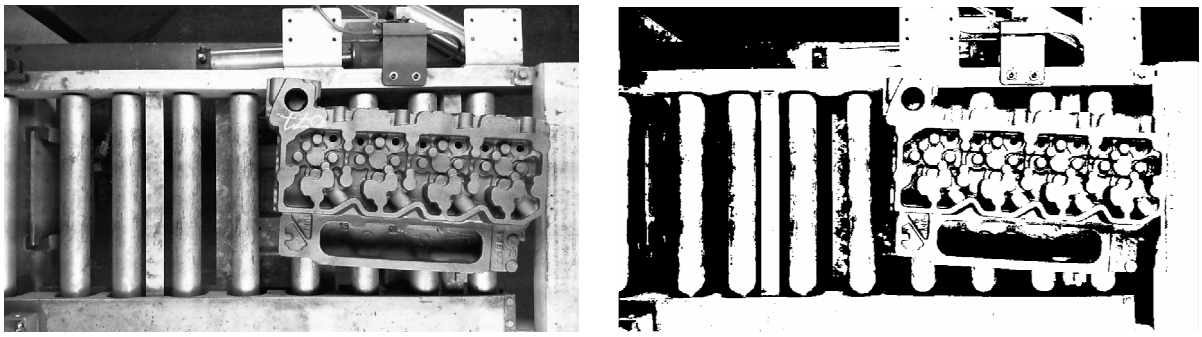
Um processo de segmentação que identifique corretamente a localização, a topologia e a forma dos objetos constitui um requisito de fundamental importância para que as informações resultantes de um sistema de análise de imagens sejam confiáveis (PEDRINI; SCHWARTZ, 2008).

2.4.1 *Thresholding*

O *thresholding* (limiarização) converte uma imagem em tons de cinza para a forma binária (GONÇALVES, 2004), ou seja, seus *pixels* podem assumir somente os valores 0 (preto) e 1 (branco). Uma função do *thresholding* é separar, dentro de uma imagem, o fundo dos objetos de interesse (ARAÚJO, 2010). Esse operador seleciona *pixels* que possuem um valor particular ou que estão dentro de uma faixa específica (NIXON; AGUADO, 2009). Uma imagem em tons de cinza pode ser binarizada utilizando um valor de *threshold T* tal que os *pixels* com intensidades maiores que *T* assumem o valor 1 (branco) e os *pixels* com intensidades menores ou iguais a *T* assumem o valor 0 (preto).

O objetivo do *thresholding* é marcar os *pixels* pertencentes às regiões de interesse com o valor 1 (branco) e o fundo da imagem com valor 0 (preto) (FACON, 2002). A tarefa mais difícil nesse processo é escolher o melhor nível de *threshold*; algumas condições podem dificultar, como imagens pobres em contraste ou com níveis desiguais de iluminação (O'GORMAN; SAMMON; SEUL, 2009). O *thresholding* mostra-se útil na etapa de segmentação de imagens, pois pode ser utilizado para encontrar objetos em uma imagem se sua intensidade de brilho for conhecida (NIXON; AGUADO, 2009), além da simplificação das imagens, tornando-as mais apropriadas à caracterização e ao reconhecimento.

Fig. 2.5: *Threshold* global, (a) imagem original e (b) imagem binarizada



(a) (b)

Fonte: produção próprio autor

O *thresholding* pode ser feito de forma global ou adaptativa. O *thresholding* global é o mais simples, ele parte de um determinado valor *T* da escala de intensidades da imagem. Todos os *pixels* com intensidades maiores assumem o valor 1 (GONZALES; WOODS, 2002), como mostrado na Fig. 2.5. A seleção de um único nível de *threshold* não é em geral adequada para imagens que contêm variações nos níveis de cinza entre objetos e fundo (PEDRINI; SCHWARTZ, 2008). Nesses casos o *threshold* adaptativo mostra-se uma alternativa mais apropriada, que usa mais de um valor de corte para a aplicação do *threshold* (GONZALES; WOODS, 2002).

O *threshold* pode ser visto como um problema estatístico cujo objetivo é minimizar o erro médio entre dois ou mais grupos de *pixels*, também chamados de classes (GONZALES; WOODS, 2002). O método de Otsu faz a seleção do valor de *threshold T* automaticamente (NIXON; AGUADO, 2009); sua principal ideia é que as classes deveriam se distinguir por meio dos valores de intensidades dos seus *pixels*. Por consequência, um valor de *threshold* que separe as classes, em termos de suas intensidades, seria o melhor valor de *threshold* a ser escolhido (GONZALES; WOODS, 2002).

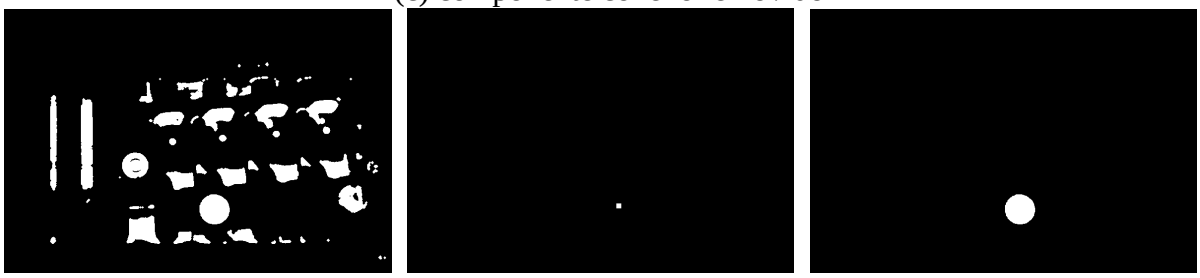
2.4.2 Segmentação por Crescimento de Regiões

A segmentação baseada em crescimento de regiões é um procedimento que agrega *pixels*, com propriedades similares, em determinadas regiões (GUO; JU; YAO, 2009; DAVIES, 2005). A abordagem básica é iniciar com um conjunto de *pixels* denominados *sementes* e, a partir deles, crescer as regiões anexando a cada ponto semente outros *pixels* que possuam propriedades similares (ou o mesmo valor para imagens binárias) e, assim, segmentar essa região do fundo da imagem. O crescimento segue até que não haja mais *pixels* similares à semente. Os *pixels* sementes podem ser escolhidos de maneira aleatória,

determinística ou selecionados pelo usuário (PEDRINI; SCHWARTZ, 2008). A Fig. 2.6 apresenta um exemplo de aplicação do crescimento de regiões.

Neste projeto a segmentação por crescimento de regiões (reconstrução de componente conexo) foi aplicada no espaço de parâmetros binarizado, extraindo as regiões que representam centros de circunferências.

Fig. 2.6: Segmentação por crescimento de regiões. (a) imagem inicial, (b) semente e (c) componente conexo removido



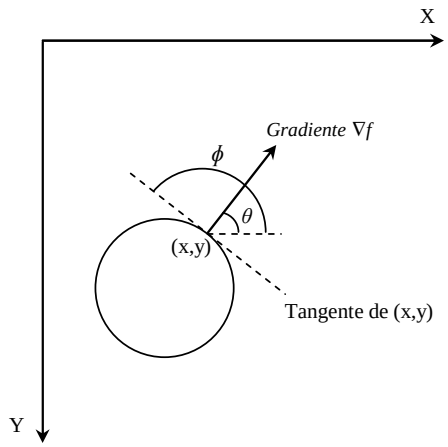
(a) (b) (c)

Fonte: produção próprio autor

2.4.3 Detecção de Bordas e Operador Gradiente

A detecção de bordas identifica as mudanças significativas nos níveis de cinza da imagem, e tais mudanças podem ser descritas por meio do conceito de derivada. O operador gradiente é um vetor cuja direção indica os locais nos quais os níveis de cinza sofrem maior variação. A Fig. 2.7 mostra que a direção do gradiente é perpendicular à direção tangente da borda (PEDRINI; SCHWARTZ, 2008).

Fig. 2.7: Gradiente em uma borda



Fonte: Pedrini e Schwartz (2008)

O vetor gradiente na posição (*x,y*) pode ser dado pelas seguintes derivadas parciais:

$$\nabla f(x,y) = \frac{\partial f(x,y)}{\partial x}\mathbf{i} + \frac{\partial f(x,y)}{\partial y}\mathbf{j} \tag{2.6}$$

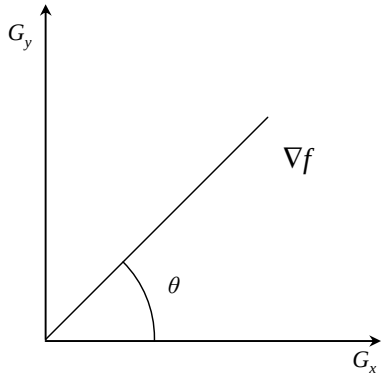
Em que **i** e **j** são os vetores unitários nas direções *x* e *y*, respectivamente.

A direção do vetor gradiente $\theta(x,y)$ no ponto (x,y) é dada por:

$$\theta(x,y) = \arctan\left(\frac{G_y}{G_x}\right) = \arctan\left(\frac{\dfrac{\partial f}{\partial y}}{\dfrac{\partial f}{\partial x}}\right) \tag{2.7}$$

Em que G_x e G_y são os operadores de realce de borda, respectivamente, no sentido vertical e horizontal. A Fig. 2.8 traz a direção do θ gradiente.

Fig. 2.8: Direção do gradiente



Fonte: produção próprio autor

Na detecção de bordas, o tamanho do vetor gradiente ∇f é importante, e ele é dado por:

$$\nabla f = \sqrt{G_x^2 + G_y^2} = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2} \tag{2.8}$$

Existem alguns operadores para detecção de bordas, um deles é o de Sobel, no qual, aplicando a convolução com os seus operadores, se identificam as bordas da imagem. A Fig. 2.4 apresenta os operadores de Sobel.

Para encontrar a imagem *G* com as bordas realçadas, é necessário fazer a Convolução Periódica, equação 2.9, para os dois operadores separadamente, ou seja,

$$\begin{aligned} Sobel_{vertical} &= \sum_{i=1}^{mn} G_{x_i} z_i \\ Sobel_{horizontal} &= \sum_{i=1}^{mn} G_{y_i} z_i \end{aligned} \tag{2.9}$$

Em que $Sobel_{vertical}$ e $Sobel_{horizontal}$ são o resultado da Convolução Periódica na direção vertical e horizontal, respectivamente. O próximo passo é encontrar a imagem resultante *G*,

ou seja, a imagem com as bordas realçadas, horizontais e verticais. Para isso aplica-se a seguinte equação:

$$G = \sqrt{\left(Sobel_{vertical}\right)^2 + \left(Sobel_{horizontal}\right)^2} \tag{2.10}$$

Para o realce de bordas utilizando os operadores de Sobel, a direção θ gradiente é calculada da seguinte forma:

$$\theta(x, y) = \arctan\left(\frac{Sobel_{vertical}(x, y)}{Sobel_{horizontal}(x, y)}\right) \tag{2.11}$$

2.4.4 Descritores de Características

Descritores representam e descrevem regiões, segmentadas de uma imagem, em uma forma adequada a processamentos computacionais (GONZALES; WOODS, 2002). Podem-se citar alguns descritores como:

- o perímetro, que representa o comprimento da borda de um objeto. Para encontrá-lo, basta contar os *pixels*, associados ao objeto de interesse, que possuam ao menos um vizinho igual a zero no caso de imagens binárias (SILVA, 2009);
- a área de um objeto, que pode ser descrita como o número de *pixels* da região de interesse (SILVA, 2009);
- o centroide, coordenada central de um objeto na imagem (SILVA, 2009). As equações para encontrar o centroide de uma região são dadas a seguir:

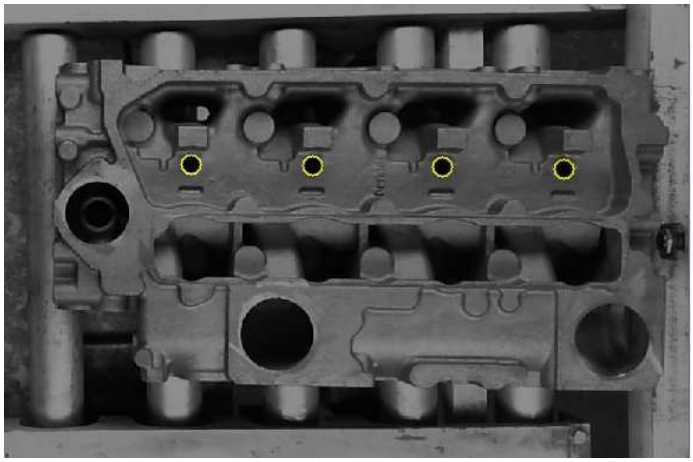
$$x_c = \frac{\sum_{i=1}^{n} x_i}{n} \;\; ; \;\; y_c = \frac{\sum_{i=1}^{n} y_i}{n} \tag{2.12}$$

Sendo x_i e y_i as coordenadas pertencentes ao objeto.

2.4.5 Transformada de Hough para Detecção de Círculos

A Transformada de Hough constitui um método de extração de parâmetros em imagens. Ela utiliza detalhes da imagem para acumular evidências do possível modelo que está sendo buscado (ZHONG *et al*., 2007; ILLINGWORTH; KITTLER, 1987). O algoritmo da transformada de Hough permite detectar padrões como linhas, círculos e elipses, mesmo se eles estiverem incompletos ou possuírem ruídos (TRESPADERNE; LÓPEZ, 2009; ILLINGWORTH; KITTLER, 1987), como se vê na Fig. 2.9.

Fig. 2.9: Círculos detectados pela Transformada de Hough em um cabeçote



Fonte: produção próprio autor

A transformada de Hough serve para detectar curvas cujas funções analíticas são conhecidas (ZHONG *et al*., 2007), mapeando detalhes, presentes no espaço da imagem, em um grupo de pontos no espaço de parâmetros (ILLINGWORTH; KITTLER, 1987). A ideia é aplicar na imagem uma transformação tal que todos os pontos pertencentes a uma mesma curva sejam mapeados num único ponto de um novo espaço de parametrização da curva procurada (ZHONG *et al*., 2007; FACON, 2002). O mapeamento é alcançado de uma maneira computacionalmente eficiente, baseado na função que descreve a forma desejada. Esse mapeamento requer menos recursos computacionais do que um casamento de padrões (NIXON; AGUADO, 2009).

O princípio básico da transformada de Hough, para detecção de círculos, é que retas perpendiculares à borda de uma circunferência se interceptam no seu centro (BOROVICKA, 2003; ILLINGWORTH; KITTLER, 1987).

Uma etapa necessária à aplicação da transformada de Hough é a detecção de bordas na imagem (DUARTE, 2003).

A transformada de Hough tem como vantagens o fato de que pode ser aplicada ao tratamento de qualquer tipo de curva com descrição analítica e apresenta robustez para imagens fortemente ruidosas (DUARTE, 2003), no entanto pode ser aplicada somente em imagens binarizadas.

Seja a equação da circunferência dada por:

$$(x-a)^2+(y-b)^2=r^2 \tag{2.13}$$

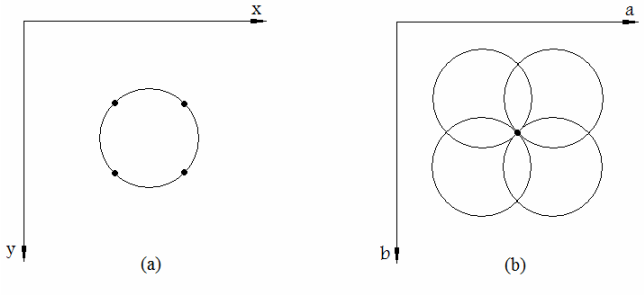
Em que (a,b) são as coordenadas do centro e r o raio da circunferência.

Supondo que (x,y) seja constante e que (a,b) seja variável, tem-se então a circunferência descrita no plano ab, e esse plano passa a ser conhecido como espaço de parâmetros. Cada

célula no espaço de parâmetros é um acumulador, no caso das circunferências, o acumulador inclui também o raio *r*, portanto a estrutura de acumulação é tridimensional (NIXON; AGUADO, 2009). Cada vez que uma circunferência intercepta um ponto no acumulador, esse ponto deve ser incrementado em 1. Se os parâmetros (*a*,*b*,*r*) são frequentemente encontrados no espaço de parâmetros, é muito provável que uma circunferência com raio *r* e centro (*a*,*b*) realmente exista na imagem. Dessa forma, picos no espaço de parâmetros corresponderão aos centros das circunferências no plano da imagem (O'GORMAN; SAMMON; SEUL, 2009; NIXON; AGUADO, 2009; DAVIES, 2005).

Neste trabalho utilizou-se o raio *r* como um parâmetro de entrada da função. Dessa forma, a estrutura do espaço de parâmetros passa a ser bidimensional, como mostrado na Fig. 2.10.

Fig. 2.10: Detecção de circunferência pela transformada de Hough. (a) Dados originais no espaço *x*, *y* e (b) células correspondentes do acumulador no espaço de parâmetros *ab*



Fonte: Pedrini e Schwartz (2008)

As equações paramétricas da circunferência em coordenadas polares são:

$$x = a + r\cos\theta$$
$$y = b + rsen\theta \qquad (2.14)$$

No espaço de parâmetros tem-se que

$$a = x - r\cos\theta$$
$$b = y - rsen\theta \qquad (2.15)$$

Dada a direção θ do gradiente em um ponto (*x*,*y*) da borda, calculam-se os valores de $sen\theta$ e $cos\theta$. Unindo as equações 2.14 e 2.15, o raio da circunferência pode ser eliminado tal que

$$b = a\tan\theta - x\tan\theta + y \qquad (2.16)$$

Partindo da direção θ do gradiente em cada ponto da borda (*x*,*y*), pode-se encontrar um valor para (*a*,*b*) e incrementar essa célula de acumulação no espaço de parâmetros *ab*, de acordo com a equação 2.16. Assim, os pontos de máximos no espaço de parâmetros corresponderão aos centros das circunferências no plano da imagem.

A função da transformada de Hough neste trabalho é segmentar os furos dos cabeçotes que serão empregados para calcular sua posição/orientação.

2.5 MORFOLOGIA MATEMÁTICA

A morfologia matemática pode ser definida como uma teoria para análise de estruturas espaciais, como forma e superfície dos objetos (SOILLE, 2004). Ela utiliza a teoria de conjuntos para representar a forma dos objetos em uma imagem (PEDRINI; SCHWARTZ, 2008). Operadores morfológicos visam extrair estruturas relevantes da imagem considerada. Alcança-se isso varrendo a imagem com um conjunto conhecido de *pixels* apropriado à análise morfológica (SILVA, 2009; SOILLE, 2004), chamado de *elemento estruturante*, e com base em seu formato e tamanho, é capaz de testar e quantificar se o *elemento estruturante* está ou não contido no objeto analisado (GONZALES; WOODS, 2002).

2.5.1 Dilatação e Erosão Binária

O princípio da dilatação binária é percorrer, ponto a ponto, o objeto analisado pelo elemento estruturante refletido (rotacionado em torno da sua própria origem). Se ao menos um ponto do elemento estruturante estiver contido no objeto, o ponto é marcado como verdadeiro e, assim, o novo objeto é criado (PEDRINI; SCHWARTZ, 2008; SOILLE, 2004; DOUGHERTY; LOTUFO, 2003; GONZALES; WOODS, 2002). Dilatação binária aumenta as regiões brancas (valor = 1) da imagem de acordo com o tamanho e forma do *elemento estruturante* (SILVA, 2009). Como resultado, tem-se um objeto maior que o original, proporcional ao elemento estruturante. A equação 2.17 mostra a dilatação para imagens binárias.

$$A \oplus B = \left\{ (\overline{B})_p \middle| p \in A \right\} \tag{2.17}$$

Em que A é a imagem inicial, B o elemento estruturante, $\overline{B}$ o elemento estruturante refletido, $(\overline{B})_p$ uma translação por p do elemento estruturante refletido e p um ponto que possui coordenadas (x,y) de A.

O princípio da erosão binária é percorrer, ponto a ponto, o objeto analisado pelo elemento estruturante, e se este estiver totalmente contido no objeto, o ponto é marcado como verdadeiro e, dessa forma, o novo objeto é criado (PEDRINI; SCHWARTZ, 2008; SOILLE, 2004; DOUGHERTY; LOTUFO, 2003; GONZALES; WOODS, 2002). A erosão binária pode ser considerada o operador dual da dilatação, aumentando as regiões pretas (valor = 0) da imagem de acordo com o tamanho e forma do *elemento estruturante* (SILVA, 2009). Como resultado, tem-se um objeto menor que o original, proporcional ao elemento estruturante. A equação 2.18 mostra a erosão para imagens binárias.

$$A\Theta B = \left\{ p \middle| (B)_p \subseteq A \right\} \tag{2.18}$$

Em que *A* é uma imagem inicial, *B* o elemento estruturante, $(B)_p$ uma translação do elemento estruturante por *p* e *p* um ponto que possui coordenadas (*x*,*y*) de A.

A Fig. 2.11 apresenta as duas operações morfológicas básicas, dilatação e erosão (DOUGHERTY; LOTUFO, 2003).

Fig. 2.11: Exemplo de dilatação e erosão em imagem binária. (a) Imagem inicial, (b) imagem dilatada e (c) imagem erodida

(a) (b) (c)

Fonte: produção próprio autor

2.5.2 Abertura e Fechamento

Abertura e fechamento são duas operações morfológicas importantes. A abertura serve para suavizar o contorno de objetos, eliminar conexões estreitas entre objetos e remover saliências. O fechamento é utilizado para fundir separações estreitas entre objetos, eliminar pequenos buracos e preencher lacunas no contorno (PEDRINI; SCHWARTZ, 2008; GONZALES; WOODS, 2002).

O princípio da abertura consiste em dilatar uma imagem previamente erodida por intermédio do mesmo elemento estruturante, em geral nem todas as estruturas são recuperadas. Abertura de A por B é igual à erosão de A por B seguida de uma dilatação por B. Assim, sua equação pode ser escrita como (SOILLE, 2004; DOUGHERTY; LOTUFO, 2003):

$$A \circ B = (A\Theta B) \oplus B \tag{2.19}$$

O efeito da abertura é o mesmo de passar uma “enceradeira” por dentro do objeto. Desse modo, o objeto resultante é menor ou igual ao original (DOUGHERTY; LOTUFO, 2003).

A ideia principal do fechamento é reconstruir a forma inicial de uma imagem dilatada por meio de uma erosão. Fechamento de A por B é igual à dilatação de A por B seguida de uma erosão por B. Sua equação é dada a seguir (SOILLE, 2004; DOUGHERTY; LOTUFO, 2003):

$$A \bullet B = (A \oplus B) \Theta B \tag{2.20}$$

O efeito do fechamento é o dual da abertura, ou seja, uma “enceradeira” por fora de tal maneira que o objeto resultante seja maior ou igual ao objeto inicial (DOUGHERTY; LOTUFO, 2003).

Neste trabalho a abertura foi usada para conectar pequenas regiões, na imagem resultante da limiarização do espaço de parâmetros, após a aplicação da transformada de Hough. O fechamento também foi aplicado em uma imagem binária, porém com o objetivo de remover pequenas regiões desnecessárias à segmentação dos furos dos cabeçotes.

2.5.3 Fechamento de Buracos

Para Soille (2004), fechamento de buracos em imagens em níveis de cinza remove todos os mínimos que não estão conectados à borda da imagem. Soille (2004) sugere a remoção desses mínimos utilizando reconstrução morfológica por erosão. O marcador empregado na reconstrução é uma imagem com valores máximos, exceto ao longo de sua borda, onde os valores da imagem original são mantidos. A Fig. 2.12 apresenta o fechamento de buracos em um sinal *f*, em que todos os mínimos regionais de *f* são removidos pela reconstrução morfológica de *f* a partir do marcador f_m.

Um conceito importante é a erosão condicional:

$$\varepsilon_f^1(f_m) = (f_m \Theta b) \vee f \tag{2.21}$$

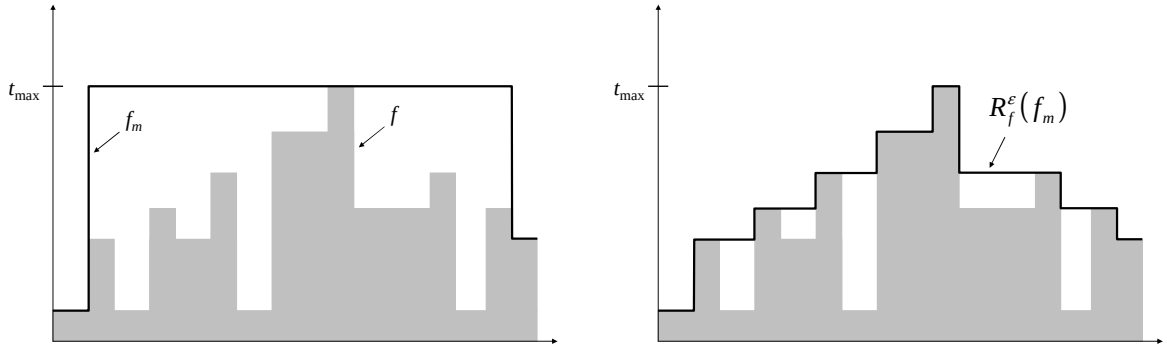
Em que $\vee$ denota um operador de máximo pontual, f_m é o marcador, *f* a imagem e *b* um elemento estruturante. Obtém-se o resultado dessa equação computando a erosão de f_m por *b* e depois selecionando o valor máximo entre a erosão e *f*, para cada ponto analisado.

O fechamento de buracos mediante reconstrução morfológica por erosão pode ser dado por:

$$FB(F) = R_f^{\varepsilon}(f_m) = \varepsilon_f^{(i)}(f_m) \tag{2.22}$$

Em que $\varepsilon_f^{(i)}(f_m) = \varepsilon_f^{(1)}[\varepsilon_f^{(i-1)}(f_m)]$ e $\varepsilon_f^{(0)}(f_m) = f_m$. As iterações devem ser feitas até que a estabilidade seja atingida, ou seja, $\varepsilon_f^{(i)}(f_m) = \varepsilon_f^{(i+1)}(f_m)$.

Fig. 2.12: (a) imagem original *f* e marcador f_m, (b) reconstrução por erosão de *f* a partir de f_m

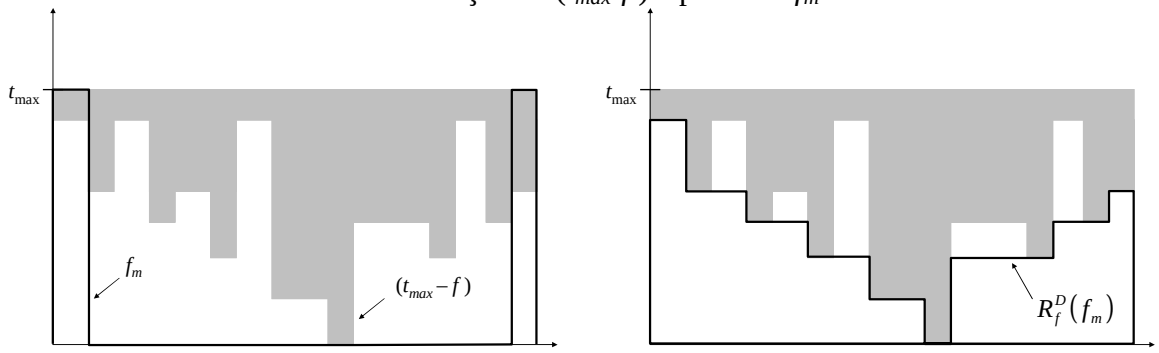


(a) (b)

Fonte: adaptado de Soille (2004)

Outra forma para o fechamento de buracos é utilizar a reconstrução morfológica por dilatação. Para tanto faz-se necessário trabalhar com o negativo da imagem. Nesse caso, o marcador possui os *pixels* da borda com valor máximo e os demais com valor mínimo. A Fig. 2.13 mostra o fechamento de buracos em um sinal *f* negado ($t_{max} - f$). Ao fim da reconstrução morfológica por dilatação, é preciso encontrar o negativo do resultado para ter o sinal inicial com os buracos fechados.

Fig. 2.13: (a) negativo da imagem inicial (t_{max}-*f*) e o marcador f_m, (b) reconstrução por dilatação de (t_{max}-*f*) a partir de f_m



(a) (b)

Fonte: produção próprio autor

Para a reconstrução por dilatação, introduz-se a dilatação condicional:

$$D_f^1(f_m) = (f_m \oplus b) \wedge f \tag{2.23}$$

Em que $\wedge$ denota um operador de mínimo pontual, f_m é o marcador, *f* a imagem e *b* um elemento estruturante. O resultado dessa equação é obtido computando a dilatação de f_m por *b* e depois selecionando o valor mínimo entre a dilatação e *f*, para cada ponto analisado.

A equação para o fechamento de buracos utilizando reconstrução morfológica por dilatação pode ser dada por:

$$FB(F) = \left(t_{\max} - R_f^D(f_m)\right) = \left(t_{\max} - D_f^{(i)}(f_m)\right) \quad (2.24)$$

Em que $D_f^{(i)}(f_m) = D_f^{(1)}[D_f^{(i-1)}(f_m)]$ e $D_f^{(0)}(f_m) = f_m$. As iterações devem ser feitas até que a estabilidade seja atingida, ou seja, $D_f^{(i)}(f_m) = D_f^{(i+1)}(f_m)$.

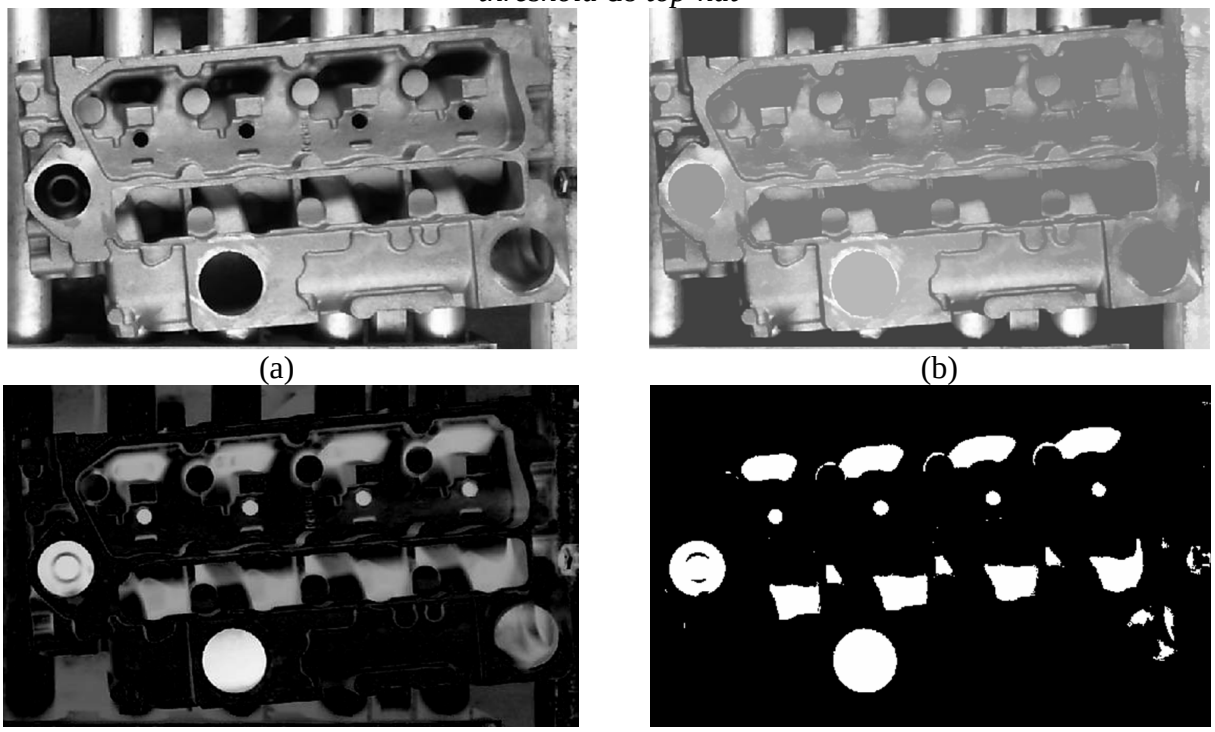
No projeto, o fechamento de buracos é a primeira operação realizada. Ele fecha os furos dos cabeçotes que serão segmentados nas etapas subsequentes.

2.5.4 *Top-Hat*

Uma abertura ou um fechamento que emprega um elemento estruturante que não se insere em componentes conexos pode ser usado para remover essas estruturas da imagem. Elas são reconstruídas por meio da subtração entre a imagem e sua abertura ou entre o fechamento da imagem e a própria imagem (SOILLE, 2004; GONZALES; WOODS, 2002). Essa diferença aritmética é a base da definição do *top-hat*.

A diferença aritmética entre uma imagem e sua abertura é chamada de *white top-hat*, e a subtração entre o fechamento e a imagem original recebe o nome de *black top-hat* (SOILLE, 2004; GONZALES; WOODS, 2002).

Fig. 2.14: (a) imagem original, (b) fechamento de buracos da imagem, (c) *top-hat* e (d) *threshold* do *top-hat*



(a) (b)

(c) (d)

Fonte: produção próprio autor

O *top-hat* pode ser útil no processo de segmentação de objetos em uma imagem, que é geralmente uma das primeiras etapas no processamento de imagens em um sistema de visão (GONZALES; WOODS, 2002).

Neste trabalho o *top-hat* foi aplicado na etapa de segmentação dos furos do cabeçote. Ele representa a subtração entre a imagem do cabeçote com os buracos fechados e a imagem inicial. A Fig. 2.14 traz a sequência de segmentação dos furos de um cabeçote utilizando o fechamento de buracos, o *top-hat* e o *threshold*.

2.6 MEDIÇÃO DE SIMILARIDADE ENTRE IMAGENS

2.6.1 Coeficiente de Correlação de Pearson

O Coeficiente de Correlação de Pearson, *r*, é usado em análises estatísticas, reconhecimento de padrões e processamento de imagens. Em processamento de imagens, ele serve para comparar duas imagens com o propósito de registro de imagens, reconhecimento de objetos e medida de disparidades. Para imagens em tons de cinza, o coeficiente de correlação de Pearson é definido como (MIRANDA NETO *et al*., 2009; MIRANDA NETO, 2007; YEN; JOHNSTON, 2005):

$$r = \frac{\sum_i (x_i - x_m)(y_i - y_m)}{\sqrt{\sum_i (x_i - x_m)^2}\sqrt{\sum_i (y_i - y_m)^2}} \quad (2.25)$$

Em que x_i é a intensidade do *i*-ésimo *pixel* da imagem 1, y_i a intensidade do *i*-ésimo *pixel* da imagem 2, x_m a média das intensidades da imagem 1 e y_m é a média das intensidades da imagem 2. O coeficiente de correlação *r* recebe o valor 1 se as duas imagens forem exatamente iguais, recebe o valor 0 se as duas imagens forem totalmente diferentes e recebe o valor -1 se uma imagem for o negativo da outra (MIRANDA NETO *et al*., 2009; MIRANDA NETO, 2007; YEN; JOHNSTON, 2005).

O coeficiente de correlação *r* recebe o valor 1 se o objeto não sofrer alteração ou recebe um valor menor que 1 se ocorrer algum movimento ou alteração. Na prática, distorções na imagem, *pixels* com ruído, pequenas variações do objeto em relação à câmera e outros fatores podem produzir um valor de *r* menor que 1, mesmo se o objeto não tiver sido movido ou se ele não sofreu nenhuma alteração (MIRANDA NETO, 2007; YEN; JOHNSTON, 2005).

Como vantagens o coeficiente de correlação de Pearson condensa a comparação de duas imagens em um simples escalar *r* e é invariante a transformações lineares em *x* e *y*. Variações no brilho ou contraste de uma imagem não afetam o resultado de *r* (YEN; JOHNSTON, 2005).

2.6.2 *Structural Similarity*

Conforme Sampat *et al*. (2009), o princípio fundamental do *Structural Similarity* (SSIM) é que o sistema visual humano se mostra altamente adaptado a extrair informações estruturais de imagens. Sendo assim, medir similaridades estruturais de imagens deveria fornecer uma boa aproximação da percepção visual de imagens. No SSIM define-se a informação estrutural de uma imagem como aqueles atributos que representam estruturas de objetos na cena visual por meio da média das intensidades e do contraste. Desse modo, SSIM compara padrões locais de intensidades de *pixels* que tenham sido normalizados pela média das intensidades e do contraste.

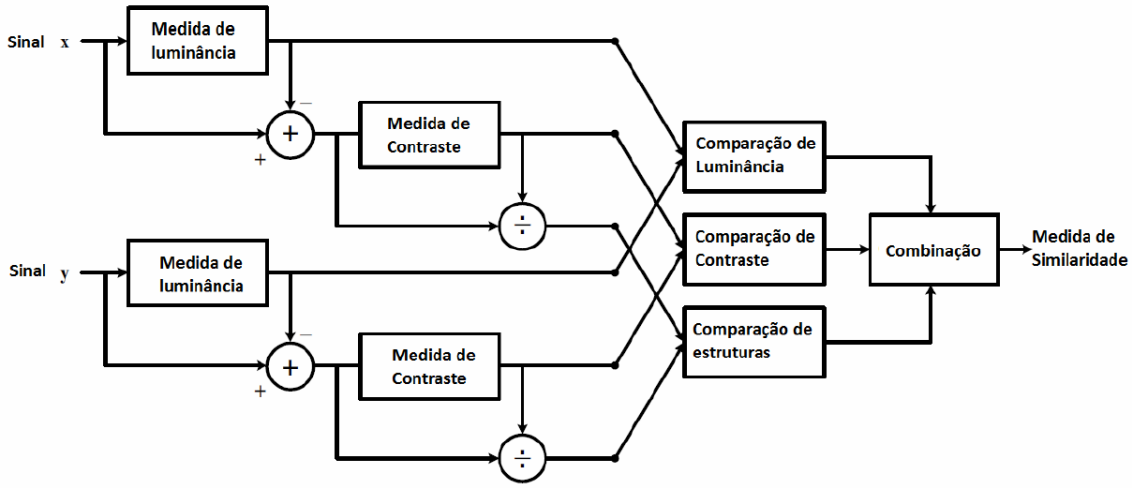
Para Ndajah *et al*. (2010), *Structural Similarity* (SSIM) é um método eficiente para medir similaridade entre imagens. Inicialmente se observou que a estrutura das imagens possui forte dependência da sua vizinhança. Essa dependência carrega informações úteis sobre a estrutura dos objetos contidos na imagem.

Sampat *et al*. (2009) comentam que, comparado com métodos baseados no sistema

visual humano, SSIM apresenta baixa complexidade computacional e desempenho superior na avaliação da similaridade entre imagens.

De acordo com Wang *et al*. (2004), SSIM compara padrões locais de intensidades de *pixels* que foram normalizados para a luminosidade e contraste. Imagens naturais são altamente estruturadas, seus *pixels* têm fortes dependências, sobretudo quando eles estão espacialmente aproximados. Tal dependência carrega informações importantes sobre a estrutura dos objetos na cena visual.

Fig. 2.15: Diagrama da SSIM



Fonte: adaptado de Wang *et al*. (2004)

A medição do índice de similaridade proposta por Wang *et al*. (2004) está demonstrada no diagrama da Fig. 2.15. Supondo que as entradas *x* e *y* são duas imagens distintas e que *x* é uma imagem sem alterações, então a medida de similaridade entre elas pode ser utilizada para medir a qualidade da imagem *y*. O diagrama apresenta a medida da similaridade dividida em três partes: comparação por luminância, comparação por contraste e comparação da estrutura.

Segundo Fonseca (2008), a primeira etapa é comparar a luminância de cada imagem. Assumindo sinais discretos, ela é estimada como a média das intensidades, mostrada nas equações 2.26 e 2.27.

$$\mu_x = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} x(i,j) \tag{2.26}$$

$$\mu_y = \frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1} y(i,j) \tag{2.27}$$

Em que x e y são imagens de entrada, M e N o tamanho delas e (i,j) percorrem todos os *pixel*s das imagens.

Obtém-se a comparação por luminância *l*(x,y) utilizando a intensidade média encontrada para cada imagem, como mostrado na equação 2.28.

$$l(x,y) = \frac{2\mu_x\mu_y + C_1}{\mu_x^2 + \mu_y^2 + C_1} \tag{2.28}$$

C_1 é uma constante inserida na equação para evitar instabilidade quando μ_x e μ_y forem próximos de zero. O valor recomendado por Wang *et al*. (2004) é:

$$C_1 = (K_1 L)^2 \tag{2.29}$$

Em que K_1 = 0,01 e L = 255 para imagens em escala de cinza de 8 bits.

Wang *et al.* (2004) e Fonseca (2008) sugerem que a comparação por contraste seja encontrada utilizando o desvio padrão de cada imagem. O desvio padrão para as imagens x e y é encontrado pelas equações 2.30 e 2.31.

$$\sigma_x = \sqrt{\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(x(i,j) - \mu_x)^2} \tag{2.30}$$

$$\sigma_y = \sqrt{\frac{1}{MN}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(y(i,j) - \mu_y)^2} \tag{2.31}$$

A comparação por contraste é apresentada na equação 2.32.

$$c(x,y) = \frac{2\sigma_x\sigma_y + C_2}{\sigma_x^2 + \sigma_y^2 + C_2} \tag{2.32}$$

C_2 é uma constante similar a C_1, com K_2 = 0,01 e L = 255 para imagens em escala de cinza de 8 bits.

$$C_2 = (K_2 L)^2 \tag{2.33}$$

Para Wang *et al.* (2004) e Fonseca (2008), após os cálculos da luminância e contraste é feita uma comparação estrutural, como mostra a equação 2.34.

$$s(x,y) = \frac{2\sigma_{xy} + C_3}{\sigma_x\sigma_y + C_3} \tag{2.34}$$

Como no caso da luminância e do contraste, uma constante muito pequena (C_3) deve ser inserida na equação da comparação estrutural *s*(*x*,*y*). Na forma discreta, σ_{xy} é dado pela equação 2.35.

$$\sigma_{xy} = \frac{1}{MN-1}\sum_{i=0}^{M-1}\sum_{j=0}^{N-1}(x(i,j) - \mu_x)(y(i,j) - \mu_y) \tag{2.35}$$

Wang *et al.* (2004) chamaram de índice SSIM a combinação das três comparações encontradas, equações 2.28, 2.32 e 2.34.

$$SSIM(x,y) = [l(x,y)]^{\alpha}[c(x,y)]^{\beta}[s(x,y)]^{\gamma} \quad (2.36)$$

Em que $\alpha > 0, \beta > 0$ e $\gamma > 0$ são parâmetros utilizados para ajustar a importância relativa de cada componente. Fazendo $\alpha = \beta = \gamma = 1$ e $C_3 = C_2/2$, pode-se simplificar a equação 2.36. O resultado encontra-se na equação 2.37.

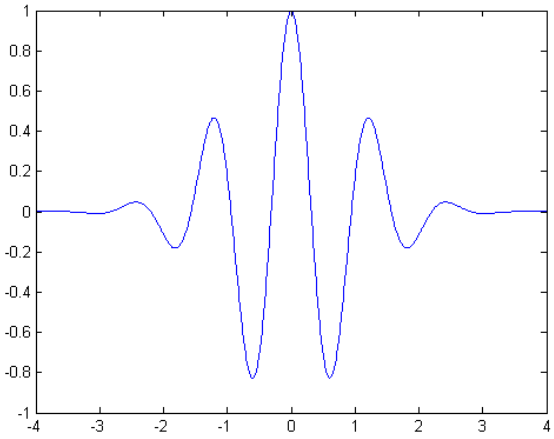
$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \quad (2.37)$$

2.6.3 *Complex Wavelet Structutral Similarity*

Segundo Fan, Wang e Wang (2010), Sampat *et al.* (2009) e Wang e Simoncelli (2005), uma desvantagem comum em índices baseados em intensidades é a sensibilidade a distorções de escala e geometria. Isso se torna um grave problema quando existem pequenas translações, rotações ou diferenças de escalas entre as imagens sendo comparadas.

Sampat *et al.* (2009) propõem um novo método chamado *Complex Wavelet Structutral Similarity* (CW-SSIM). *Wavelet* é um termo utilizado em processamento digital de sinais e significa uma breve oscilação, mostrada na Fig. 2.16, que parte do valor zero e é incrementado, depois decrementado e termina em zero novamente. Segundo Rioul e Vetterli (1991), em meados da década de 1980 os pesquisadores franceses Morlet, Grossmann e Meyer desenvolveram fundamentos matemáticos acerca dessas ondas e deram o nome de "*Ondelettes*" (*wavelets* em francês). Ela é uma onda semelhante às encontradas em um monitor cardíaco.

Fig. 2.16: Exemplo de uma *wavelet* do tipo Morlet



Fonte: http://upload.wikimedia.org/wikipedia/commons/2/23/Wavelet_-_Morlet.png. Acesso em: abr. 2012

A principal ideia por trás do CW-SSIM é que pequenas distorções geométricas na imagem levam a mudanças de fase consistentes na região dos coeficientes de *wavelets* e que um deslocamento na fase dos coeficientes não altera o conteúdo estrutural de uma imagem.

Sampat *et al.* (2009) afirmam que as vantagens do CW-SSIM são muitas e que esse método não requer correspondência explícita entre os *pixels* que estão sendo comparados. CW-SSIM é insensível a pequenas distorções geométricas, como rotações, translações e alteração de escala. CW-SSIM compara as propriedades de textura e estrutura de regiões das imagens sendo comparadas. Tais características são perdidas quando aplicados métodos baseados na geometria das imagens.

Para Sampat *et al.* (2009) e Fan, Wang e Wang (2010), CW-SSIM é uma extensão do método SSIM no domínio complexo das *wavelets*. O objetivo é desenvolver uma metodologia que não seja sensível a distorções geométricas não estruturais em uma imagem. Essas distorções são tipicamente causadas por alteração na luminosidade da cena ou pela movimentação do dispositivo de captura da imagem, que pode provocar translações na imagem.

Conforme Sampat *et al.* (2009) e Wang e Simoncelli (2005), pode-se considerar a função *wavelet* mãe como a modulação de um filtro passa-baixa, dada pela equação 2.38:

$$w(u) = g(u)e^{j\omega_c u} \tag{2.38}$$

Em que ω_c é a frequência central do filtro passa-baixa modulado e $g(u)$ a função do filtro passa-baixa.

A família de *wavelets* é composta por versões dilatadas/contraídas e transladadas da *wavelet*

mãe:

$$w_{s,p}(u) = \frac{1}{\sqrt{s}} w\left(\frac{u-p}{s}\right) = \frac{1}{\sqrt{s}} g\left(\frac{u-p}{s}\right) e^{j\omega_c \frac{(u-p)}{s}} \tag{2.39}$$

Em que $s \in R^+$ é um fator escalar e $p \in R$ um fator de translação. A transformada contínua *wavelet* de um sinal real $x(u)$ é:

$$X(s,p) = \frac{1}{2\pi} \int_{-\infty}^{\infty} X(\omega)\sqrt{s}G(s\omega - \omega_c)e^{j\omega p} d\omega \tag{2.40}$$

Em que X(ω) e G(ω) são as transformadas de Fourier de $x(u)$ e $g(u)$.

Segundo Sampat *et al.* (2009) e Wang e Simoncelli (2005), os coeficientes discretos *wavelet* são amostras da transformada contínua *wavelet*. No domínio da transformada complexa *wavelet*, pode-se supor que

$$c_x = \left\{ c_{x,i} \middle| i = 1,..., N \right\} \tag{2.41}$$

$$c_y = \left\{ c_{y,i} \middle| i = 1,..., N \right\} \tag{2.42}$$

As equações 2.41 e 2.42 apresentam dois conjuntos de coeficientes extraídos da mesma localização espacial na mesma sub-banda *wavelet* de duas imagens que estão sendo comparadas respectivamente. A equação CW-SSIM pode ser definida como

$$\tilde{S}(c_x, c_y) = \frac{2\left|\sum_{i=1}^{N} c_{x,i} c_{y,i}^*\right| + K}{\sum_{i=1}^{N} \left|c_{x,i}\right|^2 + \sum_{i=1}^{N} \left|c_{y,i}\right|^2 + K} \tag{2.43}$$

Em que c^* denota o complexo conjugado de c e K é uma constante pequena.

Sampat *et al.* (2009) e Wang e Simoncelli (2005) supõem que o método CW-SSIM seja insensível a pequenas mudanças na luminosidade, rotação e translação entre as imagens comparadas. Para comprovar essa suposição, deve-se assumir que x seja a imagem referência e y a imagem cuja similaridade está sendo comparada com a imagem referência.

Para Sampat *et al.* (2009) e Wang e Simoncelli (2005), alteração de luminosidade e contraste podem ser descritos como uma transformação linear das intensidades locais dos *pixels*, como mostra a equação 2.44.

$$y_i = ax_i + b \tag{2.44}$$

Para todos os valores de i. Em virtude da natureza de banda de passagem e linearidade da transformada *wavelet*, o efeito no domínio *wavelet* é uma escala constante para todos os coeficientes:

$$c_{y,i} = ac_{x,i} \tag{2.45}$$

Substituindo a equação 2.45 na equação 2.43, tem-se

$$\tilde{S}(c_x, c_y) = \frac{2a + K / \sum_{i=1}^{N} \left|c_{x,i}\right|^2}{1 + a^2 + K / \sum_{i=1}^{N} \left|c_{x,i}\right|^2} \tag{2.46}$$

Para imagens com características fortes (grandes coeficientes), $K / \sum_{i=1}^{N} \left|c_{x,i}\right|^2$ é muito pequeno e pode ser desconsiderado, levando a uma medição insensível. Escalonando a magnitude por um fator de 10% (a=1,1), apenas se reduz o valor de SSIM de 1 para 0,9955. O efeito manifesta-se mais em imagens com características fracas (que criam coeficientes de menor magnitude) do que em imagens com características fortes (magnitude dos coeficientes maiores).

De acordo com Sampat *et al.* (2009) e Wang e Simoncelli (2005), a translação no domínio 2-D pode ser escrita como

$$y\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = x\begin{pmatrix} u_1 + \Delta u_1 \\ u_2 + \Delta u_2 \end{pmatrix} \tag{2.47}$$

Em que Δu_1 e Δu_2 representam deslocamentos horizontais e verticais, respectivamente. Para simplificar, considere o caso 1-D:

$$y(u_1) = x(u_1 + \Delta u_1) \tag{2.48}$$

A equação 2.49 corresponde a um deslocamento de fase linear no domínio de Fourier:

$$Y(\omega) = X(\omega)e^{j\omega\Delta u} \tag{2.49}$$

Substituindo a equação 2.49 na equação 2.40, tem-se

$$Y(s, p) = \frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)\sqrt{s}G(s\omega - \omega_c)e^{j\omega(p+\Delta u)}d\omega \tag{2.50}$$

$$Y(s, p) = e^{j\omega_c \Delta u / s}.\frac{1}{2\pi}\int_{-\infty}^{\infty} X(\omega)\sqrt{s}G(s\omega - \omega_c)e^{j\omega p}e^{j(\omega - \omega_c/s)\Delta u}d\omega \tag{2.51}$$

$$Y(s, p) \approx X(s, p)e^{j\omega_c \Delta u / s} \tag{2.52}$$

Tal aproximação se mostra válida quando a translação Δu é pequena comparada com o tamanho do filtro *wavelet* e $g(u)$ varia vagarosamente. Quando $g(u)$=1, a aproximação torna-se exata.

No caso 2-D, um resultado similar pode ser obtido, e por consequência os coeficientes discretos *wavelet* $\{c_{y,i}\}$ e $\{c_{x,i}\}$ (amostras discretas de X(s,p) e Y(s,p)). Assim pode-se escrever $c_{y,i} \approx c_{x,i}e^{j\phi}$ para todos os valores de i:

$$\tilde{S}(c_x, c_y) \approx \frac{2\left|\sum_{i=1}^{N} c_{x,i} c_{x,i} e^{-j\phi}\right| + K}{\sum_{i=1}^{N} \left|c_{x,i}\right|^2 + \sum_{i=1}^{N} \left|c_{x,i} e^{j\phi}\right|^2 + K} = 1 \tag{2.53}$$

Para Sampat *et al.* (2009) e Wang e Simoncelli (2005), rotação e escala do domínio 2-D pode ser escrita como

$$y\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = x\left(\begin{pmatrix} 1+\Delta s_1 & 0 \\ 0 & 1+\Delta s_2 \end{pmatrix} \times \begin{pmatrix} \cos\Delta\theta & -\sin\Delta\theta \\ \sin\Delta\theta & \cos\Delta\theta \end{pmatrix}\begin{pmatrix} u_1 \\ u_2 \end{pmatrix}\right) \tag{2.54}$$

Em que $(1+\Delta s_1, 1+\Delta s_2)$ e $\Delta\theta$ são os fatores de escala e rotação, respectivamente. Quando $\Delta\theta$ é pequeno, têm-se $\cos\Delta\theta \approx 1$ e $\sin\Delta\theta \approx \Delta\theta$, sendo assim:

$$y\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} \approx x\begin{pmatrix} u_1 + (u_1\Delta s_1 - u_2\Delta\theta - u_2\Delta s_1\Delta\theta) \\ u_2 + (u_2\Delta s_2 - u_1\Delta\theta - u_1\Delta s_2\Delta\theta) \end{pmatrix} \tag{2.55}$$

$$y\begin{pmatrix} u_1 \\ u_2 \end{pmatrix} = x\begin{pmatrix} u_1 + \Delta u_1 \\ u_2 + \Delta u_2 \end{pmatrix} \tag{2.56}$$

Em que $\Delta u_1 = u_1\Delta s_1 - u_2\Delta\theta - u_2\Delta s_1\Delta\theta$ e $\Delta u_2 = u_2\Delta s_2 - u_1\Delta\theta - u_1\Delta s_2\Delta\theta$. Na equação 2.56, quando (u_1, u_2) não estão longe da origem, uma pequena alteração de escala e rotação pode ser localmente aproximada por uma pequena translação $(\Delta u_1, \Delta u_2)$. Baseado na análise acima, $\tilde{S}(c_x, c_y) \approx 1$.

Conforme Sampat *et al.* (2009), a respeito da análise de sensibilidade feita, conclui-se que CW-SSIM é simultaneamente insensível a alterações de luminosidade, contraste, translações, escala e rotação. CW-SSIM é resiliente apenas a pequenas distorções e resulta em pequenos valores de similaridade para grandes distorções. Do ponto de vista da similaridade estrutural, essas distorções pertencem à categoria das distorções não estruturais, causadas, por exemplo, por erros de localização, por alterações nas condições luminosas ou por movimentos na aquisição das imagens. Essas distorções não se referem a distorções na estrutura dos objetos na cena. CW-SSIM é sensível a distorções estruturais, como compressão do tipo JPEG[6], porque elas carregam variações significantes dos padrões locais de fase.

2.7 USO DE VISÃO EM TRABALHOS RELACIONADOS

Rodriguez, Norrish e Nicholson (2009) desenvolveram um sistema que gera as

[6] JPEG – método de compressão de imagens no qual o grau de redução pode ser ajustado.

trajetórias de um robô que recupera peças com solda. Utilizou-se um sistema com câmera e uma matriz de pontos de laser incidida sobre o objeto-alvo. O perfil do objeto é encontrado por meio de análises da matriz de pontos capturada pelo sistema de visão. Para mensurar posição e orientação da matriz no espaço, faz-se necessário capturar duas imagens em diferentes posições; primeiramente se captura uma imagem de todo o objeto-alvo, depois se movimenta a câmera, para uma posição predeterminada, e captura-se a segunda imagem. Essas duas imagens servem para estimar a posição e orientação do objeto no espaço. O deslocamento do mesmo ponto, de uma imagem para a outra, é obtido em termos de *pixels*, que em conjunto com sua atual posição na imagem fornecerão as informações necessárias para obter as coordenadas 3D.

Para Cheng e Denman (2005), o sistema de visão que utiliza uma câmera 2D fixa não obtém informações de profundidade da cena. Isso frequentemente resulta em problemas na identificação da posição real de um objeto 3D sob um plano. Para resolver tal problema, Cheng e Denman (2005) montaram a câmera no corpo do robô e moveram-na em diferentes posições para capturar imagens e identificar a verdadeira posição de um objeto 3D na superfície plana. Neste trabalho empregou-se o sistema de visão comercial Fanuc VisLoc. A calibração do sistema usou um padrão de calibração para encontrar o centro e a orientação do sistema de coordenadas da câmera. O sistema de visão utiliza duas ferramentas de localização que necessitam de um padrão de comparação. Esse padrão é uma imagem 2D que define a superfície superior, o tamanho e a posição do centroide do objeto a ser detectado na imagem. A ferramenta de localização 1 define quantos objetos existem na imagem. A ferramenta de localização 2 detecta quais objetos são compatíveis com o padrão de comparação e depois encontra a posição objeto no plano *XY*. O sistema de visão comunica-se com o robô pela rede *ethernet* e escreve o deslocamento do objeto detectado em um registrador de posição na área de memória do robô. Dessa forma, o robô consegue executar a pega do objeto identificado pelo sistema de visão. Os objetos manipulados pelo robô são pinos de apoio para bola de golfe. Tais pinos possuem a superfície superior circular, por isso sua orientação é desconsiderada nessa aplicação.

Gonçalves (2004) montou um sistema que captura imagens de um objeto cilíndrico. Com base nessas imagens ele encontra a sua posição no plano *X*, *Y* e as repassa a um robô que faz a pega desse objeto. A imagem capturada possui o fundo claro, enquanto os objetos são escuros. A forma utilizada para encontrar a posição do objeto no plano *X*, *Y* foi aplicar na imagem um *Thresholding* para encontrar a área e o centroide do objeto em coordenadas *X*, *Y*. Com as coordenadas *X*, *Y* do objeto calculou-se a cinemática inversa do robô para que o

objeto fosse pego. Vale lembrar que não foram aplicados cálculos para encontrar a orientação do objeto. Como se tratava de um objeto cilíndrico e essa geometria facilita a pega pelo robô, não foi necessário calcular a orientação que o robô utilizará para efetuar a pega.

Araújo (2010) desenvolveu o sistema de visão que guia um robô na indústria do calçado. A função do robô é executar a limpeza da sola do sapato antes de ser colada no couro. O robô contorna o perímetro da sola com um equipamento que executa a higienização. A imagem capturada possui um fundo claro, enquanto a sola do sapato é escura. Araújo (2010) utilizou ferramentas de processamento de imagem para segmentar o objeto do fundo da imagem e realçar seu contorno. No contorno são marcados alguns pontos que podem descrever a geometria do perímetro da sola de sapato analisada. Nesse caso, trechos mais curvos do contorno possuem mais pontos marcados, trechos com curvaturas menos acentuadas têm menos pontos marcados. Esses pontos representam as posições que o robô deverá alcançar até completar a limpeza de todo o perímetro da sola do sapato. As informações são enviadas ao robô pela rede serial. A calibração do sistema de visão encontra apenas a dimensão do *pixel* para os eixos *X* e *Y*.

Os trabalhos relatados não apresentam soluções viáveis e eficientes para o problema da pega dos cabeçotes no transportador de entrada da célula de oleação e paletização. Cada aplicação citada foi desenvolvida para resolver problemas bem definidos e em geral tratam objetos semelhantes entre si. Nos trabalhos de Araújo (2010), Cheng e Denman (2005) e Gonçalves (2004) não se considerou o problema da orientação das peças manipuladas pelo robô. Encontrar a posição/orientação dos cabeçotes é um dos objetivos deste trabalho. Rodriguez, Norrish e Nicholson (2009) e Cheng e Denman (2005) capturaram mais de uma imagem para calcular a posição das peças analisadas, pois esses trabalhos levaram em conta a profundidade da imagem. Nos quatro trabalhos relatados o objeto analisado destaca-se do fundo da imagem, enquanto os cabeçotes abordados no presente projeto possuem pouco contraste com o fundo da imagem.

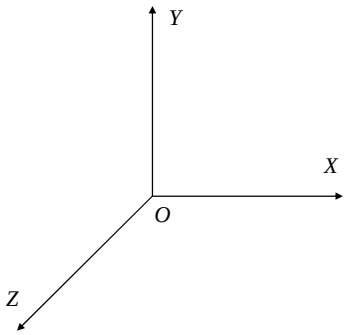
3 GEOMETRIA E PROGRAMAÇÃO DE ROBÔS

Neste capítulo concentram-se fundamentos aplicados em robótica. Será abordada a representação algébrica de sistemas de coordenadas, bem como as relações matemáticas entre dois sistemas distintos. A utilização dos sistemas de coordenadas pelo robô empregado no projeto também será apresentada. A estrutura de programação do robô usado no projeto será o último tema abordado.

3.1 MATRIZ DE ROTAÇÃO

Um sistema de coordenadas denominado *OXYZ* é definido como a combinação da origem *O* com um conjunto de eixos *X*, *Y* e *Z* perpendiculares entre si (BARRIENTOS, 2007; HOLLERBACH, 2003), como mostra a Fig. 3.1. Esse sistema pode ser deslocado e/ou rotacionado em relação a outro sistema de coordenadas.

Fig. 3.1: Representação do sistema de coordenadas *OXYZ*



Fonte: produção próprio autor

É útil expressar a relação entre dois sistemas de coordenadas, com o ponto de origem em comum, por uma matriz 3x3 cujos elementos são os cossenos diretores de cada eixo de um sistema de coordenadas expressado no outro (MARCUS, 1967).

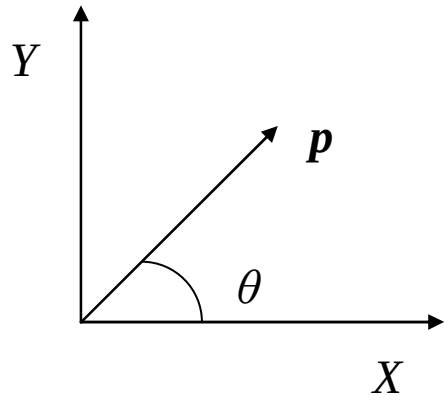
A matriz de rotação é um método mais completo para descrever a orientação de um sistema de coordenadas, graças principalmente às facilidades do uso da álgebra matricial. A matriz de rotação representa a orientação de sistemas rotacionados sobre um dos eixos do sistema de referência (BARRIENTOS, 2007).

As coordenadas de um vetor $\boldsymbol{p}$ após ser rotacionado por um ângulo θ, como se vê na Fig. 3.2, são as seguintes:

$$\boldsymbol{p} = \cos\theta\,\mathbf{i} + \sin\theta\,\mathbf{j} \tag{3.1}$$

$$\boldsymbol{p} = \begin{bmatrix} \cos\theta \\ \sin\theta \end{bmatrix} \tag{3.2}$$

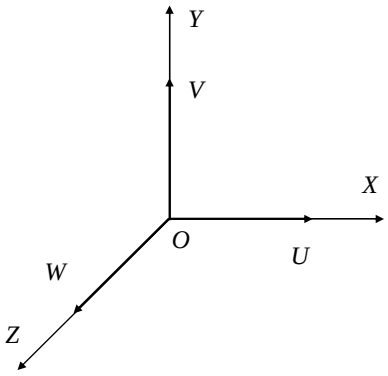
Fig. 3.2: Representação do vetor ***p*** no plano *XY*



Fonte: produção próprio autor

A rotação de um sistema de coordenadas em relação a outro não é tão simples como rotacionar um vetor no plano, porém um raciocínio análogo pode ser empregado para obtê-la. A matriz de rotação define a orientação de um sistema *OUVW* em relação ao *OXYZ*, sem deslocar a sua origem.

Fig. 3.3: Representação dos sistemas *OXYZ* e *OUVW*



Fonte: Barrientos (2007)

Seja o sistema *OXYZ* fixo no espaço e o sistema *OUVW* móvel, mostrados na Fig.3.3, em que $\mathbf{i}_x$, $\mathbf{j}_y$ e $\mathbf{k}_z$ são os vetores unitários do sistema *OXYZ* e $\mathbf{i}_u$, $\mathbf{j}_v$ e $\mathbf{k}_w$ são os vetores unitários do sistema *OUVW*, a equação 3.3 descreve a rotação do sistema móvel em relação ao fixo (BARRIENTOS, 2007).

$$\begin{bmatrix} \mathbf{i}_x \\ \mathbf{j}_y \\ \mathbf{k}_z \end{bmatrix} = Rot \begin{bmatrix} \mathbf{i}_u \\ \mathbf{j}_v \\ \mathbf{k}_w \end{bmatrix} \tag{3.3}$$

Em que

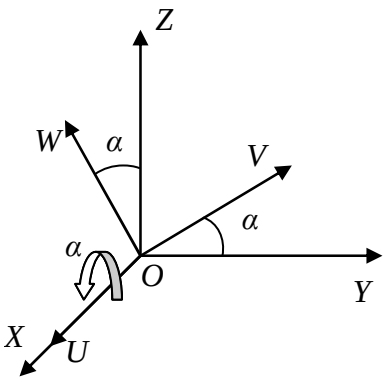
$$Rot = \begin{bmatrix} \mathbf{i}_x\mathbf{i}_u & \mathbf{i}_x\mathbf{j}_v & \mathbf{i}_x\mathbf{k}_w \\ \mathbf{j}_y\mathbf{i}_u & \mathbf{j}_y\mathbf{j}_v & \mathbf{j}_y\mathbf{k}_w \\ \mathbf{k}_z\mathbf{i}_u & \mathbf{k}_z\mathbf{j}_v & \mathbf{k}_z\mathbf{k}_w \end{bmatrix} \tag{3.4}$$

Utilizando a relação de seno e cosseno podem-se encontrar as equações para as rotações sobre os eixos fixos *OX*, *OU* e *OZ*.

A equação 3.5 descreve uma rotação α do sistema *OUVW* em relação ao sistema *OXYZ* em torno do eixo *OX*, como na Fig. 3.4.

$$Rot(x,\alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos\,\alpha & -sen\alpha \\ 0 & sen\alpha & cos\,\alpha \end{bmatrix} \tag{3.5}$$

Fig. 3.4: Rotação α do sistema *OUVW* em relação ao sistema *OXYZ*

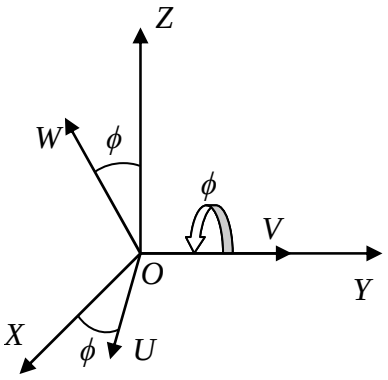


Fonte: Barrientos (2007)

A equação 3.6 descreve uma rotação ϕ do sistema *OUVW* em relação ao sistema *OXYZ* em torno do eixo *OY*, como apresentado na Fig. 3.5 (BARRIENTOS, 2007).

$$Rot(y,\phi) = \begin{bmatrix} cos\,\phi & 0 & sen\phi \\ 0 & 1 & 0 \\ -sen\phi & 0 & cos\,\phi \end{bmatrix} \tag{3.6}$$

Fig. 3.5: Rotação ϕ do sistema *OUVW* em relação ao sistema *OXYZ*

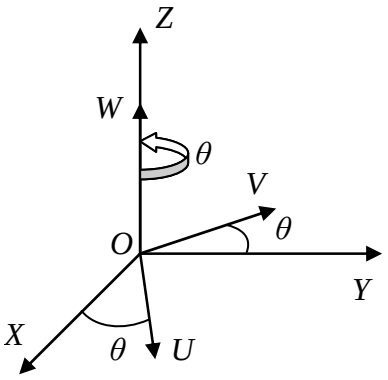


Fonte: Barrientos (2007)

A equação 3.7 apresenta uma rotação θ do sistema *OUVW* em relação ao sistema *OXYZ* em torno do eixo *OZ*, conforme Fig. 3.6 (BARRIENTOS, 2007).

$$Rot(z,\theta) = \begin{bmatrix} cos\,\theta & -sen\,\theta & 0 \\ sen\,\theta & cos\,\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

Fig. 3.6: Rotação θ do sistema *OUVW* em relação ao sistema *OXYZ*



Fonte: Barrientos (2007)

As equações 3.5, 3.6 e 3.7 são as matrizes básicas de rotação em um sistema com três dimensões. A composição dessas matrizes pode expressar a aplicação de várias rotações seguidas sobre o mesmo sistema de coordenadas (BARRIENTOS, 2007). A equação 3.8 traz a sequência de três rotações, sendo a primeira α sobre *OX*, a segunda ϕ sobre *OY* e a terceira θ sobre *OZ*.

$$T = Rot(z,\theta)Rot(y,\phi)Rot(x,\alpha) = \begin{bmatrix} C\theta C\phi & -S\theta C\alpha + C\theta S\phi S\alpha & S\theta S\alpha + C\theta S\phi C\alpha \\ S\theta C\phi & C\theta C\alpha + S\theta S\phi S\alpha & -C\theta S\alpha + S\theta S\phi C\alpha \\ -S\phi & C\phi S\alpha & C\phi C\alpha \end{bmatrix} \tag{3.8}$$

Em que $C\theta = \cos\theta$, $S\theta = sen\theta$, $C\alpha = \cos\alpha$, $S\alpha = sen\alpha$, $C\phi = \cos\phi$ e $S\phi = sen\phi$.

3.2 MATRIZ DE TRANSFORMAÇÃO HOMOGÊNEA

A matriz de transformação homogênea representa a posição e a orientação de um sistema de coordenadas em relação a outro. Ela é composta por uma rotação, uma translação, uma perspectiva e um fator de escala. Essa matriz possui o formato 4x4, como apresentado na equação 3.9 (BARRIENTOS, 2007; HOLLERBACH, 2003; OKAFOR; ERTEKIN, 2000),

$$T = \begin{bmatrix} R_{3x3} & p_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} Rotação & Translação \\ Perspectiva & Escala \end{bmatrix} \tag{3.9}$$

Se o sistema *OUVW* for somente transladado por um vetor $\boldsymbol{p} = p_x\mathbf{i}+p_y\mathbf{j}+p_z\mathbf{k}$ com relação ao sistema *OXYZ* e fizer a perspectiva = [0 0 0] e a escala = 1, tem-se a matriz básica de translação, evidenciada na equação 3.10 (BARRIENTOS, 2007).

$$T(p) = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

Dessa forma, pode-se descrever a matriz de transformação homogênea de um sistema rotacionado por um ângulo α em torno do eixo *OX*, seguido de um ângulo ϕ em torno do eixo *OY*, seguido de um ângulo θ em torno do eixo *OZ*, seguido da translação de um ponto $p = (p_x, p_y, p_z)$, como na equação 3.11 (BARRIENTOS, 2007; HOLLERBACH, 2003).

$$T = \begin{bmatrix} C\theta C\phi & -S\theta C\alpha + C\theta S\phi S\alpha & S\theta S\alpha + C\theta S\phi C\alpha & p_x \\ S\theta C\phi & C\theta C\alpha + S\theta S\phi S\alpha & -C\theta S\alpha + S\theta S\phi C\alpha & p_y \\ -S\phi & C\phi S\alpha & C\phi C\alpha & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.11}$$

3.3 QUATÉRNIO

Quatérnio é um quadrinômio imaginário que pode ser interpretado como a rotação de um vetor no espaço 3D (OLIVEIRA; PIERI; MORENO, 2010). De forma similar aos números complexos, os quatérnios possuem uma parte real e uma imaginária, todavia esta última possui três componentes: **i**, **j** e **k** (BIASI; GATTASS, 2002). Um quatérnio é capaz de representar a orientação de um sistema de coordenadas *OUVW* em relação a outro. É possível compor rotações e translações de maneira muito simples e computacionalmente econômica (BARRIENTOS, 2007). Um quatérnio pode ser escrito do seguinte modo (HOLLERBACH,

2003):

$$\boldsymbol{q} = q_0 + \mathbf{q} \tag{3.12}$$

$$q_0 = cos\left(\frac{\theta}{2}\right) \tag{3.13}$$

$$\mathbf{q} = \sin\left(\frac{\theta}{2}\right)\mathbf{i} + \sin\left(\frac{\theta}{2}\right)\mathbf{j} + \sin\left(\frac{\theta}{2}\right)\mathbf{k} \tag{3.14}$$

Essa notação apresenta um quatérnio 4x1, em que q_0 é a parte escalar e **q** a parte vetorial, que pode ser representado pela equação 3.15 (BARRIENTOS, 2007; HOLLERBACH, 2003).

$$\boldsymbol{q} = \left[q_0, q_1\mathbf{i}, q_2\mathbf{j}, q_3\mathbf{k}\right] \tag{3.15}$$

O conjugado $\boldsymbol{q}^*$ de um quatérnio é dado pela inversão do sinal da sua parte imaginária (BARRIENTOS, 2007; HOLLERBACH, 2003). Dessa forma,

$$\boldsymbol{q}^* = \left[q_0, -q_1\mathbf{i}, -q_2\mathbf{j}, -q_3\mathbf{k}\right] \tag{3.16}$$

A norma de um quatérnio ($||\boldsymbol{q}||$) é dada pelo somatório dos quadrados das suas componentes (OLIVEIRA; PIERI; MORENO, 2010; BARRIENTOS, 2007). Assim,

$$\| \mathrm{q} \| = \sqrt{q_0^2 + q_1^2 + q_2^2 + q_3^2} \tag{3.17}$$

O inverso de um quatérnio é obtido por meio da equação 3.18, que é o conjugado dividido pela norma (BARRIENTOS, 2007).

$$\boldsymbol{q}^{-1} = \frac{\boldsymbol{q}^*}{\| \boldsymbol{q} \|} \tag{3.18}$$

Um quatérnio é dito unitário quando a sua norma é igual a 1, $\| \boldsymbol{q} \| = 1$ (OLIVEIRA; PIERI; MORENO, 2010; BIASI; GATTASS, 2002). O quatérnio unitário é utilizado para representação de movimentos, pois essa característica leva a um caso particular de simplificação da regra de inversão de quatérnios para $\boldsymbol{q}^{-1} = \boldsymbol{q}^*$ (OLIVEIRA; PIERI; MORENO, 2010).

O vetor **q** que representa o sentido de rotação de um quatérnio. Pode-se, assim, definir um quatérnio que represente um giro de valor θ sobre um vetor $\boldsymbol{k}$ como sendo (BARRIENTOS, 2007):

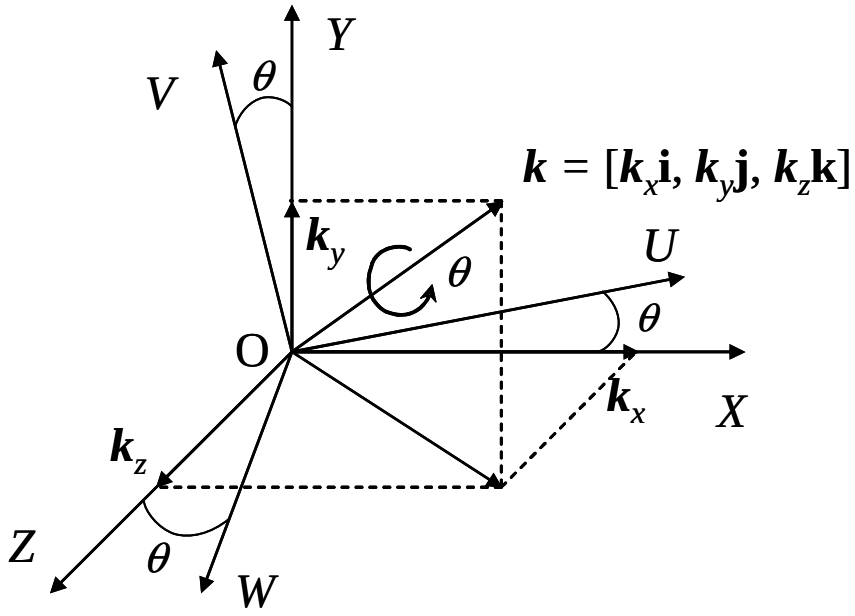
$$\boldsymbol{q} = Rot(\boldsymbol{k}, \theta) = \left(cos\left(\frac{\theta}{2}\right), \boldsymbol{k}\, sin\left(\frac{\theta}{2}\right)\right) \tag{3.19}$$

Então, para um vetor $\boldsymbol{k}$ = [k_x**i**, k_y**j**, k_z**k**], a mesma rotação sugerida anteriormente pode ser representada por meio de um quatérnio, da seguinte forma,

$$\boldsymbol{q} = \left(cos\left(\frac{\theta}{2}\right), \boldsymbol{k}_x\, sin\left(\frac{\theta}{2}\right), \boldsymbol{k}_y\, sin\left(\frac{\theta}{2}\right), \boldsymbol{k}_z\, sin\left(\frac{\theta}{2}\right)\right) \tag{3.20}$$

Na equação 3.20, $\boldsymbol{k}_x$, $\boldsymbol{k}_y$ e $\boldsymbol{k}_z$ representam as coordenadas do vetor $\boldsymbol{k}$, no sistema de coordenadas de referência, que indica a direção da rotação do quatérnio. A Fig. 3.7 ilustra a situação.

Fig. 3.7: Representação da rotação do sistema de coordenadas *OUVW*, em relação ao sistema *OXYZ*, por um ângulo θ na direção do vetor $\boldsymbol{k}$



Fonte: produção próprio autor

A matriz de transformação homogênea **T** equivalente ao quatérnio $\boldsymbol{q}$ é dada pela equação 3.21 (BARRIENTOS, 2007):

$$\mathbf{T} = 2\begin{bmatrix} q_0^2 + q_1^2 - \frac{1}{2} & q_1q_2 - q_3q_0 & q_1q_3 + q_2q_0 & 0 \\ q_1q_2 + q_3q_0 & q_0^2 + q_2^2 - \frac{1}{2} & q_2q_3 - q_1q_0 & 0 \\ q_1q_3 - q_2q_0 & q_2q_3 + q_1q_0 & q_0^2 + q_3^2 - \frac{1}{2} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.21}$$

Considerando uma matriz de transformação genérica, como apresentada na equação 3.22, a obtenção do quatérnio equivalente a essa matriz de transformação homogênea, evidenciada nas equações 3.23, pode ser encontrada igualando o traço e os elementos da diagonal principal da matriz da equação 3.21 com a equação 3.22 (BARRIENTOS, 2007).

$$\mathbf{T} = \begin{bmatrix} n_x & o_x & a_x & 0 \\ n_y & o_y & a_y & 0 \\ n_z & o_z & a_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.22}$$

$$\begin{aligned} q_0 &= \frac{1}{2}\sqrt{(n_x + o_y + a_z + 1)} \\ q_1 &= \frac{1}{2}\sqrt{(n_x - o_y - a_z + 1)} \\ q_2 &= \frac{1}{2}\sqrt{(-n_x + o_y - a_z + 1)} \\ q_3 &= \frac{1}{2}\sqrt{(-n_x - o_y + a_z + 1)} \end{aligned} \tag{3.23}$$

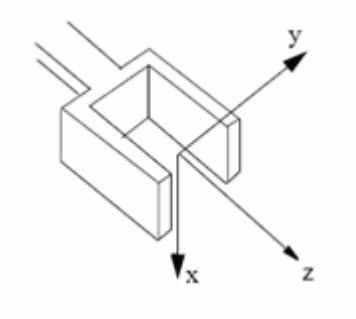
3.4 *TOOL CENTER POINT* DO ROBÔ IRB6640

A posição do robô e os seus movimentos estão sempre relacionados ao *tool center point* (TCP). Esse ponto é normalmente definido como algum lugar na ferramenta de trabalho do robô, como, por exemplo, o bico de uma pistola de pintura ou o centro de uma garra (ABB, 2007a).

No programa do robô, quando se grava um ponto, são as coordenadas do TCP que são armazenadas, além do que é o TCP que se movimenta ao longo de uma trajetória (ABB, 2007a).

A ferramenta montada no flange do robô, frequentemente, requer um sistema de coordenadas próprio, também chamado de *tool coordinate system*, para. O TCP será sempre a origem do sistema de coordenadas da ferramenta e deverá ser referenciado no ponto da ferramenta que necessita ser corretamente posicionado na área de trabalho (ABB, 2007a). A Fig. 3.8 expõe um exemplo de localização do TCP.

Fig. 3.8: Posição e orientação do TCP de uma ferramenta



Fonte: ABB (2007a)

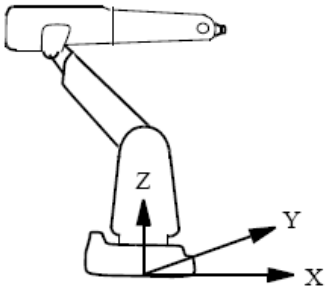
3.5 SISTEMAS DE COORDENADAS DO ROBÔ IRB6640

As posições do TCP podem ser especificadas em diferentes sistemas de coordenadas, para facilitar o programa e os possíveis reajustes. O sistema de coordenadas definido pelo usuário dependerá da tarefa que o robô vai executar. Se nenhum sistema de coordenadas for definido, os pontos serão referenciados ao sistema de coordenadas da base (*base coordinate system*) (ABB, 2007a).

3.5.1 Sistema de Coordenadas da Base

O ponto de referência do sistema de coordenadas da base está localizado na intersecção do centro da base do robô com o plano onde o robô está apoiado. Nesse sistema de coordenadas, o eixo Z é coincidente com o eixo 1 do robô e o plano XY paralelo com plano onde a base é montada, como na Fig. 3.9 (ABB, 2007a).

Fig. 3.9: Sistema de coordenadas da base

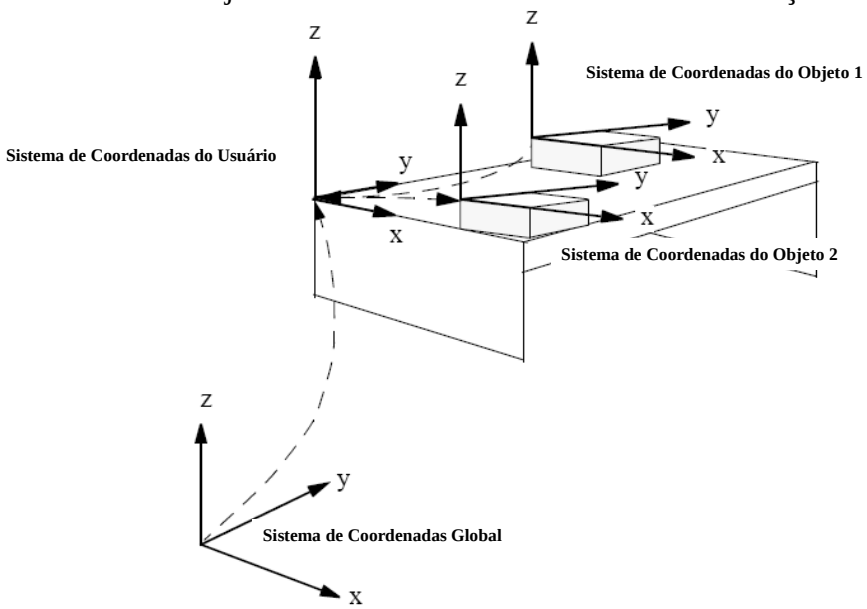


Fonte: ABB (2007a)

3.5.2 Sistema de Coordenadas do Objeto

O sistema de coordenadas do objeto serve para diferenciar estações de trabalho, porém em cada estação de trabalho pode haver vários objetos a serem manuseados. O sistema de coordenadas do objeto facilita a programação caso os objetos sejam reposicionados. Esse sistema de coordenadas é utilizado em programação *offline*, uma vez que as posições empregadas podem ser facilmente retiradas dos desenhos da célula de trabalho. O sistema de coordenadas do objeto está demonstrado na Fig. 3.10 (ABB, 2007a).

Fig. 3.10: Dois diferentes Sistemas de Coordenadas do Objeto descrevem a posição de dois diferentes objetos de trabalho localizados na mesma estação



Fonte: adaptado de ABB (2007a)

3.6 PROGRAMAÇÃO DO ROBÔ

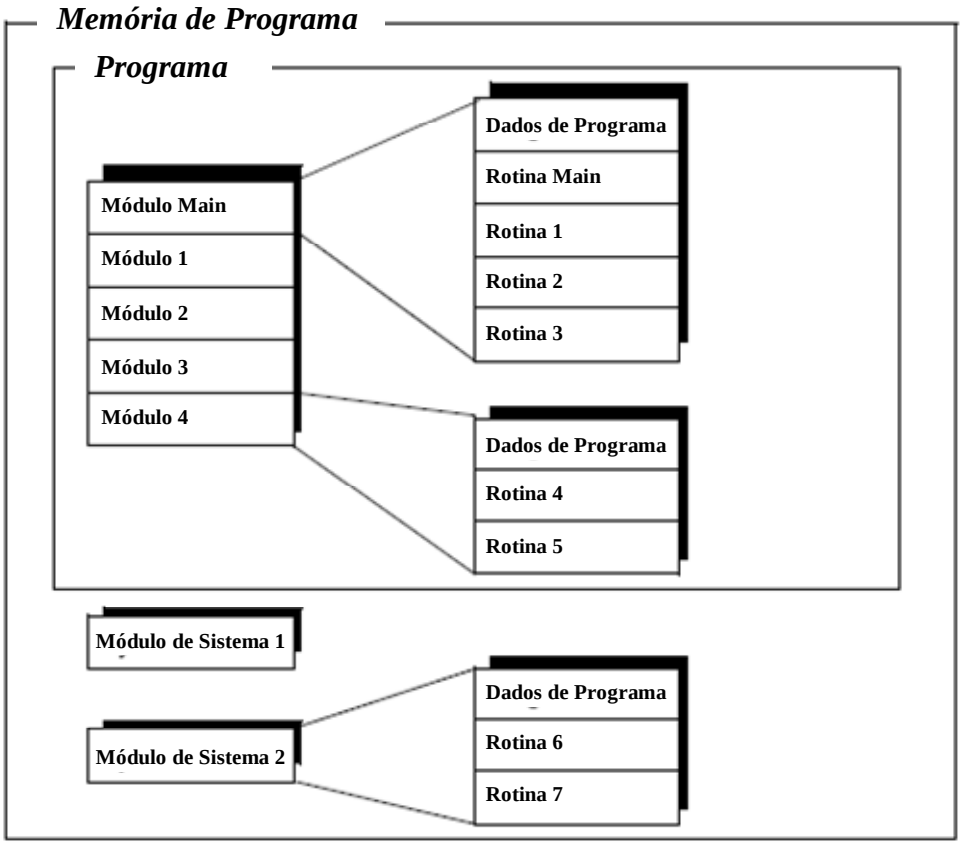
Os robôs industriais são equipamentos automatizados projetados para movimentar peças ou ferramentas sobre uma trajetória previamente estabelecida. Programar um robô significa descrever os procedimentos a serem seguidos para a execução de uma tarefa.

Em linhas gerais, o programa de um robô industrial possui instruções de lógica e instruções de movimento. As instruções de lógica processam os sinais e são responsáveis pelas tomadas de decisões. As instruções de movimento são predefinidas e, quando o robô está trabalhando, elas não se alteram. Dessa forma, ao declarar que o movimento do robô será de um ponto “A” para um ponto “B”, ele vai executar o mesmo movimento sempre que a instrução for chamada no programa.

3.6.1 A Estrutura da Linguagem de Programação do Robô ABB

A estrutura do programa do robô é constituída por um Módulo de Programa e por Módulos de Sistema. Um Módulo de Programa pode conter vários dados e rotinas. Um dos Módulos contém o procedimento de entrada, que é global, chamado de *main*. Executar o programa significa executar o procedimento *main*. O programa pode incluir muitos módulos, mas apenas um desses terá o procedimento *main*. Os Módulos de Sistema são usados para definir as rotinas comuns e os dados de sistema, como as ferramentas e as velocidades. A Fig. 3.11 apresenta a estrutura da Linguagem de Programação do Robô ABB.

Fig. 3.11: Estrutura da Linguagem de Programação do Robô ABB



Fonte: adaptado de ABB (2007a)

O programa do robô compõe-se de uma série de instruções que descrevem as suas tarefas. Desse modo, há instruções específicas para os vários comandos existentes no robô, como instruções para movimentar o robô, para deslocar um sistema de coordenadas, entre outros (ABB, 2007a). As instruções geralmente possuem um argumento associado que define o que está sendo manipulado. Existem três tipos de rotinas no programa do robô ABB, a saber: procedimentos, funções e rotinas de erro.

- Os procedimentos são utilizados como subprogramas;
- As funções retornam valores de um tipo específico e serve como argumento de uma instrução;
- As rotinas de erro constituem um meio de tratar as interrupções. Elas podem ser associadas a uma interrupção específica e são executadas automaticamente quando ela ocorrer.

Informações podem ser armazenadas em Dados, como, por exemplo, dados numéricos. Dados são agrupados em diferentes tipos, os quais descrevem diferentes tipos de informação, como ferramentas, posições e cargas. Existem três tipos de dados: *constants*, *variables* e *persistents* (ABB, 2007b).

- Uma *constant* representa um valor estático, e seu valor somente é alterado manualmente;
- Uma *variable* pode ser alterada durante a execução do programa;
- Um dado *persistent* pode ser descrito como uma variável persistente. Quando o programa é gravado, seu valor de inicialização reflete o valor atual do dado *persistent*.

3.6.2 Posicionamento do TCP Durante a Execução do Programa

As instruções de movimento controlam todo o deslocamento do robô durante a execução do seu programa. Essas instruções fornecem ao robô informações de como executar os movimentos, que são (ABB, 2007b):

- O ponto de destino do movimento (definido como a posição do TCP, a orientação da ferramenta, os parâmetros do robô e a posição dos eixos externos);
- O método de interpolação usado para alcançar o ponto de destino (interpolação por juntas, linear ou circular);
- A velocidade do robô e dos eixos externos (se possuir);
- Os dados de regiões (define como o robô e os eixos externos vão passar pelo ponto de destino);
- O sistema de coordenadas empregado no movimento (global, usuário ou objeto).

3.6.3 Instrução de Movimento, de *Offset* e de Deslocamento de Sistema de Coordenadas

Os movimentos do robô são programados como sendo de uma posição para outra, ou seja, movimentos da posição atual para uma nova posição. A trajetória entre essas duas posições é calculada pelo próprio robô. Define-se o tipo de trajetória de acordo com a instrução de posição utilizada. Os argumentos das instruções de posição são (ABB, 2007b):

- Dados de posição (posição final para o robô e eixos externos);
- Dados de velocidade (a velocidade desejada);
- Dados de região (precisão de posição);
- Dados de ferramenta (a posição do TCP);
- Dados de *work-object* (o sistema de coordenadas atual).

As instruções de movimento mais comuns são (ABB, 2007b):

- *MoveJ* (movimento orientado pelas juntas do robô);
- *MoveL* (TCP se move ao longo de uma trajetória linear);
- *MoveC* (movimentos do TCP em uma trajetória circular).

A função *Offs* adiciona um deslocamento (*offset*) a uma posição gravada no robô, expresso em relação ao sistema de coordenadas usado para executar o movimento.

A instrução de deslocamento de sistemas de coordenadas reposiciona e reorienta, momentaneamente, um sistema de coordenadas do robô. É utilizada quando todos os pontos, gravados no programa do robô e referenciados ao mesmo sistema de coordenadas, necessitam ser deslocados. Ao ser desabilitada, o sistema de coordenadas do robô retorna à sua posição e orientação iniciais. A instrução possui como atributos três coordenadas de posição (x,y,z) e os parâmetros de um quatérnio (q_1,q_2,q_3,q_4) para representar quanto o sistema será deslocado. As principais instruções de deslocamento de programa são (ABB, 2007b):

- *PDispOn* (ativa um deslocamento de programa);
- *PDispSet* (ativa um deslocamento de programa e especifica os seus valores);
- *PDispOff* (desativa um deslocamento de programa).

A instrução de deslocamento é parametrizável, assim a posição/orientação do sistema de coordenadas do robô se movimenta de acordo com o valor dos parâmetros de entrada do programa do robô. Os parâmetros podem estar armazenados no próprio programa ou podem ser carregados de um sistema externo por meio de redes de comunicação.

3.6.4 Parametrização Aplicada ao Projeto

Existem recursos que tornam o programa do robô mais robusto, um deles é utilizar programação parametrizável, ou seja, a posição e a orientação de um ponto podem variar de acordo com parâmetros de entrada no programa. Na grande maioria dos casos, esses parâmetros são inseridos no programa como sendo constantes, fazendo com que a trajetória do robô não mude enquanto ele estiver trabalhando, ou seja, o robô não consegue interagir, em tempo real, com alterações de posição e orientação dos objetos na área de trabalho.

Um sistema em tempo real é aquele que responde a estímulos externos em um tempo finito e determinado (LAPLANTE, 2004). Tais sistemas não devem ficar parados enquanto aguardam esses estímulos; eles devem continuamente ler as suas entradas a fim de interagir com o ambiente onde estão inseridos (OLDEROG; DIERKS, 2008).

Quando os parâmetros são inseridos no programa como sendo variáveis, o programa do robô torna-se mais dinâmico, as respostas ao sistema variam com os estímulos externos ao robô (OLDEROG; DIERKS, 2008; LAPLANTE, 2004). Esse tipo de programação deixa o

robô mais interativo e flexibiliza a célula de manufatura na qual ele está inserido. Assim, a célula pode processar diversos modelos de produtos diferentes sem que sejam feitas alterações na programação do robô, pois ele pode adaptar seus movimentos à medida que os parâmetros externos são fornecidos.

Em conjunto com um sistema de visão que seja capaz de encontrar a posição e orientação de objetos, é possível empregar a programação parametrizável de forma dinâmica. Dessa maneira, a trajetória do robô pode ser alterada de acordo com a posição e orientação de um objeto na área de trabalho do robô que o sistema de visão encontrar. Isso torna a utilização do robô industrial muito mais flexível, eliminando os dispositivos indexadores.

Um ponto gravado no programa do robô representa a posição e a orientação do TCP em relação a um sistema de coordenadas (conforme seção 3.4). Se o sistema de coordenadas for deslocado de uma posição para outra, todos os pontos gravados nele serão deslocados juntos. O programa desenvolvido no robô da Célula de Oleação e Paletização de Cabeçotes usou tal recurso, conforme é descrito na seção 5.7.2.

4 IDENTIFICAÇÃO E PEGA DOS CABEÇOTES

Este capítulo apresenta a Célula de Oleação e Paletização de Cabeçotes e a descrição das dificuldades encontradas na pega das peças pelo robô. Também descreve uma proposta de solução para o problema utilizando conceitos de visão computacional.

4.1 A CÉLULA DE OLEAÇÃO E PALETIZAÇÃO DE CABEÇOTES

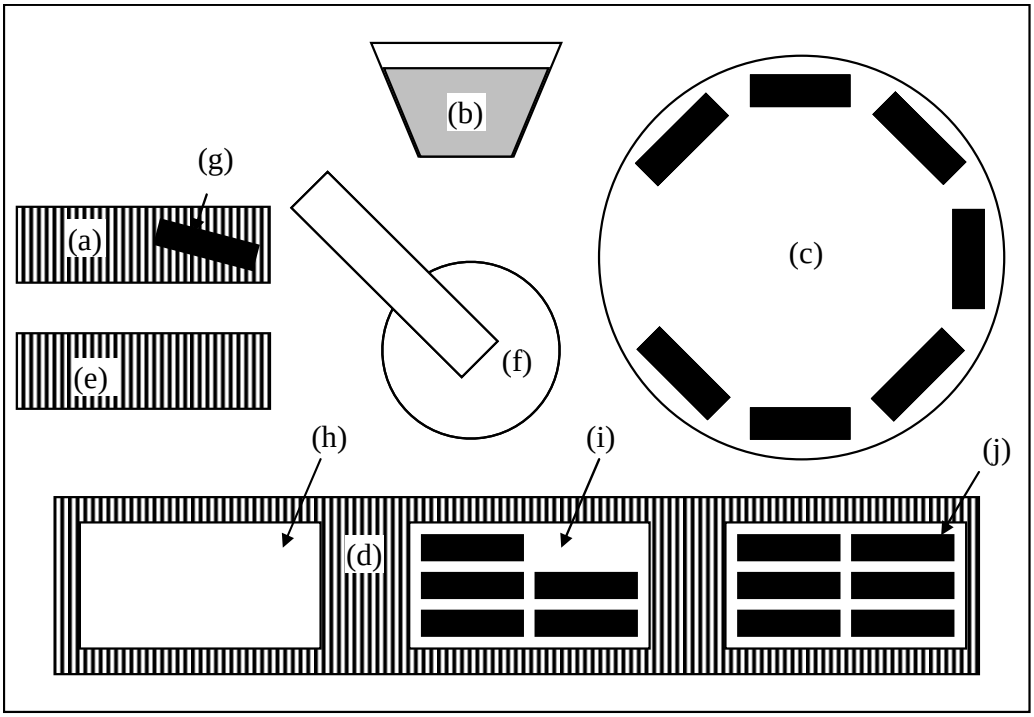
A Célula de Oleação e Paletização de Cabeçotes está em desenvolvimento e será instalada na linha de acabamento de cabeçotes fundidos E-II da Tupy S.A. Sua função será olear e paletizar as peças processadas nessa linha. A Fig 4.1 mostra o *layout* da célula.

O transportador que alimenta a célula é abastecido manualmente. Portanto, as peças são disponibilizadas ao robô em posições e orientações randômicas (Fig 4.1(g)). Outra dificuldade encontrada na célula é que muitos modelos de peças são processados na linha; ao total serão 26 modelos pertencentes a seis famílias diferentes. A Fig. 4.3 mostra cabeçotes de três famílias, é possível perceber a falta de contraste entre as peças e o fundo da imagem. As peças processadas nessa célula são brutas e podem possuir rebarbas que dificultam o seu posicionamento.

O fluxograma da Fig. 4.2 traz as principais etapas do funcionamento da Célula de Oleação e Paletização de Cabeçotes. Como pode ser visto, o robô executa várias tarefas em um único ciclo de trabalho. O tempo disponível ao robô é de 28 segundos por ciclo para executar todas as etapas do processo da célula. As principais atividades do robô são:

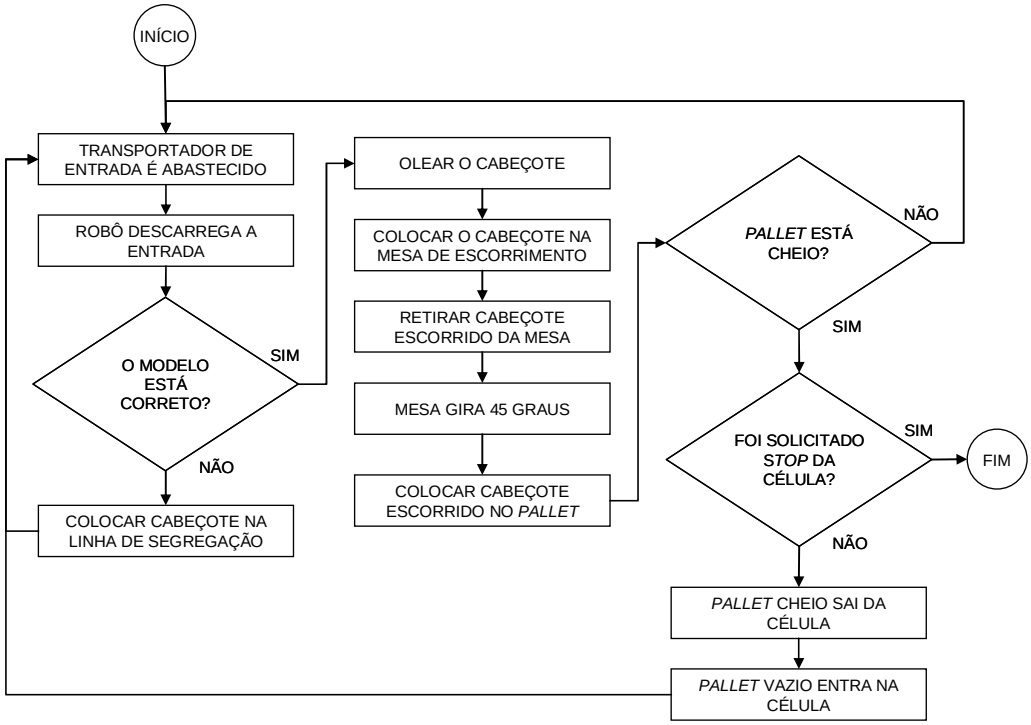
- Descarregar o transportador de entrada;
- Segregar se o cabeçote não for compatível com o modelo do *pallet*;
- Olear o cabeçote;
- Colocar o cabeçote na mesa de escorrimento;
- Retirar um cabeçote da mesa de escorrimento;
- Paletizar o cabeçote.

Fig. 4.1: *Layout* da Célula de Oleação e Paletização de Cabeçotes. (a) transportador de entrada, (b) tanque de óleo, (c) mesa de escorrimento do óleo, (d) linha de paletização dos cabeçotes, (e) linha de segregação de peças, (f) robô industrial, (g) cabeçote no transportador de entrada, (h) *pallet* vazio, (i) *pallet* sendo carregado e (j) *pallet* totalmente carregado



Fonte: produção próprio autor

Fig. 4.2: Fluxograma do funcionamento da Célula de Oleação e Paletização de Cabeçotes



Fonte: produção próprio autor

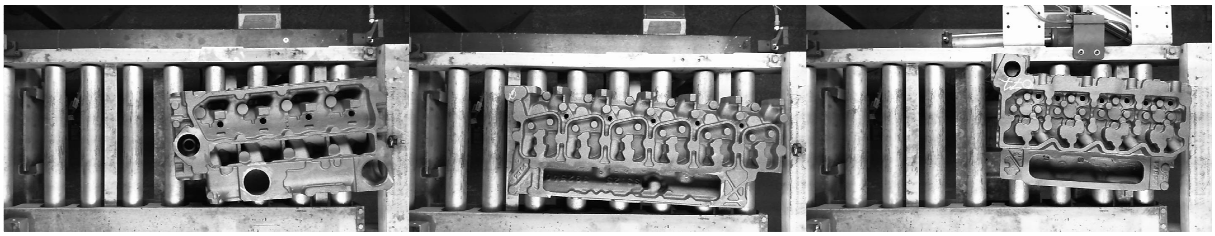
4.2 O PROBLEMA DE DESCARREGAR O TRANSPORTADOR DE ENTRADA

Para um robô que manipula peças em uma célula de trabalho é fundamental que a posição e a orientação dos objetos a serem manipulados sejam conhecidas. Aplicações mais simples utilizam dispositivos indexadores para garantir a posição e a orientação da peça com a qual o robô vai interagir (ROMANO, 2002). Esse tipo de dispositivo fixa a peça em um local conhecido pelo robô e monitora a sua posição e a orientação, informando ao robô se ela está pronta ou não para ele executar a sua tarefa.

A produção na linha de acabamento de cabeçotes não é por lote, ou seja, famílias diferentes de cabeçotes podem estar misturadas. Dispositivos indexadores geralmente não são apropriados a tal tipo de produção, uma vez que cada modelo de peça possui o seu. Na troca da produção de um modelo de cabeçote para outro, perdas significativas de tempo podem ocorrer em virtude da substituição de um indexador por outro. Algumas dificuldades na utilização desses dispositivos podem ser listadas:

- As peças operadas nessa célula são brutas;
- A célula é abastecida manualmente;
- As peças são posicionadas randomicamente para a pega pelo robô;
- O indexador não detecta se a peça está rotacionada 180 graus no transportador e nesse caso não consegue posicioná-la e/ou orientá-la;
- São muitos modelos diferentes de cabeçotes;
- Seria necessário um indexador para cada família de cabeçote;
- O tempo de ciclo da célula é muito pequeno para efetuar a troca de dispositivos indexadores.

Fig. 4.3: Exemplo de três famílias diferentes de cabeçotes manipulados pelo robô



Fonte: produção próprio autor

Na Fig. 4.3 pode-se ver que a linha de roletes, onde os cabeçotes são transportados, não possui guias. As peças ficam livres sobre a linha; essa característica aumenta a dificuldade na pega pelo robô.

O sistema terá de enviar as informações de posição/orientação ao robô sem que haja perdas de informação. Como as peças a serem manipuladas na célula em questão estarão brutas, as variações entre as peças não poderão influenciar no desempenho do sistema. A Célula de Oleação e Paletização de Cabeçotes trabalhará 24 horas por dia, e por consequência

o sistema a ser desenvolvido deverá estar disponível no mesmo período, sem comprometer a produtividade da célula.

4.3 SOLUÇÃO PROPOSTA PARA GUIAR O ROBÔ

A solução proposta divide o problema em duas partes:

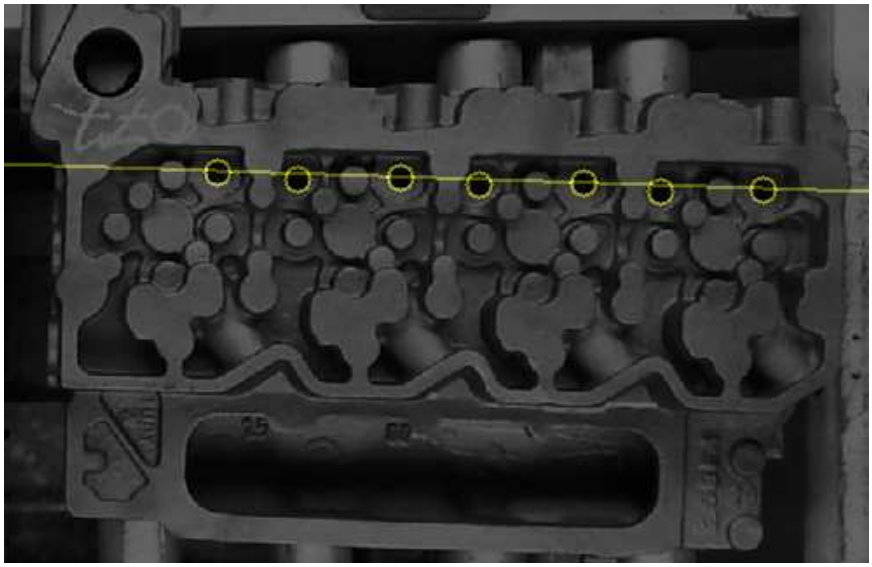
- Identificação do modelo de cabeçote;
- Encontrar a posição/orientação.

O sistema de visão proposto vai diferenciar as famílias de cabeçotes por meio de um método de medição de similaridades. Uma base de dados será utilizada para identificar os modelos. Nela algumas imagens de cada modelo de cabeçote serão armazenadas. O algoritmo terá de comparar a imagem do cabeçote que entrou na célula com as imagens da base de dados e calcular o percentual de similaridade. O maior valor de similaridade indicará o modelo do cabeçote que entrou na célula.

A Fig. 4.3 mostra que alguns modelos de cabeçotes possuem furos grandes. Essa característica não pode ser utilizada na detecção do modelo porque existem diferentes peças com o mesmo número de furos grandes. Outra dificuldade está na segmentação, pois, em virtude do tamanho desses furos, não é possível criar uma sombra uniforme no seu interior.

Uma vez identificado o modelo do cabeçote, o sistema de visão carregará parâmetros específicos, correspondentes ao modelo, para encontrar sua posição/orientação. Os centros de alguns furos dos cabeçotes indicam a sua orientação (Fig. 4.4) em relação ao sistema de coordenadas da câmera. A posição do cabeçote é encontrada por meio do cálculo de seu deslocamento em relação à posição de referência. Essas informações serão repassadas ao robô de manipulação para ele efetuar a pega dos cabeçotes que entrarem na célula.

Fig. 4.4: Orientação de um cabeçote baseada na detecção das circunferências



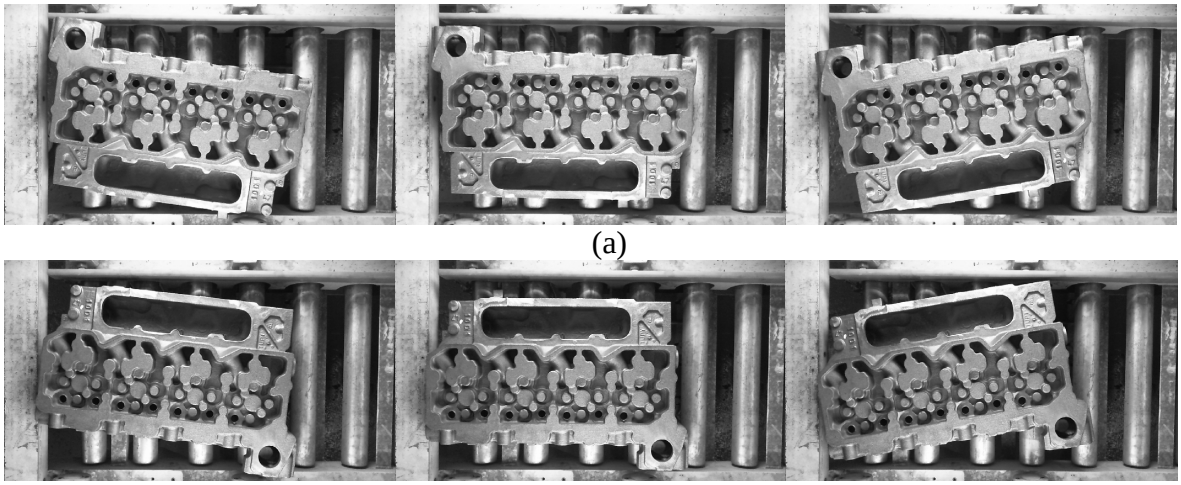
Fonte: produção próprio autor

4.3.1 Proposta para Determinar o Modelo dos Cabeçotes

Na fase de identificação da peça criou-se uma base de dados dividida em grupos; cada um deles é composto por seis imagens de cada cabeçote de motor. As seis imagens de cada grupo formam dois subgrupos, um representando as peças colocadas no sentido correto na esteira e outro representando as peças colocadas a 180º na esteira. Isso porque o operador pode abastecer a esteira com peças em qualquer orientação. Cada subgrupo possui três imagens da peça, representando três orientações distintas, ou seja, rotação positiva, negativa e sem rotação. O valor máximo de rotação é de 15º tanto no sentido positivo quanto no negativo. A restrição se dá pela largura do transportador e pelo tamanho do cabeçote.

Para cada uma das imagens da base de dados, são calculados os índices de similaridades em relação à imagem que está sendo identificada. Após encontrar os coeficientes, calcula-se a média aritmética dos índices de cada imagem dentro do subgrupo. A maior média indica a qual subgrupo a imagem pertence. Isso aumenta a robustez do processo e está coerente com a necessidade do algoritmo, já que ele precisa saber apenas a qual subgrupo o cabeçote pertence, e não a qual imagem do subgrupo a imagem comparada corresponde. A Fig. 4.5 apresenta um grupo com seus dois subgrupos de cabeçotes.

Fig. 4.5: (a) subgrupo representando as peças no sentido correto, (b) subgrupo representando as peças no sentido oposto no transportador



(a)

(b)

Fonte: produção próprio autor

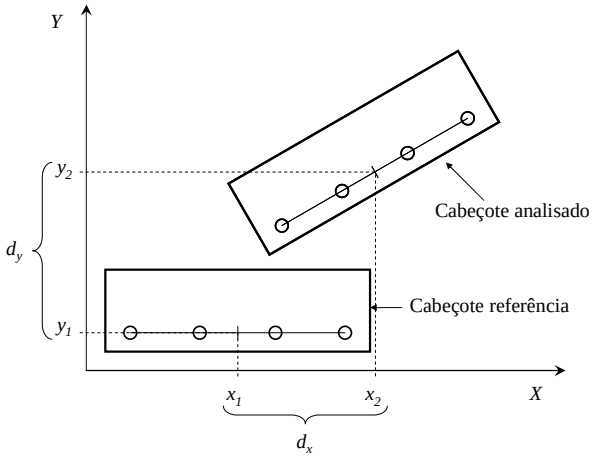
4.3.2 Proposta para Encontrar a Posição/Orientação dos Cabeçotes

Após classificar o modelo do cabeçote, o algoritmo encontra a posição dos furos utilizados no projeto. Com tais informações o sistema de visão encontrará, utilizando os quatérnios, a rotação do cabeçote em relação ao sistema de coordenadas da câmera.

O programa do robô e o sistema de visão conhecem a posição de referência para cada modelo de cabeçote. O sistema de visão calcula o deslocamento entre o cabeçote analisado e o cabeçote referência. O valor encontrado é passado ao robô. Com essa informação o robô desloca o sistema de coordenadas usado como base para a pega das peças. Dessa forma, para o robô o novo cabeçote encontra-se na mesma posição do cabeçote referência. Sempre que o sistema de coordenadas do robô for reposicionado, o ponto de pega será movimentado na mesma proporção.

Para a peça referência foi encontrada a posição do ponto central, entre os dois furos das extremidades, com relação à origem do sistema. Para cada novo cabeçote analisado, o ponto central deve ser calculado. Feito isso, basta encontrar o deslocamento (d_x,d_y) do cabeçote que está sendo analisado em relação à peça de referência. A Fig. 4.6 mostra o deslocamento entre um cabeçote analisado e o cabeçote referência.

Fig. 4.6: Deslocamento (d_x,d_y) em relação ao cabeçote referência



Fonte: produção próprio autor

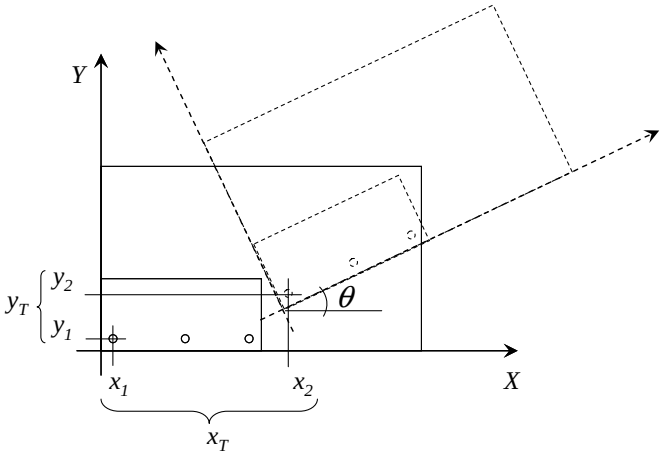
Após calcular o deslocamento do cabeçote, é preciso encontrar a orientação do sistema. Para isso, faz-se necessário saber qual o ângulo em que o cabeçote foi rotacionado. É possível encontrar o quatérnio que representa a rotação do cabeçote por meio da equação 4.1. Encontrando o quatérnio, basta passar a informação ao robô juntamente com o deslocamento encontrado. Tais dados são suficientes para que o robô possa pegar o cabeçote na esteira de entrada da célula.

$$
\begin{aligned}
q_0 &= \frac{1}{2}\sqrt{\cos\theta + \cos\theta + 1 + 1} = \cos\frac{\theta}{2} \\
q_1 &= \frac{1}{2}\sqrt{\cos\theta - \cos\theta - 1 + 1} = 0 \\
q_2 &= \frac{1}{2}\sqrt{-\cos\theta + \cos\theta - 1 + 1} = 0 \\
q_3 &= \frac{1}{2}\sqrt{-\cos\theta - \cos\theta + 1 + 1} = sen\frac{\theta}{2}
\end{aligned}
\tag{4.1}
$$

Pode-se notar nas equações 4.1 que o quatérnio representa apenas a parte de rotação do sistema de coordenadas e não depende da translação que o sistema sofreu.

A Fig. 4.7 mostra a representação de uma rotação e uma translação, em um cabeçote, e como a origem e a direção do sistema de coordenadas podem ser encontradas.

Fig. 4.7: Rotação e translação de um cabeçote e por consequência do sistema de coordenadas



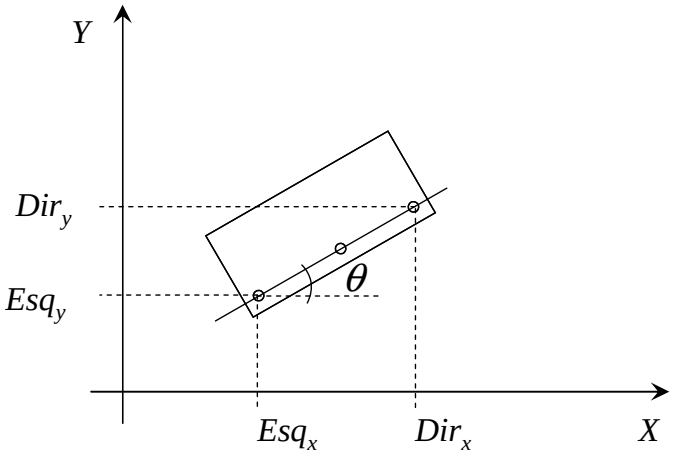
Fonte: produção próprio autor

O ângulo θ, como mostrado na Fig. 4.8, pode ser encontrado mediante a equação da tangente e está evidenciado na equação 4.2:

$$tg\theta = \left(\frac{Dir_y - Esq_y}{Dir_x - Esq_x} \right)$$
$$\theta = \arctan\left(\frac{Dir_y - Esq_y}{Dir_x - Esq_y} \right) \quad (4.2)$$

Em que (Esq_x,Esq_y) e (Dir_x, Dir_y) são as coordenadas do primeiro e último furo, respectivamente, utilizados para encontrar a orientação do cabeçote.

Fig. 4.8: Rotação de um cabeçote por um ângulo θ



Fonte: produção próprio autor

5 IMPLEMENTAÇÃO DO PROJETO

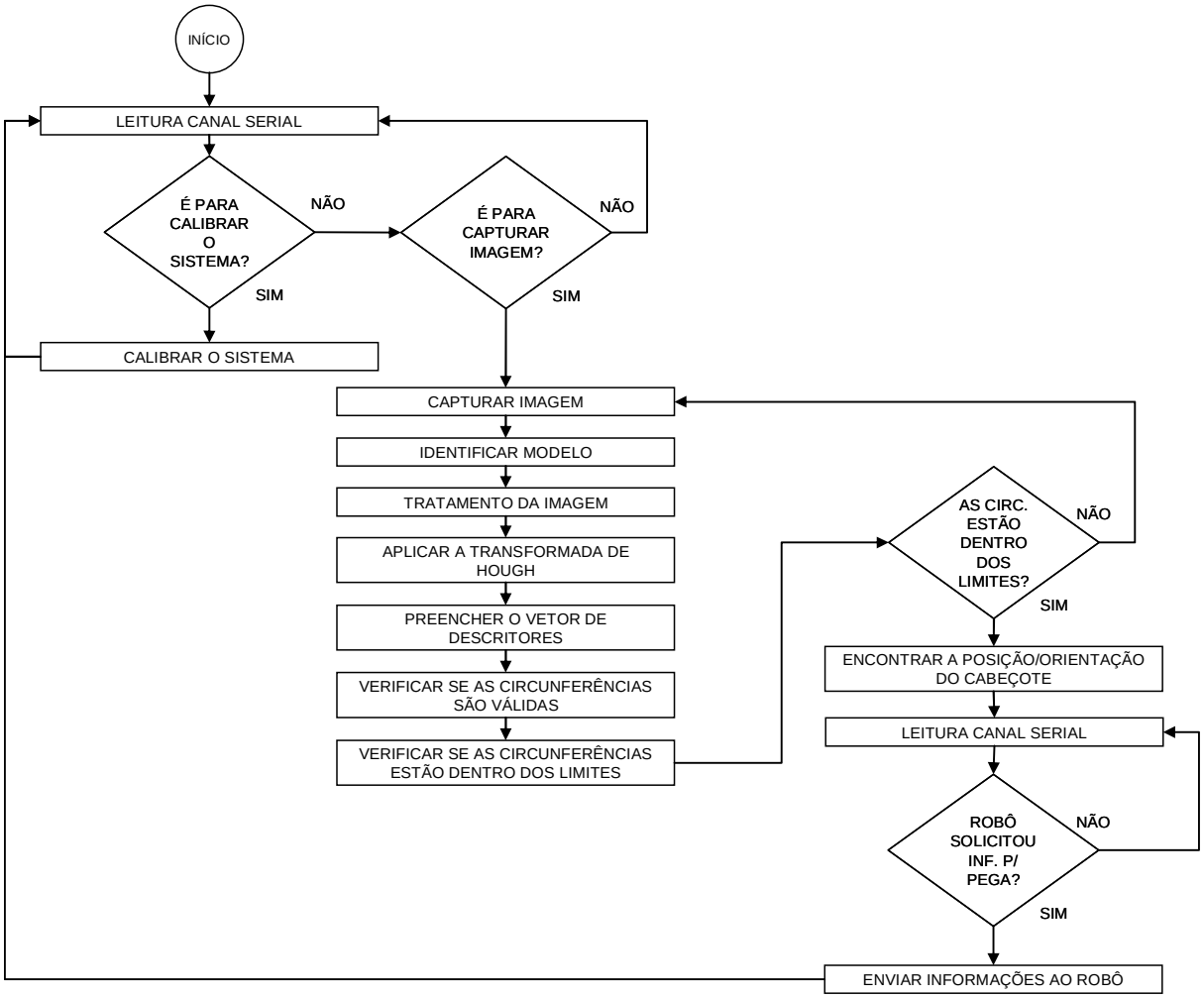
Este capítulo apresenta o desenvolvimento do *hardware* e do *software* utilizados no projeto do sistema de visão, bem como o desenvolvimento do *software* do robô. Primeiramente será feito um detalhamento do *hardware* e do *software* usados no sistema de visão. Na sequência serão descritas as implementações do *software* do sistema de visão e do *software* do robô.

5.1 VISÃO GERAL DO PROJETO

Empregando os conceitos apresentados neste trabalho, desenvolveu-se o sistema de visão que guia o robô de oleação e paletização de cabeçotes. Ele é composto de *hardware*, *software* e rede de comunicação para trocar informações entre o robô e o sistema de visão.

Os cabeçotes entram na célula através de um transportador do tipo linha de roletes. O abastecimento do transportador é feito manualmente, e por isso os cabeçotes são dispostos na linha aleatoriamente. O sistema de visão, que identifica o modelo do cabeçote e encontra a sua posição/orientação, está posicionado sobre o transportador de entrada. Depois de abastecido, o transportador desloca o cabeçote até atuar o sensor no final da linha. O robô recebe o sinal do sensor e comunica-se com o sistema de visão, solicitando que o processo de captura e processamento da imagem se inicie. Ao encontrar a posição/orientação do cabeçote, o sistema de visão aguarda a solicitação de envio das informações pelo robô. Essas informações são enviadas para o robô via comunicação serial para que dê início ao procedimento de pega do cabeçote na linha de entrada. A Fig. 5.1 apresenta o fluxograma do *software* do sistema de visão.

Fig. 5.1: Fluxograma do *software* do sistema de visão



Fonte: produção próprio autor

A iluminação do sistema de visão foi posicionada de tal forma a gerar sombras em alguns furos utilizados para encontrar a posição e a orientação dos cabeçotes. A primeira operação a ser feita é a calibração do sistema de visão, para isso usa-se uma placa com três furos que formam a origem e os eixos X e Y do sistema de coordenadas da câmera. Com o sistema calibrado é possível iniciar o procedimento para encontrar a posição e a orientação dos cabeçotes. Após a captura da imagem, o sistema de visão identifica o modelo do cabeçote. Essa informação é importante porque define os parâmetros a serem usados nas operações subsequentes. A imagem precisa ser tratada antes de aplicar a transformada de Hough, que detecta os furos utilizados no projeto. Nessa fase empregam-se uma transformação de brilho/contraste, *thresholding* e transformações morfológicas (DOUGHERTY; LOTUFO, 2003). O resultado dessas transformações é o espaço de parâmetros, cujos picos representam o centro de possíveis circunferências. No próximo passo, os picos são localizados e tratados. Efetua-se uma verificação na imagem binarizada, buscando a confirmação da existência de

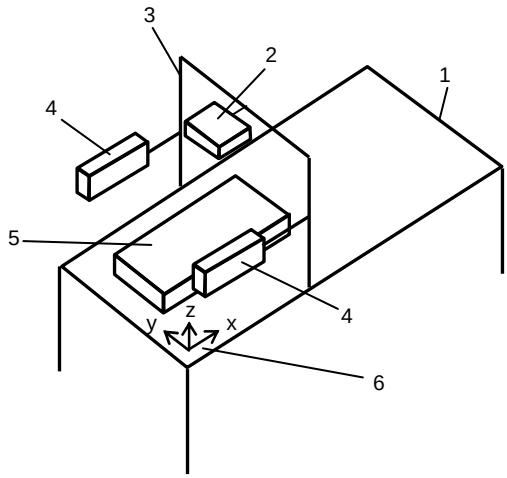
uma circunferência naquele local. Se o resultado for positivo, a coordenada de centro e o raio da circunferência são armazenados no vetor de descritores do algoritmo, caso contrário, a coordenada analisada é descartada. Após detectar todas as circunferências, realiza-se uma verificação no vetor de descritores para determinar se as posições encontradas estão na região desejada da imagem. A próxima etapa é determinar se a distância, entre a primeira e a última circunferência, está dentro do esperado. Podem ocorrer falhas na segmentação dos furos, por isso uma nova imagem é capturada caso as circunferências detectadas não estejam na região desejada. O vetor de descritores serve para encontrar a posição/orientação do cabeçote após ser tratado. As informações de deslocamento e orientação são enviadas ao robô por meio da comunicação serial para que ele possa fazer a pega do cabeçote na linha de entrada da célula.

O programa do robô foi construído de forma a solicitar as informações de deslocamento e orientação dos cabeçotes ao sistema de visão no momento de efetuar a pega. As posições de referência para cada modelo de cabeçote estão gravadas no programa do robô. Para cada cabeçote que entra na célula o robô utiliza as informações vindas do sistema de visão para executar a pega.

5.2 CONSTITUIÇÃO DO *HARDWARE* DO PROJETO

Como pode ser visto na Fig. 5.2, o *hardware* proposto é formado por uma linha de roletes, por uma *webcam* e por elementos de iluminação. A câmera está posicionada acima da linha de entrada, perpendicularmente ao cabeçote, a aproximadamente 700 mm acima da linha de roletes.

Fig. 5.2: *Layout* do protótipo. (1) linha de roletes, (2) câmera fotográfica, (3) suporte de fixação da câmera, (4) iluminação com ajuste de altura, (5) cabeçote e (6) ponto de referência para o sistema de coordenadas



Fonte: produção próprio autor

5.2.1 Iluminação

Em um sistema de visão a iluminação constitui um aspecto importante a ser observado. Uma escolha errada pode fazer com que a aquisição da imagem não seja a ideal, gerando dificuldades no seu processamento (ARAÚJO, 2010; DAVIES, 2005). Alguns tipos de iluminação podem ser utilizados. Uma delas é a frontal, onde a fonte luminosa é colocada acima do objeto, iluminando a superfície e realçando suas características, como forma e dimensão (ARAÚJO, 2010).

A fonte de iluminação deve fornecer ao sistema de visão as melhores imagens capturadas sob uma dada circunstância. Dessa forma, as imagens possuem mais contraste entre a área de interesse e o fundo. O projeto da iluminação deve garantir que fontes de luz externas não vão interferir na captura da imagem (ARAÚJO, 2010; NOVINI, 1993).

Alguns fatores devem ser considerados na escolha do tipo de iluminação que será empregado em um sistema de visão (NOVINI, 1993):

- Qual detalhe está sendo analisado na imagem?
- As imagens serão de objetos em movimento ou estacionários?
- Em qual ambiente a aplicação será instalada (óleo, poeira...)?
- Que tipo de câmera será utilizado no sistema de visão?

Existem algumas técnicas de iluminação que podem ser utilizadas em um sistema de visão (NOVINI, 1993):

- *Front light*;
- *Back light*;
- *Structured light*.

Na *front light* a iluminação está localizada na frente da região a ser inspecionada, ou seja, no mesmo lado da câmera. Essa técnica pode ser usada quando a região a ser inspecionada é a superfície ou a textura de objetos (ARAÚJO, 2010; NOVINI, 1993).

A *back light* usualmente fornece ao sistema de visão um maior contraste e pode simplificar o algoritmo. Essa técnica está limitada a alguns casos, uma vez que ela realça apenas o contorno de objetos (ARAÚJO, 2010; NOVINI, 1993).

Structured light é a fonte de luz com a forma de um holofote projetado, controlado por algum meio. O efeito pode ser alcançado de diversas formas, por exemplo, o uso de aberturas e lentes ou o uso de lasers (NOVINI, 1993).

A iluminação mostra-se um dos mais importantes parâmetros no desenvolvimento de um sistema de visão (ARAÚJO, 2010). Neste projeto ela foi posicionada com o intuito de formar sombra nos furos usados para a determinação da posição/orientação do cabeçote na linha de roletes. A sombra torna os furos escuros, facilitando o processo de segmentação. Realizaram-se vários testes com posições diferentes para a iluminação. Colocou-se em baixo do transportador (linha de roletes), por cima, inclinada e na lateral (com variações verticais). Os melhores resultados foram obtidos com a iluminação direcionada à face superior dos cabeçotes, posicionada 700 milímetros acima da linha de roletes e inclinada a 30^0 em relação à reta normal à linha de entrada. Dessa forma, os furos ficaram mais realçados e as sombras indesejadas tornaram-se menos prejudiciais.

Foram utilizadas duas luminárias contendo quatro lâmpadas fluorescentes de 40W cada.

5.2.2 Câmera

As imagens são capturadas por meio de uma *webcam* de uso comum e baixo custo, se comparada a câmeras de uso industrial. Utilizou-se no projeto uma câmera Microsoft, modelo LifeCam HD-5000 de alta resolução (HD – *high definition*), ajustada para o formato de imagem com 960X544 *pixels*.

A utilização da *webcam* comprova que uma câmera de baixo custo pode resolver

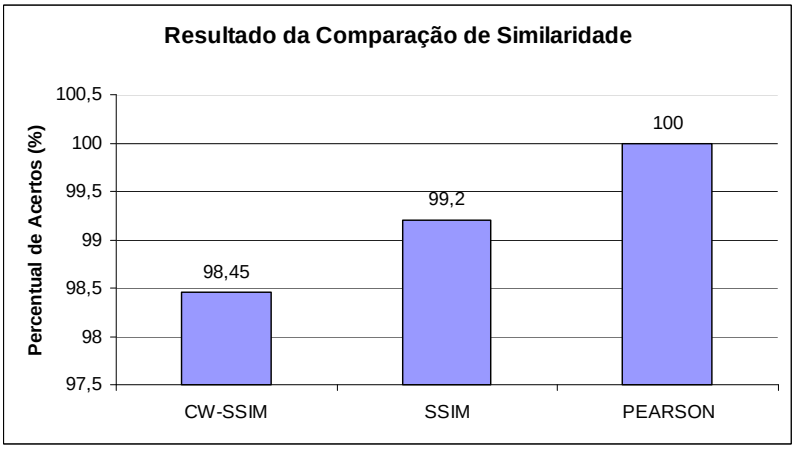
problemas em ambientes industriais.

5.3 MÉTODO DE DETECÇÃO DE MODELO DO CABEÇOTE

A primeira etapa do sistema de visão desenvolvido é determinar o modelo do cabeçote que está na área de pega do robô. Para tanto avaliaram-se três métodos de medição de similaridades, que são: correlação de Pearson (seção 2.6.1), SSIM (seção 2.6.2) e CW-SSIM (seção 2.6.3). A escolha foi baseada na ferramenta com melhores resultados. A precisão desses métodos e o tempo de processamento foram as características analisadas.

Na avaliação aplicou-se o processo exposto na seção 4.3.1. O banco de imagens foi composto por 18 imagens. Calculou-se o índice de similaridade comparando uma imagem capturada com cada imagem do banco. O resultado são 18 índices de similaridade para cada nova imagem capturada. A média entre os índices de cada subgrupo foi calculada para aumentar a robustez do processo.

Fig. 5.3: Resultado da comparação de similaridade entre os três métodos analisados

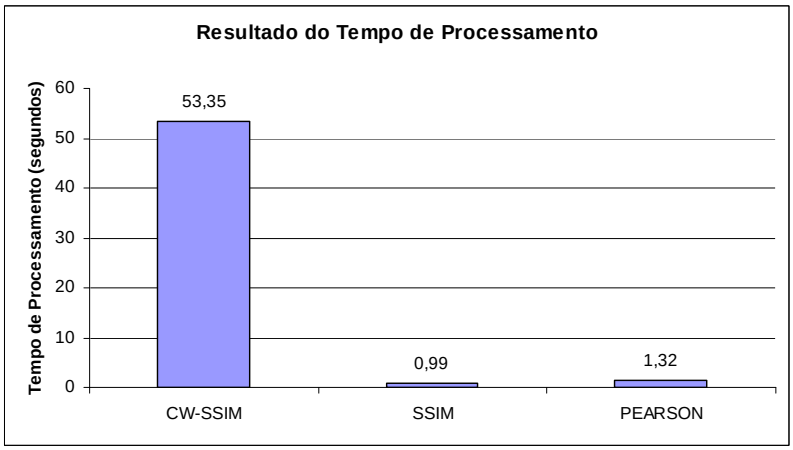


Fonte: produção próprio autor

Analisaram-se 129 imagens, e o percentual de acerto de cada método avaliado pode ser visto na Fig. 5.3. O tempo de processamento para calcular o índice de similaridade entre uma imagem e o banco de imagens foi analisado, porém ele possui alta dependência da capacidade do *hardware* empregado, bem como da implementação utilizando linguagens de programação interpretadas, como Matlab (considerando o uso de comandos iterativos ou em baixo nível). O *hardware* utilizado foi um *notebook* Toshiba, modelo Satellite A135-S2296, processador Pentium 1,6GHz dual core, 1Gb de RAM e sistema operacional Windows Vista. A Fig. 5.4 apresenta o tempo total de processamento, para os cálculos de similaridade e de média, entre uma imagem e as 18 imagens do banco. Recorreu-se ao Matlab para calcular o coeficiente de correlação de Pearson, SSIM e CW-SSIM. A implementação do CW-SSIM

(SAMPAT *et al.*, 2009) no Matlab foi cedido gentilmente pelo Prof. Mehul Sampat.

Fig. 5.4: Tempo de processamento entre os três métodos analisados

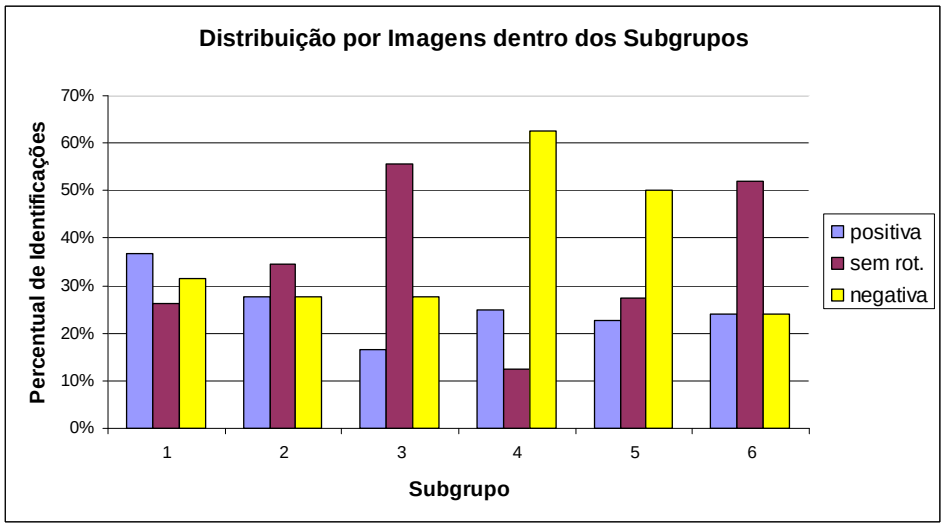


Fonte: produção próprio autor

O método escolhido para identificar o modelo dos cabeçotes foi a correlação de Pearson. Nos testes de identificação do modelo dos cabeçotes, 100% das imagens foram corretamente identificadas utilizando a correlação de Pearson. O tempo de processamento desse método está de acordo com os requisitos do projeto. O banco de imagens criado possui apenas três dos seis grupos de cabeçotes produzidos pela empresa. Os demais modelos não foram disponibilizados pela companhia durante a fase dos testes de validação. A célula vai processar cabeçotes pequenos e grandes, entretanto apenas os cabeçotes pequenos foram usados durante os testes.

Os resultados da correlação de Pearson foram analisados para verificar se a utilização de três imagens por subgrupo é realmente necessária, ou se uma imagem por subgrupo seria o suficiente. Analisar uma imagem por subgrupo reduzirá o tempo de processamento para 1/3 do tempo gasto na análise de três imagens por subgrupo, pois tal método é linear. O gráfico da Fig. 5.5 foi construído utilizando o maior valor de similaridade encontrado para os cabeçotes analisados. O resultado mostra o percentual de peças identificadas para cada imagem dos subgrupos. A distribuição no gráfico comprova a necessidade do uso das três rotações distintas, uma vez que um banco de imagens contendo cabeçotes em uma única rotação não seria suficiente para identificar as peças no transportador de entrada. Observa-se que nos subgrupos 1 e 2 existe uma distribuição quase uniforme entre as suas três imagens. Nos subgrupos 3 e 6 existe uma predominância a identificar as peças utilizando a imagem sem rotação do cabeçote. Por fim, nos subgrupos 4 e 5 o maior percentual de identificação ocorreu utilizando a imagem com o cabeçote na rotação negativa.

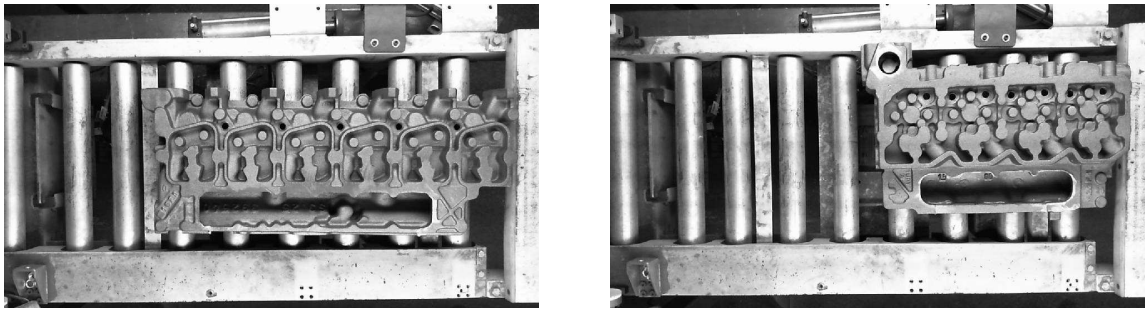
Fig. 5.5: Gráfico da distribuição de identificações por subgrupo



Fonte: produção próprio autor

A Fig. 5.3 aponta o melhor resultado para a correlação de Pearson, mesmo ela sendo um método de análise global. O fundo da imagem possui uma grande influência no resultado do cálculo de similaridade. A câmera do sistema deve possuir um campo de visão amplo para que as imagens dos cabeçotes grandes possam ser processadas corretamente. Tais cabeçotes deixam visíveis apenas dois roletes da linha de transporte (Fig. 5.6(a)). Quando um cabeçote pequeno é analisado, quatro roletes ficam expostos (Fig. 5.6(b)). Nesse caso SSIM e CW-SSIM utilizam a estrutura dos roletes na imagem para compor o índice de similaridade. Isso pode gerar um erro na identificação do modelo, pois, mesmo possuindo modelos de cabeçotes diferentes, o resultado da comparação entre duas imagens pode apontar um índice de similaridade bem grande.

Fig. 5.6: (a) cabeçote grande, (b) cabeçote pequeno



(a) (b)

Fonte: produção próprio autor

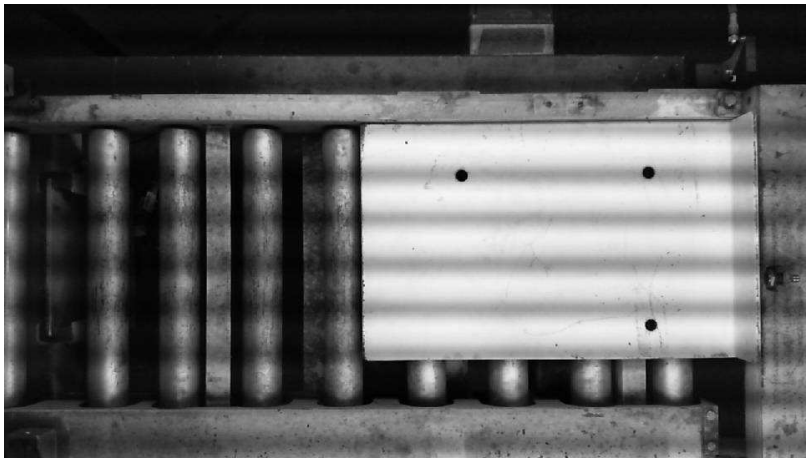
5.4 CALIBRAÇÃO DO SISTEMA DE VISÃO

A calibração da câmera constitui uma etapa necessária em um sistema de visão. Uma calibração precisa torna-se crucial em aplicações que envolvem medidas quantitativas, como mensurar a dimensão de um objeto. A calibração é o processo que determina os parâmetros intrínsecos, como distância focal, fator de escala, centro da imagem, e extrínsecos, como posição e orientação da câmera com relação a um determinado sistema de coordenadas (SIRISANTISAMRID; TIRASESTH; MATSUURA, 2005).

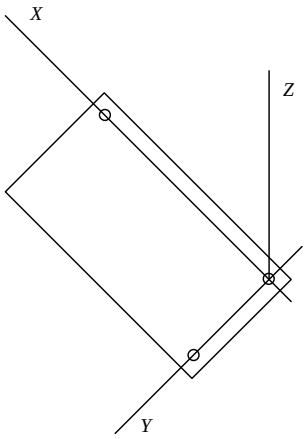
A calibração de uma câmera pode ser feita observando um padrão de calibração em diferentes orientações. O primeiro passo na calibração é fazer a aquisição de algumas imagens do padrão de calibração. O próximo passo é extrair características dos objetos da imagem e compará-las com aquelas do padrão. Uma vez estabelecida a correspondência entre os pontos na imagem e os pontos do padrão, o terceiro passo é calcular os parâmetros que determinam a dimensão de um *pixel* (WANG; WANG; WU, 2010).

Para as informações de posição e orientação geradas por um sistema de visão serem entendidas por um robô, faz-se necessário que o sistema de coordenadas do robô seja calibrado na mesma posição em que o sistema de coordenadas da câmera for calibrado (CHENG; DENMAN, 2005). No projeto utilizou-se uma placa com furos para calibrar o sistema de visão e o sistema de coordenadas do robô. Essa placa possui três furos representando, respectivamente, a origem do sistema de coordenadas, a direção do eixo *X* e a direção do eixo *Y* que serão usados pelo robô e pelo sistema de visão. A direção do eixo *Z* será perpendicular ao plano *XY*, formado pelos furos da placa de calibração, considerando a “regra da mão direita”, como se depreende da Fig.5.7.

Fig. 5.7: (a) Placa de calibração e (b) os eixos do sistema de coordenadas de referência



(a)



(b)

Fonte: produção próprio autor

Com a superposição dos sistemas de coordenadas, na área de trabalho do robô, os deslocamentos encontrados pelo sistema de visão serão os mesmos que o robô deverá utilizar para pegar as peças (CHENG; DENMAN, 2005). Em um sistema de visão 2D a calibração da câmera estabelece o sistema de coordenadas do sistema de visão (CHENG; DENMAN, 2005).

A calibração é responsável também pela conversão de *pixels* da imagem em coordenadas do mundo real, com a finalidade de encontrar a dimensão real das distâncias em uma imagem (ARAÚJO, 2010). Para isso é necessário descobrir o tamanho de um *pixel* em unidades de medidas SI[7]. Encontra-se o tamanho de um *pixel* fazendo a razão de uma medida real pela encontrada na imagem (em *pixels*) (ARAÚJO, 2010), ou seja:

$$T_{pixel} = \frac{d_{xreal}}{d_{ximagem}} \tag{5.1}$$

ou

$$T_{pixel} = \frac{d_{yreal}}{d_{yimagem}} \tag{5.2}$$

Em que T_{pixel} é o tamanho de um *pixel*, d_{xreal} e d_{yreal} uma distância conhecida no mundo real, respectivamente, nos eixos *X* e *Y*, e $d_{ximagem}$ ou $d_{yimagem}$ é a distância na imagem correspondente ao mundo real, também nos eixos *X* e *Y*.

Uma dimensão no mundo real pode, então, ser encontrada fazendo a multiplicação do tamanho do *pixel* (T_{pixel}) pela quantidade de *pixels*, correspondente à mesma dimensão, encontrada na imagem ($d_{ximagem}$ ou $d_{yimagem}$) (ARAÚJO, 2010).

$$d_{xreal} = T_{pixel} \times d_{ximagem} \quad (5.3)$$

$$d_{yreal} = T_{pixel} \times d_{yimagem} \quad (5.4)$$

Dessa forma, é possível encontrar as coordenadas físicas de qualquer ponto, por meio das coordenadas medidas em valores de *pixels* na imagem.

5.4.1 Distorção Radial

Imagens capturadas podem apresentar curvaturas indesejadas em linhas retas, particularmente nas proximidades da periferia da imagem. Por razões de simetria, distorções que surgem nas imagens tendem a envolver expansões radiais ou contrações radiais relativas ao eixo óptico do equipamento de captura de imagens (DAVIES, 2005).

O tratamento desse tipo de distorção não será abordado no trabalho, pois a câmera empregada não apresentou tal problema.

5.5 COMUNICAÇÃO SERIAL RS-232

A rede de comunicação utilizada para trocar dados entre o sistema de visão e o robô foi a serial, com protocolo RS-232. A interface RS-232 é uma porta de dados largamente usada pela sua estrutura simples e por ser de rápida configuração (GAO *et al*., 2008). O padrão RS-232 possui uma boa imunidade a ruídos por ter níveis elétricos diferenciais em suas linhas. Mesmo assim, esse padrão é destinado a aplicações de curto alcance (SILVA; ANTONIO; SANTOS, 2008).

É possível comunicar o robô com um computador utilizando o canal serial e transferir informações entre os dispositivos. Para usar o canal serial do robô IRB6640, é necessário abri-lo, fazendo isso o canal recebe um descritor que serve como uma referência quando lê ou escreve informações na rede (ABB, 2008b).

O sistema operacional do robô disponibiliza algumas funcionalidades para a comunicação serial, uma delas é baseada em informações binárias. A proposta da comunicação binária é permitir que o robô se comunique com outros dispositivos. Para manusear esse tipo de comunicação, o sistema do robô disponibiliza instruções para leitura/escrita, funções para leitura e instruções de manipulação do canal serial (ABB, 2008a).

---

[7] SI – sistema internacional de medidas.

A sequência a seguir deve ser respeitada ao utilizar o canal serial do robô IRB6640:

- Abrir o canal serial;
- Ler ou escrever no canal serial;
- Fechar o canal serial.

5.5.1 Programando a Comunicação Serial RS-232 do Robô

As instruções básicas para efetuar a comunicação serial entre o robô e um dispositivo são:

- *Open* – utilizado para abrir o canal serial para leitura/escrita;
- *Write* – utilizado para escrever no canal serial;
- *Close* – utilizado para fechar o canal serial.

A função *ReadStr* é utilizada para ler uma *String* a partir do canal serial. O tipo de dado *iodev* contém a referência a um canal serial. Ele pode ser relacionado à porta de comunicação física por meio da instrução *Open* e então utilizado para leitura/escrita.

A Fig. 5.8 traz um exemplo da comunicação serial entre o robô e uma aplicação em um PC. Nela, o robô lê um valor e o armazena na variável *data*, depois escreve o valor “1”, que é colocado na variável *confirma*. As variáveis *data* e *confirm*a são do tipo *string*, no entanto a variável *canalserial*, a ser empregada para fazer o *link* entre o canal físico e os dados, é do tipo *iodev*.

Fig. 5.8: Exemplo de algoritmo para comunicação serial do robô IRB6640

```
VAR iodev canalserial;
VAR string data;
VAR string confirma;

Open "COM1",canalserial\Read;
data:=ReadStr(canalserial\Time:=60);
Close canalserial;
confirma := ValToStr(1);
Open "COM1",canalserial\Write;
Write canalserial,confirma;
Close canalserial;
```

Fonte: produção próprio autor

5.6 *SOFTWARE* DO SISTEMA DE VISÃO

O protótipo do *software* do sistema de visão foi desenvolvido utilizando a linguagem Python (BARRY, 2010; PYTHON, 2010) e as bibliotecas Numpy (NUMPY, 2010), IA636 (SILVA; LOTUFO, 2010), PIL (PIL, 2010) e Mmorph (LOTUFO; MACHADO, 2010). Desenvolveu-se o programa do robô em linguagem RAPID, própria para os robôs ABB.

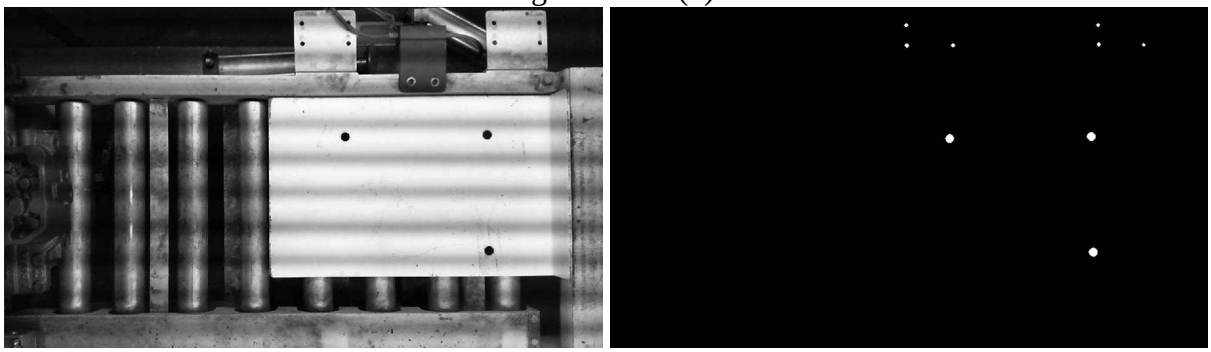
O sistema de visão desenvolvido utiliza a correlação de Pearson para identificar o modelo do cabeçote e a transformada de Hough como base para encontrar os centros dos furos que indicarão a posição e a orientação dos cabeçotes. Como a transformada de Hough pode ser aplicada somente em imagens binárias, depois de capturadas as imagens passam por várias operações que as preparam para aplicação dessa transformada. O *software* do sistema de visão (ver apêndice B) pode ser dividido em cinco partes:

- Calibração;
- Comunicação serial;
- Identificação do modelo de cabeçote;
- Segmentação dos furos;
- Cálculo da posição e orientação do cabeçote.

5.6.1 Calibração

Para detectar os furos da placa de calibração o *software* segue os passos 1 a 21 do algoritmo de segmentação de furos apresentado no fluxograma da Fig. 5.10. Na Fig. 5.9 há a imagem da placa de calibração antes e depois da segmentação dos furos.

Fig. 5.9: Placa de calibração (a) e placa de calibração binarizada, sem ruídos e com os furos segmentados (b)



(a) (b)

Fonte: produção próprio autor

Apenas os três furos da placa de calibração são armazenados no vetor de descritores, o pertencente à origem, o pertencente ao eixo *X* e o pertencente ao eixo *Y*. Após isso a inclinação do eixo *X* em relação à borda superior da imagem é encontrada. Essa inclinação representa a orientação do sistema de coordenadas da câmera e será utilizada no cálculo da orientação dos cabeçotes. É necessário descobrir o tamanho de um *pixel* no eixo *X* e no eixo *Y*. Para isso empregaram-se as equações 5.1 e 5.2. Os parâmetros de orientação do sistema de coordenadas do sistema de visão e os tamanhos de um *pixel* para o eixo *X* e para o eixo *Y* são passados ao algoritmo que calcula a posição e a orientação dos cabeçotes. O apêndice C apresenta o algoritmo de calibração do sistema de visão.

5.6.2 Comunicação Serial

O canal serial é verificado ciclicamente no algoritmo, como se vê no fluxograma da Fig. 5.1. O sistema de visão vai aguardar uma ordem vinda do robô para capturar uma imagem e iniciar seu processamento. O resultado é enviado ao robô pelo canal serial após ser solicitado pelo robô.

5.6.3 Identificação do Modelo do Cabeçote

Como abordado na seção 5.3 deste trabalho, criou-se um banco de imagens contendo três grupos de imagens. Cada grupo é subdividido em dois subgrupos. Um subgrupo representa as peças no sentido correto e o outro no sentido oposto sobre a linha de entrada da célula. O banco de imagens possui 18 imagens. Calculou-se o índice de similaridade utilizando a correlação de Pearson, pois ela apresentou os melhores resultados na comparação feita na seção 5.3. O maior índice encontrado na comparação entre uma imagem e os subgrupos indica a qual subgrupo a imagem comparada pertence.

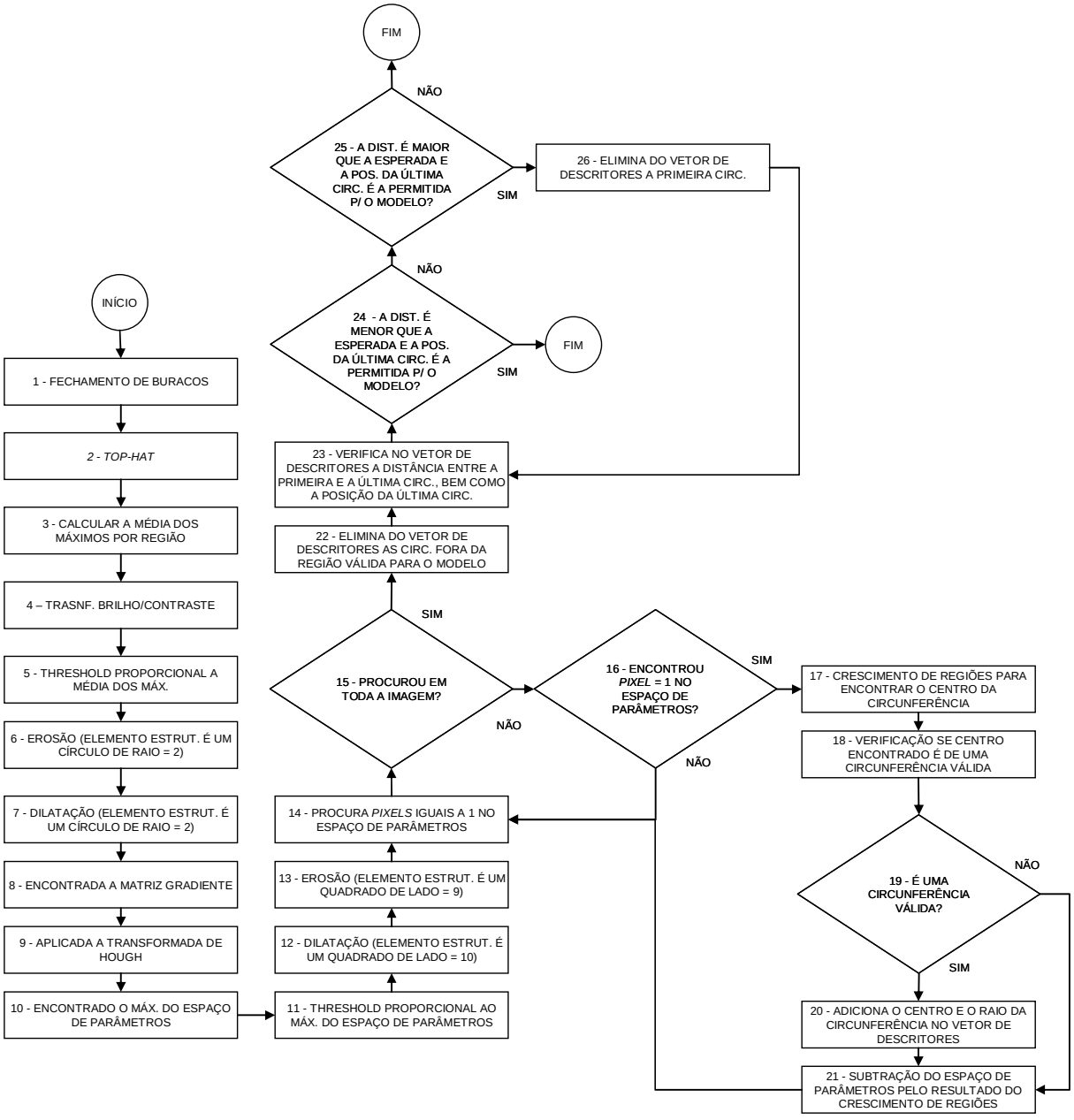
Com o modelo do cabeçote detectado, alguns parâmetros são carregados no *software* de detecção das circunferências, a listar:

- Região onde o último furo provavelmente vai ser encontrado;
- Distância máxima e mínima entre o primeiro e o último furo;
- Faixa da escala de cinza onde é feita a transformação brilho/contraste;
- Valor do *threshold* antes de aplicar a transformada de Hough;
- Valor do *threshold* do espaço de parâmetros;
- Valor do percentual de similaridade na comparação entre a circunferência encontrada e uma circunferência ideal.

5.6.4 Segmentação dos Furos

A segmentação dos furos do cabeçote segue os passos indicados no fluxograma da Fig. 5.10. A base desse processo é a transformada de Hough. Para tanto, a imagem deve ser preparada para sua aplicação. Por isso várias ferramentas de processamento de imagens são usadas antes de aplicar a transformada. A posição de cada circunferência encontrada é armazenada em um vetor de descritores. Visando garantir que apenas as circunferências válidas para cada modelo sejam guardadas, faz-se uma verificação nesse vetor. Assim, as circunferências indesejadas são eliminadas. No apêndice A há imagens das principais etapas do algoritmo de segmentação dos furos para os três modelos de cabeçotes disponibilizados pela empresa para os testes. Os passos executados pelo algoritmo são comentados no texto após o fluxograma da Fig. 5.10.

Fig. 5.10: Fluxograma da segmentação dos furos de um cabeçote



Fonte: produção próprio autor

Os passos do fluxograma são descritos na seqüência, para que haja melhor entendimento do funcionamento do algoritmo.

Passo 1: é feito o fechamento de buracos para clarear os furos do cabeçote.

Passo 2: uma subtração (*top-hat*) da imagem com os buracos fechados pela imagem inicial. Dessa forma, as regiões de baixo relevo da imagem de entrada passam a ser regiões de alto relevo após o *top-hat*. Essa subtração é condicional, ou seja, satura em 0.

Passo 3: para calcular o valor médio dos máximos por região, realizaram-se 20 divisões horizontais e 20 divisões verticais na imagem. Encontrou-se o valor máximo para cada

divisão da imagem e depois foi feita a média aritmética entre os todos máximos. O objetivo é absorver pequenas variações de luminosidade, pois, mesmo quando ela oscila, os roletes da linha de transporte refletem muita luz, em decorrência de sua geometria circular. Por conta disso, nessas regiões estão os maiores valores de intensidades da imagem. Esses valores não podem ser empregados como base para um limiar global que binarize a imagem sem perder os furos desejados.

Passo 4: feita uma transformação de brilho/contraste na imagem. A região de intensidades alterada é baseada no valor médio dos máximos encontrados no passo 3. Para cada família de cabeçote existe uma região distinta, ou seja,

família A = [1,0*max_medio*-10; 1,0*max_medio*+10];

família B = [0,6*max_medio*-10; 0,6*max_medio*+10];

família C = [1,0*max_medio*-10; 1,0*max_medio*+10].

O intuito é tornar maior a diferença entre as intensidades dos *pixels*, dos furos e da região da sua borda, preparando a imagem para a etapa seguinte. A Fig. 5.11 apresenta tal situação.

Passo 5: um *threshold* global com valor de corte proporcional ao máximo médio da imagem. O valor da proporção varia de acordo com o modelo do cabeçote analisado, ou seja,

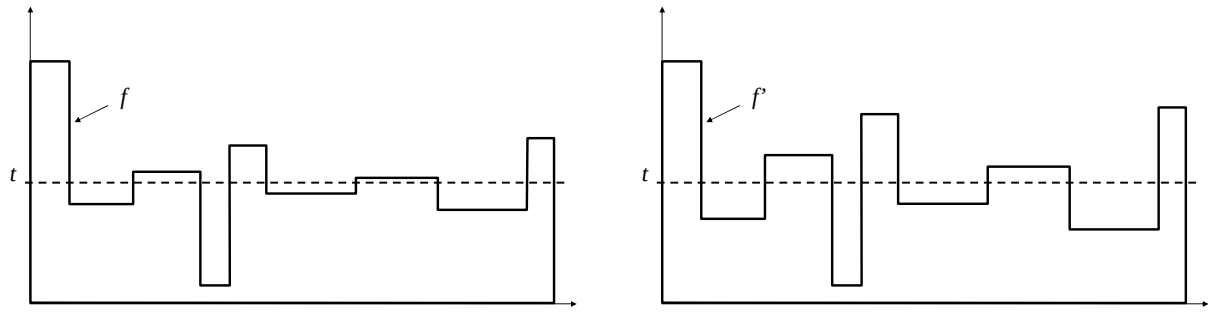
Família A = 2,0*max_medio*;

Família B = 2,0*max_medio*;

Família C = 2,2*max_medio*.

O pico médio torna o *threshold* adaptativo às possíveis variações de brilho nas imagens capturadas.

Fig. 5.11: (a) *f* é a imagem inicial, (b) *f'* é a imagem após a transformação de brilho/contraste e *t* é o valor do *threshold*



(a) (b)

Fonte: produção próprio autor

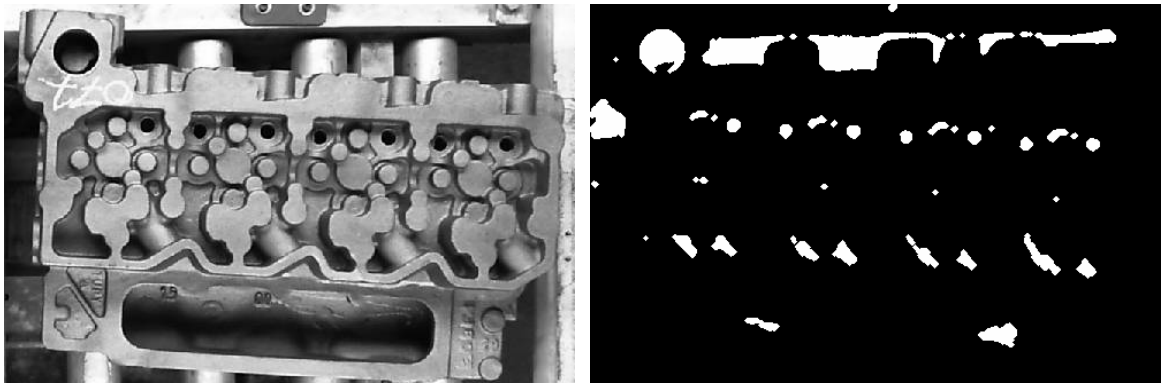
Passo 6: uma erosão com elemento estruturante circular e raio igual a 2 *pixels* é aplicada à imagem resultante dos processamentos anteriores. O objetivo dessa erosão é eliminar ruídos e

pequenas regiões que não interessam à segmentação dos furos.

Passo 7: uma dilatação com o mesmo elemento estruturante do passo 6 é efetuada para restaurar a imagem erodida.

A Fig. 5.12 apresenta a imagem original de um cabeçote e após o passo 7, com os buracos realçados e binarizados.

Fig. 5.12: Imagem inicial de um cabeçote (a) e com os buracos realçados e binarizados (b)



(a) (b)

Fonte: produção próprio autor

Passo 8: após a etapa de realce dos buracos da imagem, faz-se necessário encontrar a matriz gradiente, cujos ângulos serão usados na transformada de Hough. Para isso utilizou-se o filtro de Sobel.

Passo 9: nessa etapa aplica-se a transformada de Hough e preenche-se o espaço de parâmetros.

Passo 10: o valor máximo do espaço de parâmetros é encontrado.

Passo 11: um *threshold* global, com valor de corte proporcional ao valor encontrado no passo 10 (0,3**max_esp_par*), é aplicado no espaço de parâmetros resultante da transformada de Hough. Com isso os picos do espaço de parâmetros tornam-se pequenas regiões.

Passo 12: a saída do passo 11 é uma imagem binária com a região dos centros das circunferências muito fragmentada. Por isso realiza-se uma dilatação com elemento estruturante quadrado de lado igual a 10 *pixels*. Como resultado, as regiões de centro de circunferências passam a ser regiões maciças de *pixels*, sem descontinuidades.

Passo 13: uma erosão, com elemento estruturante quadrado de lado 9 *pixels*, é aplicada à imagem resultante do passo 12 para reduzir o tamanho dos objetos.

Passo 14: na imagem resultante do passo 13 efetua-se uma varredura em busca de *pixels* iguais a 1.

Passo 15: se toda a imagem já foi percorrida, vai para o passo 22. Caso contrário segue para o passo 16.

Passo 16: o *pixel* encontrado no passo 14 é utilizado como uma semente para o crescimento de regiões do passo 17.

Passo 17: realiza-se um crescimento de regiões a partir do *pixel* encontrado no passo anterior. O objetivo desse passo é encontrar o centro e a área da região analisada. Se a área for maior que 30 *pixels*, ela é descartada por ser muito grande e não representar um centro de circunferência.

Passo 18: com a posição encontrada no passo 17 é feita uma verificação para confirmar se naquela posição existe realmente uma circunferência válida. Assim como no algoritmo de detecção do modelo do cabeçote, a base dessa verificação é a correlação de Pearson. Ela compara uma circunferência ideal com a circunferência detectada pela transformada de Hough. Na imagem dos furos realçados e binarizados coloca-se uma circunferência ideal na posição encontrada no passo 17. Entre a região da circunferência da imagem binarizada e a circunferência ideal efetua-se uma comparação aplicando a correlação de Pearson. Para tornar a verificação mais robusta, são feitas três verificações utilizando três raios diferentes para a circunferência ideal, que são:

$$r_{circ\_ideal} = r_{hough} - dist \tag{5.5}$$

$$r_{circ\_ideal} = r_{hough} \tag{5.6}$$

$$r_{circ\_ideal} = r_{hough} + dist \tag{5.7}$$

Em que r_{circ_ideal} é o raio da circunferência ideal, r_{hough} o raio empregado pela transformada de Hough e *dist* a tolerância utilizada na transformada de Hough.

O maior valor encontrado nas três aplicações da correlação de Pearson é utilizado para verificar se a circunferência detectada é válida.

Passo 19: se o valor da comparação for maior que 0,85, o algoritmo segue para o passo 20, caso contrário, vai para o passo 21.

Passo 20: a coordenada da circunferência é armazenada no vetor de descritores.

Passo 21: subtração da imagem do passo 14 pela imagem resultante do crescimento de regiões e retorna para o passo 14.

Passo 22: com o vetor de descritores preenchido, faz-se uma verificação para determinar se as posições armazenadas estão dentro dos limites esperados para o modelo de cabeçote analisado. Se existirem circunferências fora desses limites, elas serão excluídas do vetor de descritores. Como existe uma variação muito pequena na posição de parada do cabeçote na

linha de entrada, esse recurso aumentou a robustez do algoritmo. As coordenadas do vetor de descritores que estiverem fora dos limites são removidas.

Passo 23: Outro parâmetro carregado, após a detecção do modelo do cabeçote (seção 4.3.2), é a distância entre a última e a primeira circunferência. Tal passo analisa se a distância entre a primeira e a última circunferência do vetor de descritores está compreendida entre os limites determinados para o modelo analisado, assim como se a posição do último furo também está dentro do limite relacionado ao modelo detectado.

Passo 24: se a distância entre o primeiro e o último furo for menor que a tolerável e a última circunferência estiver na região desejada, o algoritmo se encerra, caso contrário vai para o passo 25.

Passo 25: se a distância entre o primeiro e o último furo for maior que a tolerável e a última circunferência estiver na região desejada, o algoritmo vai para o passo 26, caso contrário o algoritmo se encerra.

Passo 26: a posição da primeira circunferência é removida do vetor de descritores. O algoritmo retorna para o passo 23.

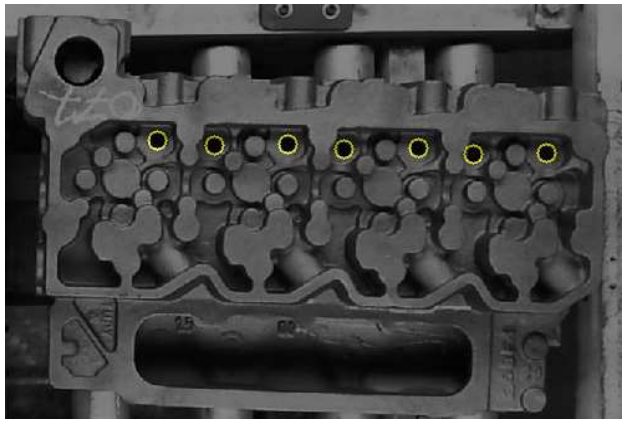
Seguindo os passos descritos, a primeira e a última circunferência de cada cabeçote é encontrada. Com elas, podem-se encontrar a orientação do cabeçote em relação ao sistema de coordenadas da câmera e o deslocamento em relação a uma posição de referência, como será descrito na seção 5.6.5.

A Fig. 5.13 mostra a imagem do cabeçote com os furos segmentados.

5.6.5 Cálculo da Posição e Orientação dos Cabeçotes

Para calcular a posição e a orientação dos cabeçotes, faz-se necessário identificar a primeira e a última circunferência do cabeçote. Fazendo isso, é possível calcular a sua rotação em relação ao sistema de coordenadas da câmera. Essa rotação é transformada em quatérnio para que o robô consiga interpretar tal informação. Após calcular a rotação, encontra-se o deslocamento do cabeçote em relação a uma posição de referência. Os cálculos para determinar a posição/orientação dos cabeçotes estão expostos na seção 4.3.2.

Fig. 5.13: Imagem do cabeçote com os furos segmentados pela transformada de Hough



Fonte: produção próprio autor

5.7 O PROGRAMA DO ROBÔ PARA EFETUAR A PEGA DOS CABEÇOTES

O programa do robô deve controlar todas as tarefas de manipulação entre as estações da Célula de Oleação e Paletização de Cabeçotes. Portanto esse programa seguirá instruções de lógica e de movimentação. Ele será dividido em sub-rotinas, algumas delas serão exclusivamente utilizadas na solução implementada por este trabalho. Há sub-rotinas que se comunicarão com o sistema de visão e aquelas que serão responsáveis por controlar os movimentos do robô, para que a pega dos cabeçotes seja feita com precisão e segurança na linha de entrada da célula.

5.7.1 A Comunicação com o Sistema de Visão

Para estabelecer a comunicação com o sistema de visão, desenvolveram-se três sub-rotinas. Uma sub-rotina envia dados ao sistema de visão, a outra recebe dados do sistema de visão e a última gerencia o recebimento das informações de deslocamento e rotação dos cabeçotes. A rotina que envia dados ao sistema de visão apenas abre o canal serial, envia dados e depois fecha o canal. A rotina que recebe dados abre o canal serial, aguarda o recebimento de dados enviados pelo sistema de visão e fecha o canal serial. Se uma falha de *Timeout* acontecer na leitura dos dados, a ocorrência é tratada por uma rotina de erro criada para tal fim. A rotina de erro tenta ler os dados no canal serial outras três vezes. Após a terceira tentativa o canal serial é fechado e uma mensagem é enviada ao TP (*teach pendant*) do robô informando que o sistema de visão não responde. A rotina que gerencia o recebimento da informação de posição e orientação dos cabeçotes inicia enviando uma mensagem ao sistema de visão dizendo que está *online*, utilizando a rotina que envia dados. Se o sistema de visão estiver *online*, uma mensagem informando que o robô está esperando os dados vindos

do sistema de visão é enviada. Após isso, o programa recebe as informações de posição e orientação do cabeçote. Se o sistema de visão não estiver *online*, o robô gera uma mensagem no TP informando que o sistema de visão não responde e a rotina é encerrada.

5.7.2 A Pega dos Cabeçotes

Cada sub-rotina de pega do cabeçote na linha de entrada inicia solicitando ao sistema de visão as informações de posição e orientação do cabeçote que será pego pelo robô. A próxima instrução é um deslocamento de sistema de coordenadas (apresentado na seção 3.6.3). O deslocamento é feito utilizando os valores recebidos do sistema de visão. Assim, a posição de pega será movimentada junto com o sistema de coordenadas do robô. Como os sistemas de coordenadas do robô e da câmera são coincidentes, o deslocamento encontrado pelo sistema de visão corresponderá ao deslocamento necessário para o robô reposicionar o sistema de coordenadas e efetuar a pega dos cabeçotes. Com o sistema de coordenadas deslocado são realizados os movimentos para efetuar a pega do cabeçote na linha de entrada. Após o robô terminar a pega na linha de entrada, o deslocamento do sistema de coordenadas é desativado e o robô continua a sequência normal do programa da célula.

6 TESTES E ANÁLISE DOS RESULTADOS

A proposta deste capítulo é expor os resultados dos experimentos realizados durante a fase de testes do projeto. O protótipo criado e os testes de *software* são descritos. Os testes com a iluminação empregada, a calibração e os cálculos da posição/orientação dos cabeçotes são apresentados. Os testes da comunicação serial e do programa do robô são mostrados no fim do capítulo.

6.1 VISÃO GERAL DOS TESTES

Todos os testes foram realizados utilizando o protótipo desenvolvido. Era desejo do autor realizar testes finais na célula projetada, em condições de operação industrial, porém ocorreram problemas, entre os quais podem ser citados os seguintes:

- Atrasos no desenvolvimento do projeto da célula robotizada;
- Atraso na concepção da garra do robô;
- O robô foi emprestado para uma das filiais da empresa por duas vezes, gerando atrasos de aproximadamente seis meses;
- O controlador do robô precisou de manutenção e ficou indisponível por aproximadamente quatro meses;
- Apenas alguns modelos de peças foram disponibilizados para os testes;
- Indisponibilidade das linhas para testes.

Após a homologação do protótipo, o *software* será implementado em C++ e executado em um computador industrial Advantech, modelo TPC-650H-N2AE, processador Intel 1,60GHz, 1Gb de RAM e sistema operacional Windows XP, dedicado ao sistema de visão. Os demais recursos de *hardware* serão os mesmos do protótipo. Os dados obtidos nos testes sugerem que o resultado será alcançado com sucesso.

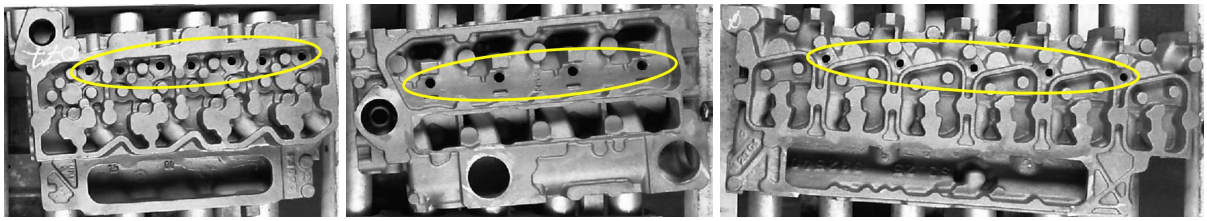
O *software* foi testado em partes distintas, que são:

- Detecção do modelo da peça;
- Segmentação dos furos;
- Cálculo da posição/orientação das peças;
- Calibração;
- Comunicação serial entre o *software* e o robô.

O *hardware* do protótipo foi desenvolvido utilizando suportes reguláveis para a câmera e iluminação. Por meio deles foi possível encontrar a melhor posição para gerar as sombras necessárias a segmentar os furos usados na determinação da posição e orientação dos cabeçotes (Fig. 6.1). Com o protótipo foi possível encontrar os parâmetros necessários para calcular a posição e a orientação de cada cabeçote. A Fig. 6.2 apresenta o *layout* do protótipo montado.

Com o protótipo fizeram-se testes de *software* e de *hardware*. Os testes de *software* foram mais numerosos em virtude da dificuldade encontrada na segmentação dos furos desejados. Durante a fase de testes o *software* do protótipo sofreu vários ajustes, aumentando a eficiência na detecção dos furos e no cálculo de posição/orientação. O maior objetivo foi torná-lo robusto suficiente para as irregularidades geométricas e as variações de brilho não influenciarem negativamente no resultado da detecção dos furos de um cabeçote para o outro. As rotações máximas dos cabeçotes estão compreendidas entre +15º e -15º. Com relação ao *hardware* não foi diferente, a posição da iluminação também sofreu mudanças durante os testes. A situação real de trabalho foi simulada. As peças foram posicionadas aleatoriamente na região de pega do robô, e com as informações vindas do sistema de visão ele fazia a sua pega e as manipulava.

Fig. 6.1: Furos utilizados para detecção da posição/orientação dos cabeçotes



Fonte: produção próprio autor

6.2 O PROTÓTIPO

O protótipo foi montado conforme a Fig. 6.2. Com ele foi possível extrair as melhores regulagens de posição da iluminação e da câmera, bem como os parâmetros necessários ao *software* para encontrar a posição e a orientação dos cabeçotes.

Fig. 6.2: Visão geral do protótipo

Fonte: produção próprio autor

6.2.1 A Determinação do Modelo do Cabeçote

Construiu-se o protótipo com o objetivo de testar e comprovar a eficiência do sistema desenvolvido. Durante os testes de determinação do modelo de cabeçote, analisaram-se 129 imagens diferentes de cabeçotes, como apresentado no item 5.3, e todos foram identificados corretamente utilizando o método correlação de Pearson. Criou-se a base de dados com apenas três dos seis modelos de cabeçotes existentes; os demais modelos não foram disponibilizados pela empresa durante o período em que se realizaram os testes.

6.2.2 A Detecção dos Furos dos Cabeçotes

Foi importante acompanhar o comportamento da transformada de Hough de acordo com a variação da iluminação e a coloração das peças analisadas. Com o passar dos dias, as peças podem ficar mais escuras, por causa da oxidação, e dificultar a detecção dos furos dos cabeçotes.

É possível dividir essa fase de testes em duas etapas distintas:

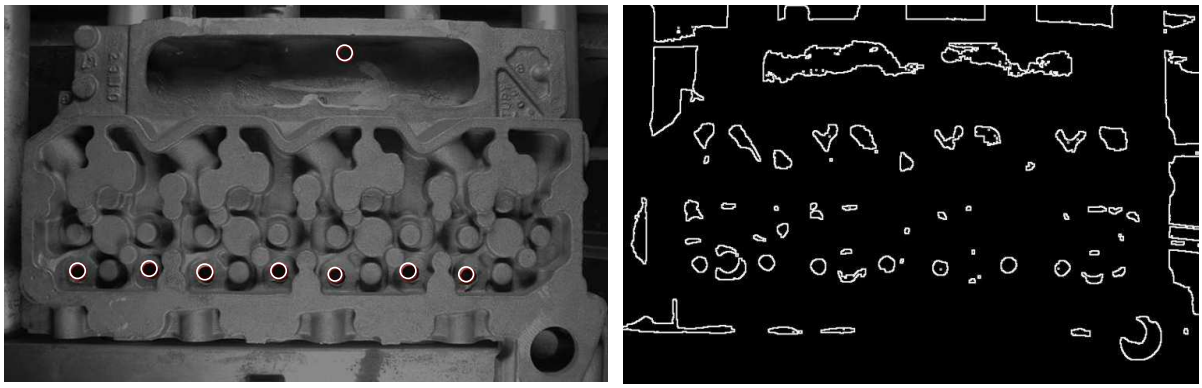
- Detecção e segmentação dos furos;
- Extração de dados e criação do vetor de descritores.

O resultado da utilização da transformada de Hough mostrou-se muito dependente da

iluminação e das primeiras etapas do processamento da imagem. Ela apresentou algumas falhas durante os testes porque os cabeçotes projetavam sombras com geometrias semelhantes a uma circunferência. Testaram-se várias posições diferentes para a iluminação com o intuito de minimizar a geração das sombras indesejadas. Nos testes iniciais, de 115 imagens analisadas, apenas sete apresentaram problemas na detecção das circunferências, ou seja, 6,08%.

Em algumas imagens capturadas ocorreram sombras indesejadas. Após as etapas do processamento de imagens, essas sombras passam a ter as mesmas características de uma circunferência. Recorreu-se a técnicas de morfologia matemática, no pré-processamento das imagens, para minimizar os efeitos dessas sombras. Além da morfologia, o *thresholding* também foi empregado com tal fim. Como o sistema proposto possui características de luminosidade constantes, o *threshold* pode ser útil no processamento de imagens. Foi necessário tratar o vetor de descritores (apresentado no item 5.6.4, passos 22 a 26) para minimizar a detecção de circunferências indesejadas. A Fig. 6.3 traz um exemplo de detecção de falsas circunferências.

Fig. 6.3: Detecção de falsas circunferências (a) e realce de borda da imagem (b)

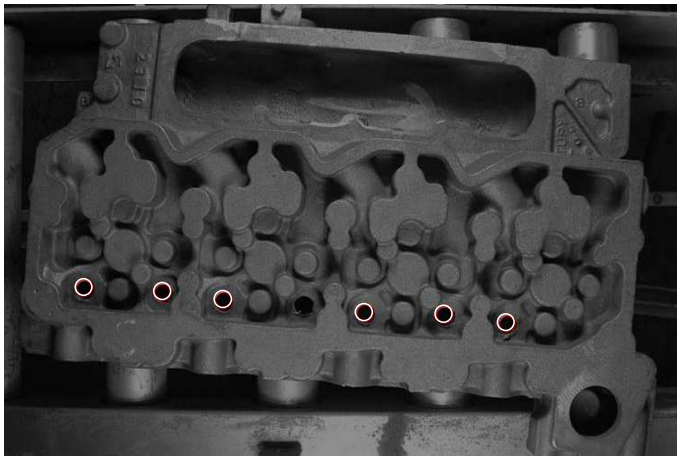


(a) (b)

Fonte: produção próprio autor

O teste do protótipo deixou claro que rebarbas nos furos podem dificultar o reconhecimento de peças. Como as peças são brutas, a ocorrência de rebarbas é bem comum. A Fig. 6.4 evidencia que as rebarbas podem interferir na detecção dos furos, pois elas alteram o seu formato na imagem. Esse problema foi minimizado com a verificação que o algoritmo faz para cada circunferência detectada (ver item 5.6.4). Ele confirma se cada círculo encontrado pela transformada de Hough representa uma circunferência válida na imagem analisada.

Fig. 6.4: Não reconhecimento de um furo em virtude de rebarba existente



Fonte: produção próprio autor

No início dos testes, todos os furos eram empregados para determinar a posição e a orientação dos cabeçotes. Como houve dificuldades em encontrar todos os furos da superfície, alterou-se o algoritmo para usar apenas os dois furos extremos. A utilização do primeiro e do último furo do cabeçote tornou o algoritmo mais robusto. Esse método melhorou a eficiência do sistema.

Com os furos detectados e suas coordenadas de centro conhecidas, encontra-se a orientação da peça em relação ao sistema de coordenadas da câmera e, por consequência, o sistema de coordenadas do robô.

O cálculo dos parâmetros de orientação dos cabeçotes por meio das coordenadas dos centros dos furos, que é uma etapa do algoritmo posterior à construção do vetor de descritores, mostrou ser uma implementação robusta, pois em todos os testes houve sucesso. No entanto esses cálculos são muito dependentes do vetor de descritores, por isso a necessidade de um reconhecimento de padrões eficiente.

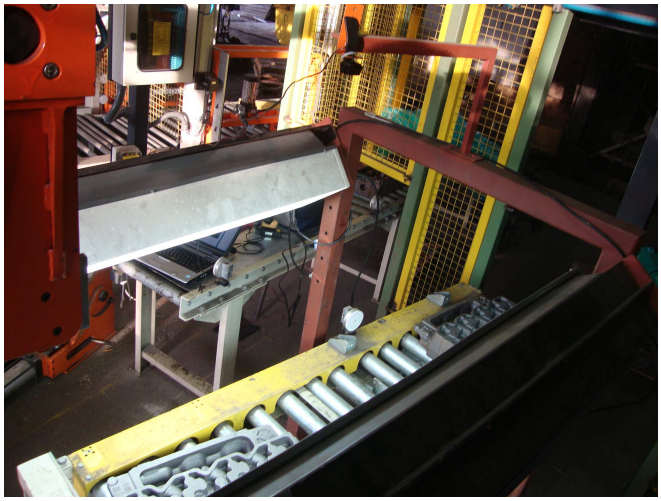
6.2.3 A Posição da Iluminação

Os testes apontaram a sombra como uma forte aliada na segmentação dos furos, bem como a necessidade do controle de posição da iluminação. É muito importante garantir que a luminosidade e o ângulo de incidência da luz na peça sejam mantidos constantes. Variações bruscas de contraste e de sombras prejudicam o resultado do processamento da imagem.

Foi preciso uma grande capacidade luminosa para realçar de forma eficiente os furos que serão utilizados para encontrar a posição e a orientação do cabeçote. O protótipo mostrou que a melhor posição para a iluminação é lateral, voltada para a face superior das peças, fazendo com que essa superfície reflita uma grande quantidade de luz e os furos formem

sombra por dentro, gerando um bom contraste com a sua borda. A Fig. 6.5 mostra o suporte para os módulos de iluminação usados no protótipo.

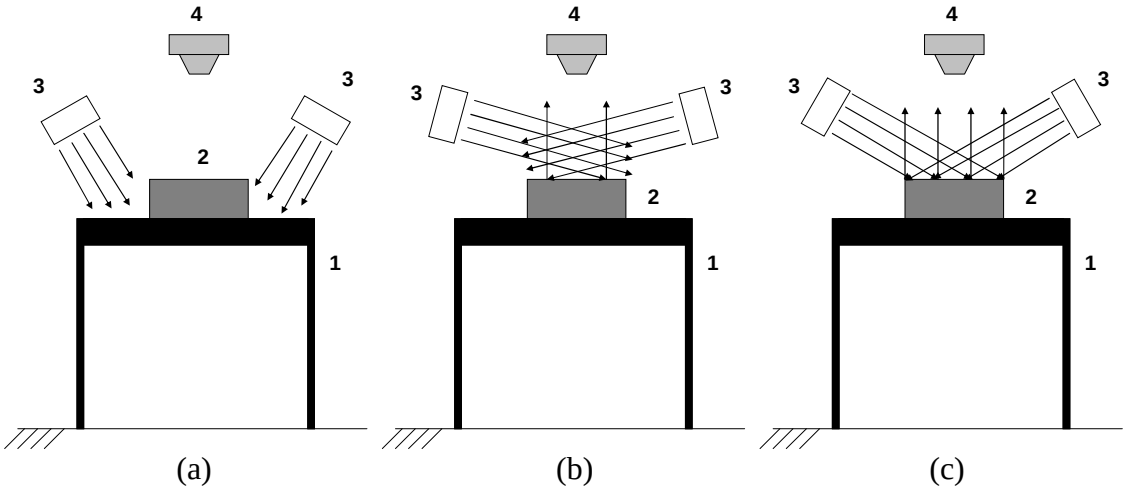
Fig. 6.5: Possíveis regulagens para a iluminação e para a câmera



Fonte: produção próprio autor

Testaram-se muitas posições diferentes para os módulos de iluminação, sempre buscando a melhor relação entre a luz refletida na superfície e a sombra formada nos furos dos cabeçotes. Dependendo da posição em que os módulos se encontravam, a superfície dos cabeçotes ficava muito iluminada, até mesmo com iluminação dentro dos furos dificultando sua segmentação. Outra posição tornava a superfície muito escura, dificultando a distinção dos furos. A melhor solução foi utilizar uma posição intermediária entre as duas citadas. A Fig. 6.6 apresenta um esquema representando as posições testadas.

Fig. 6.6: Esquema ilustrando as posições dos módulos de iluminação testadas. (a) pouca luz refletida, (b) reflexão ideal com sombra nos furos, (c) muita luz refletida. (1) linha de roletes, (2) cabeçote, (3) módulos de iluminação, (4) câmera

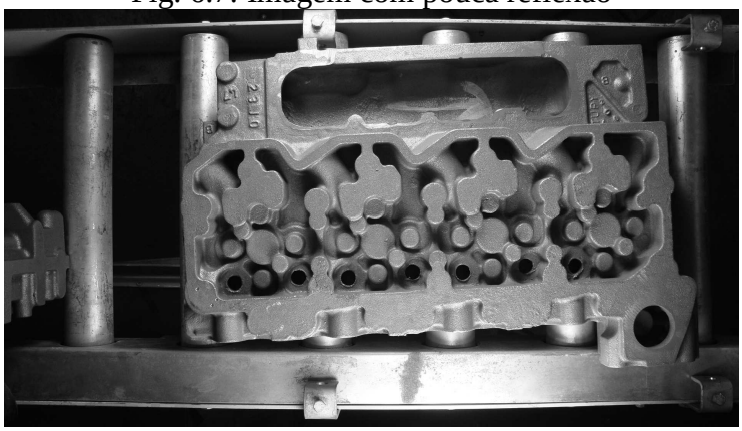


(a) (b) (c)

Fonte: produção próprio autor

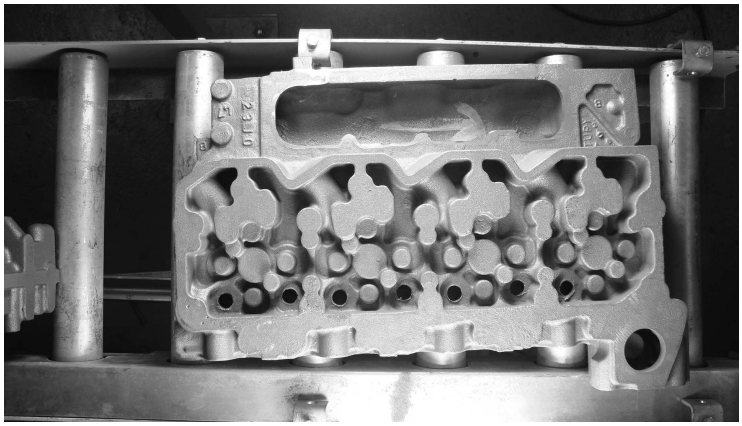
As Figs. 6.7, 6.8 e 6.9 trazem exemplos de imagens capturadas utilizando os módulos de iluminação posicionados conforme a Fig. 6.6 (a), (b) e (c), respectivamente.

Fig. 6.7: Imagem com pouca reflexão



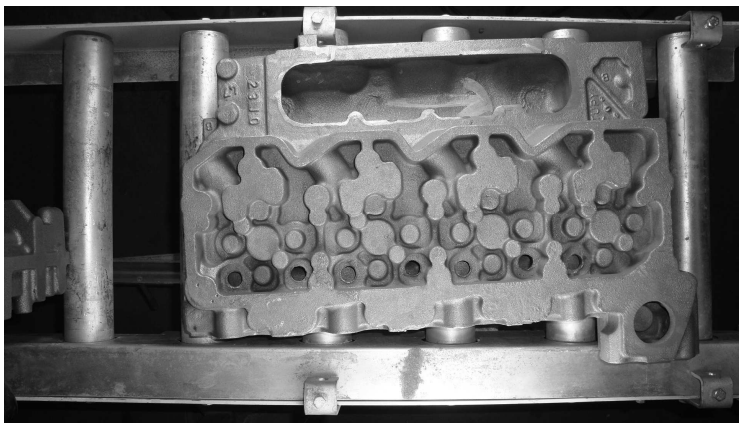
Fonte: produção próprio autor

Fig. 6.8: Imagem com bom contraste entre os furos e a superfície do cabeçote



Fonte: produção próprio autor

Fig. 6.9: Imagem com iluminação dentro dos furos



Fonte: produção próprio autor

6.2.4 A Calibração do Sistema de Visão e a Posição de pega dos Cabeçotes

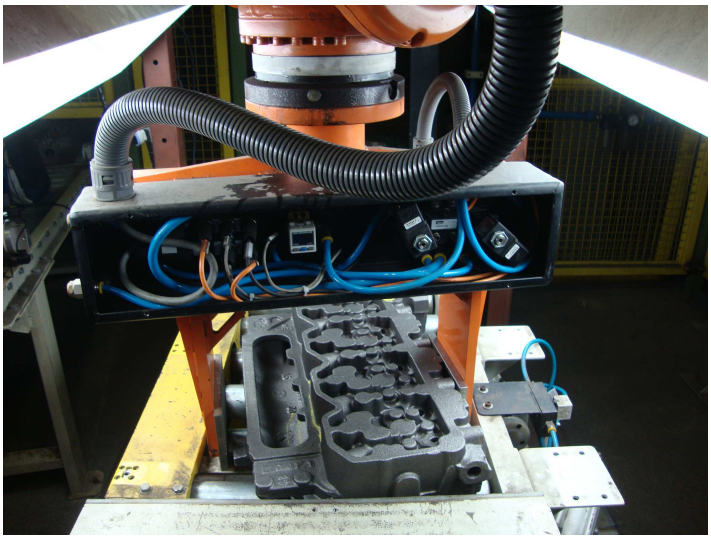
Para que os sistemas de coordenadas, da câmera e do robô, ficassem coincidentes, realizou-se a calibração de ambos com o mesmo padrão. A Fig. 5.6(a) apresenta o padrão de calibração utilizado. Não basta apenas saber onde está a origem do sistema de coordenadas, é necessário determinar uma orientação para ele, bem como mensurar o tamanho de um *pixel*, já que o robô não reconhece o *pixel* como unidade de medida. No protótipo empregou-se a conversão de *pixels* para milímetros antes de enviar as informações para o robô. Dessa forma, primeiro encontra-se o deslocamento do cabeçote em *pixels* e depois é feita a conversão para milímetros. O valor de um *pixel*, em milímetros, é encontrado no procedimento de calibração do sistema (seção 5.4).

Os testes envolvendo pega de peças pelo robô evidenciaram que o sistema de visão não necessita enviar a posição do centro de massa dos cabeçotes. Essa informação já está no programa do robô. É fácil perceber, pois o sistema de visão não cria novas posições de pega de peça, mas sim encontra o deslocamento e a rotação do cabeçote que deve ser manipulado pelo robô. Sendo assim, como a posição de pega já está memorizada no robô, resta-lhe deslocar e rotacionar o sistema de coordenadas.

A movimentação do robô até a linha de roletes é a mesma para todos os modelos de cabeçotes. Desse ponto em diante são utilizadas as informações recebidas do sistema de visão, então o robô se desloca e se reorienta para efetuar a pega da peça. Ao sair da linha de transporte, o robô retoma a trajetória padrão de saída da linha.

A pega das peças com o auxílio do sistema de visão ocorreu de forma precisa. A Fig. 6.10 apresenta a garra posicionada e orientada de acordo com o cabeçote na linha.

Fig. 6.10: Robô posicionado e orientado para pegar o cabeçote na linha

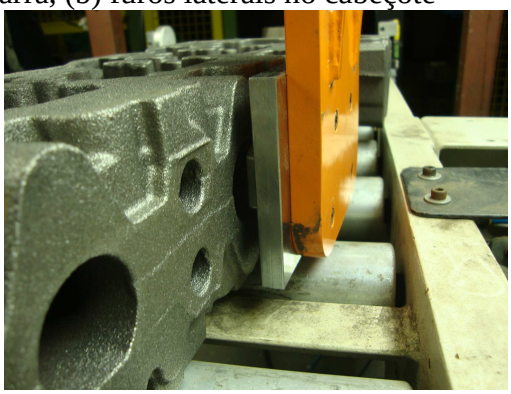


Fonte: produção próprio autor

Os cabeçotes possuem furos laterais (Fig 6.11(b)) utilizados para a pega pela garra. Por sua vez, a garra possui dois pinos que entram nos furos laterais; eles podem ser vistos na Fig. 6.11(a). Dessa forma, a fixação do cabeçote na garra será reforçada. A garra possui uma parte fixa e outra móvel. Assim, será necessário posicionar apenas o lado fixo dentro dos furos laterais. A parte móvel, acionada pneumaticamente, prende a peça por fricção.

Fig. 6.11: (a) Pinos para a fixação da peça na garra, (b) furos laterais no cabeçote

(a)



(b)

Fonte: produção próprio autor

6.2.5 A Comunicação Serial

A comunicação entre o robô e o *software* do sistema de visão deverá ser a serial, uma vez que o robô empregado no projeto não possui suporte para outras redes de comunicação. Os testes mostraram que é bem simples estabelecer a comunicação serial entre o robô e um computador. Para isso o algoritmo de comunicação foi desenvolvido utilizando instruções de escrita e leitura do canal serial, tanto do lado do computador quanto do lado do robô. Vale

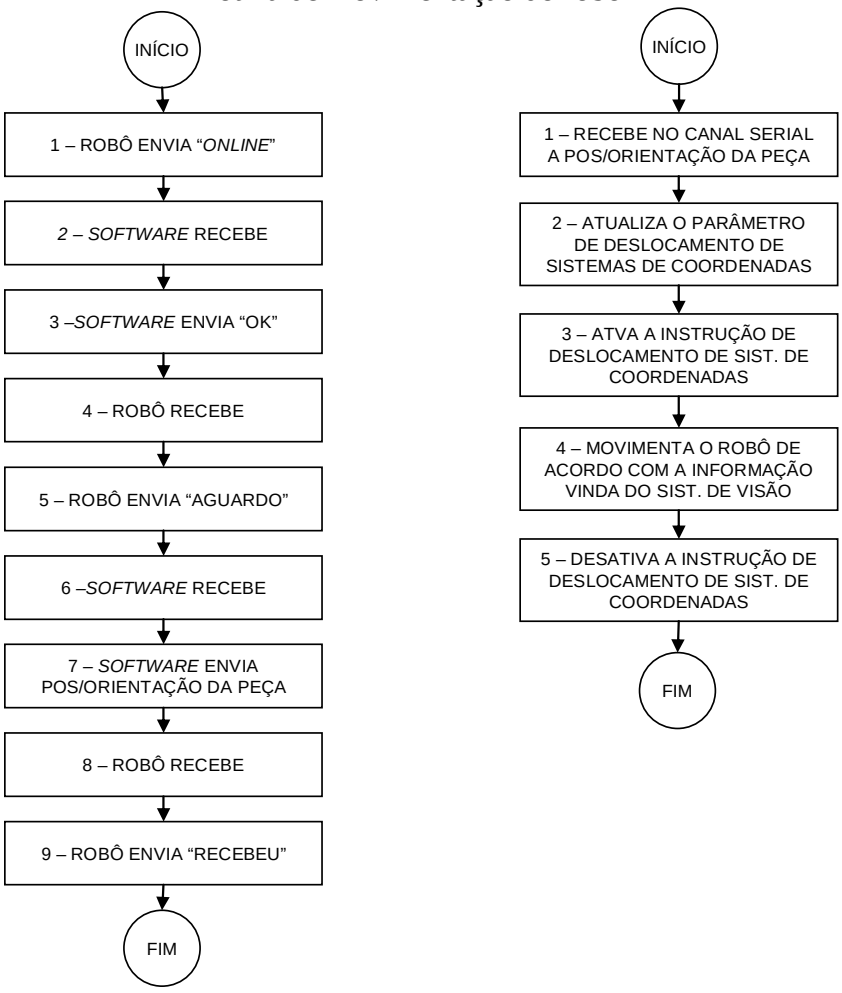
lembrar que o canal serial não permanece aberto enquanto ambos, *software* do sistema de visão e robô, estiverem executando funções que não sejam as de comunicação. No método desenvolvido quem inicia a comunicação é o robô, como se vê no fluxograma da Fig. 6.12 (a).

6.2.6 O Programa do Robô

Os módulos de movimentação e de comunicação serial do programa do robô foram testados. O fluxograma da Fig. 6.12 (b) apresenta a sequência da rotina de movimentação do robô nos testes.

A primeira operação é buscar na porta serial a informação de posição/orientação do cabeçote gerada pelo sistema de visão. Com essa informação é atualizado o parâmetro de deslocamento de sistemas de coordenadas do robô. Na sequência, o deslocamento de sistemas de coordenadas é ativado. Como apresentado no item 3.6.3, com tal instrução ativada, todas as posições gravadas no robô são deslocadas na mesma proporção. Assim, a posição de referência gravada no robô é deslocada de acordo com a posição/orientação vindas do sistema de visão. Com isso o robô pode efetuar a pega do cabeçote, deslocado e rotacionado, na linha de entrada da célula.

Fig. 6.12: (a) Fluxograma da comunicação serial entre computador e robô, (b) Fluxograma da rotina de movimentação do robô



Fonte: produção próprio autor

6.3 CUSTO DO PROTÓTIPO

Os custos dos materiais utilizados no protótipo estão descritos a seguir:

- Câmera = R$160,00;
- Módulos de iluminação fluorescente = R$355,00;
- Suporte câmera e iluminação = R$155,00;

O custo do computador industrial que será utilizado na implementação final é de R$4.850,00. No total a aplicação final custará R$5.520,00.

7 CONSIDERAÇÕES FINAIS

Neste trabalho foi apresentada uma solução para guiar um robô industrial em uma célula de manufatura robotizada abastecida manualmente. A solução proposta mostra um sistema de visão computacional capaz de identificar os modelos de peças, bem como sua posição/orientação. Uma das maiores dificuldades em manipulações robotizadas é a pega das peças. Em células abastecidas randomicamente a dificuldade aumenta. Sistemas de visão computacional minimizam o problema da pega de peças (CHENG; DENMAN, 2005) e flexibilizam a aplicação de robôs industriais.

Existem soluções prontas no mercado de sistemas de visão para essa classe de problema. Citam-se três fabricantes de sistemas de visão comerciais: Cognex, Banner e Fanuc. Trata-se de sistemas comerciais flexíveis e generalistas, que tentam resolver o maior número de problemas possível. Isso torna tais soluções comercialmente interessantes, porém, no tocante a problemas mais complexos, como guiar robôs, seu custo se eleva, em virtude de uma grande capacidade de *hardware* necessário para tal implementação, como módulos de iluminação específicos, lentes especiais para a câmera, bem como uma câmera com resolução maior que 800X600 *pixels*, que possui um custo mais elevado. Um dos objetivos da solução proposta foi utilizar um *hardware* de baixo custo, e por isso foram empregados uma *webcam*, um computador industrial e lâmpadas fluorescentes como iluminação. Soluções comerciais podem custar cinco vezes mais que o valor gasto na solução desenvolvida.

A ferramenta de medida de similaridades utilizada na implementação atende aos requisitos do projeto. Fez-se uma busca por ferramentas computacionalmente eficientes, com baixo tempo de processamento e 100% de acerto. Em princípio buscaram-se ferramentas invariantes à rotação e translação, todavia os resultados mostraram que, especificamente neste projeto, uma ferramenta de análise global (correlação de Pearson) ofereceu melhores resultados ante as demais analisadas (SSIM e CW-SSIM).

Como este projeto propõe uma solução para um problema real em uma empresa privada, ele sofreu alterações de cronograma por parte da empresa. Este projeto é, na verdade, uma pequena parte de um projeto maior da Tupy SA., que é desenvolver uma célula robotizada para oleação e paletização de cabeçotes fundidos. Como houve muitos atrasos e imprevistos no desenvolvimento da célula robotizada, a solução proposta aqui também sofreu atrasos. Isso ficou evidente uma vez que todos os testes foram feitos utilizando o protótipo do sistema de visão em conjunto com o robô da célula. Esses testes foram realizados utilizando-

se apenas três modelos diferentes de cabeçotes, pois os demais não foram disponibilizados pela empresa. Salienta-se que o protótipo foi desenvolvido na empresa e por isso os horários disponíveis para testes tiveram de obedecer aos limites impostos pela companhia. Outra dificuldade encontrada foi com relação ao tamanho e peso das peças. Elas são de difícil manuseio, por serem pesadas e possuírem arestas cortantes. Os cabeçotes fundidos possuem facilidade em oxidar sua superfície, por se tratar de peças não acabadas e sem revestimentos superficiais antioxidantes. A coloração dos cabeçotes dificultou o processamento de imagens desenvolvido. Como ela se confunde com a cor dos roletes da linha onde os cabeçotes são transportados, ficou difícil segmentá-los do fundo da imagem. Esse foi um dos motivos de utilizar um método de medição de similaridades que analisa toda a imagem para reconhecimento do modelo dos cabeçotes, em vez de extrair apenas o cabeçote da imagem para depois compará-lo a um banco de imagens.

Durante a fase de testes, o algoritmo sofreu grandes mudanças. A mais significativa foi usar apenas o primeiro e o último furo do cabeçote para determinar sua posição/orientação. A ideia inicial era encontrar todos os furos. Com base na quantidade e nos deslocamentos entre si, fazia-se o reconhecimento do modelo do cabeçote analisado, bem como era encontrada sua posição/orientação. Como era difícil encontrar todos os furos, pois os furos centrais eram mais difíceis de serem reconhecidos por causa das rebarbas existentes, decidiu-se empregar apenas os furos extremos. Outro problema encontrado nos testes foi o reconhecimento de falsas circunferências na região central do cabeçote. Esse também foi um dos motivos que levou à utilização do primeiro e do último furo na análise dos cabeçotes.

O método de calibração proposto mostrou-se eficiente para o projeto. Como as peças a serem pegas pelo robô são brutas, possuem uma boa tolerância na região de pega, ±0,5mm. Os testes evidenciaram que a calibração usada atende a esse requisito, uma vez que a pega dos cabeçotes ocorreu sem problemas.

A tarefa mais trabalhosa, durante a fase de testes, foi encontrar a melhor posição para a iluminação do sistema, uma vez que os melhores resultados da transformada de Hough ocorreram quando se conseguiu um grande contraste entre os furos e suas bordas. A transformada de Hough mostrou-se dependente da do processamento de imagem para a sua aplicação. É importante que as circunferências a serem detectadas estejam segmentadas (DUARTE, 2003). A iluminação influencia diretamente nesse resultado (ARAÚJO, 2010). A ideia foi gerar sombras no interior das circunferências para, dessa forma, segmentá-las. Porém a iluminação é fixa na linha de entrada da célula, por isso, foi necessário encontrar uma posição que obtivesse bons resultados para todos os modelos de cabeçotes analisados. Essa

tarefa não foi fácil; muitos testes aconteceram até a melhor posição ser encontrada.

A transformada de Hough para detecção de furos foi selecionada graças a sua robustez. Ela é capaz de identificar circunferências a partir de círculos danificados ou incompletos (TRESPADERNE; LÓPEZ, 2009). Tal característica, ao mesmo tempo em que é positiva, pode ser negativa, pois houve reconhecimento de falsas circunferências. Para resolver o problema, desenvolveu-se um método que verifica se o círculo detectado pela transformada de Hough realmente representa uma circunferência válida. Essa implementação aumentou a robustez do sistema.

Este trabalho resultou na publicação do artigo "Engine head blocks handling robot guided by vision system" (SEMIM *et al*., 2012), premiado com outros dois trabalhos como melhores artigos de aplicação industrial do INCOM 2012[8] – 14.ª edição do IFAC *Symposium on Information Control Problems in Manufacturing* – realizado na Romênia.

7.1 TRABALHOS FUTUROS

Possibilidades de desenvolvimento para trabalhos futuros:

- Desenvolvimento de um *software* que faça a união dos módulos desenvolvidos;
- Desenvolvimento de uma interface gráfica para o usuário;
- Utilização de outros métodos de processamento de imagem, como correlação de fase, para a detecção das circunferências e cálculo de posição/orientação dos cabeçotes;
- Desenvolvimento de um sistema de visão para identificação de peças sobre um transportador em movimento;
- Desenvolvimento de um sistema de visão para guiar um robô na montagem de componentes em peças industriais. O desafio será encontrar a posição de montagem de cada componente a ser anexado à peça em processo.

[8] INCOM 2012 – mais informações sobre os artigos premiados em <http//:www.incom2012.ro/awards.htm>.

8 REFERÊNCIAS

ABB. **IRB 6640 / IRB 6640-ID Industrial Robot**. ROB0001EN_E, 2010.

ABB ROBOTIC DIVISION. **Application manual** – robot communication and I/O control. Controller *Software* IRC5, RobotWare 5.0, 2008a.

_____. **RAPID reference manual** – RAPID overview. Rev.1, 3HAC 16580-1, Controller Software IRC5, Revision E, RobotWare 5.0, 2007a.

_____. **Technical reference manual** – RAPID instructions, functions and data types. Controller Software IRC5, RobotWare 5.0, 2007b.

_____. **Technical reference manual** – RAPID overview. Controller Software IRC5, RobotWare 5.0, 2008b.

_____. **Technical reference manual** – system parameters. Controller Software IRC5, RobotWare 5.0, 2008c.

ARAÚJO, A. M. de O. **Sistema robotizado para execução automática do processo de halogenação para a indústria do calçado**. 2010. Dissertação (Mestrado)–Universidade do Minho, Escola de Engenharia, Minho, 2010.

BARRIENTOS, A. **Fundamentos de robótica**. 2. ed. McGraw Hill, 2007.

BARRY, P. **Head first Python**. O'Reilly Media, 2010. 496 p.

BIASI, S. C. de; GATTASS, M. **Utilização de quatérnios para representação de rotações em 3D**. Rio de Janeiro: PUC, 2002.

BOROVICKA, J. **Circle detection using hough transforms** – notas de aula da Universidade de Bristol. 2003. Disponível em: <http://linux.fjfi.cvut.cz/~pinus/study.html>. Acesso em: nov. 2010.

BORTOLINI, D. **Utilização da análise de componentes principais para medida de similaridade entre imagens**. 2010. Monografia (Conclusão do Curso de Ciências da Computação)–Universidade do Estado de Santa Catarina, 2010.

CHENG, F. S.; DENMAN, A. A study of using 2D vision system for enhanced industrial robot intelligence". *In*: IEEE INTERNATIONAL CONFERENCE ON MECHATRONICS & AUTOMATION, 2005, Cataratas do Niágara. **Anais**…. p. 1.185-1.189.

CHEN, H. *et al*. Flexible assembly automation using industrial robots. *In*: IEEE INTERNACIONAL CONFERENCE ON TECHNOLOGIES FOR PRACTICAL ROBOT APPLICATIONS – TePRA, 2008, Woburn. **Anais**… p. 46-51.

DAVIES, E. R. **Machine vision**. 3. ed. Morgan Kaufmann, 2005. 934 p.

DOUGHERTY, E. R.; LOTUFO, R. A. **Hands-on morphological image processing**. Washington: SPIE – The International Society for Optical Engineering, 2003. 272 p.

DUARTE, G. D. Uso da transformada de Hough na detecção de círculos em imagens digitais. **Thema Revista Científica do Centro Federal de Educação Tecnológica**, Pelotas, v. 4, p. 51-58, 2003.

FACON, J. **Processamento e análise de imagens**. Curitiba: Pontifícia Universidade Católica do Paraná, 2002. 116 p.

FAN, G.; WANG, Z.; WANG, J. CW-SSIM kernel based random foret for image classification. *In*: VISUAL COMMUNICATIONS AND IMAGE PROCESSING CONFERENCE, 2010, Huangshan, Anhui, 2010. **Anais**…

FONSECA, R. N. da. **Algoritmos para avaliação a qualidade de vídeo em sistemas de televisão digital**. 96 f. 2008. Dissertação (Mestrado em Engenharia Elétrica)–Escola Politécnica, Universidade de São Paulo, São Paulo, 2008.

GAO, J. *et al*. Damaged mechanism research of RS232 interface under electromagnetic pulse. *In*: IEEE INTERNATIONAL CONFERENCE ON COMPUTER SCIENCE AND SOFTWARE ENGINEERING, 2008. **Anais**… 1.119-1.122.

GAO, Y.; REHMAN, A.; WANG, Z. CW-SSIM based image classification. *In*: IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING, 18., 2001, Thessaloniki. **Anais**… p. 1.249-1.252.

GARRIDO, L. O. **Hierarchical region based processing of images and video sequences: application to filtering, segmentation and information retrieval**. 2002. Tese (Doutorado)– Department of Signal Theory and Communications, Universidade Politécnica da Catalunha, Barcelona, 2002.

GOLNABI, H.; ASADPOUR, A. Design and application of industrial machine vision systems. **Robotics and Computer-Integrated Manufacturing**, v. 23, p. 630-637, 2007.

GONÇALVES, V. D. ***Software* para controle de um manipulador robótico auxiliado por um sistema de visão**. 2004. Dissertação (Mestrado em Engenharia Mecânica)–Universidade de Taubaté, Taubaté, 2004.

GONZALEZ, R. C.; WOODS, R. E. **Digital image processing**. 2. ed. Nova Jersey: Prentice Hall, 2002. 793 p.

GUO, D.; JU, H.; YAO, Y. Research of manipulator motion planning algorithm based on vision. *In*: INTERNATIONAL CONFERENCE ON FUZZY SYSTEMS AND KNOWLEDGE DISCOVERY, 6., 2009, Tianjin. **Anais**… Tianjin: Departament of Control Engineering Chengdu University of Information Technology, 2009. p. 420-425.

HOLLERBACH, J. M. **Lecture notes** – introduction to robotics. School of Computing, The University of Utah, 2003.

ILLINGWORTH, J.; KITTLER, J. The adaptive hough transform. **IEEE Transactions on Pattern Analysis And Machine Intelligence**, v. 9, n. 5, 1987.

JAMALUDDIN, M. H. *et al*. Vision guided manipulator for optimal dynamic performance. *In*: STUDENT CONFERENCE ON RESEARCH AND DEVELOPMENT – SCOReD, 4., 2006, Malásia. **Anais**...

LAPLANTE, P. A. **Real-time systems design and analysis**. 3. ed. John Wiley & Sons, 2004. 505 p.

LLOYD, J. E. *et al*. "**Model-based Telerobotics with Vision**". University of British Columbia, Canada. 1997

LOTUFO, R. A. **Fast course of mathematical morphology**. 1997. Disponível em: <http://www.dca.fee.unicamp.br/~lotufo/khoros/mmach/tutor/util/fast.html>.

_____; MACHADO, R. C. **SDC morphology toolbox for python**. Disponível em: <http://www.mmorph.com/pymorph/morph/index.html>. Acesso em: fev. 2010.

MARCUS, S. C. Remembering the transformation matrix for rotation of an orthogonal coordinate system. *In*: IEEE TRANSACTIONS ON AEROSPACE AND ELECTRONIC SYSTEMS, 1967. **Anais**…

MIRANDA NETO, A. de. **Navegação de robôs autônomos baseada em monovisão**. 2007. Dissertação (Mestrado em Engenharia Mecânica)–Universidade Estadual de Campinas, Departamento de Mecânica Computacional, Campinas, 2007.

_____ *et al*. Self-organizing maps for environments and states mapping of an autonomous navigation system based on monocular vison. *In*: INTERNATIONAL CONGRESS OF MECHANICAL ENGINEERING – COBEM, 20., 2009, Gramado. **Anais**…

NDAJAH, P. *et al*. **SSIM image quality metric for denoised images**. Advances in visualization, imaging and simulation. Departament of Electrical and Electronics Engineering, Universidade de Niigata, 2010.

NIXON, M. S.; AGUADO, A. S. **Feature extraction and image processing**. 2. ed. Elsevier, 2009. 406 p.

NOVINI, A. Fundamentals of machine vision ligthing. *In*: IEEE WESCON CONFERENCE RECORD, 1993. **Anais**… p. 44-52.

NUMPY. **Scientific computing tools for Python – Numpy**. Disponível em: <numpy.scipy.org>. Acesso em: out. 2010.

O'GORMAN, L.; SAMMON, M. J.; SEUL, M. **Practical algorithms for image analysis**. 2. ed. Cambridge University Press, 2009. 349 p.

OKAFOR, A. C.; ERTEKIN, Y. M. Derivation of machine tool error models and error compensation procedure for three axes vertical machineing center using rigid body kinematics. **International Journal of Machine Tools & Manufacture**, p. 1.199-1.213, 2000.

OLDEROG, E.; DIERKS, H. **Real-time systems:** formal specification and automatic verification. Cambridge University Press, 2008. 320 p.

OLIPHANT, T. E. **Guide to NumPy**. 2006. 378 p.

OLIVEIRA, A. S. de; PIERI, A. R. de; MORENO, U. F. Análise cinemática via quatérnios duais. *In*: CONGRESSO BRASILEIRO DE AUTOMÁTICA, 18., 2010, Bonito. **Anais**… p. 1.640-1.647.

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

PIL, Python Imaging Library (PIL). Disponível em: <**http://www.pythonware.com/products/pil/**> Acesso em: jul. 2010.

PRATT, W. K. **Digital image processing**. Nova Jersey: John Wiley & Sons., 2007.

PYTHON, Python Programming Language – Official website. Disponível em: <http://www.python.org/>. Acesso em: out. 2010.

RIOUL, O.; VETTERLI, M. Wavelets and signal processing. **IEEE SP Magazine**, p. 14-38, 1991.

RODRIGUEZ, D. A.; NORRISH, J.; NICHOLSON, A. Robot programming for non repetitive repair operations using vision systems. *In*: INTERNATIONAL CONFERENCE, 8., 2009. **Anais**… University Trends in Welding Research. p. 671-675.

ROMANO, V. F. **Robótica industrial:** aplicação na indústria de manufatura e processos. São Paulo: Edgard Blücher, 2002. 256 p.

SALEMBIER, P.; OLIVERAS, A.; GARRIDO, L. Antiextensive connected operators for image and sequence processing. **IEEE Transactions on Image Processing**, v. 7, n. 4, p. 555-570, 1998.

SAMPAT, M. P. *et al*. Complex wavelet structural similarity: a new image similarity index. **IEEE Transactions on Image Processing**, v. 18, n. 11, 2009.

SEMIM, R. C. *et al*. Engine head blocks handling robot guided by vision system. *In*: INCOM 2012 - IFAC SYMPOSIUM ON INFORMATION CONTROL PROBLEMS IN MANUFACTURING, 14., 2012, Bucareste. **Anais**… Laxenburg: International Federation of Automatic Control, 2012. v. 14. p. 859-864.

SILVA, A. G. **Notas de aula**. Disponível em: <http://parati.dca.fee.unicamp.br/adesso/wiki/PI-UDESC/10s2_pdi/view/>. Acesso: set. 2010.

_____. **Uso de árvore de componentes para filtragem, segmentação e detecção de padrões em imagens digitais**. 2009. Tese (Doutorado)–Universidade Estadual de Campinas, Faculdade de Engenharia Elétrica e de Computação, Campinas, 2009.

_____; LOTUFO, R. A. **IA636 – Image Processing Course**. Disponível em: <http://www.dca.fee.unicamp.br/ia636/README.html>. Acesso em: out. 2010.

SILVA, C. A. M.; ANTONIO, S. dos S.; SANTOS, T. B. dos. **Sistema de monitoramento remoto de ambientes**. Belém: Instituto de Estudos Superiores da Amazônia / Engenharia da Computação, 2008.

SILVA, R. K.; RIBEIRO, S. R. A. Importância da alteração do histograma de imagem de alta resolução (PAN) para fusão de imagens digitais pelo método de componentes principais. **Ambiência - Revista do Setor de Ciências Agrárias e Ambientais**, v. 5, n. 1, p. 27-36, jan./abr. 2009.

SIRISANTISAMRID, K.; TIRASESTH, K.; MATSUURA, T. Determine calibration parameters with satisfied constraints for coplanar camera calibration. *In*: TENCON 2005 IEEE REGION 10, 2005. **Anais**… p. 1-5.

SOILLE, P. **Morphological image analysis:** principles and applications. 2. ed. Berlim: Springer, 2004. 391 p.

STIVANELLO, M. E.; GOMES, P. C. R. Inspeção visual industrial automatizada por análise de forma com descritores de Fourier e redes neurais artificiais. *In*: SEMINÁRIO DE COMPUTAÇÃO, 15., 2006, Blumenau. **Anais**... Blumenau: Universidade Regional de Blumenau, 2006. p. 29-40.

TRESPADERNE, F. M.; LÓPEZ, E. F. **Visually guided robot for radiator sealing**. *In*: CONFERENCE ON EMERGING TECHNOLOGIES & FACTORY AUTOMATION, 2009, Palma de Mallorca. **Anais**... p. 983-989.

WANG, Z. *et al*. Image quality assessment: from error visibility to structural similarity. **IEEE Transaction on Image Processing**, v. 13, n. 14, 2004.

WANG, Z.; SIMONCELLI, E. P. Translation insensitive image similarity in complex wavelet domain. *In*: IEEE INTERNATIONAL CONFERENCE ACOUSTIC, SPEECH AND SIGNAL PROCESSING, 2005, Filadélfia, v. II. **Anais**… p. 573-576.

WANG, Z.; WANG, Z.; WU, Y. Recognition of corners of planar checkboard calibration pattern image. *In*: IEEE CHINESE CONTROL AND DECISION CONFERENCE, 2010. **Anais**… p. 3.224-3.228.

YEN, E. K.; JOHNSTON, R. G. **The ineffectiveness of the correlation coeficient for image comparisons**. Los Alamos National Laboratory, 2005. Disponível em: <http://library.lanl.gov/cgi-bin/getfile?00418797.pdf>.

ZHONG, F. *et al*. One method of bearing outside-diameter detection based on hough transform. *In*: IEEE International Conference on Automation and Logistics, 2007, Jinan. **Anais**… p. 1.382-1.385.

***Site* consultado**

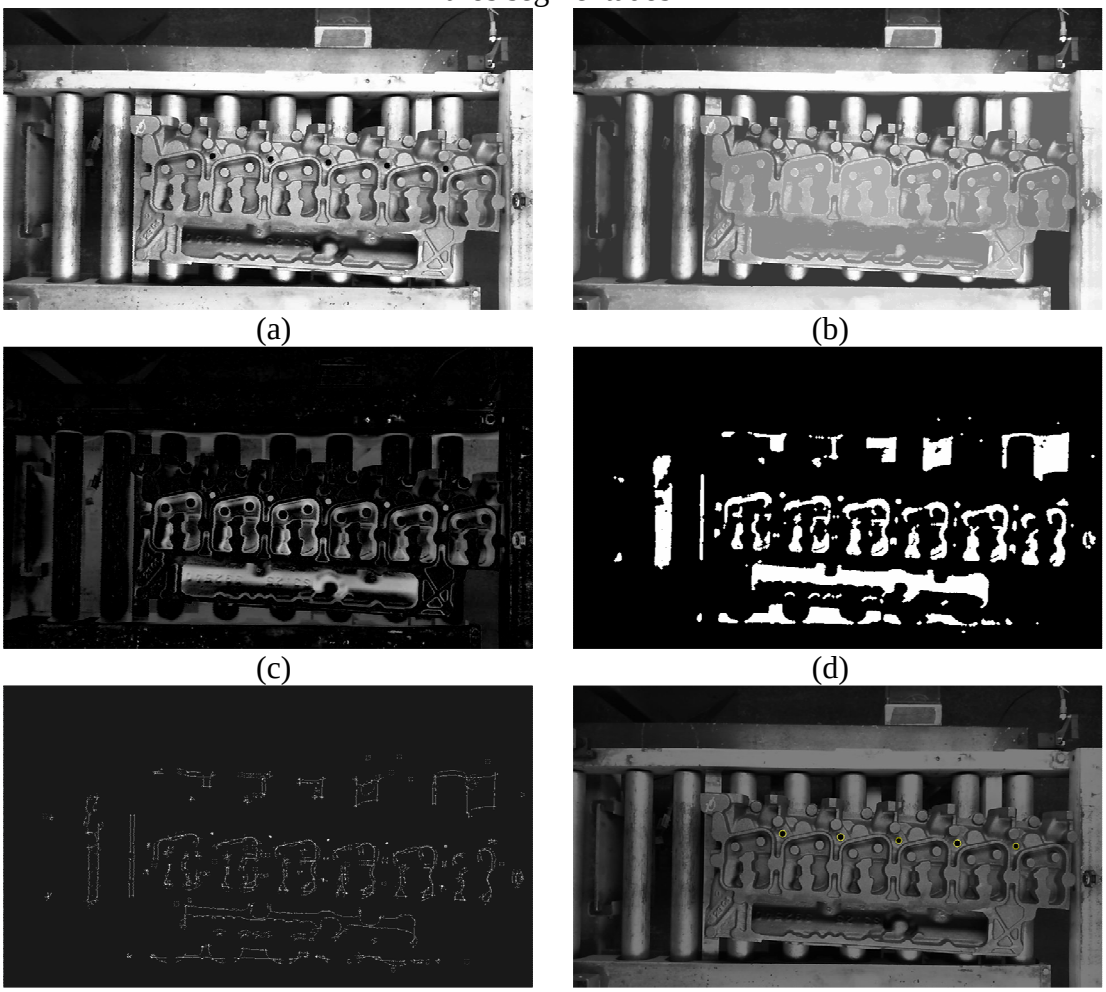
http://upload.wikimedia.org/wikipedia/commons/2/23/Wavelet_-_Morlet.png. Acesso em: abr. 2012

APÊNDICE A – Imagens da segmentação dos furos

As imagens mostram, para três famílias de cabeçotes (A, B e C), a sequência das etapas de segmentação dos furos dos cabeçotes, são elas:

- Imagem inicial;
- Fechamento de buracos;
- Subtração *top-hat*;
- *Threshold* da subtração *top-hat*;
- Espaço de parâmetros;
- Furos segmentados.

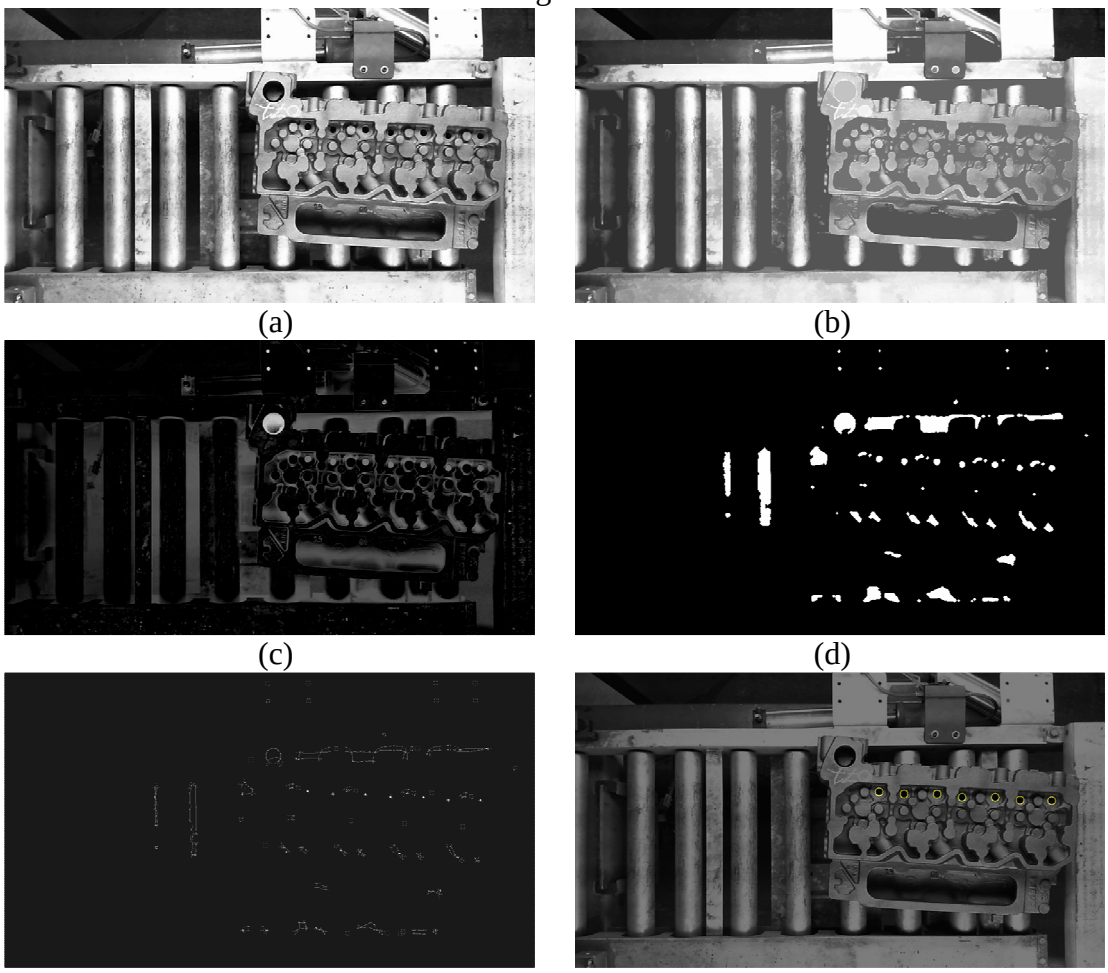
Fig. A.0.1: Sequência de segmentação dos furos para a família A: (a) imagem inicial, (b) fechamento de buracos, (c) *top-hat*, (d) *threshold* do *top-hat*, (e) espaço de parâmetros e (f) furos segmentados



(a) (b) (c) (d) (e) (f)

Fonte: produção próprio autor

Fig. A.0.2: Sequência de segmentação dos furos para a família B: (a) imagem inicial, (b) fechamento de buracos, (c) *top-hat*, (d) *threshold* do *top-hat*, (e) espaço de parâmetros e (f) furos segmentados



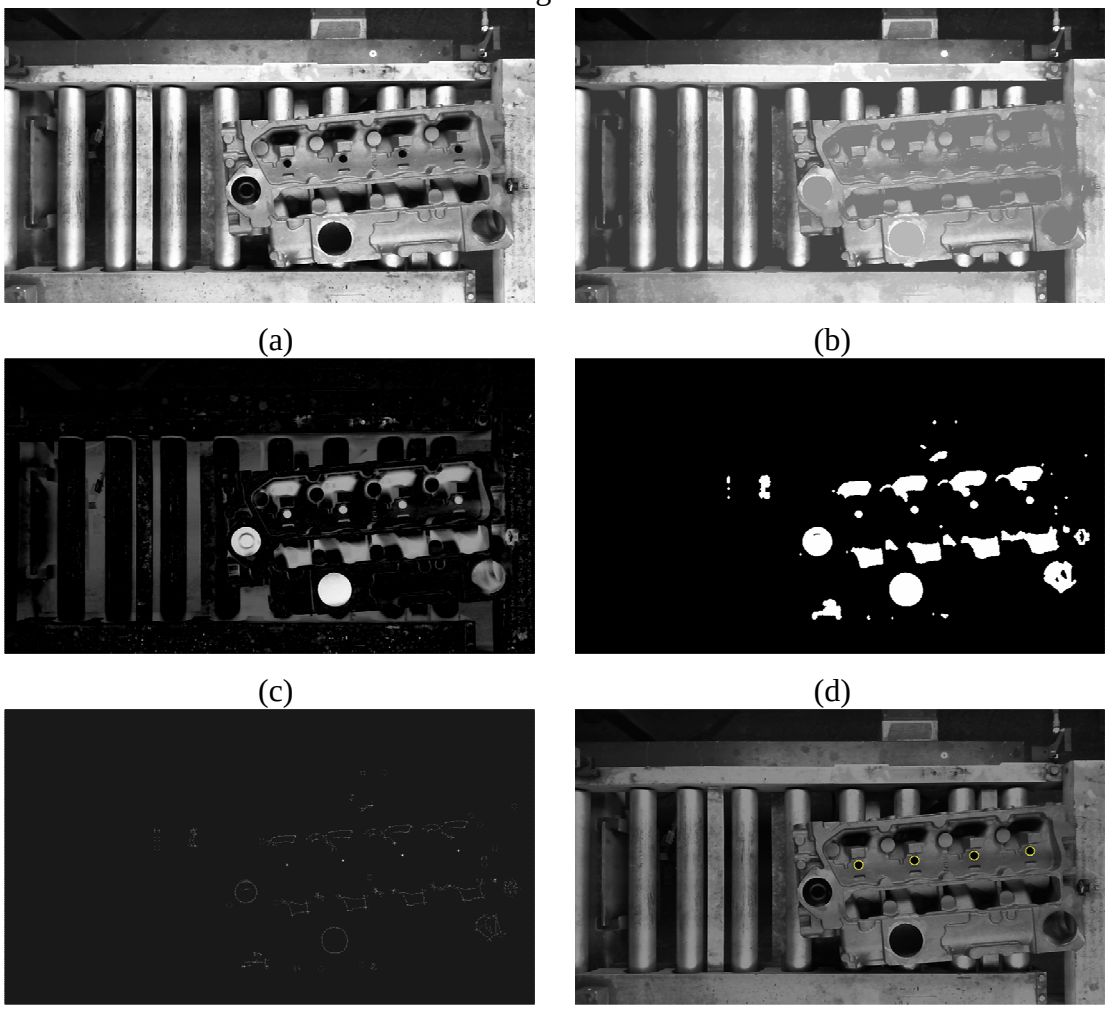
(a) (b)

(c) (d)

(e) (f)

Fonte: produção próprio autor

Fig. A.0.3: Sequência de segmentação dos furos para a família C: (a) imagem inicial, (b) fechamento de buracos, (c) *top-hat*, (d) *threshold* do *top-hat*, (e) espaço de parâmetros e (f) furos segmentados



(a) (b)

(c) (d)

(e) (f)

Fonte: produção próprio autor

APÊNDICE B – Protótipo do *software* de localização do cabeçote

O protótipo do *software* para segmentação das circunferências e cálculo de posição/orientação utilizado nos testes foi desenvolvido em Python.

```
from adpil import *
from time import *
from ia636 import *
from numpy import *
from morph import *
from math import *
from time import *
import serial
from SerialClient import *
from SerialUtils import *

################################################
# ENCONTRA A MATRIZ GRADIENTE
def sobel(f):
   wx = [[1,2,1],[0,0,0],[-1,-2,-1]]
   wy = [[1,0,-1],[2,0,-2],[1,0,-1]]
   gx = convolucao(f, wx)
   gy = convolucai(f, wy)
   mag = abs(gx + gy*1j)
   theta = arctan((gy)/(gx))
   return mag,theta

################################################
# CRESCIMENTO DE REGIÕES
def crescimento(f, fila_semente, dif, len_max):
   g = zeros(f.shape)
   f = f.astype('uint8')
   d = 0
   area = 0
   perimetro = 0
   sup = 0
   linha_sup = f.última_linha - 1
   linha_inf = 0
   coluna_esq = f.última_coluna - 1
   coluna_dir = 0
   while (len(fila_semente) > 0):
      x,y = fila_semente.remove(primeiro_elemento)
      for i (-1 até 1):
         for j (-1 até 1):
            if  f(x,y) - f(x+i,y+j) = dif and g(x+i,y+j) = 0:
               fila_semente.adiciona(x+i,y+j)
               g(x+i,y+j) = 1
               area = área + 1
               if linha_sup > (x+i):
                  linha_sup = x+i
               if linha_inf < (x+i):
                  linha_inf = x+i
               if coluna_esq > (y+j):
                  coluna_esq = y+j
               if coluna_dir < (y+j):
                  coluna_dir = y+j
               perimetro = perimetro + 1
      if area > len_max:
         fila_semente = []
   linha = linha_sup + (linha_inf - linha_sup)/2
   coluna = coluna_esq + (coluna_dir - coluna_esq)/2

   return g, (linha,coluna), area, perimetro,

################################################
# COMPARA UMA CIRCUNFERÊNCIA DA IMAGEM F COM UMA CIRCUNFERÊNCIA IDEAL UTILIZANDO CORRELAÇÃO
# DE PEARSON
# f = imagem realce de bordas limiarizado
# centro[0] = número da linha em f onde está o centro da circunferência
# centro[1] = número da coluna em f onde está o centro da circunferência
# raio = valor do raio a ser verificado
# T = valor de similaridade que deve ser encontrado entre a circunferência desejada e a encontrada na imagem
```

```
def proc_raio(f, centro, raio, T, dist):

   A = 2*raio + 7
   r = 0
   x = 0
   y = 0
   xm = 0
   ym = 0
   sx = 0
   sx2 = 0
   sy = 0
   sy2 = 0
   sxsy = 0
   r_maior = 0
   raio_hough = raio
   coord = centro
   linha_sup = f.última_linha - 1
   linha_inf = 0
   coluna_esq = f.última_coluna - 1
   coluna_dir = 0
   linha_ini = centro[0] - A/2
   linha_fim = centro[0] - A/2 + A
   coluna_ini = centro[1] - A/2
   coluna_fim = centro[1] - A/2 + A

   if (linha_ini < 0): linha_ini = 0
   if (linha_fim >= f.última_linha): linha_fim = f.última_linha-1
   if (coluna_ini < 0): coluna_ini = 0
   if (coluna_fim >= f.última_coluna): coluna_fim  = f.última_coluna-1
#região em f onde se encontra a circunferência a ser inspecionada
   regiao = f[linha_ini:linha_fim, coluna_ini:coluna_fim]
#procura o centro do objeto, recortado da imagem original, para alinhar a circ ideal com ele e depois fazer a comparação.
   for i (0 até regiao.última_linha-1):
      for j (0 até regiao.última_coluna-1):
         if regiao[i,j] == 1:
            if linha_sup > i:
               linha_sup = i
            if linha_inf < i:
               linha_inf = i
            if coluna_esq > j:
               coluna_esq = j
            if coluna_dir < j:
               coluna_dir = j
   linha = linha_sup + (linha_inf - linha_sup)/2
   coluna = coluna_esq + (coluna_dir - coluna_esq)/2
   centro = (linha,coluna)
#faz a correlação de Pearson entre a circ ideal e o objeto da imagem original, os dois devem estar com os centros alinhados.
   for k (0 até (dist+2)):
      g = criar_circulo((raio-1)+k, centro(x,y))
      for m (0 até regiao.última_linha-1):
         for n (0 até regiao.última_coluna-1):
            x = x + regiao[m,n]
            y = y + g[m,n]
      xm = 1.*x/(m*n)
      ym = 1.*y/(m*n)
      for m (0 até regiao.última_linha-1):
         for n (0 até regiao.última_coluna-1):
            sx2 = sx2 + (regiao[m,n]-xm)*(regiao[m,n]-xm)
            sy2 = sy2 + (g[m,n]-ym)*(g[m,n]-ym)
            sxsy = sxsy + (regiao[m,n]-xm)*(g[m,n]-ym)
      r = 1.*sxsy/(sqrt(sx2)*sqrt(sy2))
      if r >= T and r > r_maior:
         r_maior = r
         raio_hough = (raio-1) + k
      elif r < T:
         raio_hough = 0
   return raio_hough
################################################
# ALTERA O BRILHO/CONTRASTE NA REGIÃO ENTRE P1 E P2
def brilho_contraste(img,p1,p2):

            xp1, yp1 = p1
            xp2, yp2 = p2
            mapa = vetor 256 posições
            x = vetor 256 posições com valores de 0 a 256 e passo 1
            mapa[0:xp1] = 1.0*yp1/xp1 * x[0:xp1]
```

```
            mapa[xp1:xp2] = yp1 + (x[xp1:xp2] - xp1) * (yp2 - yp1)/(xp2-xp1)
            mapa[xp2:256] = yp2 + (x[xp2:256] - xp2) * (255 - yp2)/(255 - xp2)
            mapa = (mapa+0.5)
            img_out = mapa[ img ]
            return img_out

################################################
# ENCONTRA O MÁXIMO MÉDIO DE UMA IMAGEM POR MEIO DOS MÁXIMOS REGIONAIS
def bina(img, div):

   g = matriz de zeros na dimensão de img
   M = img.última_linha
   N = img.última_coluna
   m = M/div
   n = N/div
   pico_acum = 0

   for i (0 até div-1):
      for j (0 até div-1):
         lin_ini = i*m
         lin_fim = (m + m*i)
         col_ini = j*n
         col_fim = (n + n*j)
         if (lin_ini < 0): lin_ini = 0
         if (lin_fim >= img.última_linha-1): lin_fim = img.última_linha-1
         if (col_ini < 0): col_ini = 0
         if (col_fim >= img.última_coluna-1): col_fim = img.última_coluna-1
         aux = img[lin_ini:lin_fim, col_ini:col_fim]
         pico_max = extrai o valor maximo de (aux)
         pico_acum = pico_acum + pico_max
   pico = pico_acum/(div*div)
   return pico

################################################
# ORDENA UM VETOR DE CIRCUNFERÊNCIAS PELA COLUNA, DA MENOR => MAIOR
def ordena(circ):
   circ_novo = circ
   r = 0

   while r < circ.tamanho:
      temp1 = circ_novo[r]
      if r > 0:
         temp1 = circ_novo[r]
         temp2 = circ_novo[r-1]
         if temp1[1] < temp2[1]:
            temp3 = circ_novo.remove(r-1)
            circ_novo.adiciona(temp3)
            r = 0
         else:
            r = r + 1
      if r == 0: r = r +1
   circ = circ_novo
   return circ

################################################
# SEGMENTAÇÃO DAS CIRCUNFERÊNCIAS
# Parâmetros da função:
# - img: imagem de entrada, onde os círculos serão procurados
# - r1: valor do primeiro raio a ser procurado
# - dist: tolerância, para mais e para menos, que será utilizada pra encontrar o raio r1
# - passo: incremento que será utilizado, no algoritmo de Hough, para procurar o raio r1
def hough_circle(img, r1, dist, passo):
   esp_par = matriz de zeros na dimensão de img
   out = matriz de zeros na dimensão de img
   raio = [r1]
   esp = []
   centro = []
   circ = []

   temp = fechamento de buracos de (img)
   temp = temp - img
   pico_med = bina(temp, 20)
   temp = brilho_contraste(temp,((1*pico_med)-10,10),((1*pico_med)+10,240))
   pico = extrai o valor maximo de (temp)
   temp = threshold com valor de corte igual a 2.0*pico_med
   temp = erosão com elemento estruturante circular de raio = 2
```

```
    temp = dilatação com elemento estruturante circular de raio = 2
    img_borda,theta = sobel(temp)
    img_borda_bin = temp
    for i (0 até img_borda_bin.última_inha-1):
        for j (0 até img_borda_bin.última_coluna-1):
            if 0 <= (i-1) and (i+1) < img_borda_bin.última_linha-1:
                if 0 <= (j-1) and (j+1) < img_borda_bin.última_coluna-1:
                    if img_borda_bin[i,j] > 0 and (img_borda_bin[i-1,j] == 0 or img_borda_bin[i,j-1] == 0 or img_borda_bin[i+1,j] == 0 or
img_borda_bin[i,j+1] == 0):
                        r = r1
                        while r <= (r1+dist):
                            a = (i - r1 * sin(pi+theta[i,j]))
                            b = (j - r1 * cos(pi+theta[i,j]))
                            if a < esp_par.shape[0]:
                                if b < esp_par.shape[1]:
                                    esp_par[a,b] = esp_par[a,b] + 1
                            r = r + passo
#Procura os máximos no espaço de parâmetros.
    picos = extrai o valor maximo de (esp_par)
    spot = 0.3*picos
    esp_aux = threshold com valor de corte igual a “spot”
    esp_aux = dilatação com elemento estruturante quadrado de tamanho 10x10
    esp_aux = erosão com elemento estruturante quadrado de tamanho 9x9
    for i (0 até esp_aux.última_linha-1):
        for j (0 até esp_aux.última_coluna-1):
            if esp_aux[i,j] == 1:
                n = 0
                r_tot = 0
                a_tot = 0
                b_tot = 0
                semente = [(i,j)]
                aux, centro, area, perimetro = crescimento(esp_aux, semente, 0, 30)
                if area <= 30:
                    raio_hough = proc_raio(temp, centro, r1, 0.85, dist)
                    if raio_hough != 0:
                        circ.adiciona((centro[0],centro[1],raio_hough))
                esp_aux = esp_aux - aux
#Ordena o vetor com os centros de circunferências, pela coluna, da menor para a maior.
    circ = ordena(circ)
    for i (0 até circ.tamanho-1):
        a,b,c = circ[i]
#se a coluna for menor que 530 elimina o ponto porque não pode ser o primeiro.
        if b < 530:
            circ[i] = (0,0,0)
#Elimina os círculos encontrados muito fora da região permitida, que foram marcados com coordenadas iguais a zero.
    i = 0
    while i < circ.tamanho:
        a,b,c = circ[i]
        if a == 0 and b == 0 and c == 0:
            eliminado = circ.remove(i)
            i = -1
        i = i + 1
#Zera as coordenadas das circunferências que estão fora da reta imaginária que representa a inclinação do cabeçote.
    dec = 0
    for i (0 até circ.tamanho-1):
        if i == (circ.tamanho-1):
            a,b,c = circ[i-1]
            d,e,f = circ[i]
            aux = 2
#volta, um a um, no vetor para procurar uma coordenada de centro que seja diferente de zero.
            while a == 0 and b ==0:
                a,b,c = circ[i-aux]
                aux = aux + 1
            if (((d-a) < -40 or (d-a) > 40) or ((e-b) > 120) or (e-b) < 50):
                eliminado = circ.remove(i)
        if i < circ.tamanho-1:
            a,b,c = circ[i]
            if i == 0:
                meio_esq = 0
                meio_dir = 0
                d,e,f = circ[i+1]
                m,n,o = circ[i+2]
                if (((d-a) < -40 or (d-a) > 40) or ((e-b) > 120) or (e-b) < 40):
                    meio_esq = 1
                if (((m-d) < -40 or (m-d) > 40) or ((n-e) > 120) or (n-e) < 40):
                    meio_dir = 1
```

```
#se o primeiro não combina com o segundo e nem com o terceiro, ele é excluído.
            if (meio_esq == 1 and meio_dir == 0) or (meio_esq == 1 and meio_dir == 1):
              circ[i] = ((0,0,0))
#analisa os pontos do vetor, entre o primeiro e o último.
        else:
            d,e,f = circ[i-1]
            aux = 2
        esq_zero = 0
#Volta, um a um, no vetor para procurar uma coordenada de centro que seja diferente de zero.
#Somente executa se as coordenadas do ponto i-1 forem zero.
        while d == 0 and e ==0:
#se o primeiro ponto do vetor tiver as coordenadas iguais a zero, utiliza o ponto i+1 do vetor.
            if i-aux <= 0:
                d,e,f = circ[i+1]
#flag que marca se todos os pontos à esquerda de i possuem as coordenadas iguais a zero.
                esq_zero = 1
            else:
                d,e,f = circ[i-aux]
            aux = aux + 1
#procura por pontos muito fora da linha imaginária que simboliza a inclinação do cabeçote.
        if i > 0 and (((d-a) < -40 or (d-a) > 40) or ((b-e) > 120) or (b-e) < 40) and esq_zero == 0:
            circ[i] = ((0,0,0))
#procura por pontos muito próximos um do outro.
        elif i > 0 and ((b-e) < 20) and esq_zero == 0:
            circ[i] = ((0,0,0))
#analisa o caso em que o primeiro ponto teve suas coordenadas zeradas.
        elif esq_zero == 1:
            meio_esq = 0
            meio_dir = 0
            d,e,f = circ[i+1]
#compara o ponto i com o ponto i+1.
            if (((d-a) < -40 or (d-a) > 40) or ((e-b) > 120) or (e-b) < 40):
                meio_esq = 1
#verifica o limite do vetor, se atingir descarta o ponto i+2.
            if i+2 < len(circ):
                m,n,o = circ[i+2]
#compara o ponto i com o ponto i+2.
                if (((m-d) < -40 or (m-d) > 40) or ((n-e) > 120) or (n-e) < 40):
                    meio_dir = 1
#analisa o caso em que i+2 > len(circ).
            else:
                meio_dir = 1
#se o ponto i não combina com o ponto i+1 e nem com o ponto i+2, ele é excluído.
            if (meio_esq == 1 and meio_dir == 0) or (meio_esq == 1 and meio_dir == 1):
                circ[i] = ((0,0,0))
#Elimina os círculos encontrados fora da reta imaginária, que foram marcados com coordenadas iguais a zero.
    i = 0
    while i < circ.tamanho:
        a,b,c = circ[i]
        if a == 0 and b == 0 and c == 0:
            eliminado = circ.remove(i)
            i = -1
        i = i + 1
#Verifica a distância entre o primeiro e o último furo, ser for menor que 290, falta pelo menos um furo. A posição do último furo
#é verificada, se estiver entre as colunas 850 e 890 ele é realmente o último. Nesse caso, encontramos o ângulo theta entre o
#último e o antepenúltimo furo para poder estimar a posição do primeiro.
#Se a distância entre o primeiro e o último furo for maior que 320 e a coluna do último estiver entre 850 e 890, o primeiro furo é
#removido do vetor.
    eliminar = 0
    stop = 0
    a,b,c = circ[0]
    d,e,f = circ[circ.tamanho-1]
    while stop == 0:
        if e > 890:
            eliminar = circ.remove(len(circ)-1)
        if (e-b) < 290:
            if (850 < e < 890):
                a,b,c = circ[circ.tamanho-2]
                d,e,f = circ[circ.tamanho-1]
                theta = arctan((d-a)/(e-b))
                x = (e - 315*cos(theta))
                y = (d - 315*sin(theta))
                circ.adiciona((y,x,r1))
                stop = 1
            else:
                stop = 1
```

```
        elif (e-b) > 320 and (850 < e < 890):
            eliminar = circ.remove(0)
        else:
            stop = 1
        a,b,c = circ[0]
        d,e,f = circ[circ.tamanho-1]
    circ = ordena(circ)

    return circ, result

################################################
# CÁLCULO DA POSIÇÃO/ORIENTAÇÃO DO CABEÇOTE
def incl_reta(f, h, pos, ref):
  w = pos.tamanho
  L,C = dimensoes de f
  dir_j = 0
  esq_j = f.última_coluna
  alpha = 0
  beta = 0
  theta = 0
  if pos.tamanho > 0:
#Procura os dois pontos extremos.
      for i (0 até w-1):
        m,n,r = pos[i]
        if n > dir_j:
          dir_i = 1.*m
          dir_j = 1.*n
        if n <= esq_j:
          esq_i = 1.*m
          esq_j = 1.*n
    alpha = esq_i
  if dir_i != esq_i and dir_j != esq_j:
      beta = ((dir_i-esq_i)/(dir_j-esq_j))
#o resultado da função atan é em radianos, é necessária a conversão feita na linha abaixo.
      aux = arctan((dir_i-esq_i)/(dir_j-esq_j))
#aqui fazemos a inclinação do cabeçote menos a inclinação do sistema de coordenadas (em radianos) em relação à borda
#superior da imagem.
      theta = (aux)-(-0.0134521033127)
      print 'Theta = ', theta
      q0 = .5*sqrt(cos(theta)+cos(theta)+2)
      q1 = 0
      q2 = 0
      if beta > 0:
          q3 = -.5*sqrt(-cos(theta)-cos(theta)+2)
      else:
          q3 = .5*sqrt(-cos(theta)-cos(theta)+2)
#esse valor deve vir da calibração (em graus)
      theta = (180*aux/pi)-(-1.26181753889)
 #coordenadas do ponto zero do sistema de coordenadas de referência.
  zero_i = 205
  zero_j = 769
#essa função encontra o deslocamento, da peça referência, com relação ao ponto zero do sistema.
  if ref == 1:
#primeiro ponto do vetor de coordenadas.
      prim = pos[0]
#segundo ponto do vetor de coordenadas.
      segu = pos[w-1]
      ponto_ref_i = (((segu[0]-prim[0]))/2 + prim[0])
      ponto_ref_j = ((segu[1]-prim[1])/2 + prim[1])
      desloc_ref_i = ponto_ref_i - zero_i
      desloc_ref_j = zero_j - ponto_ref_j
      print 'Deslocamento cabeçote referencia em relação ao ponto zero: ', (desloc_ref_i,desloc_ref_j)
#essa função utiliza o deslocamento encontrado acima.
  else:
      desloc_ref_i = 14
      desloc_ref_j = 65
  prim = pos[0]
  segu = pos[w-1]
  if zero_i >= (((segu[0]-prim[0]))/2 + prim[0]):
#encontra a coordenada i do ponto central da reta imaginária que representa a inclinação do cabeçote.
      ponto_i = zero_i - (((segu[0]-prim[0]))/2 + prim[0])
  elif zero_i < (((segu[0]-prim[0]))/2 + prim[0]):
      ponto_i = (((segu[0]-prim[0]))/2 + prim[0]) - zero_i
  if zero_j >= ((segu[1]-prim[1])/2 + prim[1]):
#encontra a coordenada j do ponto central da reta imaginária que representa a inclinação do cabeçote.
      ponto_j = zero_j - ((segu[1]-prim[1])/2 + prim[1])
```

```
  elif zero_j < ((segu[1]-prim[1])/2 + prim[1]):
      ponto_j = ((segu[1]-prim[1])/2 + prim[1]) - zero_j
#o 12 é uma constante de deslocamento da garra em mm - o qto a garra precisa se afastar da peça para chegar à posição de
#pega sem colidir com ela.
  if beta >= 0:
#deslocamento do cabeçote em relação ao ponto zero (pixels) X o valor de um pixel em milímetros.
      desloc_i = ((ponto_i-desloc_ref_i)*1.2568493657800892) - 0
      desloc_j = (ponto_j-desloc_ref_j)*1.25 + (-5)
  else:
      desloc_i = ((ponto_i-desloc_ref_i)*1.2568493657800892) + 12
      desloc_j = (ponto_j-desloc_ref_j)*1.255718996331973
  return g, desloc_j,desloc_i,q0,q1,q2,q3

################################################
# COMUNCAÇÃO SERIAL
def comserial(m,n,o,p,q,r):

    SERIALCOMM_BAUD_RATE = 9600
    SERIALCOMM_BYTE_SIZE = serial.EIGHTBITS
    SERIALCOMM_PARITY = serial.PARITY_NONE
    SERIALCOMM_STOP_BITS = serial.STOPBITS_ONE
    SERIALCOMM_TIMEOUT = 0.5 #2500ms

    print '[CLIENT] Selected a port below using the number:\n'

    portsList = listAvailableSerialPortsNames()
    i = int(0)
    for portName in portsList:
        print i, ") ", portName
        i+=1

    portID = int(raw_input('Enter the number: '))

    serialComm = SerialClient(portsList[portID], SERIALCOMM_BAUD_RATE, SERIALCOMM_BYTE_SIZE,
SERIALCOMM_PARITY, SERIALCOMM_STOP_BITS, SERIALCOMM_TIMEOUT)
    recebeLista = []
    recebe = ''
    envia = 1
    cont = 0

    serialComm.serial.abrir()
    while recebe <> '0':
#recebe do robô o número de caracteres da mensagem.
        recebeSize = serialComm.serial.ler()
        if recebeSize<>'':
            for i (0 até recebeSize-1):
                recebeLista.adiciona(serialComm.serial.ler())
            recebeLista = []
        if recebe == online:
            envia = 'ok\n'
            serialComm.serial.escreve(envia)
            recebe = ''
            cont = 0
        if recebe == aguardo:
            envia = ('[['+str(m)+','+str(n)+','+'0],['+str(o)+','+str(p)+','+str(q)+','+str(r)+']]'+'\n')
            serialComm.serial.escreve(envia)
            recebe = ''
            cont = 0
        if recebe == recebeu':
            serialComm.serial.fechar()
            recebe = '0'
        if cont>50:
            serialComm.serial.fechar()
            recebe = '0'
        cont = cont + 1
    return

################################################
f = ler uma imagem (Imagem5.jpg)
centros, img = hough_circle(f, 6, 1, 0.2)
a,b,c,d,e,f,g = incl_reta(f,img,centros, 0)
comserial(b,c,d,e,f,g)
```

APÊNDICE C – Protótipo do *software* de calibração do sistema

O protótipo do *software* de calibração do sistema de visão é uma continuação do apresentado no apêndice B.

```
from adpil import *
from time import *
from ia636 import *
from numpy import *
from morph import *
from math import *

################################################
# SEGMENTAÇÃO DAS CIRCUNFERÊNCIAS
# Parâmetros da função:
# - img: imagem de entrada, onde os círculos serão procurados
# - r1: valor do primeiro raio a ser procurado
# - dist: tolerância, para mais e para menos, que será utilizada para encontrar o raio r1
# - passo: incremento que será utilizado, no algoritmo de Hough, para procurar o raio r1
def hough_circle(img, r1, dist, passo):

    esp_par = zeros(img.shape)
    out = zeros(img.shape)
    raio = [r1]
    esp = []
    centro = []
    circ = []
    temp = mmclohole(img)
    temp = mmsubm(temp, img)
    pico = max(ravel(temp))
    temp = temp > 0.45*pico
    elem = iacircle((5,5), 2, (2,2))
    temp = mmero(temp,elem.astype('bool'))
    temp = mmdil(temp,elem.astype('bool'))
    temp = temp.astype('uint8')
    img_borda,theta = iasobel(temp)
    img_borda_bin = temp
    for i in range(img_borda_bin.shape[0]):
        for j in range(img_borda_bin.shape[1]):
            if 0 <= (i-1) and (i+1) < img_borda_bin.shape[0]:
                if 0 <= (j-1) and (j+1) < img_borda_bin.shape[1]:
                    if img_borda_bin[i,j] > 0 and (img_borda_bin[i-1,j] == 0 or img_borda_bin[i,j-1] == 0 or img_borda_bin[i+1,j] == 0 or
img_borda_bin[i,j+1] == 0):
                        for k in raio:
                            r = k - dist
                            while (k-dist) <= r <= (k+dist):
                                a = (i - r * sin(pi+theta[i,j]))
                                b = (j - r * cos(pi+theta[i,j]))
                                if a < esp_par.shape[0]:
                                    if b < esp_par.shape[1]:
                                        esp_par[a,b] = esp_par[a,b] + 1
                                r = r + passo
#procura os picos no espaço de parâmetros.
    picos = max(ravel(esp_par))
    spot = 0.6*picos
    esp_aux = esp_par > spot
    elem = ones((10,10))
    esp_aux = mmdil(esp_aux, elem.astype('bool'))
    elem = ones((9,9))
    esp_aux = mmero(esp_aux, elem.astype('bool'))
    for i in range(esp_aux.shape[0]):
      for j in range(esp_aux.shape[1]):
        if esp_aux[i,j] == 1:
            n = 0
            r_tot = 0
            a_tot = 0
            b_tot = 0
            semente = [(i,j)]
            aux, centro, area, perimetro = crescimento(esp_aux, semente, 0)
            if area < 20:
                raio_hough = proc_raio(img_borda_bin, centro, r1, 0.8)
                if raio_hough != 0:
```

```
                circ.append((centro[0],centro[1],r1))
            esp_aux = esp_aux - aux
    print 'len(circ):', len(circ)
    return circ

################################################
# CÁLCULO DA POSIÇÃO/ORIENTAÇÃO DO SISTEMA DE COORDENADAS E TAMANHO DO PIXEL
# Parâmetros da função:
# - dxreal: distância entre os centros dos furos que representam o eixo X e o ponto Zero do sistema
# - dyreal: distância entre os centros dos furos que representam o eixo Y e o ponto Zero do sistema
def incl_reta(f, pos, dxreal, dyreal):

  w = len(pos)
  L,C = f.shape
  x_i = f.shape[0]
  x_j = f.shape[1]
  y_i = 0
  y_j = 0
  zero_i = y_i
  zero_j = x_j
  alpha = 0
  beta = 0
  theta = 0
  if len(pos) > 0:
#procura os pontos referentes aos eixos X e Y.
      for i in range(w):
        m,n,r = pos[i]
        if m > y_i:
           y_i = 1.*m
           y_j = 1.*n
        if n <= x_j:
           x_i = 1.*m
           x_j = 1.*n
#procura o ponto referente ao Zero do sistema.
      for i in range(w):
        m,n,r = pos[i]
        if m < y_i and n > x_j:
           zero_i = 1.*m
           zero_j = 1.*n
  else:
      print 'Não encontrou círculos'
      return g, (alpha,beta)
  alpha = x_i
#Cálculo dos coeficientes de inclinação do eixo X, lembrando j = representa o eixo X e i = representa o eixo Y.
#Verifica se as coordenadas da circunferência do eixo X são diferentes das da circunferência do ponto Zero.
  if x_i != zero_i and x_j != zero_j:
      beta = ((zero_i-x_i)/(zero_j-x_j))
      aux = atan((zero_i-x_i)/(zero_j-x_j))
      theta = (180*aux/pi)
#encontra o tamanho de um pixel para o caso de theta ser negativo.
  if (x_i>=zero_i and y_j>=zero_j):
      dipixel = dyreal/(y_i-zero_i)
      djpixel = dxreal/(zero_j-x_j)
#encontra o tamanho de um pixel para o caso de theta ser positivo.
  elif ( x_i<zero_i and y_j<zero_j):
      thetax = atan((zero_i-x_i)/(zero_j-x_j))
      thetay = atan((zero_j-y_j)/(zero_i-y_i))
      dipixel = dyreal/((y_i-zero_i)*cos(thetay))
      djpixel = dxreal/((zero_j-x_j)*cos(thetax))

  print 'Tamannho de um pixel em mm (X,Y): ', (djpixel,dipixel)
  print 'Angulo theta (inclinaçao) da reta do eixo X: ', theta
  print 'Ponto zero do sistema: ', (zero_i,zero_j)

  return (zero_i,zero_j), theta, (djpixel,dipixel)

################################################
f = ler uma imagem (calibracao.jpg)
centros, img = hough_circle(f, 7, 1, 0.5)
incl_reta(f,centros,280,230)
```