CENTRO FEDERAL DE EDUCACAO TECNOLOGICA DE MINAS GERAIS

***Campus* DIVINÓPOLIS**

BACHARELADO EM ENGENHARIA MECATRÔNICA

Desenvolvimento de um Sistema de Baixo Custo para Rejeição de Objetos Indesejáveis em Linhas de Produção Usando Visão Computacional

Fady Aboujaoude

Divinópolis, 2014.

Fady Aboujaoude

Desenvolvimento de um Sistema de Baixo Custo para Rejeição de Objetos em Linhas de Produção Usando Visão Computacional

Monografia de Trabalho de Conclusão de Curso apresentada ao Colegiado de Graduação em Engenharia Mecatrônica como parte dos requisitos exigidos para obtenção do título de Engenheiro Mecatrônico.

Eixos de Formação: Computação, Eletrônica, Mecânica.

Orientador: Prof. Dr. Renato de Sousa Dâmaso

Divinópolis, 2014.

Centro Federal de Educação Tecnológica de Minas Gerais
CEFET-MG / Campus Divinópolis
Curso de Engenharia Mecatrônica

Monografia intitulada "*Desenvolvimento de um Sistema de Baixo Custo para Rejeição de Objetos Indesejáveis em Linhas de Produção Usando Visão Computacional* ", de autoria do graduando Fady Aboujaoude, aprovada pela banca examinadora constituída pelos seguintes professores:

_____

Prof. Dr. Renato de Sousa Dâmaso - CEFET-MG / Campus Divinópolis - Orientador

_____

Prof. M.Sc Cláudio Parreira Lopes - CEFET-MG / Campus Divinópolis

_____

Prof. M.Sc Jhonatan Fernando Oliveira- CEFET-MG / Campus Divinópolis

_____

Prof. Dr. Valter Junior de Souza Leite
Coordenador do Curso de Engenharia Mecatrônica
CEFET-MG / Campus Divinópolis

Divinópolis - Abril de 2014

Resumo

Nesse trabalho será abordada a construção de um sistema que utiliza visão computacional, emprega técnicas de processamento de imagens para monitorar uma linha de produção (através de uma *webcam*) e, com auxilio de um dispositivo mecânico, remover objetos indesejáveis. Os objetos analisados estavam sobre uma esteira rolante onde foi acoplado o dispositivo mecânico de descarte cuja comunicação com o programa computacional foi feita utilizando um cabo USB. O sistema utilizou elementos acessíveis, de forma a reduzir o custo final do sistema, para que fosse viável a instalação deste em empresas de pequeno porte. O programa desenvolvido tem a opção de inserir novos perfis de produtos para possibilitar a inclusão de novos produtos na linha. A interface com o usuário foi simplificada para facilitar a sua operação. Foram usados também programas intermediários: OpenCV e EmguCV, para tratar a sequencia de imagens vindas da câmera, facilitando o processamento posterior, e também suas APIs (ou Interface de Programação de Aplicativos) já existentes. As APIs foram comparadas, de modo a obter-se os que melhor se enquadram nas necessidades do sistema. O critério utilizado para análise da viabilidade do produto foi seu aspecto físico. Na parte da construção mecânica, o atuador responsável por rejeitar o objeto é constituído por um motor de passo com uma chapa fixada no seu eixo. Assim, quando houver necessidade de ser descartado, o motor cria um desvio na linha retirando o objeto da esteira. A esteira foi construída usando tubos de aço 1060 de perfil quadrado e um motor CC com redução, típico de um motor de vidro elétrico automotivo. Este foi ligado ao computador e controlado pelo mesmo programa. O programa principal mostrou-se capaz de reconhecer objetos e rejeitar objetos com defeitos. O dispositivo mostrou-se capaz de comunicar com o programa principal assim como controlar os dois motores.

Palavras-chave: visão computacional; OpenCV; EmguCV; comunicação USB

Abstract

This work presents an integrated system that uses computer vision, to employ image processing techniques, in order to monitor a production line (via webcam). With the aid of a mechanical device, the system will then remove unwanted objects. The objects are analyzed on a conveyor belt which is coupled to mechanical disposal device whose communication with the computer program was made using a USB cable. The system used accessible elements, to reduce final cost, so that its installation in small businesses would be feasible. The developed program has the option to enter new product profiles so that new products may be included in the production line. The user interface was simplified to facilitate operation. Intermediate programs were also used, from the OpenCV and EmguCV libraries, to treat the image sequences generated by the camera. APIs were compared, so as to obtain one that best fit the needs of the system. In the automatic analysis process the criterion used to identify the feasibility of the product was the products physical aspect. The actuator responsible for rejecting the object consists of a stepper motor with a plate fixed on its shaft, which creates detour on the conveyor belt removing the undesired product. The conveyor belt itself was built using steel and a typical power window DC motor. The main program was capable of recognizing objects and reject objects with defects. The device proved to be able to communicate with the main program as well as controlling the two motors.

Key-words: computer vision; OpenCV; EmguCV; USB communication

Sumário

Lista de Figuras

Lista de Tabelas

Acrônimos

SURF	*Speeded Up Robust Features* (Características Robustas Aceleradas)
SIFT	*Scale-invariant feature transform* (Transformada de Características Invariantes a Escala)
API	*Application Programming Interface* (Interface de Programação de Aplicativos)
OPENCV	*Open Computer Vision* (Visão Computacional Aberta)
EMGUCV	*Emgu Computer Vision* (Visão Computacional Emgu)
USB	*Universal Serial Bus* (Bus Serial Universal)
PIC	*Peripheral Interface Controller* (Controlador Periférico de Interface)
ISO	*International Organization for Standardization* (Organização Internacional para Padronização)
POO	Programação Orientada a Objetos
GUI	*Graphic User Interface* (interface gráfica do usuário)
IDE	*Integrated Development Environment* (Ambiente Integrado de Desenvolvimento)
AID	Ambiente Integrado de Desenvolvimento
RAD	*Rapid Application Development* (Desenvolvimento Rápido de Aplicação)
WPF	*Windows Presentation Foundation* (Fundação Windows de Apresentação)
XAML	Extensible Application Markup Language()
HTML	HyperText Markup Language (Linguagem de Marcação de Hipertexto)
CSS	Cascading Style Sheets
UML	Unified Modeling Language
LoG	*Laplacian of Gaussian* (Laplaciano de Gaussiano)
DoG	*Difference of Gaussian* (Diferença de Gaussiano)
CLI	Command-Line Interface
MIPS	Million Instructions Per Second
PWM	*Pulse Width Modulation* (Modulação por Largura de Pulso)
CCP	*Capture/Compare/PWM* (Captura/Comparação/PWM)

ECCP	*Enhanced CCP* (CCP Aprimorada)
SPI	Serial Peripheral Interface Bus (Bus Serial de interface Periférico)
RAM	Random Access Memory (Memória de acesso aleatório)
ROM	Read-Only Memory (Memoria Somente-Leitura)
EEPROM	Electrically-Erasable Programmable Read-Only Memory (Memoria Somente-Leitura Programável e Apagável Eletricamente)
HLVD	High/Low-Voltage Detect (Detecção de Tensão Alta/Baixa)
EUSTART	Enhanced Universal Synchronous Asynchronous Receiver Transmitter (Transmissor/Receptor Universal Síncrono/Assíncrono Aprimorado)
PID	Product Identification (Identificação do Produto)
VID	Vendor Identification (Identificação do Vendedor)
DLL	Dynamic-link library (Biblioteca de Vínculo Dinâmico)
CC	Corrente Contínua

Capitulo 1. Introdução

O presente trabalho consiste no desenvolvimento de um sistema para a análise de objetos em uma linha de produção por meio de visão computacional. Para isso, foi desenvolvido um programa computacional e um dispositivo periférico para ser acoplado a uma esteira transportadora de objetos, que de acordo com a análise removerá o produto que não esteja dentro dos parâmetros previamente estabelecidos.

1.1 JUSTIFICATIVA

O mercado consumidor está cada vez mais exigente em relação à qualidade dos produtos. Por esta razão, indústrias têm sido motivadas a investirem em processos de inspeção de qualidade (STIVANELLO, 2004). Esses investimentos são feitos com mais frequência por indústrias de médio a grande porte, desfavorecendo cada vez mais indústrias menores, que muitas vezes não possuem capital suficiente para realizá-los. Por este motivo, a análise de objetos em uma linha de produção por meio de visão computacional, utilizando um sistema de baixo custo, é de grande valia para empreendimentos menores, pois utiliza componentes acessíveis que facilitam a aquisição deste dispositivo. Para isso, o presente trabalho propõe o desenvolvimento de um programa computacional e de um dispositivo periférico para ser acoplado a uma esteira transportadora de objetos, como esquematizado na Figura 1.1.

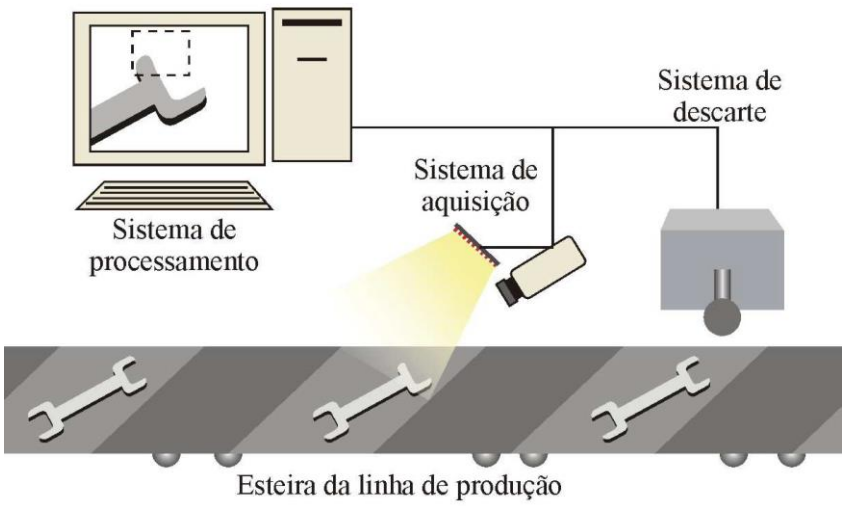


Figura 1.1 - Exemplo do dispositivo a ser desenvolvido (STIVANELLO, 2004)

O procedimento convencional de inspeção de qualidade é realizado manualmente, utilizando um operador que controla o produto final, como mostrado na

Figura 1.2. Por ser uma atividade extremamente repetitiva e podendo exigir um esforço físico excessivo do funcionário, este tipo de inspeção acarreta uma série de problemas como falta de precisão, alta rotatividade dos trabalhadores e também a falta de inspeção de alguns produtos. Segundo Stivanello (2004), no decorrer da jornada de trabalho há um declínio significativo de efetividade desta tarefa.

Figura 1.2 - Exemplo de inspeção manual na indústria de cerâmica (STIVANELLO, 2004)

Os problemas relacionados a este tipo de inspeção aumentam o custo de produção, pois acrescenta gastos extras com o “retrabalho”, transporte desnecessário e multas devido ao descumprimento dos padrões legais estabelecidos na comercialização (MACHADO, 2009)**.**

Machado (2009) demonstra uma saída para a redução dos gastos através da automatização dos processos de controle de qualidade mediante o uso de uma inspeção uniformizada com medidas objetivas. A automatização possibilita o aumento da produção, pois a inspeção pode ser feita de forma mais veloz, dependendo do processo de controle de qualidade, e a redução do custo de mão-de-obra.

Umas das partes mais dispendiosas dos sistemas automáticos de inspeção são os dispositivos que analisam o produto e processam a informação do sensor desse sistema. Por exemplo, um seletor de grãos utiliza um sensor óptico por translucidez de grande porte, que é relativamente caro. Se este sensor for substituído por uma *webcam*, o custo do dispositivo que analisa o produto diminuiria drasticamente, sem levar em conta que o processamento desta imagem é muito mais acessível, por ser feito por um *software* instalado em um computador convencional. Devido ao baixo custo de uma câmera que pode ser acoplada a um computador pessoal, o uso de visão computacional poderá ser uma solução para sistemas automáticos de inspeção.

Os estudos relacionados à visão computacional são vastos, e a área de aplicação desta tecnologia é ainda mais ampla. Existem infinitas possibilidades de como utilizar essa tecnologia, especialmente quando acoplada a outras já existentes. Molz (2001) faz uma alegação da importância do estudo na área de visão computacional:

> "Visão tem sido um foco de atenção para pesquisadores desde o início da computação, pois este é um dos mais notáveis sistemas de percepção dos seres humanos. A pesquisa no campo de emulação das capacidades visuais, através do uso de computadores, estende-se até os dias de hoje e por isto tornou-se uma área de aplicação que compreende áreas como automação industrial, robótica e processamento de documentos." (MOLZ, 2001).

Pautando-se nas informações apresentadas, o presente trabalho visa desenvolver um sistema que possa analisar objetos sobre uma esteira móvel utilizando-se visão computacional para remover objetos de qualidade questionável através de dispositivos mecânicos. A análise da imagem será feita usando o conceito SURF (*Speeded Up Robust Features*) que é um detector de características robustas inspirado no conceito SIFT (*Scale-Invariant Feature Transform*). Analogamente, poderá servir de plataforma para estudos posteriores pelos alunos do curso de Engenharia Mecatrônica otimizando o processo, ou aprimorando-o para outros fins. O sistema proposto deverá ser um sistema de baixo custo, de modo a viabilizar sua inserção no mercado.

Estudos no ramo de incorporação de diferentes tecnologias são importantes porque colocam em uso sua teoria e trazem muitos benefícios econômicos, além de poder abrir novas possibilidades de estudo. Incorporar novas tecnologias é um desafio mediante o desenvolvimento de técnicas inovadoras que preencham as lacunas existentes ainda não exploradas, pois permite a otimização de processos já existentes, bem como possibilita a obtenção de processos cada vez mais eficientes e rápidos.

Este trabalho está dividido em 6, capítulos cujo conteúdo é exposto a seguir, e, ainda, das considerações finais. Na Revisão da Literatura é feita uma revisão bibliográfica dos diversos temas que fundamentam o desenvolvimento de *software* de inspeção automatizada. Com a revisão são obtidas especificações para desenvolver o

sistema. Em Metodologia e Desenvolvimento serão abordadas a programação da interface GUI (*Graphic User Interface*), na qual são incorporadas funções de várias APIs (*Application Programming Interface*) encontrados em programas intermediários como o OpenCV e EmguCV. Após o desenvolvimento do programa “base” de obtenção e tratamento de imagens, será programada e incorporada uma função de comparação e análise de imagens. A outra parte deste capitulo consiste no desenvolvimento do arranjo eletrônico e mecânico de acionamento. Nos últimos capítulos, Testes Realizados e Resultados e Discussões, testes e correções são feitas após o programa estar finalizado, assim como a, discussão dos resultados obtidos em relação aos objetivos propostos.

1.2 OBJETIVOS

1.2.1 Objetivo Geral

Desenvolver um dispositivo de baixo custo que utiliza visão computacional para identificar peças defeituosas em uma linha de produção.

1.2.2 Objetivos Específicos

- Acoplar um sistema de visão computacional de baixo custo a uma esteira em uma linha de produção;
- Analisar imagens oriundas de uma *webcam* como controle de qualidade;
- Programar um PIC para receber informações via USB;
- Utilizar comunicação USB para controlar o rejeito de produtos defeituosos;
- Criar um programa em C# de fácil utilização para o usuário final;
- Projetar e construir um arranjo mecânico que retire da esteira peças defeituosas;
- Fornecer uma plataforma de estudos para futuros alunos do curso de Engenharia Mecatrônica.

Capítulo 2. Revisão de Literatura

2.1 LINGUAGENS DE PROGRAMAÇÃO

Uma linguagem de programação é uma linguagem artificial projetada para comunicar instruções a uma máquina, especialmente um computador. Linguagens de programação podem ser utilizadas para criar os programas que controlam o comportamento de uma máquina (AABY, 2004).

A descrição de uma linguagem de programação é geralmente dividida em dois componentes da sintaxe (forma) e semântica (significado). Alguns idiomas são definidos por um documento de especificação, como por exemplo, a linguagem de programação C é especificada por um padrão ISO. (ISO/IEC, 2011).

2.1.1 C++

A linguagem C++ é um tipo de linguagem de programação de uso geral considerado de nível intermediário por apresentar recursos de alto nível e baixo nível (SCHILDT, 1998). C++ é uma evolução da linguagem C introduzindo o POO (Programação Orientada a Objetos) com a utilização de classes. É uma das linguagens de programação mais comuns e é implementado em uma variedade de *hardwares* e sistemas operacionais (TIOBE, 2014). Suas aplicações incluem *software* de sistemas, *software* de aplicação, *drivers* de dispositivos, *software* embutido, servidor de alto desempenho, aplicações de cliente e *software* de entretenimento, como os videogames (STROUSTRUP, 2013). Existem vários grupos de programadores que fornecem compiladores gratuitos para C++ (*GNU Project*, LLVM, *Microsoft* e *Intel*). Outras linguagens de nível relativamente mais alto, como C# e Java, foram notavelmente influenciadas por C++ (NAUGLER, 2007).

2.1.2 C#

C# é uma linguagem de programação de uso geral desenvolvido pela *Microsoft* para ser executado em plataformas .NET. A intenção da *Microsoft* foi criar uma linguagem simples, moderna, orientada a objetos, e de uso geral (ECMA, 2012). A linguagem é utilizada no desenvolvimento de componentes de *software* adequados

para a implantação em ambientes distribuídos. Em relação à criação de GUI *(*Interface Gráfica do Usuário), C# tem uma ampla biblioteca de funções de fácil implementação, assim como, vários tutoriais no próprio site da *Microsoft.*

2.2 AMBIENTE INTEGRADO DE DESENVOLVIMENTO

O Ambiente Integrado de Desenvolvimento (IDE) é um programa de computador que reúne características e ferramentas de apoio ao desenvolvimento de *software* com o objetivo de agilizar este processo. Geralmente, os IDEs facilitam a técnica de RAD (*Rapid Application Development*), que visa a maior produtividade dos desenvolvedores. Um exemplo de um IDE é o *Microsoft Visual Studio 2012*.

2.2.1 *Microsoft Visual Studio*

O *Microsoft Visual Studio* é um pacote de programas da *Microsoft* para desenvolvimento de *software* especialmente dedicado ao .NET *Framework* e às linguagens *Visual Basic*, C, C++, C# e J#. Também é um grande produto de desenvolvimento na área *web*, usando a plataforma do ASP.NET. As linguagens com maior frequência nessa plataforma são: VB.NET e o C#" (KENNEDY, 2013).

O *software Visual Studio* inclui uma série de *designers* visuais para auxiliar no desenvolvimento de aplicações. Essas ferramentas incluem:

- *Windows Forms Designer*: o *designer* de *Windows Forms* é usado para construir aplicações GUI usando *Windows Forms*. O *layout* pode ser controlado, alojando os controles dentro de outros recipientes. Os controles que exibem dados (como caixa de texto, caixa de lista, exibição de grade, etc) podem ser ligado a fontes de dados, como bancos de dados ou consultas. Controles ligados a dados podem ser criados arrastando itens da janela *Data Sources* em uma superfície de design (*MICROSOFT*, 2013). A interface do usuário está relacionada com o código usando um modelo de programação orientada a eventos. O *designer* gera códigos do tipo C# ou VB.NET para a aplicação.
- WPF *Designer:* o *Designer WPF* é usado para criar interfaces de usuário segmentação *Windows Presentation Foundation.* Ele suporta todas as funcionalidades do WPF, incluindo a ligação de dados e gerenciamento de *layout* automático. Ele gera código XAML.

- *Web designer* / desenvolvimento*: Visual Studio* inclui um editor de web-*site* e *designer*, arrastando e soltando *widgets*. É usado para o desenvolvimento de aplicações ASP.NET e suporta HTML, CSS e JavaScript.
- *Designer* de Classe*:* o *Designer* de Classe é usado para criar e editar as classes (incluindo seus membros e seu acesso) utilizando modelagem UML. O *Designer* de Classe pode gerar código C# e VB.NET. Ele também pode gerar diagramas de classes.
- *Designer* de Dados*:* o *designer* de dados pode ser usado para editar graficamente esquemas de banco de dados, incluindo tabelas digitadas, chaves primárias e estrangeiras e restrições. Ele também pode ser usado para criar consultas de exibição gráfica.

2.3 VISÃO COMPUTACIONAL

A visão computacional é um campo que inclui métodos para a aquisição, processamento, análise e compreensão de imagens (e até dados em 3D do mundo real). A finalidade dessas análises e processamento é a obtenção de informações numéricas ou simbólicas, como por exemplo, na forma de decisões. (SHAPIRO, 2001). Esse “entendimento” de imagem pode ser visto como a extração de informações úteis a partir de dados de imagem usando modelos construídos com a ajuda da geometria, física, estatística e teoria da aprendizagem (FORSYTH, 2003) .

Um dos campos de aplicação mais utilizados de visão computacional é na indústria, às vezes chamada de *Machine Vision* (Sistema de Visão), onde a informação é extraída com a finalidade de apoiar um processo de fabricação. Um exemplo é o controle de qualidade onde os detalhes ou produtos finais estão sendo inspecionados automaticamente a fim de encontrar defeitos.

2.3.1 *Machine Vision*

Visão de Máquina (*Machine Vision*) é uma área de visão computacional que emprega uma variedade de tarefas de visão, como medição e processamento, que podem ser executadas usando uma variedade de métodos. É muito utilizada em processos agrícolas para remoção de materiais indesejáveis a partir de material a granel, como no processo chamado de triagem óptica. Alguns exemplos típicos de tarefas de visão por computador são apresentados abaixo.

- Reconhecimento: uma tarefa clássica de processamento de imagem e *Machine Vision* é a de determinar se os dados da imagem contém algum objeto específico;
- Análise de movimento: tarefa relacionada com a estimativa de movimento, onde uma sequência de imagens é processada para produzir uma estimativa da velocidade, em cada um dos pontos na imagem;
- Reconstrução de ambientes: dado algumas imagens de uma cena, ou um vídeo, a tarefa de reconstrução de ambientes visa computar um modelo 3D do ambiente. No caso mais simples, o modelo pode ser um conjunto de pontos em 3D. Métodos mais sofisticados produzem um modelo completo de superfície 3D.

2.3.2 Métodos de um sistema de visão computacional

A organização de um sistema de visão computacional depende muito da sua aplicação. Alguns sistemas são aplicações *stand-alone* que resolvem uma tarefa específica como a de detecção, enquanto outros constituem um subsistema de um projeto maior. Por exemplo, subsistemas para o controle de atuadores mecânicos.

Muitas funções são exclusivas para a aplicação. Existem, no entanto, funções típicas, que são encontradas em muitos sistemas de visão por computador e normalmente são executadas na seguinte ordem:

1. Aquisição
2. Pré-processamento
3. Extração de características
4. Processamento de alto nível
5. Tomada de decisão

- **Aquisição de imagens:** uma imagem digital é produzida por um ou vários sensores de imagem, que incluem vários tipos de câmeras sensíveis à luz, sensores de variação de aparelhos de tomografia, radar, câmeras ultrassônicos. Dependendo do tipo de sensor, o resultando de dados de imagem é uma imagem 2D comum, um volume 3D, ou uma sequência de imagens. Os valores de *pixel* normalmente correspondem à intensidade da luz em uma ou várias bandas espectrais (imagens de cinza ou imagens coloridas), mas também pode estar relacionado a várias medidas físicas, tais como a profundidade, absorção

ou reflexão de ondas sonoras ou ondas eletromagnéticas, ou ressonância magnética nuclear (DAVIES, 2005).

- **Pré-processamento:** para poder processar imagens em sistemas de visão computacional elas precisam ser pré-processadas. Este pré-processamento assume um importante papel para extração de certas informações de uma imagem. Por exemplo, uma imagem pode ter muitas informações não utilizadas para a análise, que podem influenciar de forma indesejada a decisão a ser tomada. Informações não utilizadas abrangem detalhes como cor, reflexos, e até objetos no fundo da imagem. Usando o pré-processamento adequado estas informações podem ser ignoradas ou até retiradas (ZUECH, 1988). Este procedimento tende a melhorar a tomada de decisão posterior à análise. De maneira geral o pré-processamento visa melhorar a qualidade da imagem para facilitar o processamento seguinte. Alguns métodos de pré-processamentos estão descritos abaixo.
- **Operação de realce:** o objetivo deste tipo de pré-processamento é destacar características importantes da imagem usando técnicas de contraste, destaque de contorno e suavização. Uma das técnicas de realce é ilustrado pela imagem da Figura 2.1 (técnica de contraste por histograma). O contraste por histograma é realizado através da definição de uma função de transferência que mapeia a faixa de valores de uma imagem original para novos valores radiométricos mais bem distribuídos num domínio de valores radiométricos pré-estabelecidos (INPE, 2007). Existem outras técnicas de realce, por exemplo, realce de imagens multiespectrais onde é possível realizar contrastes individuais das bandas de uma imagem multiespectral.

Figura 2.1 - Exemplo da aplicação do filtro realce sobre uma imagem de satélite (INPE, 2007).

- **Restauração de imagens:** esta operação de pré-processamento tem como objetivo deixar a imagem o mais próximo possível da cena real. Por exemplo, quando uma imagem é tirada por uma máquina digital, a imagem digitalizada pode apresentar ruídos indesejados que prejudicam alguns detalhes. Essa técnica pode ser aplicada também a imagens que apresentam ausência de foco ou movimento na imagem capturada.

As características de uma imagem em vários níveis de complexidade são extraídos a partir dos dados de imagem (DAVIES, 2005). Exemplos típicos de tais características são:

- **Segmentação**: um processo de separação de objetos (cada um com atributos uniformes) do resto da imagem. Esse processo cria partições da imagem baseado em uma variável, por exemplo, a intensidade da cor cinza. Existem vários tipos de segmentação. Nesta seção serão apresentados alguns dos processos mais utilizados (ZUECH, 1988).
- **Limiarização:** este processo de segmentação atribui a cor “branca” para cada *pixel* na imagem que tem o seu valor de cinza acima de um valor especificado e atribui a cor preta para os *pixels* que apresentam um valor de cinza abaixo deste. Este valor especificado se chama *threshold* (ou limiar) e é um valor de cinza de uma imagem monocromática. Áreas mais claras que o limiar tornam-se brancas e áreas mais escuras tornam-se pretas. A imagem resultante é chamada de imagem binária porque em vez de usar um *byte* para representar

cada *pixel*, é usado um *bit* com valor 0 se a região for preta e 1 se for branca. (ZUECH, 1988). Um exemplo de uma imagem segmentada usando limiarização pode ser observado na Figura 2.2.

Figura 2.2 - Exemplo de limiarização (INPE, 2007)

- **Detecção de bordas**: outra técnica de segmentação muito utilizada é a de detecção de bordas de um objeto. Nesse processo são utilizados filtros com métodos matemáticos que identificam pontos na imagem onde o seu brilho muda de forma drástica. Esses pontos são organizados em um conjunto de linhas e curvas. Segundo Gonzalez (2000), a detecção de bordas pode ser obtida usando filtros por derivadas, sendo o “Operador de Sobel” um desses filtros. Um exemplo do seu uso é mostrado na Figura 2.3. Nessa figura há também outros filtros que tem como objetivo detectar bordas.

Figura 2.3 **-** Detecção de Bordas: A) sem filtro, B) operador sobel, C) operador Roberts, D) operador Prewitt (WIKIPEDIA, 2013)

- **Processamento de alto nível:** nesta etapa a entrada é tipicamente um pequeno conjunto de dados, por exemplo, um conjunto de pontos ou uma região de imagem que se supõe conter um objeto específico (DAVIES, 2005). Esse tipo de processo garante que os dados satisfaçam pressupostos específicos e aplicação baseada em modelo; estima parâmetros específicos de aplicação, tais como orientação do objeto ou tamanho do objeto; reconhece imagens classificando um objeto e o detectado em diferentes categorias; permite o registro de imagens por comparar e combinar duas visões diferentes do mesmo objeto.
- **Tomada de decisão:** toma a decisão final necessária para a aplicação, como por exemplo, passar em testes de inspeção automáticos.

2.4 DETECTORES DE PONTO DE INTERESSE

A busca de correspondências em imagens discretas pode ser dividida em três etapas principais. Na primeira etapa, "pontos de interesse" são selecionados em locais distintos em uma imagem, como cantos, bolhas, e junções-T. A propriedade mais valiosa de um detector de pontos de interesse é a sua repetição, ou seja, encontrar de forma confiável os mesmos pontos de interesse em diferentes condições de visualização. Em seguida, a vizinhança de cada ponto de interesse é representada por um vetor característico (descritor). Este descritor tem que ser distinto e, ao mesmo tempo, robusto em relação a ruído e deformações geométricas. Finalmente, os vetores descritores são equiparados entre duas imagens diferentes. A correspondência é normalmente baseada na distância entre os vetores de descritores (BAY, 2008).

Existem vários detectores de pontos de interesse, o mais utilizado é o detector de cantos de Harris (HARRIS, 1988), mas o resultado desse detector varia em relação à escala e/ou distância do objeto. Lindbergh (1998) introduziu o conceito de detectores invariantes a escala, e com isso deu caminho para vários outros detectores (LOWE, 1999) (KADIR e BRADY, 2001).

Existem também diversos descritores: invariantes a momento (MINDRU, 2004), utilizando derivadas Gaussianas (HAAR, 1994), características complexas (BAUMBER, 2000), e SURF (BAY, 2008).

O conceito de pontos de interesse, também chamado de pontos chaves ou pontos característicos, é amplamente usado para resolver muitos problemas no reconhecimento de objetos, registro de imagens, controle visual, reconstrução 3D,

entre outros. Ele se baseia na ideia de que, em vez de olhar para a imagem como um todo, pode ser vantajoso selecionar alguns pontos especiais na imagem e realizar uma análise local sobre eles. Estas abordagens funcionam bem, desde que tenha uma quantidade suficiente de tais pontos nas imagens. Esses pontos são características estáveis que podem ser localizados com precisão (LAGANIÉRE, 2011).

2.4.2 SURF (*Speeded Up Robust Features*)

SURF é um algoritmo de visão computacional que descreve e detecta pontos chaves de uma imagem. Foi desenvolvido em 2006 por um grupo de pesquisadores e apresentado na 9ª Conferencia Europeia de Visão Computacional (BAY, 2008). Desde então, vem sendo muito utilizado no ramo de visão computacional pela sua rapidez em relação ao processamento e também sua invariância em relação à escala e orientação do objeto sendo analisada. O detector SURF realiza duas tarefas básicas, sendo elas: detecção de pontos chaves em uma imagem e descrição destes pontos.

Detectores de pontos chaves:

Pontos chaves são características consideradas importantes dentro de uma imagem. Por exemplo, cantos e arestas entre outros (Figura 2.4). Um computador consegue achar estes pontos analisando variações bruscas de intensidade de cor. Uma das primeiras tentativas para detectar estes pontos foi feito por Harris e Stephens (1988), onde analisaram o conceito de cantos utilizando uma perspectiva matemática.

Figura 2.4 – Exemplos de pontos de interesse representados em A, B, C, e D (OPENCV, 2014)

Harris e Stephens (1998) propuseram encontrar a diferença de intensidade em relação a um deslocamento, segundo a seguinte equação:

$$\mathrm{R} = \det(\mathrm{M}) - \mathrm{k}(\mathrm{trace}(\mathrm{M}))^2 \quad (1)$$

Onde,

$$\mathrm{M} = \sum_{\mathrm{x,y}} w(\mathrm{x},\mathrm{y}) \begin{bmatrix} \mathrm{I_x I_x} & \mathrm{I_x I_y} \\ \mathrm{I_x I_y} & \mathrm{I_y I_y} \end{bmatrix} \quad (2)$$

I_x e I_y são as derivadas das imagens nas direções X e Y e w(x,y) define o tamanho da área sendo analisada.

Encontrando os autovalores (λ_1, λ_2) para M e considerando:

$$\det(\mathrm{M}) = \lambda_1 \lambda_2 \quad (3)$$

$$\mathrm{trace}(\mathrm{M}) = \lambda_1 + \lambda_2 \quad (4)$$

Foram deduzidas as seguintes regras:

1. Quando $|R|$ é pequeno, o que acontece quando λ_1 e λ_2 são pequenos, a região é plana.
2. Quando R < 0, o que acontece quando $\lambda_1 >> \lambda_2$ ou vice-versa, a região é de uma aresta.
3. Quando R é grande, o que acontece quando λ_1 e λ_2 são grandes e $\lambda_1 \sim \lambda_2$, a região é um canto.

O detector de Harris e Stephens provou-se muito útil e eficiente, mas tinha uma desvantagem, não era invariante à escala. Por exemplo, um canto pode não parecer um canto a uma escala maior. Para tentar superar esta desvantagem, Lowe (2004), criou outro algoritmo de detecção de pontos chaves, SIFT. Este algoritmo detecta pontos chaves e já executa a próxima tarefa (descrita na próxima seção) de descrever estes pontos.

Para aumentar a precisão em achar pontos chaves, o SIFT utiliza um conceito chamado filtragem de escala-espaço. Neste conceito o LoG (Laplaciano de Gauss) é

encontrado de uma imagem com vários σ. O LoG atua como um detector de partículas de vários tamanhos (o tamanho da partícula depende do σ). O σ pode ser considerado um parâmetro de escala. O algoritmo retorna então um vetor de pontos chaves potenciais (x, y, σ). Uma vez que estes pontos de chaves potenciais são achados, eles são passados mais uma vez por um processo de filtragem utilizando uma expansão de Taylor. Mesmo com uma precisão aumentada, mais uma vez apresentou uma desvantagem, o LoG não podia ser calculado diretamente, e então, utilizava um método numérico chamado Diferença de Gaussiano. Tal método é considerado computacionalmente pesado. Então, para tentar melhorar a velocidade em que estes pontos são computados, foi criado o algoritmo SURF.

O algoritmo SURF foi criado visando aumentar a velocidade em achar pontos chaves sem perder a precisão conquistada anteriormente pelo SIFT. Isso foi feito pelos criadores usando duas novas técnicas: imagens integrais (Figura 2.5) e uma outra aproximação para LoG denominado *Box Filtering*.

Imagens integrais realçam pontos de interesse, pois realçam a diferença de intensidade de cores nos *pixels*. Este método é demonstrado pela Equação (5) e através da figura 2.5, como se segue.

$$s(x,y) = i(x,y) + s(x-1,y) + s(x,y-1) - s(x-1,y-1) \quad (5)$$

5	2	3	4	1
1	5	4	2	3
2	2	1	3	4
3	5	6	4	5
4	1	3	2	6

Imagem Original

0	0	0	0	0	0
0	5	7	10	14	15
0	6	13	20	26	30
0	8	17	25	34	42
0	11	25	39	52	65
0	15	30	47	62	81

Imagem Integral

Figura 2.5 **-** Exemplo de imagem integral

A vantagem de se usar o método de LoG, mostrada na Equação (6), é que pode ser calculado para diferentes escalas em paralelo e de forma independente, em contraste com a Diferença de Gaussiano que dependia de uma escala anterior para o cálculo da próxima. O algoritmo SURF mostrou-se rápido e preciso em achar pontos

chaves e, por isso, é um dos detectores mais utilizados. A utilização da Equação (6) em um ponto chave é mostrada na Figura 2.6.

$$LoG(x,y) = -\frac{1}{\pi\sigma^4}\left[1 - \frac{x^2+y^2}{2\sigma^2}\right]e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (6)$$

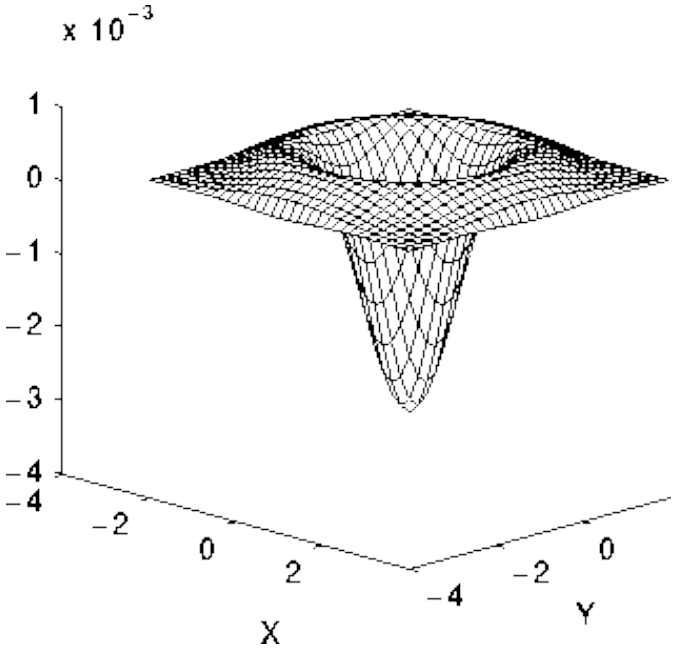


Figura 2.6 - Exemplo de utilização de LoG em um ponto chave

Descritores de pontos chaves

Na extração de pontos chaves, obtém-se informações sobre a sua posição, e a sua área de cobertura (geralmente aproximada por um círculo ou elipse) na imagem. Embora as informações sobre a posição de pontos chaves podem ser úteis, não diz muito sobre os próprios pontos chaves. Para isso é preciso definir o quão diferente ou semelhante é um ponto chave para o outro, utilizando-se descritores. Estes descritores resumem em formato vetorial, de comprimento constante, algumas características sobre os pontos chaves, como por exemplo, a sua intensidade em direções especificas.

Para se obter os descritores no algoritmo SURF, as características a serem descritas para definir o quão diferente é um ponto chave de outro são as variações de intensidade de cor em 8 direções. Para isso uma vizinhança de tamanho 16x16 é definida ao redor do ponto chave considerado. Esta vizinhança é dividida em 16 sub-blocos de tamanho 4x4. Para cada sub-bloco, 8 vetores em orientações distintas são criados onde são atribuídos os valores de variação de intensidade da cor em relação à

direção (Figura 2.7). Estes vetores direcionais são representados por um vetor numérico de 64 posições para formar o descritor do ponto chave.

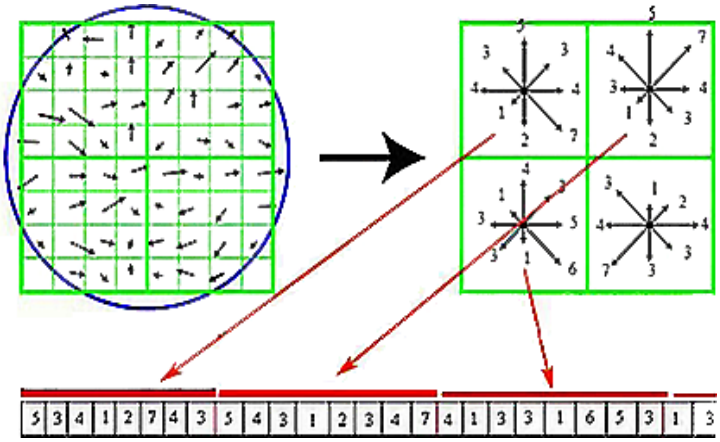


Figura 2.7 – Exemplo de vizinhança de ponto chave e geração de vetor numérico (OPENCV, 2014).

2.5 BIBLIOTECAS OPENCV E EMGUCV

2.5.1 OpenCV

OpenCV (*Open Source Computer Vision Library*) é uma biblioteca livre ao uso acadêmico e comercial, para o desenvolvimento de aplicativos na área de visão computacional (OPENCV, 2013). Essa biblioteca possui mais de 2500 algoritmos otimizados, desde os mais simples até os mais modernos, tais como os de *Machine-Learning* (Aprendizagem de Máquina).

As utilidades do seu uso são vastas. O OpenCV é utilizado no desenvolvimento de diversos aplicativos desde programas simples de colagem de imagens até programas mais complexos como os de auxilio de navegação robótica. Ele é disponibilizado em versões como C++/C/Python e Java, e pode ser usado em múltiplos sistemas operacionais como *Windows, Linux, Android* e *MacOS.*

Delai (2012), faz uma afirmação da importância do uso de OpenCV nos trabalhos acadêmicos:

> ”A gratuidade da OpenCV, o baixo custo de maquinas com capacidade de processamento cada vez maiores e a crescente qualidade das câmeras torna possível o desenvolvimento de sistemas sofisticados de visão com baixo investimento e custo de operação. A utilização de câmeras pode

inclusive substituir outros sensores e sistemas mais caros, complexos e menos genéricos.”

Dependendo da versão, o OpenCV pode variar em relação a sua estrutura, mas a maioria das versões são similares e baseadas em módulos (OPENCV, 2013). Os módulos básicos são:

- ***Core* (núcleo)**: inclui definições de estruturas de dados básicos, como arranjos multidimensionais (MAT- matrizes) e funções básicas usadas por todos os outros módulos.
- **Imgproc**: um módulo de processamento de imagem, que inclui filtragem de imagens, transformações geométricas, conversão de espaço de cores, histogramas, entre outros .
- **Vídeo:** um módulo de análise de vídeo, que inclui a estimativa de movimento, subtração de fundo, e os algoritmos de rastreamento de objeto.
- **Calib3d: a**lgoritmos geométricos básicos de vistas múltiplas, calibração de câmera estéreo e elementos de reconstrução 3D.
- **Features2d**: descritores de imagens.
- **Objdetect:** detecção de objetos e instâncias das classes pré-definidas.
- **HighGUI:** uma interface para captura de vídeo e imagem.

2.5.2 EmguCV

EmguCV tem como função principal adaptar o código na biblioteca do OpenCV para que possa ser chamado de plataformas e linguagens compatíveis com .NET como C#, VB, VC++, IronPython, entre outros (EMGUCV, 2013). A versão mais recente é o EmguCV 2.9. A estrutura deste *wrapper* (empacotador) pode ser observada na Figura 2.8.

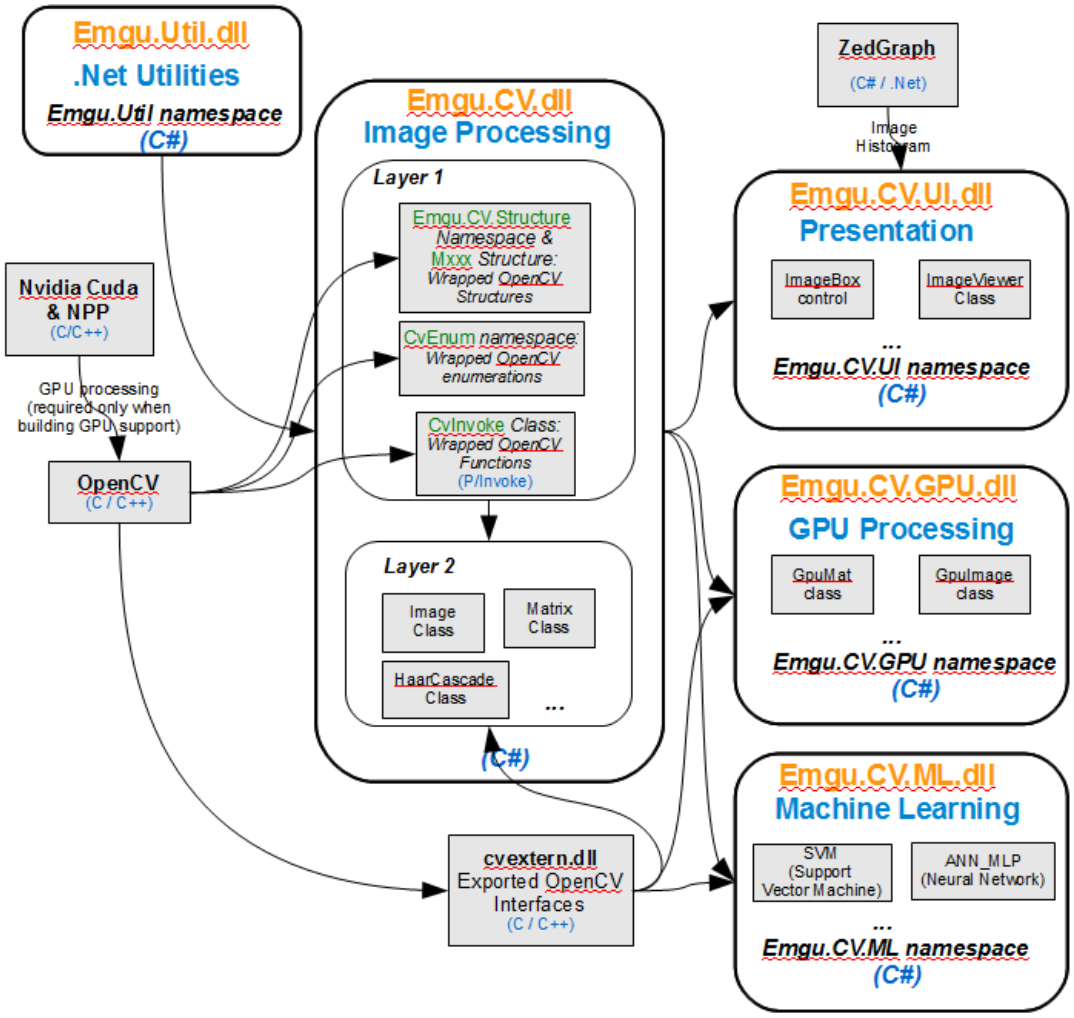


Figura 2.8 **-** Camadas de Empacotamento do EmguCV (EMGUCV, 2013)

A estrutura de empacotamento do EmguCV pode ser dividida em duas camadas básicas: a primeira contém o mapeamento das funções, estruturas, e enumeração do OpenCV; e a segunda contém classes que utilizam as vantagens da estrutura .NET. Por exemplo, pode ser compilado em “mono”, que é uma implementação do CLI, e, assim, pode ser executado em qualquer sistema operacional (EMGUCV, 2013).

2.6 PIC

Os PICs são uma família de microcontroladores fabricados pela *Microchip Technology*, que processam dados de 8 *bits*, de 16 *bits* e, mais recentemente, de 32 *bits* (MICROCHIP, 2013). Seu nome é oriundo de *Programmable Interface Controller* (Controlador de Interface Programável). Contam com extensa variedade de modelos e periféricos internos. Possuem alta velocidade de processamento devido a sua arquitetura *Harvard* e conjunto de instruções *RISC*, com recursos de programação por

memória *flash* e EEPROM. Os microcontroladores PIC têm famílias com núcleos de processamento de 12 *bits*, 14 *bits* e 16 *bits*, e trabalham em velocidades de 0kHz (ou DC) a 48MHz e velocidades de 16 MIPS em alguns modelos. Há o reconhecimento de interrupções tanto externas como de periféricos internos. Funcionam com tensões de alimentação de 1.8 a 6V e os modelos possuem encapsulamento de 6 a 100 pinos.

Os processadores PIC apenas aceitam linguagem de máquina (*assembly*) para sua programação. No entanto a programação pode ser feita em linguagens de alto nível utilizando-se compiladores. Existem ainda diversas IDEs disponíveis para a programação, entre elas a mais utilizada é o *MPLAB,* disponibilizado de modo gratuito pela própria *Microchip*.

Para gravar o programa no microcontrolador é utilizado um dispositivo dedicado. É comum que tais dispositivos também possuam capacidade de “debuggar” o programa, que auxilia muito na fase de testes dos sistemas. É possível ainda encontrar diversas placas de desenvolvimento que já possuem um *hardware* pré-montado de modo a agilizar o projeto de um produto, permitindo que se comece a desenvolver o *software* em paralelo com o *hardware*.

2.6.1 PIC 18F4550

O PIC18F4550 é um microcontrolador de 8 *bits* com arquitetura *Harvard* e conjunto de instruções tipo RISC, ele possui uma memória interna de 32 KB para armazenamento do programa residente e 2048 *bytes* de memória RAM. Sua tensão de alimentação pode ser de 4 a 5,5 Volts e sua frequência de operação é de até 48MHz, a esta frequência ele é capaz de executar até 12 milhões de instruções por segundo (MICROCHIP, 2013).

O PIC 18F4550 possui 40 pinos dos quais 35 podem ser configurados como portas I/O e diversos periféricos tais como memória EEPROM de 256 bytes, um módulo CCP e ECCP um módulo SPI e um módulo I2C. Possui também 13 conversores A/D com 10 *bits* de resolução cada e tempo de amostragem programável, 02 comparadores analógicos, uma comunicação EUSART, um *timer* de 8 *bits* e três *timers* de 16 *bits* cada, um módulo de detecção de tensão alta/baixa (HLVD) e um módulo USB 2.0 com a capacidade de operar nos modos *low-speed* (1,5Mbps) ou *full-speed* (12Mbps) (MICROCHIP, 2013).

2.7 COMUNICAÇÃO VIA USB COM O PIC 18F4550

Para ligar um dispositivo USB em um computador primeiro é preciso escrever e compilar um *firmware* para o PIC, nesse caso o PIC 18F4550 (Figura 2.9). A *Microchip,* fabricante de microcontroladores, fornece uma biblioteca que possibilita comunicação USB que pode ser utilizada, modificada e redistribuída justamente para este fim (MICROCHIP, 2013). O *firmware* tem de executar duas tarefas importantes: enumeração de dispositivos e a comunicação com o *host*. Nos tópicos seguintes encontra-se as descrições destas tarefas.

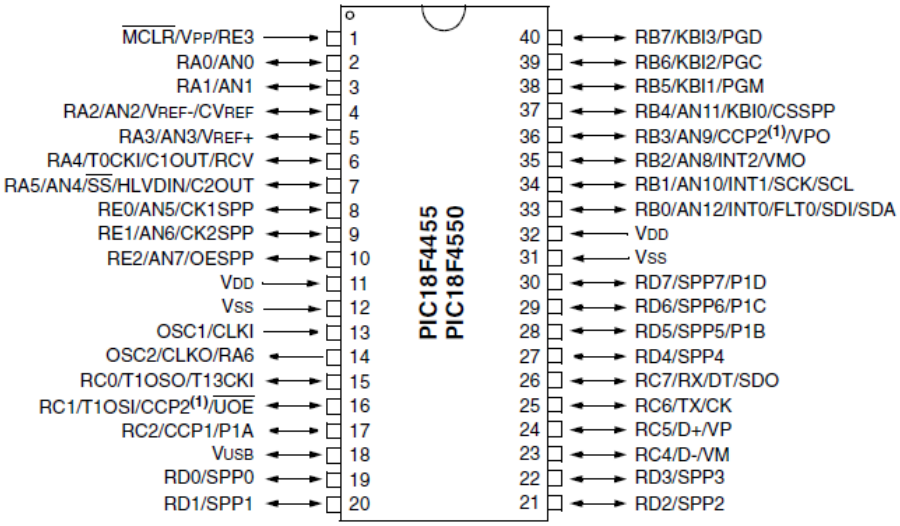


Figura 2.9 **–** Diagrama de pinagens do PIC 18F4550 (MICROCHIP, 2013)

2.7.1 Enumeração de dispositivos

A enumeração do dispositivo USB é a comunicação inicial com o *host* do USB (nesse caso o computador), quando o dispositivo informa ao *host* o que é e como ele deseja comunicar. A comunicação USB é feita usando *endpoints*, que enviam informações tanto para o *host* ou ao dispositivo. Assim como a criação de canais de comunicação, o dispositivo deve passar também o seu nome e outros dois valores importantes: o VID e PID.

O VID é a identificação do fornecedor e identifica o fabricante do dispositivo. A obtenção de um VID particular é altamente dispendiosa. Como um dos objetivos desse projeto é ser de baixo custo, foi usado o VID da *Microchip* para conter as despesa.

O PID é o ID do produto. Junto com a VID eles formam um identificador exclusivo para o dispositivo. Quando o dispositivo é enumerado pela primeira vez, o *Windows* irá armazenar a combinação VID e PID do dispositivo.

2.7.2 Comunicação com o *host*

A segunda tarefa importante que o *firmware* realiza é a comunicação real entre o *host* e o dispositivo. Cada evento de comunicação é identificado por um "comando". Ao usar o padrão HID genérico (que será utilizado nesse caso), o "comando" comunica ao *host* e ao dispositivo como interpretar a informação que é passada com o comando. Essa informação pode ser qualquer sinal (por isso o nome "genérico"). Uma vez que o dispositivo é enumerado, o *host* irá pesquisar periodicamente. Em cada pesquisa, o *host* pode tanto enviar um comando de enviar dados para o dispositivo, bem como um comando de receber dados a partir do dispositivo.

Capitulo 3. Metodologia e Desenvolvimento

3.1 ESPECIFICAÇÕES DO PROJETO

Antes de desenvolver o protótipo, foi necessário definir parâmetros e especificações para se poder prosseguir com o projeto. Essas especificações são enumeradas a seguir.

1. Analisar imagens de produtos capturadas por uma câmera de vídeo, com o objetivo de rejeitar aqueles que possuam defeitos;
2. A inspeção deve ser baseada na semelhança do produto sendo analisado com o produto padrão;
3. Possuir a opção de inserir novos perfis de produtos de modo a ter um maior espectro de reconhecimento de objetos sem a necessidade de uma nova implementação;
4. Realizar as análises em tempo real;
5. Interagir com um sistema de descarte, de maneira a retirar um produto defeituoso imediatamente após a detecção da imperfeição;
6. Ser de baixo custo;
7. Servir de plataforma para futuros estudos pelos alunos de Engenharia Mecatrônica.

Com a definição dessas especificações foi iniciado o desenvolvimento do protótipo visando cumprir essas propostas.

3.2 RECURSOS UTILIZADOS

3.2.1 *Softwares* e bibliotecas

Softwares: MPLAB® X (IDE e XC8 Compiler v1.30) *; Microsoft Visual Studio Express* 2012

Bibliotecas: OpenCV ; EmguCV *; usbGenericHidCommunications*

3.2.2 Hardware

Notebook HP PAVILION DV6 (2.2 GHz, Intel Core i7-2670Q,; 8GB); motor de vidro elétrico MABUCHI, 8 Dentes, 12V; *webcam* com resolução QVGA (340x240); motor de passo unipolar, 5V, 1.8 DEG/STEP, 75 ohm; PIC 18F4550; componentes eletroeletrônicos (resistores, capacitores, etc.).

3.3 ESCOLHA DAS LINGUAGENS DE PROGRAMAÇÃO

Foi realizada a escolha de duas linguagens computacionais, C e C#, para o desenvolvimento do programa principal e a programação do PIC, as quais são descritas a seguir.

3.3.1 Linguagem do programa principal

Inicialmente, a linguagem de programação C++ foi escolhida devido ao seu amplo uso no meio acadêmico (STROUSTRUP,1994). No entanto, essa linguagem é de um nível relativamente baixo para desenvolver programas do tipo GUI, de modo que uma substituição para a linguagem C# foi necessária para melhor se adequar a proposta deste trabalho. Além disso, há diversos dados bibliográficos ligados à visão computacional utilizando a linguagem C#, o que pode auxiliar indivíduos com pouco conhecimento na área que venham utilizar esta plataforma em estudos futuros.

3.3.2 Linguagem de programação do PIC

Como foi destacado na revisão de literatura, os PICs aceitam apenas a linguagem de máquina (*Assembly*) para sua programação. No entanto, a programação de um PIC para comunicação via USB totalmente em *Assembly* é uma tarefa extensa e desnecessária, uma vez que podem ser utilizadas linguagens de alto nível (com seus respectivos compiladores), que necessitam de menos linhas de código para realizar as mesmas funções. Com tudo, devido à facilidade de programação assim como uma gama de bibliotecas, foi escolhido a linguagem C para a programação do PIC 18F4550.

3.4 ESCOLHA DOS *SOFTWARES*

Após escolher as linguagens que seriam utilizadas, foram escolhidos os *softwares* para o desenvolvimento dos programas.

3.4.1 Software de desenvolvimento do programa principal

O programa de desenvolvimento escolhido foi o *MS Visual Studio Express*, por ser um ambiente de desenvolvimento integrado gratuito, onde pode ser utilizada a licença GNU (*General Public License*) (EMERSON, 2013). Essa característica garante ao usuário direitos de usar, estudar, e modificar o *software* para qualquer fim, desde que as modificações feitas sejam acessíveis para outros usuários (STALLMAN, 2011). Este programa pode ser adquirido pela *internet* com muita facilidade e possibilita o desenvolvimento de uma interface GUI, conforme previsto no protótipo final.

3.4.2 *Software* de desenvolvimento do programa do PIC

O programa de desenvolvimento escolhido foi o MPLAB® X (IDE), por ser um ambiente de desenvolvimento integrado gratuito. Este programa é adquirido facilmente pela *internet* e permite o desenvolvimento de programas para o PIC utilizando a linguagem C e o compilador MPLAB® XC8 Compiler v1.30, também adquirido no mesmo site, gratuitamente.

3.5 ESCOLHA DA BIBLIOTECA DE VISÃO COMPUTACIONAL

A biblioteca de visão computacional EmguCV foi escolhida por ser a única gratuita que possa ser utilizada em conjunto com *Microsoft Visual Studio 2012* e também por ter um vasto acervo de tutoriais na *internet* que facilitaram a execução deste trabalho.

3.6 ESCOLHA DA BIBLIOTECA PARA PROGRAMÇÃO DO PIC

Durante a revisão bibliográfica encontrou-se uma biblioteca, com a licença GNU, de um *driver* genérico para dispositivos que utilizavam comunicação USB. Essa

biblioteca é disponibilizada pela *Microchip* com vários exemplos que podem ser modificados para fins acadêmicos.

3.7 DESENVOLVIMENTO DO *SOFTWARE* PRINCIPAL

Antes de iniciar-se o desenvolvimento do programa principal, foi realizado um esboço do fluxo de informações para definir-se a base estrutural do programa. A Figura 3.1 apresenta esse fluxo de informações de modo simplificado.

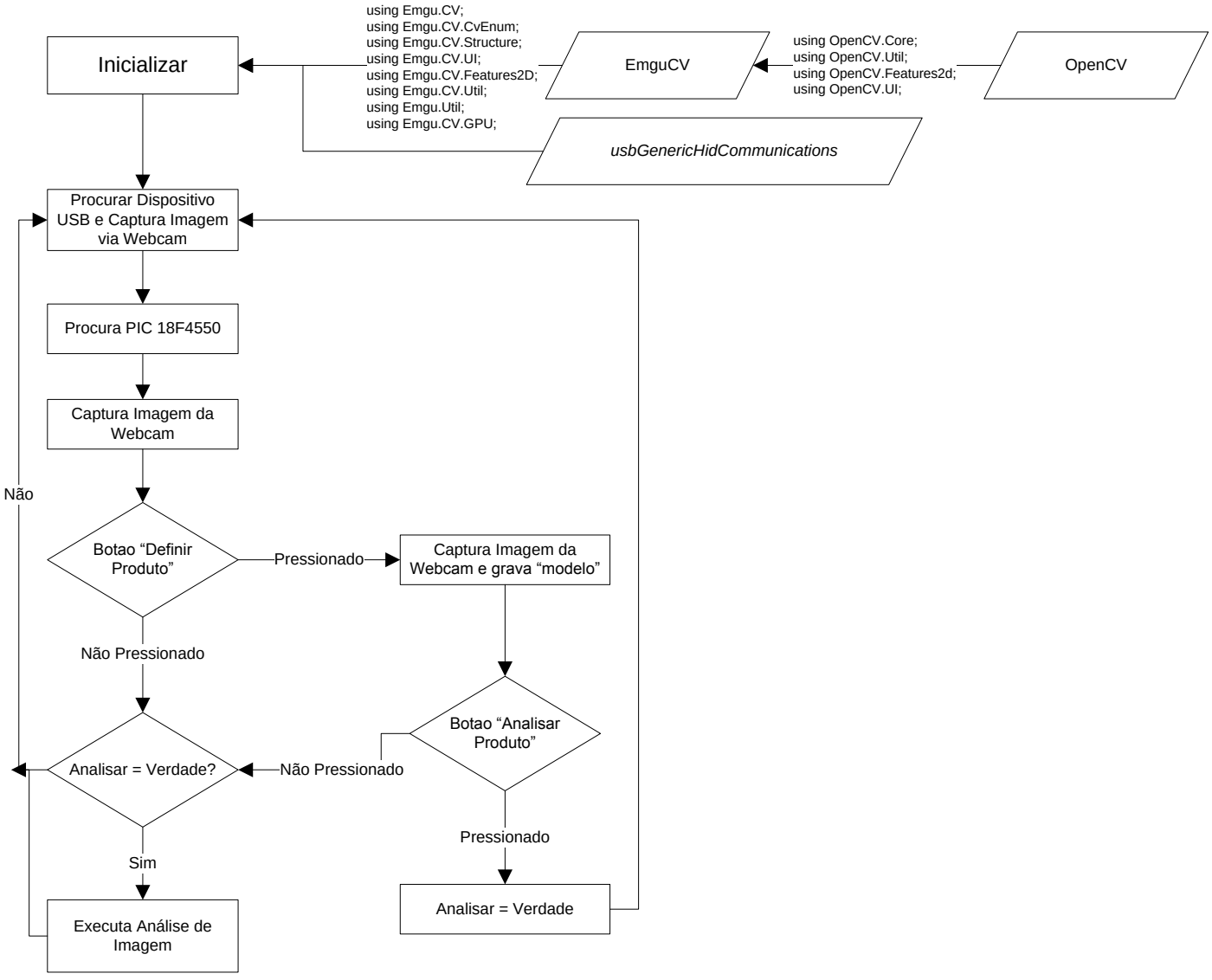


Figura 3.1 - Fluxograma de Informações do programa principal

Analisando o fluxo de informações, notou-se que existem duas tarefas básicas: a de análise de imagens e a de comunicação com o PIC via USB. Essas duas tarefas requerem funções e variáveis específicas. Estas funções podem ser utilizadas mais de uma vez com as definições de variáveis diferentes. Deste modo, foram criadas duas classes: *DrawMatches* para a análise de imagens e *USBReferenceDevice* para a comunicação via USB.

3.7.1 Análise de Imagens

O funcionamento do programa principal é baseado em duas funções, a análise de imagem e comunicação com o PIC. Nesta seção serão descritos os métodos usados para a análise de imagens.

Para a análise de imagens é necessário que estas sejam obtidas previamente por uma *webcam*, sendo imprescindível a criação de uma conexão com o dispositivo utilizado. Para mostrar a imagem sendo capturada pela *webcam* é preciso criar um objeto na janela principal chamado de *ImageBox* que é disponibilizado no DLL do EmguCV. Com esse objeto é possível atribuir a imagem com o seguinte código:

```
1  public Capture capWebcam = null;
2  public Image<Bgr, Byte> imgWebcam;
3  public Image<Bgr, Byte> imgModelo;
4  capWebcam = new Capture(0);
5  imgWebcam = capWebcam.QueryFrame();
6  imgModelo = capWebcam.QueryFrame();
7  ImageBoxWebCam.Image = imgWebcam;
8  ImageBoxModelo.Image = imgModelo;
```

Este código permite mostrar uma imagem capturada pela *webcam*, mas se limita por não mostrar um fluxo contínuo de imagens advindas desta câmera (vídeo). Para superar esta limitação é necessário englobar esses comandos em uma função, e colocar essa função nas tarefas a serem realizadas no tempo ocioso do programa. O código a seguir mostra como isso foi feito:

```
1  Application.Idle += processFrameAndUpdateGUI;
2
3  void processFrameAndUpdateGUI(object sender, EventArgs arg)
4      {
5          imgWebcam = capWebcam.QueryFrame();
6          ImageBox.Image = imgWebcam;
7      }
```

Com os códigos acima o programa principal mostrará um vídeo oriundo da *webcam* e outra imagem estática, podendo ser a imagem padrão (ou modelo) a ser analisado. A captura das imagens, portanto, permitirá que sua análise seja realizada.

Para realizar essa análise foi utilizado o algoritmo SURF, disponibilizado na biblioteca EmguCV, para detectar os pontos chaves e computar os descritores. Para utilizar esse algoritmo é preciso primeiramente definir algumas variáveis importantes:

```
1  SURFDetector surfCPU ;                          \\ Detector Surf Iniciado;
2  Image<Gray,Byte> imgWebcamGray ;                \\ Imagem Cinza para analisar;
3  Image<Gray,Byte> imgModelGray ;                 \\ Imagem Cinza Modelo;
4  modelKeyPoints = new VectorOfKeyPoint();        \\ Vetor de pontos chaves do Modelo;
5  observedKeyPoints = new VectorOfKeyPoint();  \\ Vetor de pontos chaves da imagem da Cam;
6  Matrix<float> modelDescriptors ;                \\ Matriz de descritores do Modelo;
7  Matrix<float> observedDescriptors ;             \\ Matriz de descritores da imagem da Cam;
```

Com essas variáveis, pode ser realizado o cálculo dos descritores utilizando o código seguinte:

```
1  surfCPU = new SURFDetector(300, false);
2  imgWebcamGray = imgWebcam.Convert<Gray,Byte>();
3  imgModelGray = imgModel.Convert<Gray,Byte>();
4  modelDescriptors = surfCPU.DetectAndCompute(modelImageGray, null, modelKeyPoints);
5  observedDescriptors = surfCPU.DetectAndCompute(observedImageGray, null, observedKeyPoints);
```

As matrizes de descritores possibilitaram a realização de uma análise de semelhança usando um objeto denominado *BruteForceMatcher*, que encontra a semelhança entre cada elemento da matriz. Depois de encontradas as semelhanças, utiliza-se uma função da biblioteca *Features2DToolbox*, que usa a constante

uniquenessThreshold (limiar de singularidades) para delimitar as semelhanças significativas.

```
1 BruteForceMatcher<float> matcher ;
2 matcher.Add(modelDescriptors);
3 matcher.KnnMatch(observedDescriptors, indices, dist, k, null);
4 mask = new Matrix<byte>(dist.Rows, 1);
5 mask.SetValue(255);
6 Features2DToolbox.VoteForUniqueness(dist, uniquenessThreshold, mask);
```

Após delimitadas as semelhanças significativas, é utilizada uma função da mesma biblioteca para gerar uma matriz de homografia, que descreve a orientação do objeto encontrado.

```
1 homography = Features2DToolbox.GetHomographyMatrixFromMatchedFeatures(modelKeyPoints,
  observedKeyPoints, indices, mask, 2);
```

Com as informações obtidas é possível gerar uma imagem utilizando a função *Features2DToolbox.DrawMatches*, que demonstra os pontos chaves do objeto por meio de círculos e o identifica com um retângulo ao seu redor, concluindo assim a análise de imagem.

3.7.2 Comunicação USB

Para realizar a comunicação do programa com o dispositivo de descarte foi utilizada a porta USB. Este processo foi realizado com ajuda de uma biblioteca genérica de USB obtida *online* (MICROCHIP, 2014). Essa biblioteca usufrui da licença GNU que permite sua utilização para fins acadêmicos e comerciais, contanto que o seu código seja disponibilizado para o usuário final.

Quando deseja-se abrir uma conexão com o dispositivo USB é preciso primeiramente criar um objeto de referência com os dados de identificação do vendedor (VID) e do produto (PID). Esses dados são atribuídos ao PIC via código e podem ser alterados. Nesse caso esses valores foram 0x04D8 e 0x0080, sendo VID e PID respectivamente. Este processo foi realizado da seguinte forma:

```
1  theReferenceUsbDevice = new usbReferenceDevice(0x04D8, 0x0080);
2  theReferenceUsbDevice.usbEvent += new usbReferenceDevice.usbEventsHandler(usbEvent_receiver);
```

Com o código acima é criado o objeto de referência. Posteriormente (linha 2) uma condição é elaborada para tratar eventos de comunicação, entre o dispositivo e o computador. Nesta condição, uma função *usbEvent_receiver* é chamado para verificar os dados desta comunicação, identificando se estes advém do PIC referenciado. Dentro do objeto *usbReferenceDevice* criou-se funções que enviam informações (2 *Bytes = 16 bits*) para o dispositivo que as interpreta.

3.7.3 Estrutura final

A estrutura final do programa pode ser visualizada na Figura 3.2. Nota-se que existem bibliotecas externas (*Externals*) que devem estar na mesma pasta que o programa principal.

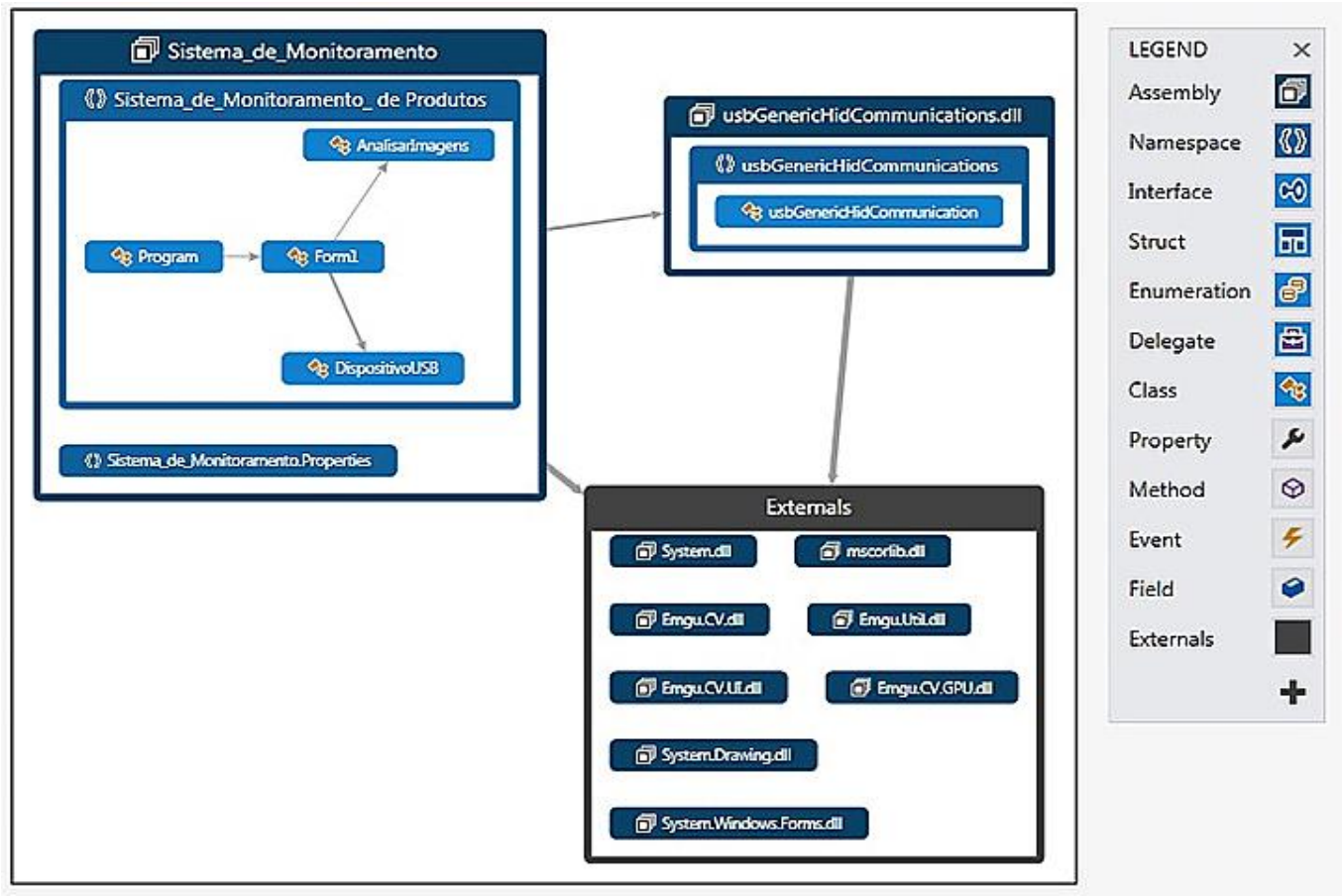


Figura 3.2 - Estrutura Final do Programa

3.8 DESENVOLVIMENTO DO PROGRAMA PARA O PIC

Como mencionado na Seção 2.7, uma biblioteca foi utilizada para facilitar a comunicação entre o computador e o dispositivo. Uma vantagem do uso dessa biblioteca é que o dispositivo emprega dados de um *driver* de USB HID genérico que pode ser reconhecido em qualquer computador.

3.8.1 Criando o programa principal

A biblioteca utilizada no programa principal consiste em dois arquivos que são o *HardwareProfile.h* e o *debug.h*. O primeiro atribui nomes mais usuais para portas do PIC assim como algumas funções básicas, como por exemplo:

```
1  #define mStatusBobina0          LATDbits.LATD0
2  #define mStatusBobina1          LATDbits.LATD1
3  #define mStatusBobina0_on()     mStatusBobina0 = 1;
4  #define mStatusBobina1_on()     mStatusBobina1 = 1;
```

O segundo arquivo seta o tamanho do pacote a ser passado pela fase de *debug* pelo dispositivo (128 ou 1 *byte*).

O código que executa os comandos principais, como o de ligar o motor de passo, está localizado no arquivo source “main.c”, na função *processUsbCommands*. Esta função é responsável por determinar o comando sendo recebido e enviar ou receber dados, conforme apropriado. Este processo é bastante simples, uma vez que a biblioteca USB cuida de toda a complexidade de comunicação de baixo nível. Um exemplo desse código é mostrado a seguir.

```
1  if(!HIDRxHandleBusy(USBOutHandle))
2  {
3          switch(ReceivedDataBuffer[0])
4          {
5                  case 0x80:  // Energizar Bobina 0
6          sprintf(debugString, " Bobina 0 Energizado");
7                  debugOut(debugString);
8                  mStatusBobina0_Toggle();
9          break; ...
```

A função *Main* (corpo principal do programa) simplesmente "chama" a biblioteca USB para executar todas as tarefas de dispositivos de baixo nível e, em seguida, executa a função *processUsbCommand* mais uma vez.

```
1  void main(void)
2  {
3          // Inicializa o PIC “setando” as portas TRISA TRISB....
4     initialisePic();
5          // Se o dispositivo foi removido  e o dipositivo estiver em modo Suspenso
6         // tenta conectar de novo
7     #if defined(USB_INTERRUPT)
8        USBDeviceAttach();
9     #endif
10         // Inicializa as funcoes de debug
11    debugInitialise();
12       // Ligar o LED de Energia
13    mStatusLEDPower_on();
14         // Loop de processamento principal
15    while(1)
16    {
17       #if defined(USB_POLLING)
18                          //Executa Tarefas de baixo nível do USB
19                USBDeviceTasks();
20       #endif
21
22          // Processa Commandos USB
23       processUsbCommands();
24        }
25 }
```

3.9 DESENVOLVIMENTO DO DISPOSITIVO MECÂNICO PARA REJEITAR OBJETOS

O dispositivo mecânico de destinado a rejeitar objetos indesejáveis rejeito foi desenvolvido para ser acoplado a uma esteira já existente fornecida pelo Laboratório de Robótica do CEFET-MG, *Campus Divinópolis*. Para isso, foi necessário dimensionar o dispositivo levando em consideração características da esteira (Figura 3.3) como comprimento e largura. As dimensões físicas da esteira podem ser consultadas no Anexo IV. Uma estrutura de madeira foi confeccionada para sustentar a câmera sobre a esteira e suas especificações podem ser consultadas no Anexo I.

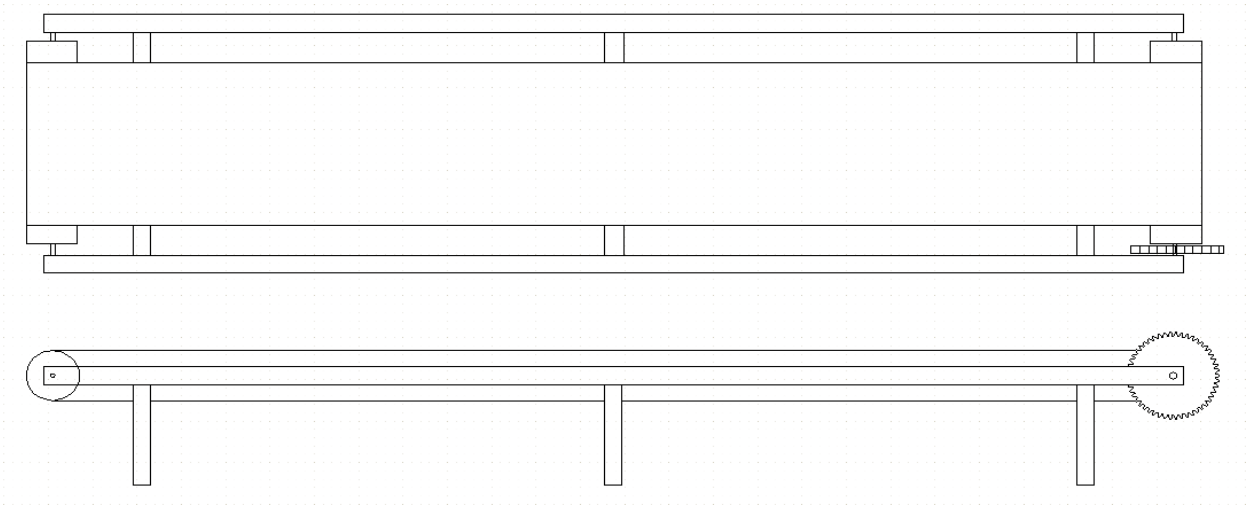


Figura 3.3 - Modelo da esteira utilizada

Acoplado à esteira encontra-se um motor Mabuchi JC/LC-578VA (Figura 3.4) com a função de movimentá-la durante o processo de análise dos objetos. Este motor será controlado pelo programa principal com a utilização de um relé e uma fonte de computador externa.

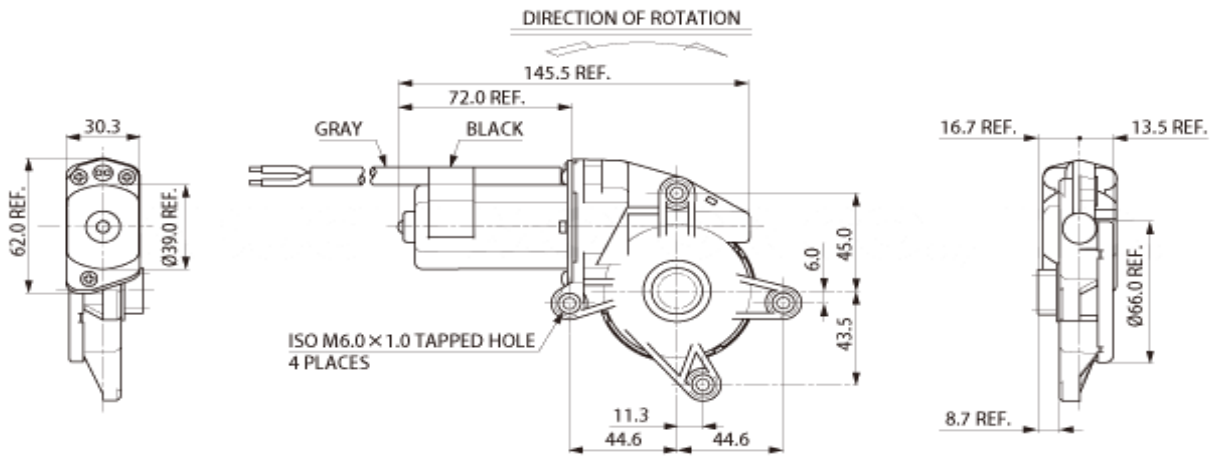


Figura 3.4 - Representação ilustrativa do motor acoplado à esteira (MABUCHI, 2014)

3.9.1 Dispositivo de Rejeito

Para o desenvolvimento do dispositivo de rejeito, parafusou-se um motor de passo a uma estrutura de alumínio com dimensões compatíveis com a estrutura lateral da esteira (Figura 3.5). Este mecanismo foi acoplado lateralmente à estrutura da esteira devido ao seu formato. As dimensões dos elementos que compões o dispositivo podem ser consultadas nos anexos II e III.

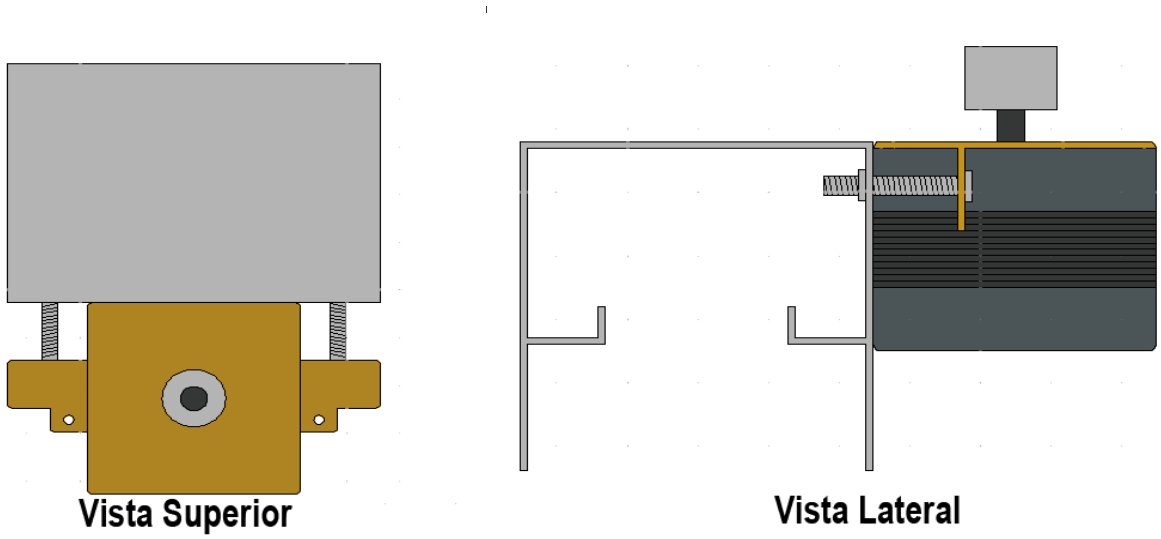


Figura 3.5 - Esquema ilustrativo do dispositivo de rejeito.

Para rejeitar os objetos incompatíveis com o padrão preestabelecido, inseriu-se junto ao motor de passo uma lâmina de plástico para atuar como barreira, conforme pode se observar na Figura 3.6.

Figura 3.6 - Lamina de plástico acoplada ao dispositivo com função de barrar objetos indesejáveis.

Após realizadas estas etapas, obteve-se o sistema montado, como esquematizado na Figura 3.7. Entende-se por área útil da câmera o campo de visão máximo que este equipamento possui da esteira. Esta área foi calculada (67mm x 90mm) para poder definir-se o tamanho máximo do objeto a ser analisado. Vale ressaltar que esse tamanho pode variar com a aproximação e afastamento da câmera da esteira. A área útil da câmera demarca a área útil da esteira (Figura 3.7 em vermelho).

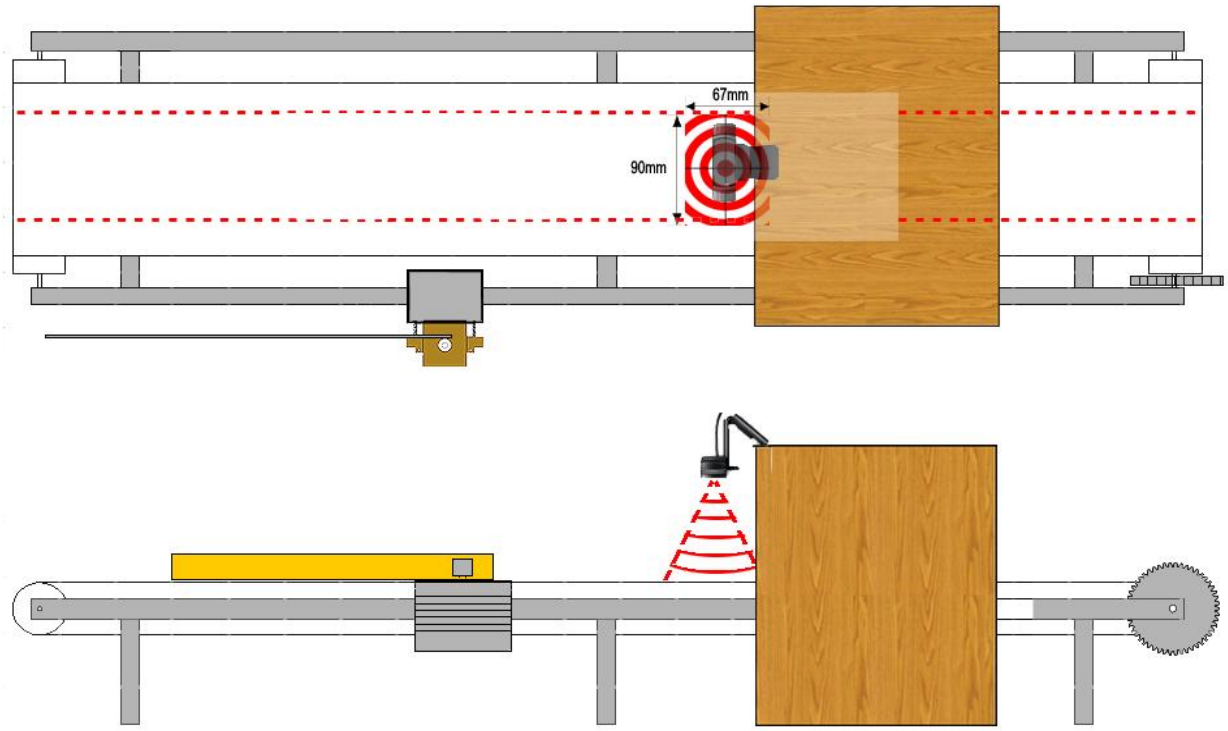


Figura 3.7 - Esquema representativo do sistema montado

3.9.2 Acionamento do dispositivo

O sistema de rejeito é acionado quando o sistema de visão computacional reconhece que não há na esteira um produto similar ao padrão, ocorrendo o início da contagem regressiva do tempo para ocorrer a obstrução da esteira. Esse tempo é suficiente para que o objeto antecessor ao objeto a ser descartado ultrapasse a linha inicial de obstrução do dispositivo. O dispositivo de rejeito deve bloquear a área útil da esteira extraviando o produto indesejável para a sua região periférica. É crucial salientar que o dispositivo deve realizar esta tarefa antes que o objeto sucessor a aquele a ser descartado chegue à linha de obstrução máxima do dispositivo de rejeito. Este processo pode ser melhor visualizado na Figura 3.8.

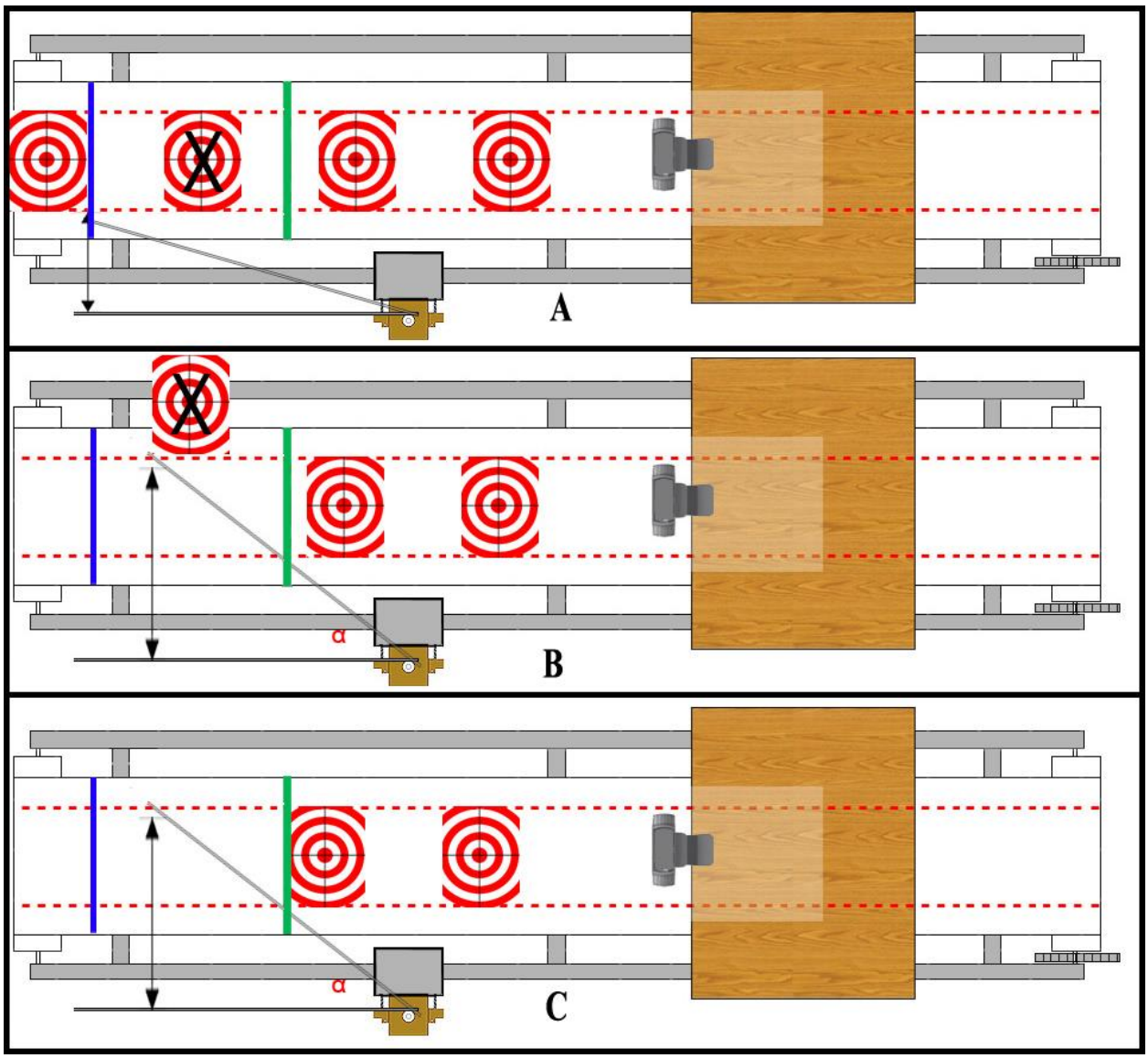


Figura 3.8 – Acionamento do dispositivo de rejeito.

Na Figura 3.8, em azul tem-se a linha inicial de obstrução. Em verde tem-se a linha de obstrução máxima. A) Reconhecimento do objeto indesejável e inicio da obstrução da área útil da esteira pelo dispositivo de rejeito. B) Remoção do objeto indesejável como obstrução máxima da área útil da esteira pelo dispositivo de rejeito. C) Retomada do dispositivo de rejeito para posição inicial.

Para se rejeitar o produto indesejável presente na esteira, deve-se analisar parâmetros como a velocidade em que o produto está sendo transportado, a distância em que o dispositivo se encontra da câmera e a velocidade em que este é acionado. Desta forma, para se obter a velocidade em que o produto é transportado, calculou-se primeiramente a velocidade angular do motor. Em seguida foi calculada a velocidade angular do eixo da esteira através da razão de redução entre as engrenagens e

finalmente, a velocidade linear da esteira foi calculada usando o perímetro do seu rolo transportador. Estes cálculos foram realizados como se segue:

Dados:	Cálculos:
r_1=0,6cm±0,1cm r_2=4,7cm±0,1cm V_{em} = 49rpm r_3=3cm±0,1cm V_{le} = ?	$\frac{r_1}{r_2} = \frac{0.6}{5.2} = 0{,}1154 = \eta$ $V_{ee} = V_{em} * \eta = 49rpm * 0{,}1154 = 5{,}65rpm$ $V_{le} = Vee * 2 * \pi * r_3 = 5{,}.65 * 2 * 3{,}14 * 3 = 106{,}5cm/min$ $\frac{106{,}5m}{min} * \frac{1m}{100cm} * \frac{1min}{60s} = 0.0178 \pm 0{,}0018m/s$

Onde:

r_1 = raio da engrenagem do motor CC

r_2 = raio da engrenagem da esteira

r_3 = raio do rolo da esteira

V_{em} = velocidade angular do eixo do motor (MABUCHI, 2014)

V_{le} = velocidade linear da esteira

Para conferir a veracidade dos cálculos, a velocidade linear da esteira foi medida de forma experimental, onde mensurou-se uma distância percorrida em relação ao tempo. Este teste foi realizado em três vezes de modo a minimizar possíveis erros. Os resultados destes testes são apresentados na Tabela 3.1.

Tabela 3.1 – Resultados dos testes de velocidade linear

Teste	**Distância (m)**	**Tempo (s)**	**Velocidade Linear (m/s)**
1	0,20	12,34	0,0162
2	0,20	12,45	0,0161
3	0,30	18,59	0,0161
Total	0,7	43,38	0,0161

Uma média do valor experimental encontrado foi calculada (0,0161m/s) e está dentro do erro estimado (±0,0018m/s) da velocidade linear calculada (0,0178m/s).

A distância do dispositivo à câmera foi mensurada com a utilização de uma régua onde obteve-se um valor de 40cm.

A eficácia do sistema de rejeição depende ainda da velocidade em que esse dispositivo é acionado e retoma a sua posição inicial. Para se calcular esta velocidade é necessário conhecer-se previamente o tempo máximo que o dispositivo tem para ser acionado, rejeitar o objeto indesejável e retomar a sua posição inicial, sem interferir nos demais objetos que estejam na esteira. O cálculo do tempo máximo é demonstrado abaixo:

Dados:	Cálculos:
$L_1 = 154\text{mm}$ $L_2 = 296\text{mm}$ $L_3 = 400\text{mm}$ $L_5 = 400\text{mm}$ $d + \Delta d = 300\text{mm}$ $V_{le} = 16{,}1\text{mm/s}$ $L_4 = ?$ $L_6 = ?$ $t_1 = ?$ $t_{max} = ?$	$$\boldsymbol{\alpha_1} = \operatorname{sen}^{-1}\frac{154}{400} = \mathbf{22,64°}$$ $$\boldsymbol{L_4} = \sqrt{(400^2 - 154^2)} = \boldsymbol{369,17mm}$$ $$L_4 + L_5 = 369.17mm + 400mm = 739.17mm$$ $$\boldsymbol{t_1} = \frac{739.17mm}{16.1mm/s} = \boldsymbol{45.9s}$$ $$\boldsymbol{\alpha_2} = \operatorname{sen}^{-1}\frac{296}{400} = \mathbf{47.73°}$$ $$\boldsymbol{L_6} = \sqrt{(400^2 - 296^2)} = \boldsymbol{269.04mm}$$ $$\boldsymbol{d} = L_4 - L_6 + \frac{L_2 - L_1}{\tan\alpha_2} = 369.17 - 269.04 + 104.5 = \boldsymbol{204.6mm}$$ $$\Delta\boldsymbol{d} = 300 - 204.6 = \boldsymbol{95.4mm}$$ $$\boldsymbol{t_{max}} = \frac{95.4mm}{16.1mm/s} = \boldsymbol{5.93s}$$

Onde:

L_1 = Distância da posição inicial até a linha da área útil da esteira

L_2 = Distância da posição inicial até a posição final

L_3 = Comprimento da lamina de plástico do dispositivo

L_5 = Distância do dispositivo até a câmera

t_1 = Tempo para objeto antecessor ultrapassar a linha de obstrução inicial

t_{max} = Tempo para objeto sucessor atingir a linha de obstrução máxima

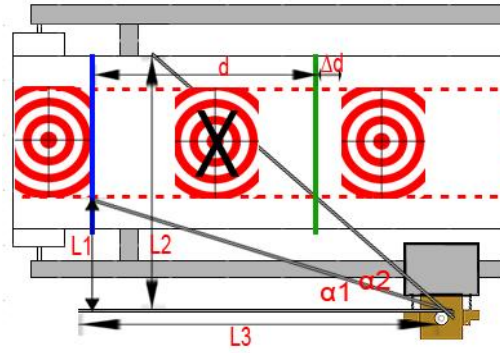


Figura 3.9 - Esquema ilustrativo das variáveis calculadas

Com o tempo máximo pôde-se calcular a velocidade mínima do dispositivo. Estes cálculos são demonstrados abaixo:

Dados:	Cálculos:
$t_{max} = 5.93s$ $\alpha_1 = 22.64°$ $\alpha_2 = 47.73°$ $V_d = ?$	$$V_d = \frac{47.73° - 22.64°}{5.93s} = \frac{25.09°}{5.93s} = 4.23°/s$$

Onde:

$\alpha_2 - \alpha_1$ = Deslocamento angular necessário para descarte

t_{max} = Tempo máximo para descarte

V_d = Velocidade angular do dispositivo

Como encontrado nos cálculos, a velocidade mínima do motor para descartar objeto deve ser de 4.23°/s. Cada passo do motor de passo utilizado é de 1.8° (1.8°/passo). Isto significa que o motor deve dar no mínimo 2.35 passos por segundo $\left(\frac{4.23°/s}{1.8°/passo}\right)$. Para isso o programa deve enviar a sequência de pulsos que formam o passo no mínimo a cada 0.43s. Para obter-se um fator de segurança de 3, o tempo entre os passos foi reduzido para 0.15s ou 150ms.

Com isso, pôde-se calcular o tempo entre o reconhecimento da peça defeituosa e o acionamento do dispositivo.

Dados:	Cálculos:
t_1= 45,9s V_{du} = 12°/s α_1 = 22,64° t_{total} = ?	$$t_{total} = t_1 - \frac{22{,}64°}{12°/s} = 44s$$

Onde:

α_1 = Deslocamento do dispositivo até área útil da esteira

t_1 = Tempo para objeto antecessor ultrapassar a linha de obstrução inicial

t_{total} = Tempo do reconhecimento do objeto defeituoso até o inicio de funcionamento do dispositivo

3.10 DESENVOLVIMENTO DO CIRCUITO DO DIPOSITIVO

Um circuito foi montado para que o PIC pudesse comunicar-se com a porta USB. Esse circuito é padrão e foi encontrado em um tutorial disponibilizado pelo Departamento Engenharia Elétrica e Eletrônica da Universidade do Oriente Médio na Turquia (ERDOGAN, 2013). O circuito foi montado no programa *Proteus* (Anexo V) e, posteriormente, foi confecionada.

Para acionar o motor da esteira de 12V foi utilizado uma fonte de um computador de mesa que tem as especificações mostradas na figura 3.10, pois a tensão de 5V fornecida pela porta USB não foi suficiente para acionar este motor.

Figura 3.10 – Imagem representativa da fonte utilizada para alimentar o motor da esteira (GUIMARÃES, 2012)

3.10.1 Descrição do circuito

O PIC18F4550 foi energizado utilizando o próprio VCC e GND da porta USB. Isso significa que o dispositivo extrairá toda sua energia a partir do *host* (PC). Um capacitor de 470nF é necessário para dar estabilidade a tensão no interior do PIC podendo operar o seu circuito de USB interno. O valor desse capacitor recomendado pelo *datasheet* é de no mínimo 220nF. No caso deste trabalho utilizou-se um capacitor de 470nF, segundo especificações encontradas no tutorial. No entanto, não encontrou-se argumentos que justificassem esse valor de capacitor utilizado. Um cristal de 12MHz é utilizado em conjunto com um PLL (multiplicador de frequência) interno para aumentar o *clock* para 48Mhz (frequência necessária para a comunicação USB). As conexões D+ e D- utilizados para comunicação são conectados às portas 24 e 25 respectivamente (MICROCHIP, 2013), sendo responsáveis pelo envio/recebimento de pacotes de dados para o computador.

Para acionar a esteira utilizou-se um relé, na posição normalmente aberta, onde foi aplicada uma tensão de 12V da fonte de computador. Este relé foi acionado pelo PIC que energiza a sua bobina interna fechando o contato que libera os 12V, ligando assim a esteira. O motor de passo foi acionado ligando o seu fio comum ao terminal de 5V do USB e seus outros 4 fios (1, 2 ,3 e 4) (Figura 3.11) a outros 4 transistores (TBJ) ligados à terra (GND). Estes TBJs foram acionados pelas portas D1, D2, D3, e D4 do PIC, onde liberam a passagem de corrente do seu fio comum para terra (nesse caso, o D-). O esquema do circuito feito no *Proteus* pode ser consultado no Anexo V.

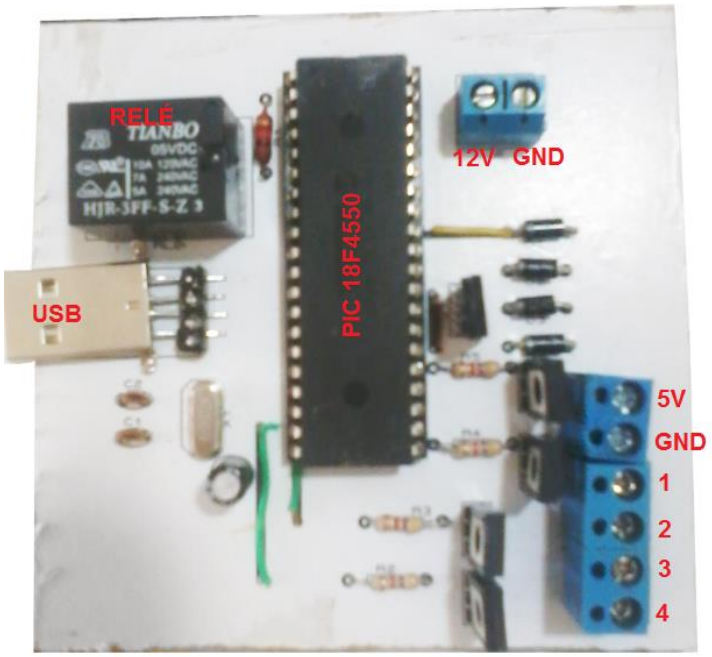


Figura 3.11 – Circuito confeccionado

Após a construção do circuito, este e a fonte foram colocadas dentro de uma caixa para aumentar a sua mobilidade e seu aspecto estético. Na Figura 3.12 é mostrado o circuito devidamente montado, onde observam-se duas saídas, uma de 12V e uma GND, que são ligadas ao motor da esteira. Nota-se, ainda, um interruptor e uma tomada de 110V para o fornecimento de energia elétrica ao circuito, assim como conectores para interligá-lo ao dispositivo de rejeito e ao computador com o programa principal.

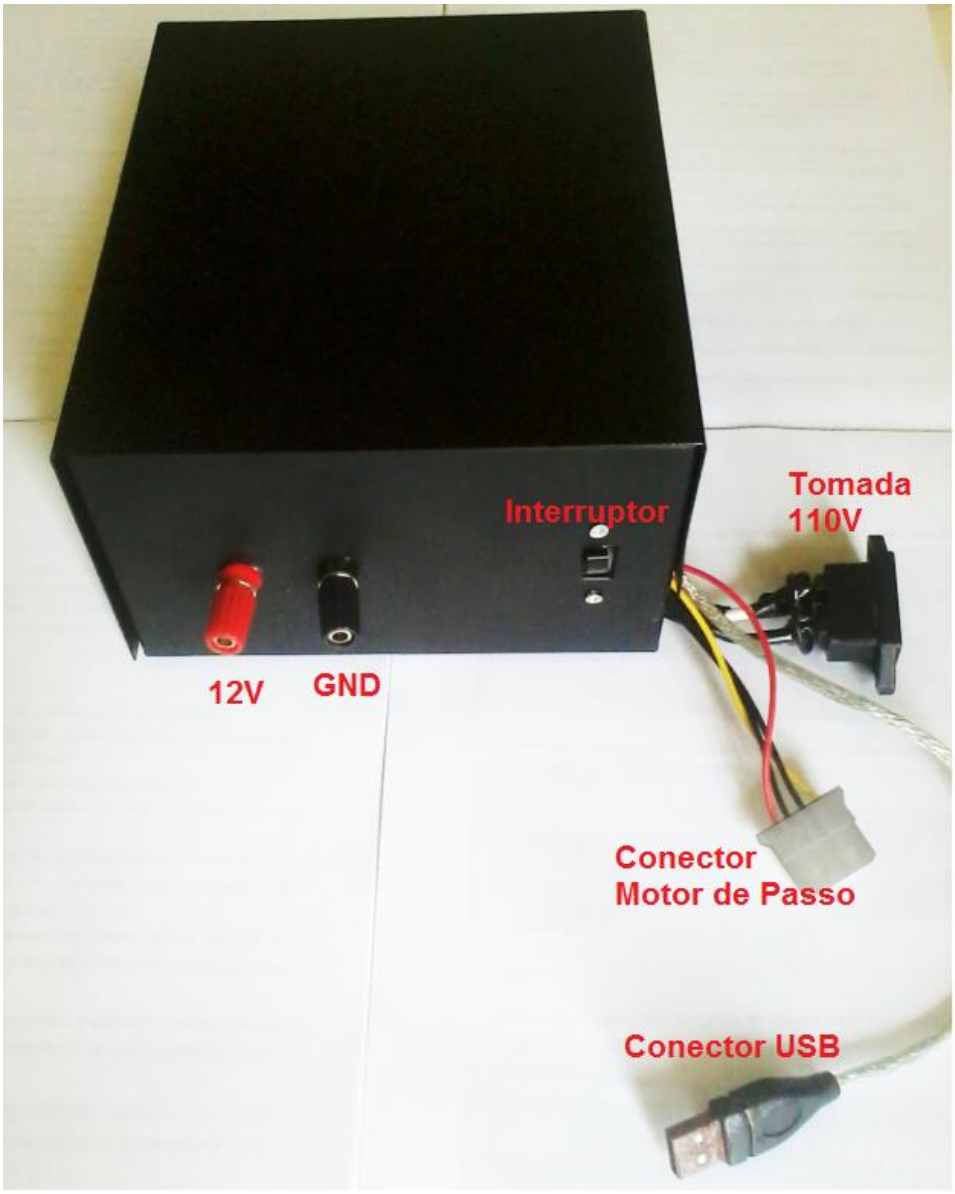


Figura 3.12 – Circuito devidamente montado

Capitulo 4. Testes Realizados

Para verificar o cumprimento dos requisitos do projeto, assim como os objetivos desse trabalho, foram realizados diversos testes. A verificação dos requisitos é feita de forma independente, e depois em conjunto. Por exemplo, foi verificado somente a eficiência do reconhecimento de imagens, em seguida somente a comunicação via USB, e finalmente as duas tarefas em conjunto. Os testes que foram realizados são:

1. Verificar eficiência de reconhecimento de imagens
2. Verificar eficácia da programação do PIC em receber/enviar informações via USB;
3. Verificar eficácia do PIC em controlar os periféricos (motor de passo e motor CC 12V);
4. Verificar o funcionamento do dispositivo como um todo;

4.2 RECONHECIMENTO DE IMAGENS

O primeiro teste realizado visou analisar a eficiência da tarefa de reconhecimento. Os parâmetros usados para verificação foram:

1. Reconhecer o objeto de vários ângulos (programa é invariante a rotação);
2. Reconhecer o objeto de várias distâncias (invariante a escala);
3. Reconhecer o objeto com variação da iluminação;
4. Conseguir gravar e reconhecer novos perfis de objetos com facilidade;
5. Rejeitar peças defeituosas.

Todos os testes foram feitos nas mesmas condições, alterando-se apenas a variável em análise. A imagem de um objeto foi capturada em uma sala de aula do CEFET-MG com as luzes acesas às 11h30 do dia 30/01/2014. O objeto encontrava-se a 15 cm da *webcam*, a um ângulo normal ao objeto. Essa imagem (Figura 4.1) foi utilizada como modelo para o restante dos testes nesta seção.

Figura 4.1 **-** Imagem padrão capturada pelo dispositivo

4.1.1 Invariância em relação à rotação

Utilizando a imagem modelo, o programa principal foi configurado para analisar o vídeo capturado pela *webcam*, o qual avaliou um objeto colocado em diferentes ângulos em relação ao campo de visão da câmera. Esses ângulos foram registrados e foi verificado o reconhecimento ou não do objeto (Figura 4.2).

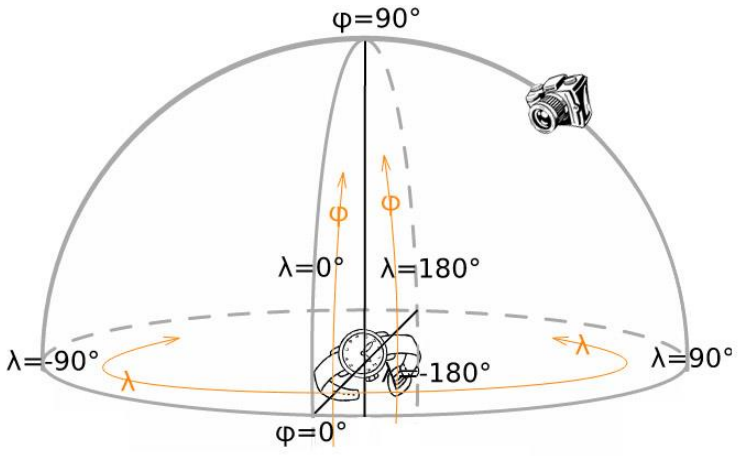


Figura 4.2 **-** Esquema do teste de invariância à rotação (WIKIPEDIA, 2006 - modificado).

4.1.2 Invariância em relação à escala

Utilizando-se novamente a imagem modelo para reconhecimento, o programa principal foi configurado para analisar o vídeo proveniente da *webcam*, onde foi analisado um objeto a distâncias no intervalo de 0cm a 30cm. Essas distâncias foram registradas e foi verificado o reconhecimento ou não do objeto (Figura 4.3).

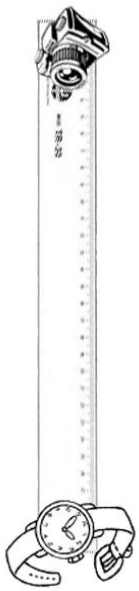


Figura 4.3 **-** Esquema do teste de invariância a escala (CHARITY, 2004 - Modificado)

4.1.3 Invariância à iluminação

O teste de invariância a iluminação verificou se a luminosidade do ambiente influenciaria no reconhecimento do objeto a ser analisado pelo dispositivo. Esse teste foi realizado em uma sala com uma lâmpada fluorescente de 60W às 20h00 do dia 31/01/2014. Primeiramente, o objeto foi analisado pelo programa com as luzes acesas (Figura 4.4), e logo após, com as luzes apagadas (Figura 4.5).

Figura 4.4 **-** Captura do objeto com as luzes acesas

Figura 4.5 **-** Captura do objeto com as luzes apagadas

4.1.4 Inclusão de novos perfis

A fim de se verificar a amplitude do programa em reconhecer diferentes imagens, fez-se o teste de inclusão de novos perfis. Este teste visa verificar a capacidade do programa em obter imagens modelos de novos objetos, assim como reconhecê-los. Esse teste consistiu na captura de diversas imagens de novos objetos (Figuras 4.6 – 4.9) e da análise e reconhecimento destes.

Figura 4.6 - Imagem padrão de uma chave universal

Figura 4.7 - Imagem padrão de um tubo de creme dental

Figura 4.8 - Imagem padrão de um telefone celular

Figura 4.9 - Imagem padrão de uma cartela de remédios

4.1.5 Rejeito de peças defeituosas

Esse teste tem como objetivo verificar a precisão do programa em rejeitar um objeto com imperfeições em relação a um objeto padrão previamente cadastrado. Para realizar este teste foi definido no programa principal uma imagem modelo. Essa imagem foi utilizada posteriormente como padrão comparativo com os objetos que apresentam imperfeições, onde o programa rejeitaria (ou não reconheceria) os produtos defeituosos.

A realização deste teste foi efetuada de duas formas: a primeira com a modificação da imagem do objeto digitalmente, e a segunda com a alteração física de outros objetos.

O teste de rejeito com a modificação digital da imagem do objeto foi realizado utilizando uma foto padrão (Figura 4.10) e alterando-a deixando apenas uma parcela

de suas características originais (Figuras 4.11 4.12) e posteriormente avaliando a capacidade do programa em rejeitar um objeto defeituoso em relação as características da foto original mantidas.

Figura 4.10 – Imagem padrão usado para testes estatísticos (GIEVES & HAWKES, 2006)

Figura 4.11- Imagem Analisada com 90% das características da foto padrão

Figura 4.12 - Imagem analisada com 80% das características da foto padrão

A segunda forma foi utilizar um objeto padrão (Figura 4.13 e Figura 4.15) e deformar ou atribuir defeitos fisicamente (Figura 4.14 e Figura 4.16) a este, avaliando a capacidade do programa rejeitar um objeto defeituoso na prática.

Figura 4.13 – Imagem de um tubo de pasta de dente sem defeito

Figura 4.14- Imagem de um tubo de pasta de dente com defeito

Figura 4.15- Imagem de uma lata de creme de barbear sem defeito

Figura 4.16 - Imagem de uma lata de creme de barbear com defeito

4.3 VERIFICAÇÃO DA EFICÁCIA DA PROGRAMAÇÃO DO PIC

Nesse teste foram realizados testes de envio e recebimento do programa principal para o PIC.

Para verificar se o PIC estava sendo reconhecido pelo computador, foi colocada no programa principal uma Barra de *status*. Quando dispositivo era reconhecido, o programa alterava o texto da barra para dispositivo conectado. Esse reconhecimento se deve ao uso do VID e PID. O código utilizado para realizar esse teste é mostrado a seguir:

```
1  theReferenceUsbDevice = new usbReferenceDevice(0x04D8, 0x0080);
2  .
3  .
4  .
5  private void usbEvent_receiver(object o, EventArgs e)
6          {
7            if (theReferenceUsbDevice.isDeviceAttached)
8             {
9               this.usbToolStripStatusLabel.Text = "USB Device is attached";
```

A verificação do envio e recebimento de informações foi realizada para verificar se a programação do PIC obteve êxito. Dentro da programação do PIC existem códigos que ao receberem um sinal de inicialização do computador enviam uma mensagem de volta, indicando que houve o recebimento do código, e com isso infere-se que há uma comunicação bilateral efetiva entre o computador e o PIC.

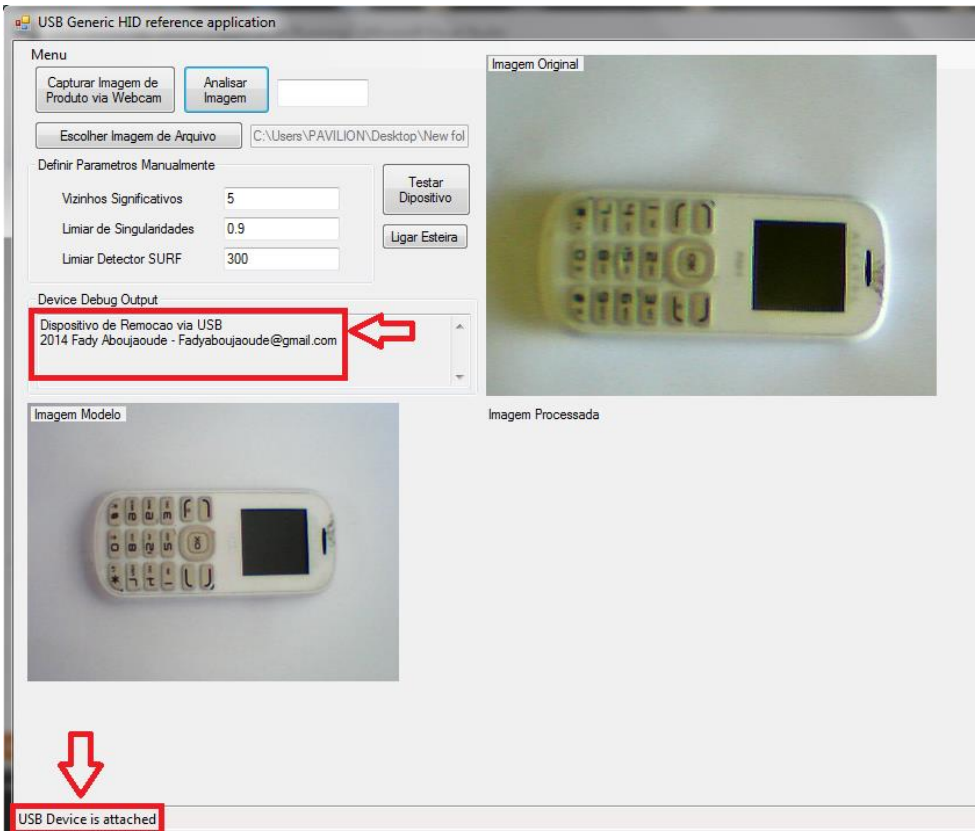


Figura 4.17 - Imagem da tela do programa com indicação de comunicação bilateral destacado

Capitulo 5. Resultados e Discussão

Com o intuito de se conhecer sobre a funcionalidade do programa e possíveis fatores que pudessem prejudicá-lo, os testes descritos no capítulo anterior foram realizados e nesta seção terão seus resultados discutidos.

Analisando-se diferentes ângulos de orientação da *webcam* (Tabela 5.1), notou-se que o programa desenvolvido tem uma invariância completa em relação ao eixo de rotação ʎ, ou seja, a câmera reconhece o objeto independentemente da variação do ângulo desse eixo. Essa invariância completa é devido a uma função chamada *BruteForceMatcher,* que atua em conjunto com uma matriz de homografia. O *BruteForceMatcher* acha os descritores de uma imagem, semelhantes aos descritores de outra imagem. Este fato ocorre pela utilização de um método denominado *KNNMatcher* (*K-Nearest-Neighbor-Matcher*). Esse método pega um descritor e busca outros “K” descritores mais próximos a eles, e em seguida encontra as distâncias do descritor original em relação aos demais (LAGANIÉRE, 2011). Esses dados são salvos para esse descritor distinto e então são comparados como o mesmo conjunto de dados para todos os descritores de outra imagem até identificar um descritor em comum (EMGUCV, 2013). Após encontrar todos os descritores e seus semelhantes, é calculada uma matriz de homografia. Se esta existir, o objeto pode ser encontrado, assim como a sua orientação (EMGUCV, 2013). Observou-se ainda que há uma invariância menor em relação ao eixo de rotação ϕ.

Tabela 5.1 - Resumo dos resultados do teste de invariância em relação à rotação.

Orientação da *webcam*		**Verificação de reconhecimento**
ϕ	**λ**	
90^o	90^o	Objeto reconhecido
80^o	90^o	Objeto reconhecido
70^o	90^o	Objeto reconhecido
60^o	90^o	Objeto não reconhecido
50^o	90^o	Objeto não reconhecido
50^o	50^o	Objeto não reconhecido
90^o	90^o	Objeto reconhecido
90^o	80^o	Objeto reconhecido
90^o	70^o	Objeto reconhecido
90^o	60^o	Objeto reconhecido
90^o	50^o	Objeto reconhecido
80^o	80^o	Objeto reconhecido
70^o	70^o	Objeto reconhecido
60^o	60^o	Objeto reconhecido

A fim de se observar o efeito que diferentes distâncias exerceriam sobre o reconhecimento de objetos pelo programa, afastou-se gradualmente a câmera do objeto a ser analisado num intervalo de 5 a 30cm (Tabela 5.2). Pôde-se observar que a distâncias pequenas (5 a 10cm) não houve reconhecimento do objeto pelo dispositivo, o que pode estar relacionado com o foco da imagem sendo analisada. A distâncias maiores do objeto em relação à câmera (20 a 30cm) notou-se o mesmo fato, o que também pode estar relacionado à qualidade da imagem obtida. Os melhores intervalos observados para análise da imagem estão compreendidos em uma faixa média dos pontos de 10 a 20cm. Isso deve ao fato de que a imagem padrão do objeto em análise foi obtida em 15cm com um foco ajustado para essa distância. O teste então foi repetido da mesma forma ajustando-se o foco da *webcam* a cada 5cm. Analisando os resultados do teste repetido (Tabela 5.3), foi observado que o programa obteve uma invariância do reconhecimento do objeto em relação à distância da *webcam*, o que está relacionado à utilização de uma escala gaussiana durante a obtenção dos descritores de uma imagem (MORDVINTSEV, 2014). O fato do teste anterior não apresentar a mesma conclusão se deve ao desfoco da imagem, que

acarretou o ocultamento de algumas de suas características, impedindo seu reconhecimento pelo detector SURF e sua posterior análise.

Tabela 5.2 - Resultados do primeiro teste de invariância à escala

Distancia da *webcam*	**Verificação de reconhecimento**
Até 5cm	Objeto não reconhecido
5 a10cm	Objeto não reconhecido
10 a 15cm	Objeto reconhecido
15 a 20cm	Objeto reconhecido
20 a 25cm	Objeto não reconhecido
25 a 30cm	Objeto não reconhecido

Tabela 5.3 - Resultados do teste de invariância à escala repetido com ajuste de foco

Distancia da *webcam*	**Verificação de reconhecimento**
Até 5cm	Objeto não reconhecido
5 a 10cm	Objeto reconhecido
10 a 15cm	Objeto reconhecido
15 a 20cm	Objeto reconhecido
20 a 25cm	Objeto reconhecido
25 a 30cm	Objeto reconhecido

Com o objetivo de se identificar o efeito provocado pela luminosidade no reconhecimento de imagens, fez-se uma análise da detecção dos objetos pelo programa principal em ambientes com diferentes condições de luz (Figuras 5.1 e 5.2). Conforme os resultados do teste de iluminação (Tabela 5.4), notou-se que na ausência de luminosidade não houve reconhecimento da imagem pelo programa, o que pode estar relacionado à formação da imagem. Diminuindo-se a luminosidade de um ambiente, deteriora-se a qualiade da imagem obtida pela câmera, consequentemente acarretando um menor número de pontos chaves reconhecidos pelo programa dificultando deste modo, a identificação do objeto.

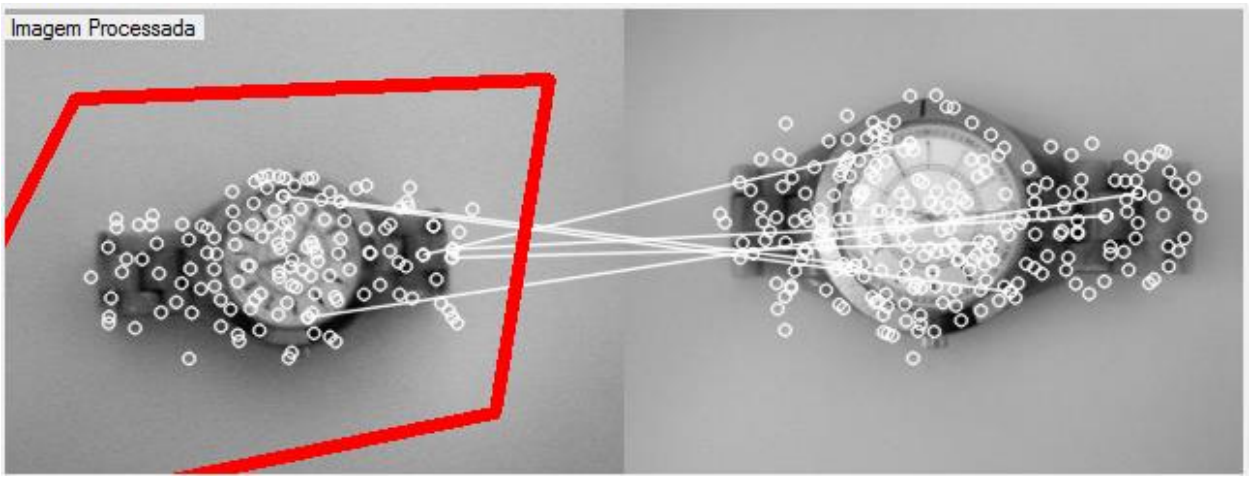


Figura 5.1 - Resultado do teste de reconhecimento do objeto com baixa luminosidade

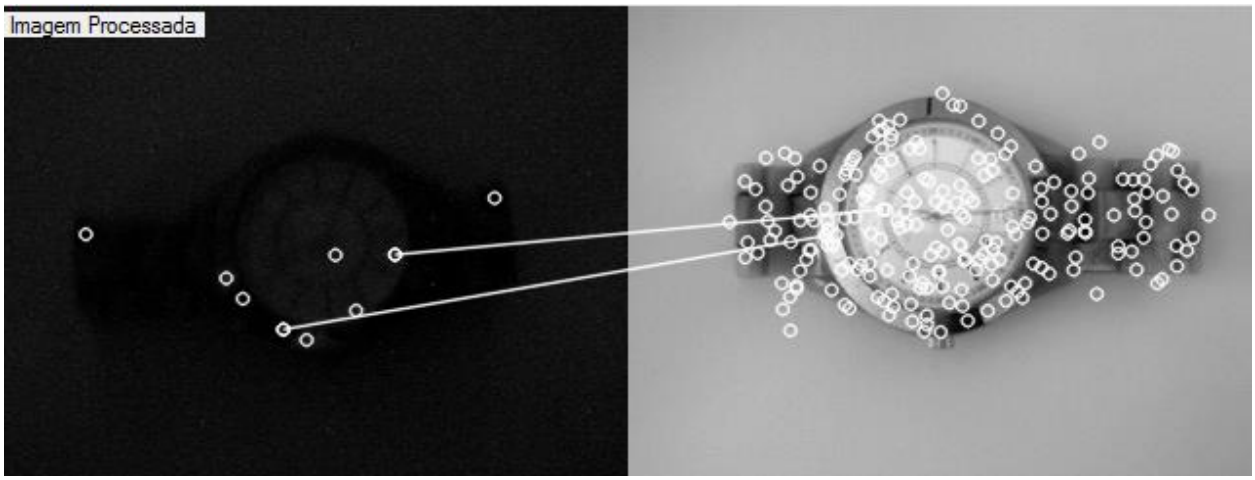


Figura 5.2 - Resultado do teste de reconhecimento do objeto na ausência de luminosidade

Tabela 5.4 **-** Resultados do teste de invariância a iluminação

Distância da *webcam*	**Verificação de reconhecimento**
Imagem modelo (Controle)	Objeto reconhecido
Pouca iluminação (as 20h00)	Objeto reconhecido
Ausência de iluminação (as 20h00)	Objeto não reconhecido

O teste de inclusão de novos perfis se deu com a captura de imagens aleatórias de diversos produtos, onde estabeleceu-se, portanto, o padrão destes. Com as imagens padrão, o programa analisou os mesmos objetos em distâncias e orientações diferentes reconhecendo-os ou não. Analisando os resultados deste teste (Tabela 5.5) pode ser visto que 3 em 4 produtos novos foram reconhecidos sem alterar a programação, o que pode estar associado ao ajuste limiar utilizado neste teste. De acordo com Laganière (2011), pode-se obter mais detalhes diminuindo o valor do

limiar, e com isso detectar mais pontos chaves, aumentando a gama de reconhecimento de imagens. Ainda segundo Laganière (2011), diminuindo-se drasticamente o valor do limiar pode-se comprometer o reconhecimento de imagens, uma vez que detalhes irrelevantes para a análise serão detectados. Esta informação foi aplicada no detector SURF e os resultado obtidos (Tabela 5.6) foram de fato melhores do que os resultados anteriores (Tabela 5.5). As Figuras 5.3 à 5.6 mostram os resultados visuais do computador.

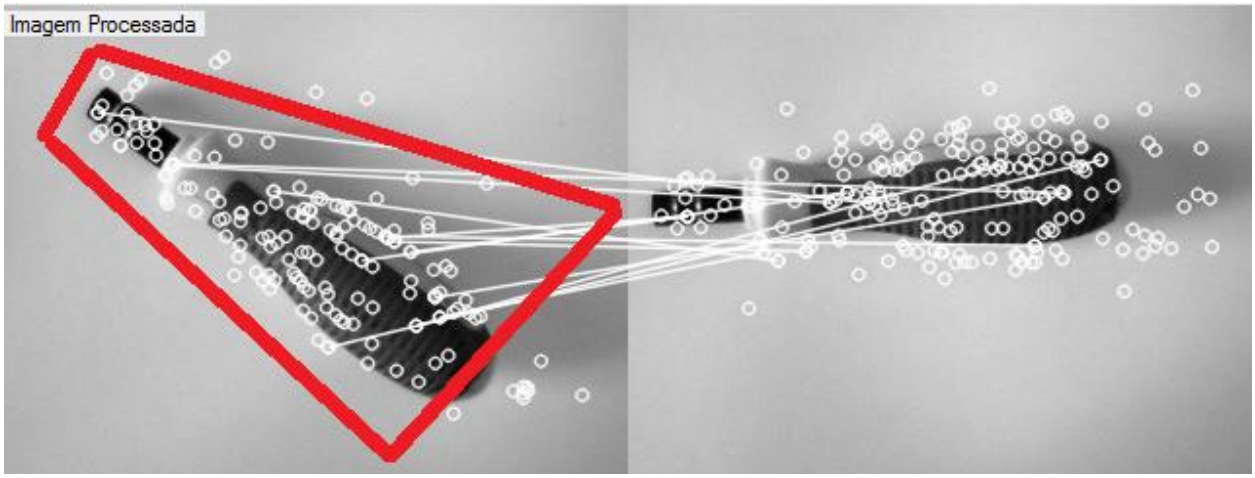


Figura 5.3 - Resultado do primeiro teste de reconhecimento de um novo objeto

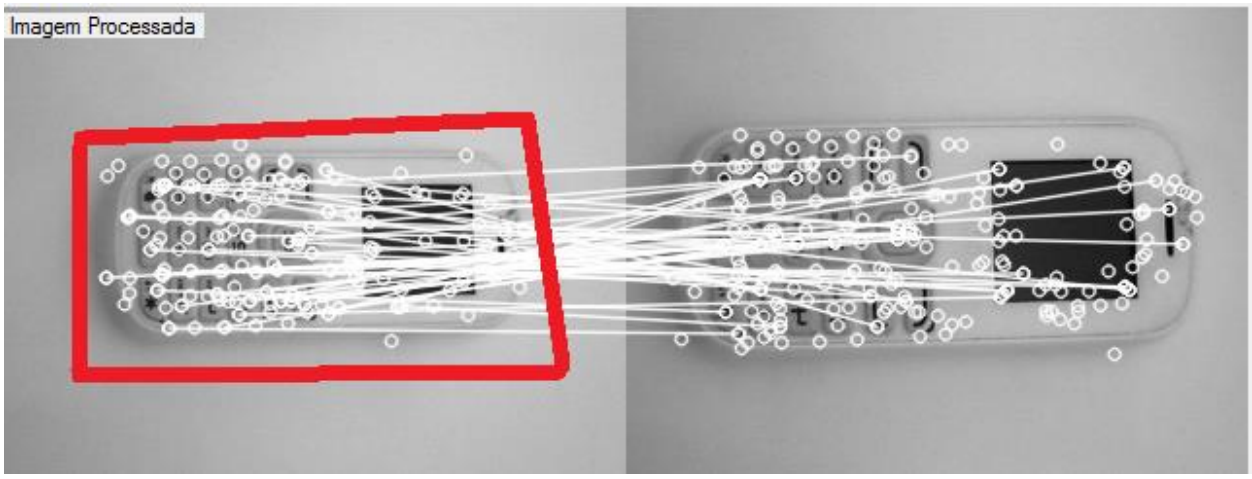


Figura 5.4 - Resultado do segundo teste de reconhecimento de um novo objeto

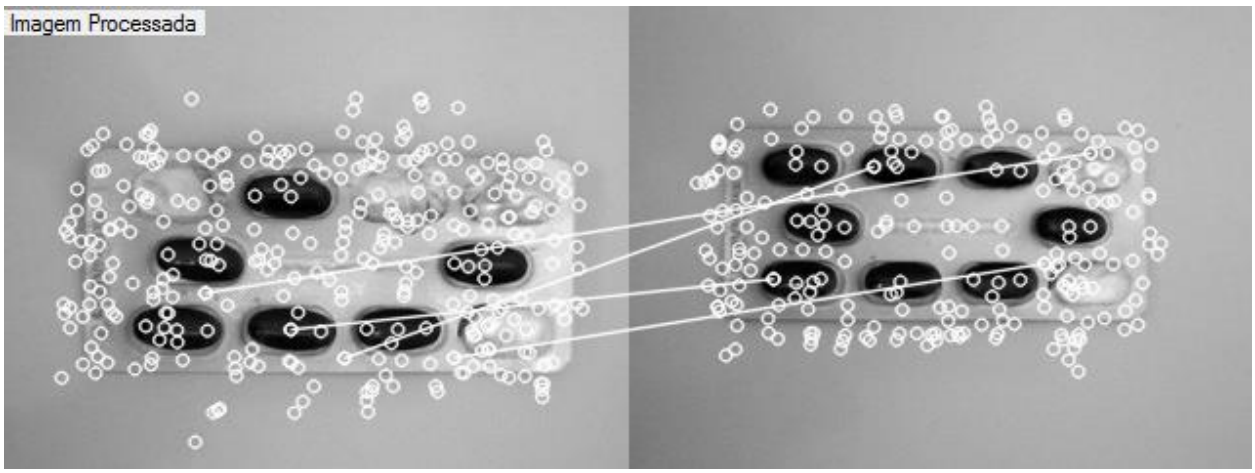


Figura 5.5 - Resultado do terceiro teste de reconhecimento de um novo objeto

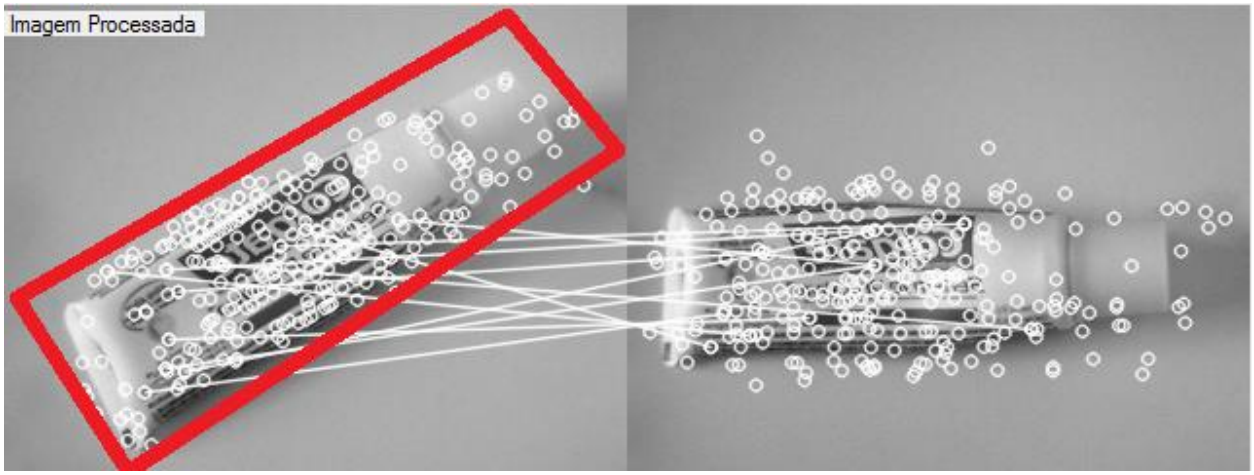


Figura 5.6 **-** Resultado do quarto teste de reconhecimento de um novo objeto IV

Tabela 5.5 **-** Resultados do teste inicial de inserção de novos perfis no programa principal

Produto	**Teste 1**	**Teste 2**	**Teste 3**	**Teste 4**
Chave universal	Reconhecido	Reconhecido	Não reconhecido	Reconhecido
Celular de botão	Reconhecido	Reconhecido	Reconhecido	Reconhecido
Cartela de remédio	Não reconhecido	Reconhecido	Não reconhecido	Reconhecido
Tubo de creme dental	Reconhecido	Reconhecido	Reconhecido	Reconhecido

Tabela 5.6 **-** Resultados do segundo teste de inserção de novos perfis no programa principal

Produto	**Teste 1**	**Teste 2**	**Teste 3**	**Teste 4**
Chave universal	Reconhecido	Reconhecido	Reconhecido	Reconhecido
Celular de botão	Reconhecido	Reconhecido	Reconhecido	Reconhecido
Cartela de remédio	Reconhecido	Reconhecido	Reconhecido	Reconhecido
Tubo de creme dental	Reconhecido	Reconhecido	Reconhecido	Reconhecido

Para se avaliar a capacidade do programa em rejeitar objetos diferentes da imagem padrão, foram capturadas imagens modelos de dois objetos e, posteriormente, foram apresentados os mesmos objetos com alguma imperfeição física ao programa principal para analise.

Os resultados deste teste estão representados nas Figuras 5.7 à 5.10, onde pôde-se observar que o programa realizou o reconhecimento e rejeição das peças corretas. Segundo Pereira (2012), a rejeição do objeto se deve à ausência de pontos chaves correspondentes aos da imagem original. Essa ausência de pontos resulta em um vetor de descritores diferente do vetor original. Quando esse vetor diferente for analisado não será reconhecido como semelhante ao objeto original. Estes testes foram realizados 10 vezes e são apresentados na Tabela 5.7.

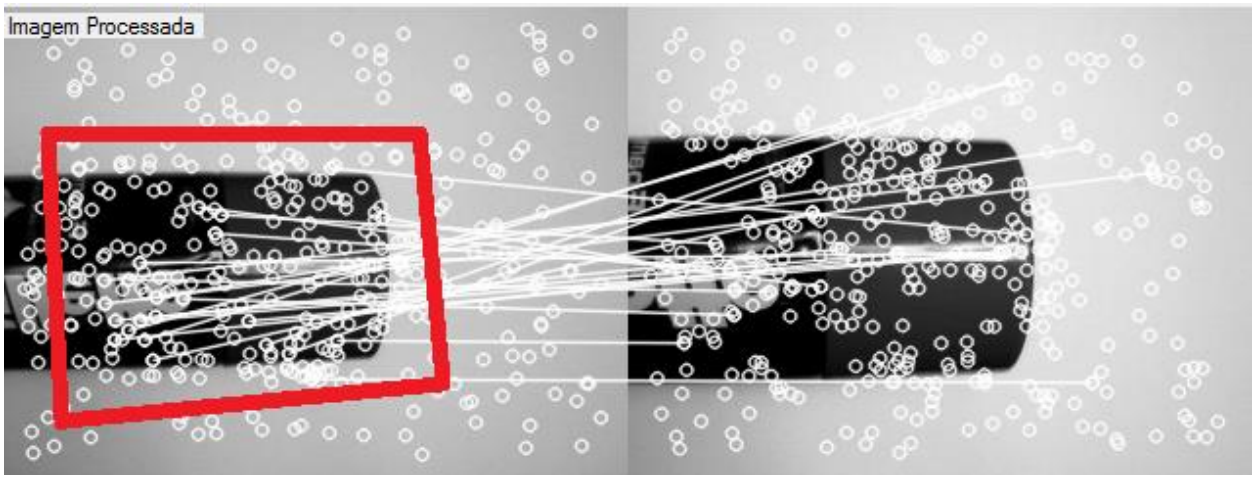


Figura 5.7 - Resultado do teste de rejeito para o primeiro objeto sem defeitos

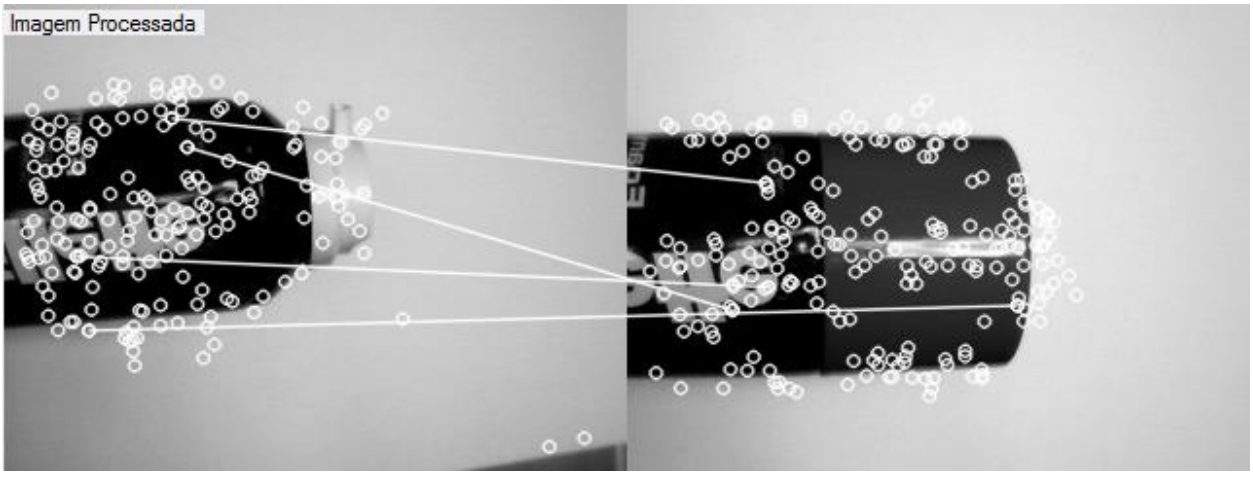


Figura 5.8 - Resultado do teste de rejeito para o primeiro objeto com defeitos.

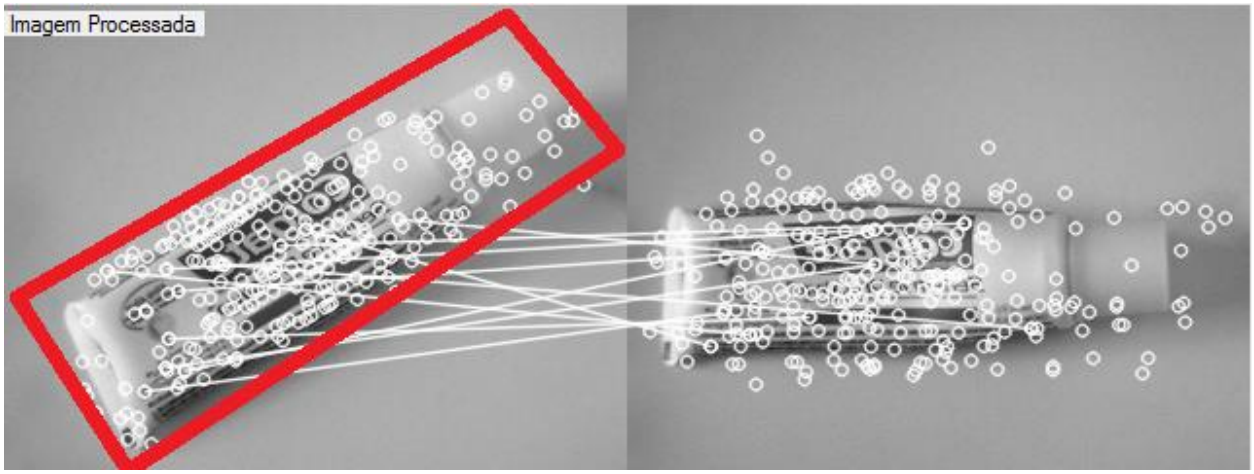


Figura 5.9 - Resultado do teste de rejeito para o segundo objeto sem defeitos.

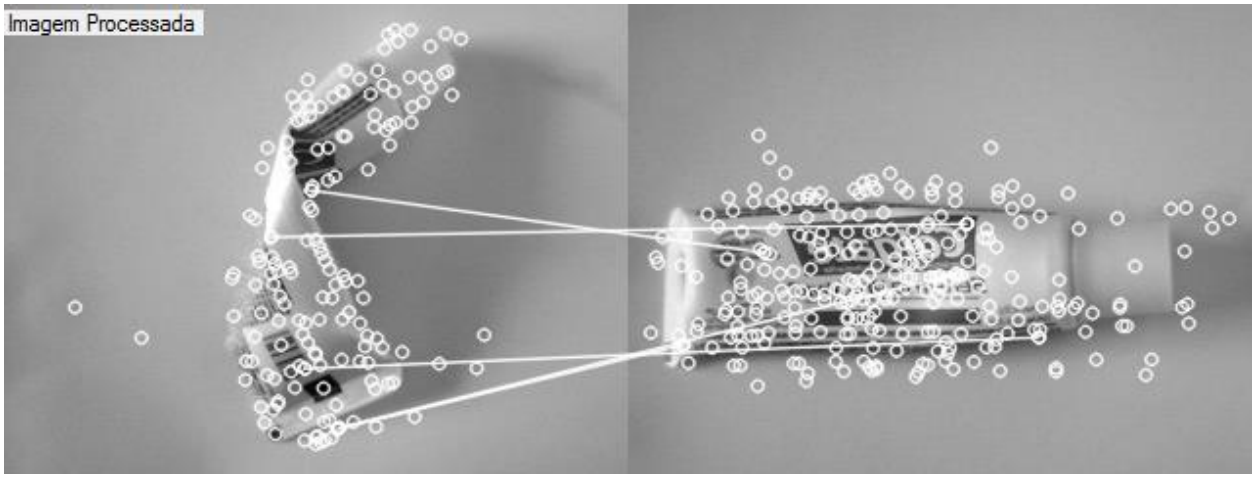


Figura 5.10 - Resultado do teste de rejeito para o segundo objeto com defeitos.

Tabela 5.7 - Resultados dos testes de objetos com defeitos na prática, em que R se refere ao reconhecimento e N ao não reconhecimento da imagem

	Testes									
Objeto	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Lata de creme de barbear (s/ defeito)	R	R	R	R	R	R	R	R	R	R
Lata de creme de barbear (c/ defeito)	N	N	N	R	N	N	N	N	N	R
Tubo de creme dental (s/ defeito)	R	R	R	R	R	R	R	R	R	R
Tubo de creme dental (c/ defeito)	N	N	R	N	N	R	N	N	N	N

Analisando os resultados apresentados na tabela 5.7, o programa apresentou uma eficácia em rejeitar os objetos em 80% das vezes, ou seja, rejeitou 16 produtos defeituosos de um total de 20 produtos defeituosos apresentados.

Capitulo 6. Conclusões e Perspectivas

Este trabalho apresentou um método para a análise de produtos e o desenvolvimento de um protótipo de software para inspeção industrial automatizada. Para realizar tal tarefa foram utilizadas diversas técnicas de processamento de imagens como o detector SURF, o BruteForceMatcher, e o KNNMatcher.

Os métodos e linguagens utilizados para o processamento e reconhecimento das imagens mostraram-se efetivos e atenderam a proposta inicial de um desenvolvimento de um protótipo de baixo custo.

O programa apresentou comunicação bilateral eficaz com o PIC, que por sua vez controlou a esteira e o motor de passo.

Em relação à rotatividade, notou-se uma invariância completa ao eixo ʎ e uma invariância menor ao eixo ϕ.

Observou-se que há algumas limitações que interferem na eficiência do reconhecimento de imagens pelo programa criado. Notou-se que em ausência de luminosidade não há reconhecimento dos objetos pelo programa devido a ausência de pontos chaves necessários para análise.

A distância em que o objeto se encontra da câmera é outro fator que influenciou no desempenho do projeto. Desta forma, a distância ideal para o reconhecimento das imagens depende do foco ajustado pelo operador.

Quanto à inserção de diferentes objetos para reconhecimento pelo programa, notou-se que o limiar interfere na definição dos detalhes reconhecidos pelo detector SURF, sendo necessário alterá-lo de acordo com o objeto em análise.

O teste de rejeito de objetos reais que apresentavam imperfeições em relação à imagem padrão foram descartados em 80 % dos testes realizados.

Em futuros estudos deverão ser abordados novos tópicos como a alteração do algoritmo de reconhecimento para verificar se haverá um aumento na precisão do sistema, o estabelecimento de um ambiente controlado onde os objetos serão analisados em condições idênticas, tais como iluminação e posição. Poderá ser abordado também uma maneira de tornar o sistema mais veloz, aproximando-o cada vez mais das necessidades de uma indústria. Deste modo, as limitações apresentadas pelo programa desenvolvido neste trabalho poderão ser supridas, permitindo assim sua efetiva inserção em linhas de produção de empresas.

Referências Bibliográficas

AABY, A. A. (2004). *Introduction.* Eastern Mediterranean University, Disponível em: <http://www.emu.edu.tr/aelci/Courses/D-318/D-318-Files/plbook/intro.htm> Acesso em: 25 Jan 2014 10h30

BAUMBER. (2000). Reliable feature matching across widely separated views. *Computer Vision and Pattern Recognition*, (pp. 774-781).

BAY, H. (2008). SURF: Speeded Up Robust Features. *Computer Vision and Image Understanding*, pp. 346-359.

CHARITY, M. (2004). CCRI Faculty Web - *Some printable paper rulers*. Disponivel em: < http://www.vendian.org/mncharity/dir3/paper_rulers/UnstableURL/ruler_foot.pdf> Acesso em: 06 Jan 2014.

DAVIES, R. (2005). *Machine Vision: Theory, Algorithms, Practicalities* 1 ed., Vol. 1. Morgan Kaufmann.

DELAI, R. L. (2012). *VISÃO COMPUTACIONAL COM A OPENCV – MATERIAL APOSTILADO E VEÍCULO SEGUIDOR AUTÔNOMO.* Instituto Mauá de Tecnologia, Mauá.

ECMA. (2012). *C# Language Specification,* 4 ed., Ecma International. .

EMERSON. (2013). Wikipedia *Dev-C++*. Disponivel em: <http://en.wikipedia.org/wiki/Dev-C%2B%2B> Acessado em: 15 Ago 2013.

EMGUCV. (2013). *Main Page*. EMGUCV Disponivel em: <http://www.emgu.com/wiki/index.php/Emgu_CV> Acessado em: 10 Jan 2014.

ERDOGAN, B. (2013). *Middle East Technical University* - Departamento of Electrical and Electronics Engineering, Disponivel em: <http://www.eee.metu.edu.tr/~design/lib/exe/fetch.php?media=lecture_notes:tutorial_1_-_implementation_of_a_usb_based_pic-to-pc_communication.pdf> Acessado em: 20 Dez 2013.

FORSYTH, D. A. (2003). *Computer Vision, A Modern Approach* (1 ed., Vol. 1). Prentice Hall.

GIEVES, & HAWKES. (2006, 10 5). *GIEVES AND HAWKES*. Disponivel em: http://gievesandhawkes.com/images/Buckshot%20brogue.jpg Acessado em: 20 Jan 2014.

GONZALES, R. C. (2000). *Processamento de imagens digitais* (1 ed.). São Paulo: Edgard Blucher,.

GUIMARÃES, J. (2012). *Manutenção Básica de Computadores.* IFET-RN.

HAAR, R. (1994). General Intensity Transformations and Differential Invariants. *Journal of Mathematical Imaging and Vision , 4*, 171-187.

HARRIS, C. &. (1988). A combined corner and edge detector. *Proceedings of the Alvey Vision Conference*, (pp. 147 – 151). Romsey, Reino Unido.

INPE. (2007). *Processamento Digital de Imagens: Realce de Imagens Digitais*. Disponivel em: <http://www.dpi.inpe.br/~carlos/Academicos/Cursos/Pdi/pdi_realce.html#s1_2> Acessado em: 21 Mai 2013.

ISO/IEC. (2011). Programming languages — C. *ISO/IEC 9899:201x*, p. 701.

KADIR, & BRADY. (2001). Scale Saliancy and Image Description. *International Journal of Computer Vision, 45*(2), 83-105.

KENNEDY, S. T. (2013). *F# at Microsoft Research*. Disponivel em: <http://research.microsoft.com/en-us/projects/fsharp/default.aspx> Acessado em: 20 Dez 2013.

LAGANIÈRE, R. (2011). *OpenCV 2 Computer Vision Application Programming Cookbook.* Birmingham, UK: Packt Publishing Ltd.

LINDEBERG. (1998). Feature detection with automatic scale selection. *International Journal of Computer Vision*, pp. 79 – 116.

LOWE. (1999). Object Recognition on scale invariant interest points. *International Conference on Computer Vision.* Toronto Canada.

LOWE, D. (2000). Towards a computational model for object recognition in IT cortex. *Biologically Motivated Computer Vision*, (pp. 20-31).

MABUCHI. (2014). *JC/LC-578VA - DATASHEET.* MABUCHI.

MACHADO, D. S. (2009). *Sistema de Inspeção Visual Automática Aplicado ao Controle de Qualidade de Ovos em Linhas de Produção.* CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS, Belo Horizonte.

MCHUGH, S. (2014). Cambridge In Colour - *HISTOGRAMAS - LUMINÂNCIA E COR EM HISTOGRAMAS*. Disponivel em: <http://www.cambridgeincolour.com/pt-br/tutorials/histograms2.htm> Acessado em: 20 Jan 2014.

MICROCHIP. (2013). Disponivel em: <http://www.microchip.com/stellent/idcplg?IdcService=SS_GET_PAGE&nodeId=2680&dDocName=en537044> Acessado em: 10 Jan 2014,

MICROCHIP. (2013). *32 bit PIC Microcontroller Solutions* 1 ed., Vol. 1. (MICROCHIP, Ed.) Chandler AZ USA: MICROCHIP.

MICROCHIP. (2014). *Microchip Libraries for Applications*. Disponivel em: http://www.microchip.com/pagehandler/en-us/devtools/mla/archives.html Acessado em: 07 Dez 2013,

MICROSOFT. (2013). Developer Network - *Binding Controls to Data in Visual Studio*. Disponivel em: <http://msdn.microsoft.com/en-us/library/ms171923.aspx> Acessado em: 15 Jan 2014,

MICROSOFT. (2013). Microsoft Developer Network: *Classes (C# Programming Guide)*. Disponivel em: <http://msdn.microsoft.com/en-us/library/x9afc042.aspx> Acessado em: 15 Jan 2014

MINDRU. (2004). Moment invariants for recognition under changing viewpoint and illumination. *Computer Vision and Image Understanding, 94*, 3-27.

MIYADAIRA, A. N. (2009). *Microcontroladores Pic18: Aprenda e Programe em LINGUAGEM C* 1 ed., Vol. 1. São Paulo: Erica.

MOLZ, R. F. (2001). *Uma Metodologia para o Desenvolvimento de Aplicações de Visão Computacional utilizando um projeto conjunto de Hardware e Software.* UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL, Porto Alegre.

MORDVINSTEV, A. (2014). *Introduction to SIFT (Scale-Invariant Feature Transform).* Disponivel em: OpenCV-Python Tutorials: <https://opencv-python-tutroals.readthedocs.org/en/latest/py_tutorials/py_feature2d/py_sift_intro/py_sift_intro.html?highlight=scale> Acessado em: 10 Jan 2014,

NAUGLER, D. (2007, 05). C# 2.0 for C++ and Java programmer. *Journal of Computing Sciences in Colleges*, pp. 22-27.

OPENCV. (2013). OpenCV: *About OpenCV*. Disponivel em:<http://opencv.org/about.html> Acessado em: 17 Ago 2013.

OPENCV. (2014). OpenCV 3.0.0-dev documentation*: Understanding Features*. Disponivel em: <http://docs.opencv.org/trunk/doc/py_tutorials/py_feature2d/py_features_meaning/py_features_meaning.html>Acessado em 25 Jan 2014.

ORLANDINI, G. (2012). *Desenvolvimento de Aplicativos Baseados em Técnicas de Visão Computacional para Robô Móvel Autônomo.* Piracicaba: Universidade Metodista de Piracicaba.

PEREIRA, V. F. (2012). *Reconhecimento de Objetos Baseado em Contexto Utilizando a Lógica de Descrição Probabilística CRALC.* São Paulo – SP – Brasil: Escola Politécnica — Universidade de São Paulo.

SCHILDT, H. (1998). *C++ The Complete Reference* (3 ed.). (Osborne, Ed.) McGraw-Hill.

SHAPIRO, L. G. (2001). *Computer Vision* (1 ed., Vol. 1). Prentice Hall.

STALLMAN, R. (2011). GNU Operating System: *About the GNU Operating System*. Disponivel em: <http://en.wikipedia.org/wiki/GNU_General_Public_License>, Acessado em: 16 Ago 2013.

STIVANELLO, M. E. (2004). *INSPEÇÃO INDUSTRIAL ATRAVÉS DE VISÃO COMPUTACIONAL.* UNIVERSIDADE REGIONAL DE BLUMENAU, CENTRO DE CIÊNCIAS EXATAS E NATURAIS. BLUMENAU: Disertação de Mestrado.

STROUSTRUP, B. (1994). *The Design and Evolution of C++* (1 ed.). Addison-Wesley.

STROUSTRUP, B. (2013). Stroustrup: *C++ Applications*. Disponivel em: <http://www.stroustrup.com/applications.html> Acessado em: 12 Fev 2013,

TIOBE. (2014). TIOBE Software: *TIOBE Index for January 2014*. Disponivel em: <http://www.tiobe.com/index.php/content/paperinfo/tpci/index.html> Acessado em: 10 Jan 2014.

WIKIPEDIA. (2006, 12 4). WIKIPEDIA: *Geographic coordinates sphere*. Disponivel em: <http://commons.wikimedia.org/wiki/File:Geographic_coordinates_sphere_2.svg> Acessado em: 13 Jan 2014.

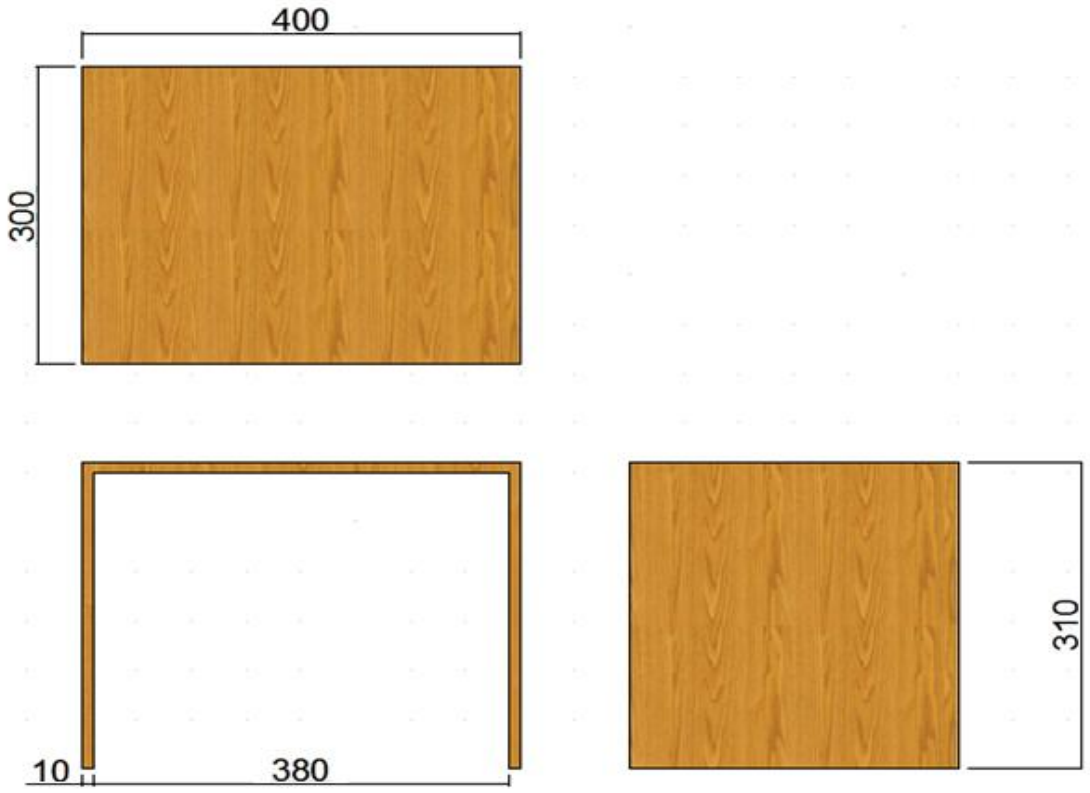
WIKIPEDIA. (2013, 05 12). Wikipedia: *Sobel Operator*. Disponivel em: <http://en.wikipedia.org/wiki/Sobel_operator> Acessado em: 15 Jan 2014.

ZAHN, C. T. (1972, 03). Fourier Descriptors for Plane Close Curves. *IEEE Trans. Computers, C-21*, pp. 269-281.

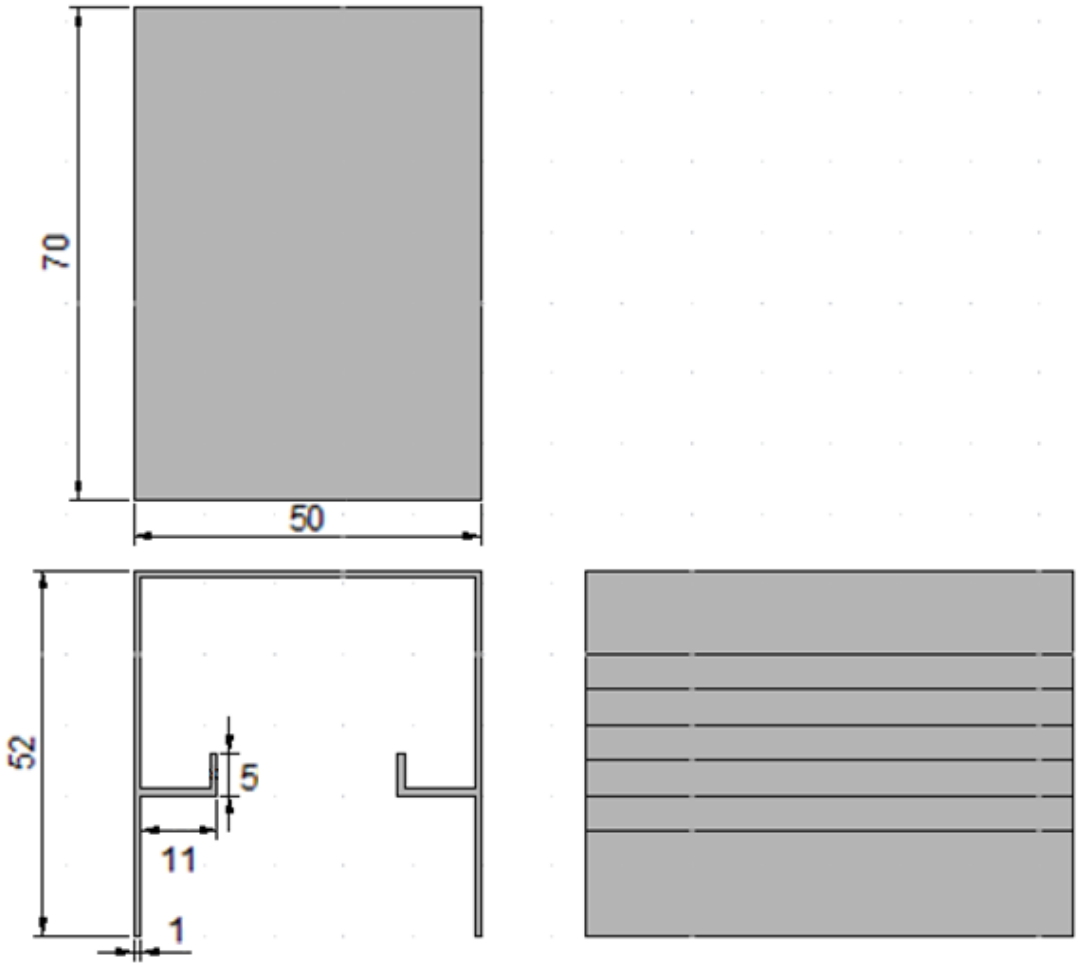
ZUECH, N. (1988). *Understanding and Applying Machihe Vision* (2 ed.). Yardley, Pennsylvania: Marcel Dekker INC.

ANEXOS

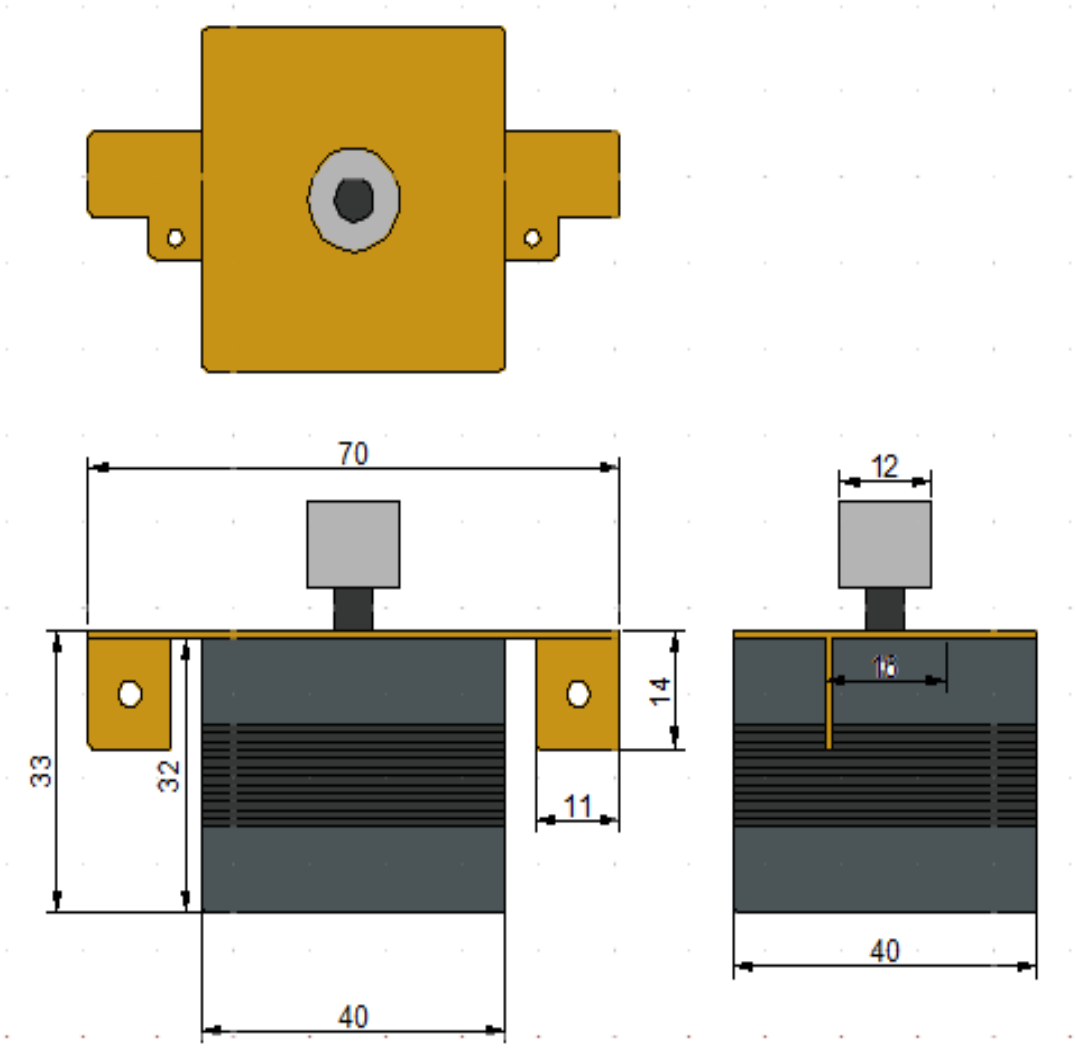
ANEXO I - Estrutura de madeira usada para sustentar a câmera



ANEXO II - Estrutura de alumínio usada para fixar o motor de passo na esteira



ANEXO III - Dimensões do motor de passo utilizado no dispositivo de rejeito



ANEXO IV - Dimensões da esteira

ANEXO V - Circuito Eletrônico simulado no programa *Proteus*