ÁREA DE INFORMÁTICA

TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS – ADS

ANDRÉ BRAZ DA SILVA, LEONARDO CAETANO DOS SANTOS

CONTROLE DE DISPOSITIVOS ROBÓTICOS USANDO VISÃO COMPUTACIONAL

TRABALHO DE CONCLUSÃO DE CURSO – TCC

CARAGUATATUBA

2014

CONTROLE DE DISPOSITIVOS ROBÓTICOS USANDO VISÃO COMPUTACIONAL

Trabalho de Conclusão de Curso apresentado como requisito parcial à obtenção do título de Tecnólogo da Área de Informática, do Instituto Federal de São Paulo.

Orientadora:
Prof.ª M.ª Juliana Matheus Grégio Pereira

Coorientador:
Prof. Dr. Ederson Rafael Wagner

CARAGUATATUBA
2014

S586c SILVA, André Braz da
Controle de dispositivos robóticos usando visão computacional / André Braz da Silva; Leonardo Caetano dos Santos. – Caraguatatuba: IFSP-CAR, 2014.
97f.

Monografia (Tecnólogo) - Instituto Federal de São Paulo, Câmpus Caraguatatuba, 2014.

1. Visão Computacional. 2. Camshift. 3. Robótica. 4. OpenCV. I. Santos, Leonardo Caetano dos. II. Título

CDD 005.3

TERMO DE APROVAÇÃO

CONTROLE DE DISPOSITIVOS ROBÓTICOS USANDO VISÃO COMPUTACIONAL

por

ANDRÉ BRAZ DA SILVA
LEONARDO CAETANO DOS SANTOS

Este Trabalho de Conclusão de Curso (TCC) foi apresentado em 15 de Dezembro de 2014 como requisito parcial para a obtenção do título de Tecnólogo em Análise e Desenvolvimento de Sistemas (ADS). Os candidatos foram arguidos pela Banca Examinadora composta pelos professores abaixo assinados, a qual após deliberação, considerou o trabalho aprovado.

Prof.ª M.ª Juliana Matheus Grégio Pereira
Orientadora

Prof.ª M.ª Luciana Brasil Rebelo dos Santos
Presidente

Prof. M.e Lucas Venezian Povoa
Membro

“O verdadeiro perigo não é que computadores começarão a pensar como homens, mas que homens começarão a pensar como computadores.”.
(HARRIS, Sydney J., 1988)

AGRADECIMENTOS - ANDRÉ

Quero primeiramente agradecer a Deus, por ser meu guia, meu amigo, meu porto seguro, meu refúgio e a força que me leva além.

Agradeço aos meus pais pelo carinho, pela paciência na minha ausência, pela educação que me deram e continuam me dando e por terem feito de tudo para que eu chegasse até aqui.

Agradeço aos meus professores, desde aqueles que me deram aula na minha infância até os de hoje, grandes mestres que me fizeram ter gosto pelo estudo. Agradeço especialmente à Prof.ª Juliana e ao Prof. Ederson, que mais do que orientar, acreditaram na nossa capacidade.

Agradeço a todos que apoiaram, que incentivaram, aos colegas de sala pelas dicas e conselhos.

AGRADECIMENTOS - LEONARDO

Primeiramente a Deus que permitiu que tudo isso fosse possível, não apenas nesse tempo como estudante, mas que sempre me ajudou a superar as dificuldades, se mostrando um grande mestre.

A todos os professores que me inspiraram e me proporcionaram o conhecimento racional e de caráter e por não terem apenas me ensinado, mas por terem me feito aprender.

Aos meus pais pelo amor, carinho e incentivo direcionado a mim por palavras e abraços desde minha infância até o presente momento.

E a todos que direta e indiretamente participaram da minha estrada, me orientando e incentivando a seguir em frente, o meu muito obrigado.

RESUMO

Este trabalho de conclusão de curso (TCC) tem como finalidade apresentar um sistema que através da Visão Computacional, possa identificar eventos e, sem a intervenção humana, tomar decisões utilizando dispositivos robóticos conectados a um computador. O sistema identifica os eventos por uma câmera convencional, capturando-os quadro a quadro, e processando-os por meio da biblioteca de Visão Computacional OpenCV (*Open Source Computer Vision*) que é distribuída gratuitamente e possui documentação farta na internet, com exemplos e aplicações práticas. Foi estudado o algoritmo Viola-Jones para fazer detecção de faces e o algoritmo *Camshift* para rastreamento de objetos. Uma face é detectada, gerando uma região de interesse que envia para o algoritmo *Camshift* a posição do centro desta região nos quadros do vídeo capturado, caracterizando um rastreamento. Os dados de entrada do controlador são resultados obtidos pelos algoritmos de Visão Computacional e este sistema de controle manipula os dispositivos robóticos.

Palavras-chave: Visão Computacional. *Camshift*. Robótica. OpenCV.

ABSTRACT

This conclusion work aims to introduce a system which through computer vision, can identify events and, without human intervention, make decisions using robotic devices connected to a computer. The system identifies the events by a conventional camera, capturing them frame by frame, and processing them through the computer vision library OpenCV (Open Source Computer Vision) which is distributed for free and has done internet documentation with examples and practical applications. It was studied Viola-Jones algorithm to make tracking and the algorithm for object tracking Camshift. A face is detected, generating a region of interest that it sends to the Camshift algorithm the center position of this region in the frames of the captured video, featuring a trace. The input data of the controller are results obtained by computer vision algorithms and this control system manipulates the robotic devices.

Keywords: Computer Vision. Camshift. Robotics. OpenCV.

LISTA DE FIGURAS

Figura 1 – Digitalização de Imagens 20
Figura 2 – Técnicas de Processamento de Imagens 21
Figura 3 – Espaço de Cores RGB 21
Figura 4 – Espaço de Cores HSV 22
Figura 5 – Exemplo de Histograma 23
Figura 6 – Sequência de eventos Camshift 26
Figura 7 – Exemplo de classificadores Haar 27
Figura 8 – Exemplo de filtro de cascata 27
Figura 9 – Exemplo de Sistema de Malha Aberta 28
Figura 10 – Exemplo de Sistema de Malha Fechada 28
Figura 11 – Diagrama do controle PID 30
Figura 12 – Sistema Robótico 31
Figura 13 – Exemplo de Atuador - Braço Robótico 33
Figura 14 – Componentes internos de um Servomotor 34
Figura 15 – Exemplo de controle por PWM em Servomotor 36
Figura 16 – *Arduino Uno* 37
Figura 17 – Exemplo de Monitor Serial 38
Figura 18 – Esquema de *Arduino* com Servomotor 39
Figura 19 – *Raspberry Pi* Modelo B 39
Figura 20 – Esquema completo *Raspberry Pi* Modelo B 41
Figura 21 – Área de Trabalho do Raspibian 42
Figura 22 – Montagem dos Equipamentos 47
Figura 23 – Diagrama das partes do Desenvolvimento 48
Figura 24 – Tela de Captura de Imagem 49
Figura 25 – (a) Detecção da face frontal (b) Não detecção em perfil 50
Figura 26 – Seleção da Região de Interesse 51
Figura 27 – Seleção da região de interesse e Histograma 51
Figura 28 – Imagem resultante da função *backprojection* 52
Figura 29 – Retângulo formado sobre objeto ratreado 52
Figura 30 – Conexão do *Arduino* com Servomotores 56
Figura 31 – (a) Detecção de faces (b)Interferência de Detecção de Faces 58
Figura 32 – (a) Camshift sem interferência (b) Camshift com interferência 59
Figura 33 – Exemplo de Margem de erro com Setpoint 60

LISTA DE QUADROS

Quadro 1 – Especificações do servomotor Tower Pro 35
Quadro 2 – Especificações do *Arduino Uno* 37
Quadro 3 – Especificações ASUS A45A 44
Quadro 4 – Efeito de cada parâmetro PID sobre o processo 53

LISTA DE TABELAS

Tabela 1 – Testes de Rastreamento com imagens de resolução 640x480 57
Tabela 2 – Testes de Rastreamento com imagens de resolução 320x240 58

LISTA DE SIGLAS

ASCII	*American Standard Code for Information Interchange*
BGR	*Blue Green Red*
CSI	*Camera Serial Interface*
DC	*Direct Current*
DSI	*Display Serial Interface*
GPU	*Graphics Processing Unit*
HD	*Hard Disk*
HDMI	*High-Definition Multimedia Interface*
HSV	*Hue Saturation Value*
I^2C	*Inter-Integrated Circuit*
I^2S	*Integrated Interchip Sound*
I/O	*Input/Output*
IDE	*Integrated Development Environment*
IFSP	Instituto Federal de São Paulo
IP	*Internet Protocol*
JTAG	*Joint Test Action Group*
PDF	*Probability Distribution Image*
PID	Proporcional Integral Derivativo
PWM	*Pulse Width Modulation*
RCA	*Radio Corporation of America*
RGB	*Red Green Blue*
SD	*Secure Digital*
SPI	*Serial Peripheral Interface*
UART	*Universal Asynchronus Receiver/Transmitter*
USB	*Universal Serial Bus*

LISTA DE ABREVIATURAS

bit	*Binary Digit*
cm	Centímetro
fps	*Frame per Second*
g	Grama
GB	*Gigabyte*
GND	*Ground*
GPIO	*General Purpose Input/Output*
KB	*Kilobyte*
kgf	Quilograma-Força
kOhms	*Kiloohms*
mA	*Miliampère*
MB	*Megabyte*
MHz	*Megahertz*
mm	Milimetro
p	*Progressive Scan*
RJ	*Registered Jack*
seg	Segundo
TRS	*Tip-Ring-Sleeve*
V	*Volt*

LISTA DE ACRÔNIMOS

3D	Três Dimensões
ARM	*Advanced RISC Machine*
LAN	*Local Area Network*
MIPI	*Mobile Industry Processor Interface*
OpenCV	*Open Source Computer Vision*
pixel	*Picture Element*
RAM	*Random Access Memory*
ROI	*Region Of Interest*
SOC	*System On a Chip*
XML	*eXtensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO 17
1.1 OBJETIVOS 18
1.2 ORGANIZAÇÃO DO TEXTO 18
2 REVISÃO BIBLIOGRÁFICA 19
2.1 VISÃO COMPUTACIONAL 19
2.1.1 Estrutura de um Sistema de Visão Computacional 20
2.2 IMAGEM DIGITAL 20
2.2.1 Processamento de Imagem Digital 21
2.2.2 RGB e BGR 21
2.2.3 HSV 22
2.2.4 Histograma 22
2.3 OPENCV 22
2.3.1 Rastreamento de Objetos 23
2.3.1.1 *Camshift* 24
2.3.1.2 *Viola-Jones* 26
2.4 CONTROLE 27
2.4.1 Controle Proporcional 29
2.4.2 Controle Integral 29
2.4.3 Controle Derivativo 29
2.4.4 Controle Proporcional-Integral-Derivativo 30
2.5 DISPOSITIVOS ROBÓTICOS 31
2.5.1 Sensores 32
2.5.1.1 Câmera 32
2.5.2 Atuadores 33
2.5.2.1 Servomotores 34
2.6 *ARDUINO* 36
2.6.1 Comunicação Serial 37
2.6.2 Comunicação com Dispositivos Robóticos 38
2.7 *RASPBERRY PI* 38
2.7.1 Raspbian 41
2.8 TRABALHOS RELACIONADOS 42
2.8.1 Detecção e Acompanhamento de Movimento Através de uma Câmara de Vídeo 42
2.8.2 Rastreamento em Tempo Real com Câmera de Pan-Tilt . 43
2.8.3 Sistema de Captura de Movimentos para Comandar Equipamentos Eletrônicos 43
3 DESENVOLVIMENTO 44
3.1 ESTRUTURA DE *SOFTWARE* 44

3.1.1 CONFIGURAÇÕES NO LINUX 45
3.1.2 CONFIGURAÇÕES NO WINDOWS 45
3.2 ESTRUTURA DE HARDWARE 46
3.3 FLUXO DO SISTEMA 47
3.3.1 CAPTURA DA IMAGEM 47
3.3.2 SELEÇÃO DA REGIÃO DE INTERESSE 49
3.3.2.1 DETECÇÃO DE FACE 49
3.3.2.2 SELEÇÃO COM O MOUSE 50
3.3.3 RASTREAMENTO 50
3.3.4 CONTROLE 53
3.3.5 ENVIO DA DADOS PARA O *Arduino* VIA SERIAL 54
3.3.5.1 COMUNICAÇÃO NO RASPBIAN 54
3.3.5.2 COMUNICAÇÃO NO MICROSOFT WINDOWS 54
3.3.5.3 MÉTODO DE ENVIO DOS DADOS 54
3.3.6 AÇÃO DOS DISPOSITIVOS ROBÓTICOS 55
3.4 RESULTADOS 55
3.4.1 TESTES DE CONTROLE – *Raspberry Pi* 56
3.4.2 TESTES DE CONTROLE – NOTEBOOK 57
3.4.3 TESTES DE DETECÇÃO DE FACES 58
3.4.4 TESTES DE RASTREAMENTO 59
3.4.5 TESTES DE PERDA DE RASTREAMENTO 59
3.4.6 TESTES DE ESTABILIDADE DOS SERVOS 59
4 CONCLUSÃO 61
4.1 LIMITAÇÕES 61
4.2 TRABALHOS FUTUROS 62

REFERÊNCIAS 63
ANEXO A INSTRUÇÕES DE INSTALAÇÃO DO OPENCV NAS DISTRIBUIÇÕES LINUX BASEADOS EM DEBIAN 68
ANEXO B INSTRUÇÕES DE INSTALAÇÃO DO OPENCV NO WINDOWS 7 . 70
ANEXO C TABELA ASCII 71
ANEXO D CÓDIGO FONTE DA BIBLIOTECA ESERIAL.H - LINUX 75
ANEXO E CÓDIGO FONTE DA BIBLIOTECA SERIAL.CPP - WINDOWS . . . 79
ANEXO F CÓDIGO FONTE DA BIBLIOTECA SERIAL.H - WINDOWS 83
APÊNDICE A CÓDIGO FONTE DO SISTEMA DA CÂMERA SEGUIDORA - WINDOWS(OPÇÕES PARA LINUX EM COMENTÁRIOS) 85
APÊNDICE B CÓDIGO DO *ARDUINO* 97

1 INTRODUÇÃO

A visão computacional é a ciência que emula a visão humana, possibilitando as máquinas enxergarem e interagirem com o ambiente externo. É a mais importante tecnologia para o futuro de desenvolvimento de robôs inteligentes(BUDIHARTO; GROUP, 2014).

Essa área se concentra no campo de inteligência artificial e aquisição e processamento de imagens, ou seja, é o conceito de ensinar as máquinas a como enxergar(BUDIHARTO; GROUP, 2014).

Atualmente, são utilizados *mouses*, teclados, telas *touchscreens*, entre outros periféricos para se interagir com um computador. Porém, com o crescimento da visão computacional, graças aos computadores mais potentes, às câmeras melhores e de custo acessível e aos algoritmos de visão começarem a amadurecer(BRADSKI; KAEHLER, 2013), a tendência da Interação Homem-Computador é diminuir, cada vez mais, os fios e os toques, os dispositivos anteciparão suas necessidades de informação, numa interação mais natural e intuitiva (GARBIN, 2010).

O videogame *Microsoft XBox 360* com seu *Kinect* já demonstra como será esse futuro, permitindo a comunicação total com alguns jogos usando o próprio corpo como controle(MICROSOFT, 2014a). Além da área de entretenimento, a segurança também é uma área onde a visão computacional vem ganhando espaço, desde a identificação de rostos de pessoas em filmagens de câmeras de segurança, até mesmo em sistemas de desbloqueio facial em celulares e *tablets*(RENATO, 2014). Na Medicina também é muito utilizada para reconhecimento de padrões em imagens(BRADSKI; KAEHLER, 2013).

Além desses campos de aplicação da Visão Computacional, um outro campo onde ela pode ser aplicada é a robótica. Para os sistemas robóticos, a visão é o sensor mais importante, pois através dela é possível obter mais informações e estas são mais detalhadas do que em outros tipos de sensores(BUDIHARTO; GROUP, 2014). É possível fazer com que um robô desvie de obstáculos, siga um trajeto, identifique pessoas, expressões e objetos, entre outras aplicações.

Para desenvolver sistemas robóticos baseados em visão computacional, não é necessário desembolsar grandes quantidades de dinheiro, isso porque existem meios de baixo custo para alcançar esse objetivo. Um exemplo é a biblioteca de Visão Computacional OpenCV (*Open Source Computer Vision*), que é disponibilizada gratuitamente e é compatível com *Linux*(ITSEEZ, 2014). Além do *software*, o *hardware*, também apresenta opções de baixo custo, como a plataforma de prototipagem eletrônica livre *Arduino*(ARDUINO, 2014c).

1.1 Objetivos

O objetivo deste trabalho é o desenvolvimento de um sistema de visão computacional, que a partir do algoritmo de rastreamento *Camshift*, possa controlar dispositivos robóticos. Visto que sistemas de interação homem-computador são incomuns ou inacessíveis à maioria das pessoas, esse sistema foi testado em um *notebook* e também em um *Raspberry Pi* para verificar os limites de processamento.

A aplicação prática desse sistema é denominada câmera seguidora, cujo objetivo é detectar um objeto de interesse a partir de uma câmera convencional e rastreá-lo, de forma que a câmera, sendo movida horizontalmente e verticalmente por servomotores, acompanhe a trajetória desse objeto de interesse, fazendo com que este fique posicionado no centro da imagem capturada, mesmo que esteja em movimento.

1.2 Organização do Texto

No capítulo 2 será apresentado a revisão bibliográfica que contém os conceitos dos principais tópicos desse trabalho e também os trabalhos relacionados. No capítulo 3 é apresentado o desenvolvimento do trabalho, mostrando a estrutura utilizada para o desenvolvimento do sistema e os resultados obtidos. No capítulo 4 são apresentados as conclusões, as limitações encontradas e os trabalhos futuros.

2 REVISÃO BIBLIOGRÁFICA

Neste capítulo são apresentados os conceitos dos principais tópicos para o desenvolvimento do trabalho, abordando principalmente a Visão Computacional, o Controle PID e os dispositivos robóticos. Ao final do capítulo são apresentados uma breve análise de trabalhos relacionados que utilizam conceitos semelhantes aos apresentados nesse trabalho.

2.1 Visão Computacional

O sentido da visão pode ser considerado o meio pelo qual o ser humano pode obter mais informações do ambiente a sua volta(MOLZ, 2001).

O olho humano consegue perceber e identificar objetos com grande velocidade, graças ao córtex visual, uma das partes mais complexas do cérebro humano. Alguns cientistas concentram seus estudos na tentativa de entender o funcionamento dessa parte do cérebro para então trazer tais ideias para a Visão Computacional(MILANO; HONORATO, 2010).

A Visão Computacional é a emulação da visão humana no computador, ou seja, fazer com que a partir de imagens captadas de uma câmera, o computador possa pensar e tomar decisões sobre aquilo que está vendo. A quantidade de informações que podem ser obtidas através de Visão Computacional, assim como na visão humana, é muito grande, gerando muitas possibilidades, por essa razão é a tecnologia mais importante no futuro em desenvolvimento de interatividade de robôs(BUDIHARTO; GROUP, 2014).

Estas muitas possibilidades podem ser divididas em três níveis de acordo com os seus propósitos(GONZALEZ; WOODS, 2006):

Processamento de nível baixo: Operações primitivas como processamento de imagens para reduzir ruídos, melhoramento de contraste, nitidez da imagem, etc. A principal característica é que tanto a entrada como a saída são imagens.

Processamento de nível intermediário: Envolve tarefas como segmentação(particionar uma imagem em regiões ou objetos), descrição de objetos para reduzí-los para uma forma adequada para o processamento do computador e reconhecimento de objetos individuais. A principal característica é que a entrada geralmente é uma imagem, mas as saídas são atributos extraídos destas imagens.

Processamento de nível alto: Envolve funções cognitivas normalmente associados com a visão, como análise de imagens, reconhecimento e rastreamento de objetos ou faces e funções semelhantes.

2.1.1 Estrutura de um Sistema de Visão Computacional

A Visão Computacional ou de máquina, compreende técnicas e métodos para manipulação de imagem. A imagem é capturada por uma câmera, características são extraídas e decisões são tomadas através dessas informações(BECKER, 2013).

O sistema de visão é formado por(FACON, 2005):

Sensores de visão Realiza a captura da imagem (câmeras, sensores, etc.).

Hardware de digitalização de imagens Cria uma imagem digitalizada, representada por uma matriz de números, cujo elementos são chamados *pixels* (*Picture Element*).

Computador Processa a imagem e possibilita a escolha da informação adequada nas imagens, a ser interpretada e a partir da qual serão tomadas as decisões.

2.2 Imagem Digital

Como mencionado no capítulo anterior, as imagens são capturadas e passam por um processo de digitalização, ou seja, onde um ser humano vê uma imagem comum, o computador vê uma matriz de números(BRADSKI; KAEHLER, 2013).

Na figura 1 é apresentado um exemplo em (a) de uma imagem binarizada digitalizada, onde os *pixels* têm valores 0 ou 1, em (b) é apresentado uma imagem em escala de cinza, onde os números representam a intensidade de luz, 0 para o valor mais escuro e 255 para o valor mais claro, em (c) é apresentado uma imagem colorida, que é representada por três valores, que nos monitores mais comuns são RGB (*Red Green Blue*), que serão apresentados em 2.2.2.

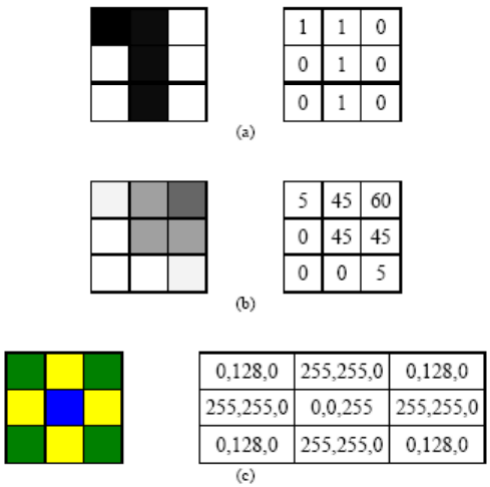

Figura 1 – Digitalização de Imagens

Fonte**:** Oshiro e Goldschmidt (2008)

2.2.1 Processamento de Imagem Digital

O processamento de imagem envolve técnicas aplicadas em determinadas imagens ou vídeos para a extração de informações que são divididas em técnicas de análise e de melhoria. Segundo Manzi (2007), as técnicas de análise estão relacionadas à parametrização e descrição das informações dentro da imagem, e a de melhorias relaciona-se ao tratamento utilizado para corrigir defeitos ou ressaltar informações a fim de facilitar a análise.

Na figura 2 é apresentado um exemplo de processamento de imagens com técnica de melhoria para deixar a imagem mais nítida, e um exemplo de técnica de análise para identificar o número da placa e o modelo do carro.

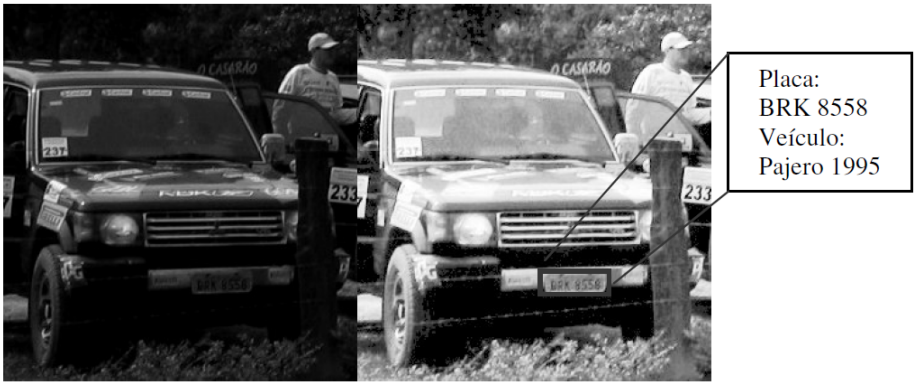

Figura 2 – Técnicas de Processamento de Imagens

Fonte**:** Marengoni e Stringhini (2009)

2.2.2 RGB e BGR

O modelo RGB é formado pelas cores vermelha, verde e azul e se baseia em um sistema de coordenadas cartesianas(BECKER, 2013). É o mais utilizado em equipamentos eletrônicos, mas por possuir um espaço de cor uniforme, não se faz eficiente quando é necessário a seleção de áreas de interesse(SOUSA, 2013). Na figura 3 é apresentado visualmente o conceito do RGB.

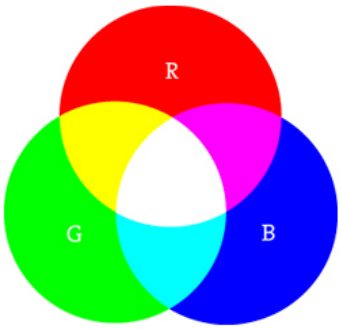

Figura 3 – Espaço de Cores RGB

Fonte**:** Adaptado de Leparmentier (2014)

O OpenCV utiliza o modelo BGR (*Blue Green Red*) que possui similaridades com o RGB, porém a ordem das cores é inversa, sendo esta azul, verde e vermelho. O motivo é pelo simples fato desse espaço de cores ter uma leitura mais rápida na memória.(BRADSKI; KAEHLER, 2013).

2.2.3 HSV

O HSV (*Hue Saturation Value*) possui os componentes matiz, saturação e brilho. A matiz representa a cor da imagem, a saturação mostra a distância da cor e o brilho mostra a proporção de luz refletida pelo objeto(RÉZIO, 2008). A figura 4 é uma representação de como funciona o espaço de cores HSV.

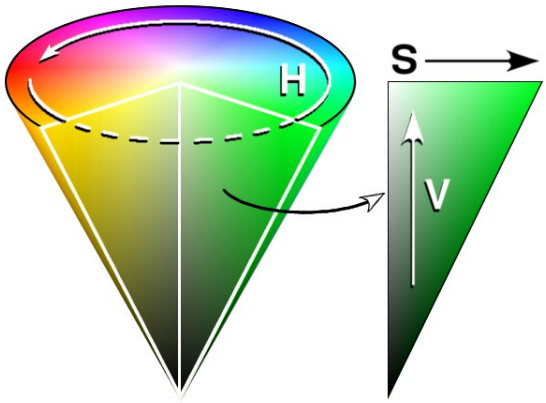

Figura 4 – Espaço de Cores HSV
Fonte**:** Wikimedia (2014)

2.2.4 Histograma

Oshiro e Goldschmidt (2008) define o histograma da seguinte maneira:

> Um histograma em processamento de imagens é semelhante a um gráfico de frequências, onde cada pilha representa uma cor identificada em um *pixel* de uma imagem. É muito utilizado para cálculos estatísticos em imagens e recuperação de imagens(OSHIRO; GOLDSCHMIDT, 2008).

Uma exemplo é representado na figura 5, onde em (a) é representado uma imagem colorida qualquer e em (b) o seu histograma correspondente. Pode-se perceber que de acordo com a quantidade de *pixels* de uma determinada cor, o histograma é montado, gerando assim um gráfico de frequência de cores em uma imagem.

2.3 OpenCV

A OpenCV é uma biblioteca de Visão Computacional que foi iniciada pela *Intel* em 1998 e se tornou uma das ferramentas mais usadas na indústria de Visão Computacional(ITSEEZ, 2014). A empresa liberou no ano de 2000 para a comunidade *Open*

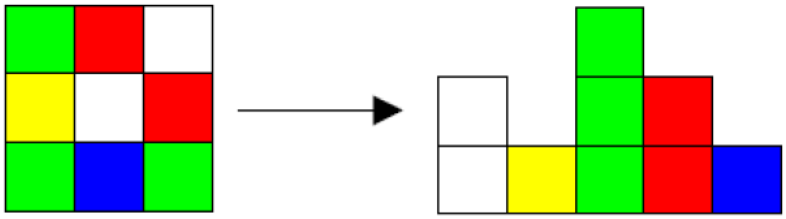

Figura 5 – Exemplo de Histograma
Fonte**:** Oshiro e Goldschmidt (2008)

Source uma versão *beta* da biblioteca(SILVA, 2011). Depois de se tornar de código aberto, OpenCV passou por vários anos em desenvolvimento ativo na *Willow Garage* e *Itseez*, e agora é mantida pela Fundação OpenCV(BRADSKI; KAEHLER, 2013).

OpenCV é escrita em C e C++ e possui compatibilidade com *Windows*, *Linux*, *Mac OS X*, *iOS*, e *Android*. Possui um desenvolvimento ativo a fim de dar suporte para programadores de *Python*, *Ruby*, *Matlab*, entre outras linguagens(SOUZA, 2011).

É formada por cinco grupos de funções. E são eles(MARENGONI; STRINGHINI, 2009):

- Processamento de Imagens
- Análise Estrutural
- Análise de Movimento e Rastreamento de Objetos
- Reconhecimento de Padrões
- Calibração de Câmera e Reconstrução de três dimensões (3D)

2.3.1 Rastreamento de Objetos

O rastreamento consiste em reconhecer um padrão em uma sequência de imagens (vídeo) e localizar esse padrão na imagem, rastreando sua posição(MARENGONI; STRINGHINI, 2009). Ainda segundo Marengoni e Stringhini (2009) uma das áreas que mais despertam estudos em Visão Computacional, sendo um dos grandes motivos o rastreamento do movimento humano, que permite, por exemplo, uma maior Interação Humano-Computador.

No seção abaixo, uma explicação de uma técnica de rastreamento, a *Camshift*, que foi utilizada nesse projeto e após, uma breve explicação sobre o algoritmo de detecção de faces *Viola-Jones* (mais detalhes em (VIOLA; JONES, 2001)).

2.3.1.1 *Camshift*

O *Camshift* é uma função do *OpenCV* para acompanhamento de objetos em uma sequência de quadros(SOUZA, 2011). Através dessa técnica é possível rastrear cores de objetos e definir padrões usando cores como características padronizadas(FERNANDES, 2012).

O *Camshift* é derivado do algoritmo *Meanshift*, este que foi criado para encontrar picos em uma distribuição de probabilidade(BRADSKI; KAEHLER, 2013).

O princípio de funcionamento baseia-se na procura do valor máximo local da densidade de probabilidade de uma variável. Implementa uma análise num subconjunto de valores determinado por uma janela em torno de um ponto de partida definido. Encontra o maior valor dentro desta janela, desloca a janela para o ponto determinado anteriormente e o processo é repetido (PEIXOTO, 2012).

Como a distribuição de probabilidade de um objeto em uma cena de vídeo pode mudar e se mover dinamicamente com o decorrer do tempo, Bradski (1998) modificou tal algoritmo para que este fosse robusto a tais alterações, portanto o *Camshift* é capaz de seguir uma distribuição de probabilidade que se altere dinamicamente(CAMPOS, 2011). Resumindo, o *Meanshift* é para rastreamento em imagens estáticas, enquanto *Camshift* é para rastreamento em vídeos, ambientes que mudam constantemente.

As etapas do *Camshift* são as seguintes:

Criação de Histograma: É obtida uma região de interesse que é a região da imagem capturada onde o objeto a ser rastreado se encontra. Para produzir a distribuição de probabilidade do objeto é criado um histograma (seção 2.2.4). A informação que é distribuída no histograma são os valores do canal Matiz da região de interesse.

Backprojection: De acordo com Peixoto (2012), para a utilização do algoritmo *Camshift* é necessário o cálculo da PDF (*Probability Distribution Image*) e esta pode ser determinada utilizando qualquer método que associe o valor de um *pixel* com a probabilidade deste *pixel* pertencer ao objeto.

Um destes métodos é chamado histograma projeção de fundo (*Histogram Back-Projection*). Essa técnica recorre a comparação de histogramas, ou seja, à relação entre o histograma da região de interesse e o histograma da imagem capturada. Desta forma pretende-se aumentar a diferença entre o *background* e o objeto, levando a uma localização mais confiável deste (PEIXOTO, 2012).

Cálculo do centro da região rastreada: Para viabilizar o rastreamento com o *Camshift* é ainda necessário que o tamanho da região de interesse se adapte ao alvo que está

sendo rastreado, uma vez que o tamanho ideal da região varia conforme o objeto se encontra mais perto ou mais afastado da câmera (RÉZIO, 2008).

Para isso é necessário primeiramente encontrar o momento de ordem zero que pode ser interpretado como a área da região de interesse(RÉZIO, 2008).

Com esse cálculo é possível encontrar a janela da área rastreada, bem como o centro dessa área.

Dado que $I(x, y)$ é a intensidade de probabilidade discreta em (x, y) dentro da região de interesse. O momento de ordem zero é dado pela equação:

$$M_{00} = \sum_{x} \sum_{y} I(x, y) \tag{2.1}$$

O primeiro momento para x e y:

$$M_{10} = \sum_{x} \sum_{y} xI(x, y); M_{01} = \sum_{x} \sum_{y} yI(x, y) \tag{2.2}$$

A localização da área de interesse é dada por:

$$x_c = \frac{M_{10}}{M_{00}}; y_c = \frac{M_{01}}{M_{00}} \tag{2.3}$$

Onde os valores de x_c e y_c são calculados até que não haja deslocamento significativo. Então a área de rastreamento, através desses cálculos, é obtida.

Cálculo da janela (ou marcação) da região rastreada: Na última etapa ocorre o cálculo do tamanho e o ângulo do objeto para cada mudança na marcação da região de interesse. Isso é realizado analisando a escala e a orientação que melhor se adequa aos *pixels* da região de alta probabilidade da nova região de interesse marcada como objeto. Essa etapa é exclusiva do *Camshift* se comparado com o *MeanShift*, sendo uma das diferenças que são notáveis quando se observa o resultado do rastreamento (CAMPOS, 2011).

Como resultado desse cálculo o *Camshift* gera um objeto da classe *RotatedRect*, um retângulo que pode ser rotacionado. Esse retângulo está rotacionado de acordo com as probabilidades encontradas na imagem. A função *ellipse* recebe esse objeto e é desenhada na imagem, no local do objeto rastreado.

Na figura 6 é apresentado a sequência de eventos para rastreamento de uma região de interesse, onde é iniciado com a estimação da região de interesse (ROI (*Region Of Interest for ellipse estimation*)) e termina no cálculo da ellipse (*Ellipse calculation*).

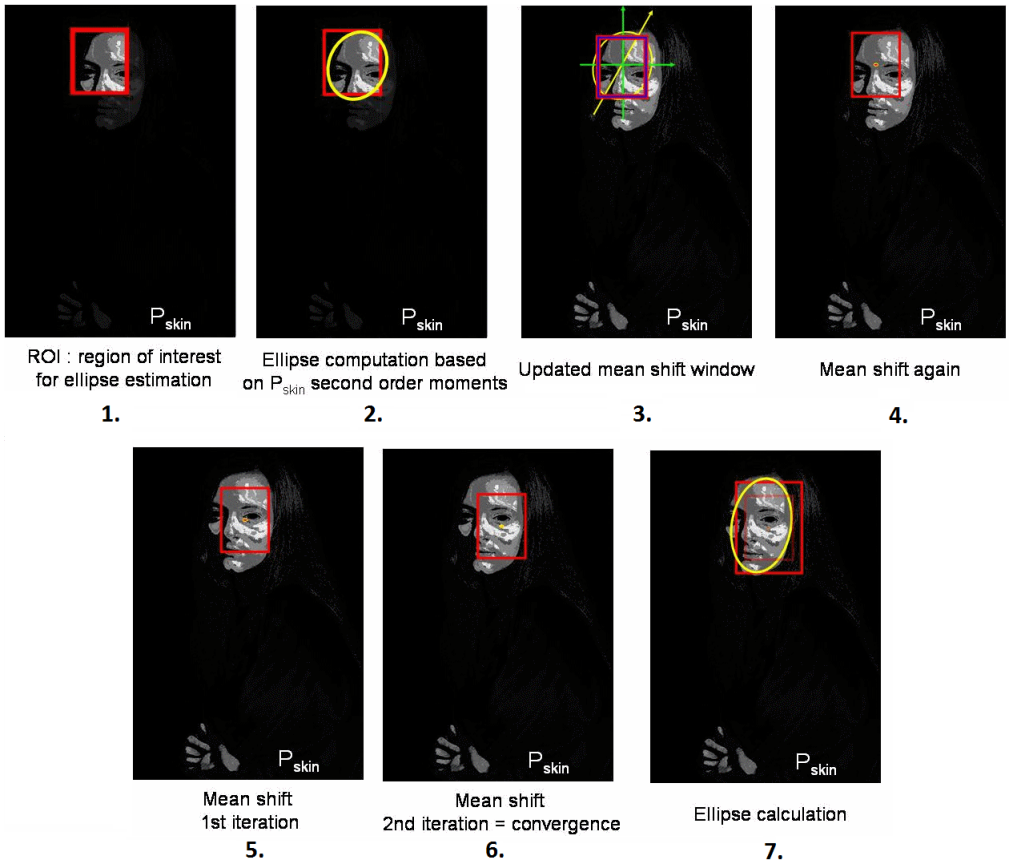

Figura 6 – Sequência de eventos Camshift

Fonte**:** adaptado de OpenCV (2014c)

2.3.1.2 *Viola-Jones*

(CAMPOS, 2011) explica o algoritmo *Viola-Jones* da seguinte maneira:

> *Viola-Jones* (VIOLA; JONES, 2001) é um método de detecção de faces que através de um aprendizado automático, aplica formas de detectar o objeto de desejo com muita rapidez. As três contribuições principais que o distingue dos outros algoritmos e o torna computacionalmente mais eficiente, são elas: a imagem integral, o algoritmo de aprendizado baseado em *AdaBoost* e um método para combinar classificadores de complexidade crescente baseados em *Haar-like features* para criar um filtro em cascata eficiente(CAMPOS, 2011).

Nessa técnica apresentada acima, são utilizadas características retangulares, chamadas de classificadores *Haar*, que possuem o formato em quadrado e duas dimensões, onde um lado é claro e o outro é escuro (figura 7).

Segundo Silva (2011), a presença de uma característica *Haar* é determinada subtraindo o valor médio do *pixel* da região escura, do valor médio do *pixel* da região

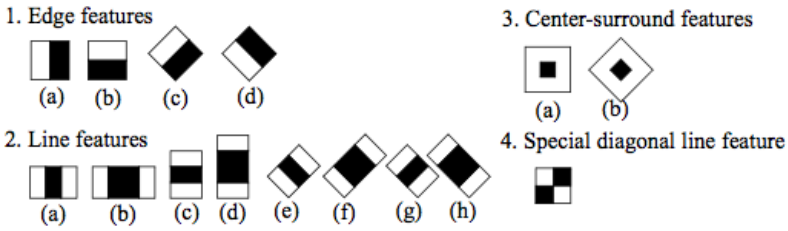

Figura 7 – Exemplo de classificadores Haar

Fonte**:** Lienhart, Kuranov e Pisarevsky (2003)

clara. Se a diferença estiver acima de um limite (estabelecido durante o processo de aprendizagem), a característica é dada como presente.

Para uma melhor eficácia na detecção de diversas características, é aplicada a técnica Integral de Imagem, onde para se obter o valor integral de um determinado *pixel*, é preciso realizar a soma de todos os *pixels* que estão a esquerda e acima(VIOLA; JONES, 2001).

Para se obter um bom resultado na aprendizagem de máquina, é utilizado o método *AdaBoost* (VIOLA; JONES, 2001), que combina classificadores denominados fracos para se obter classificadores fortes. Os classificadores fracos são combinados e é atribuído peso para cada um deles. Essa associação ajustada é denominada como classificador forte (SILVA, 2011).

Viola-Jones agrega os classificadores fortes, formando um filtro. Se houver falha em um filtro, o local é marcado como *"false"*, mas se a região passar por todos os filtros é classificada como *"true"* (figura 8), assim é gerado um documento no formato XML (*eXtensible Markup Language*) que contém todas essas características identificadas.

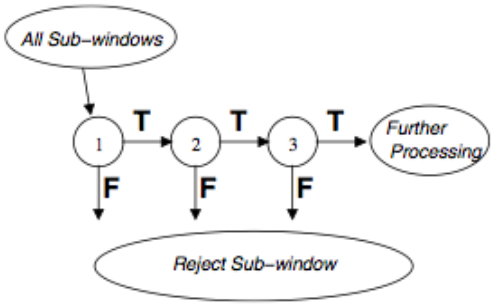

Figura 8 – Exemplo de filtro de cascata

Fonte**:** Viola e Jones (2001)

2.4 Controle

Um sistema de controle é uma interconexão de componentes ou processos configurada de forma a produzir uma determinada resposta (OLIVEIRA, 2013).

De modo geral, objetivam manter um sinal de saída em um comportamento pré-especificado a partir da aplicação de sinais adequados na entrada de controle, além de busca a minimização ou eliminação total dos efeitos causados pela ação de pertubações sobre o comportamento do sinal de saída (OGATA, 2010). A realimentação de um sistema de controle, o caracteriza como um sistema de malha fechada, enquanto a não realimentação, caracteriza o sistema de controle como um sistema de malha aberta (OLIVEIRA, 2013). Na figura 9 é apresentado um exemplo genérico de um sistema de malha aberta.

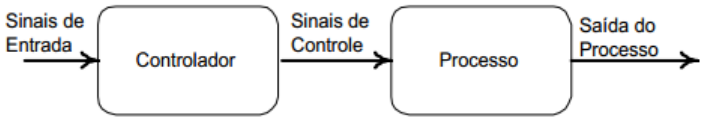

Figura 9 – Exemplo de Sistema de Malha Aberta

Fonte**:** Fuentes (2005)

Em um sistema de malha aberta os sinais de entrada são estabelecidos de forma que o sistema proporcione a saída desejada(FUENTES, 2005). Porém, o sistema não é realimentado com o estado de saída do processo. Na figura 10 há um exemplo genérico de um sistema de malha fechada que realiza a realimentação do sistema.

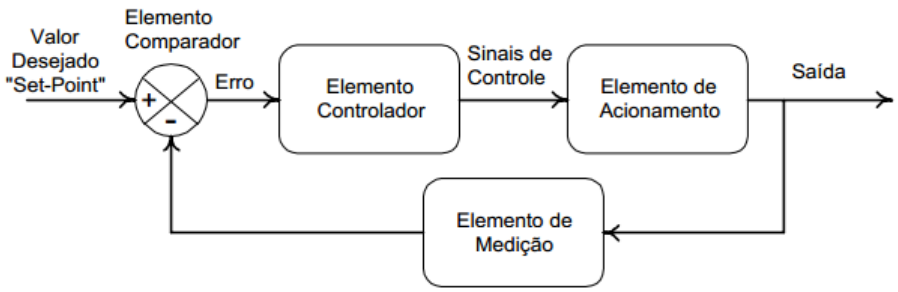

Figura 10 – Exemplo de Sistema de Malha Fechada

Fonte**:** Fuentes (2005)

Em um sistema de malha fechada, a grandeza a ser controlada é monitorada através de um elemento de medição. Esta informação(Variável de Processo - PV) é comparada com o valor desejado (*setpoint* - SP) através de um elemento comparador produzindo-se um sinal de erro (e). Este sinal de erro é processado pelo elemento controlador gerando sinais de controle que atuarão sobre o processo. Os elementos atuadores do processo agirão de tal maneira que, sejam corrigidos eventuais desvios, causados por modificações nas condições de operação, ou pertubações no processo(FUENTES, 2005). A equação 2.4 apresenta o cálculo do erro.

$$e = SP - PV \tag{2.4}$$

O Elemento Controlador é o responsável por manipular o sinal de erro de forma a modificar o sistema e fazer com que ele apresente as características de desempenho desejadas(OLIVEIRA, 2013). Existem várias formas de se realizar tal manipulação, uma delas é o controlador PID (Proporcional-Integral-Derivativo).

2.4.1 Controle Proporcional

A ação de controle proporcional, como o próprio nome sugere, apresenta um sinal resultante do controle proporcional ao sinal de erro(OGATA, 2010). Assim quanto menor o erro, menor o sinal resultante e quanto maior o erro, maior o sinal resultante. A equação 2.5 apresenta o cálculo da ação de controle proporcional, onde $u(t)$ é o sinal de controle gerado, K_p é a constante proporcional, e $e(t)$ é o sinal de erro entre o *setpoint*(SP) e a Variável de Processo (PV).

$$u(t) = K_p e(t) \tag{2.5}$$

2.4.2 Controle Integral

A ação de controle integral é proporcional à integral do sinal de erro atual(MOORE, 1999). Isoladamente não é considerado uma técnica de controle por não ser empregado separado da ação proporcional(OGATA, 2010).

De acordo com o histórico do sinal de erro, a integral elimina o erro, ou seja, deixa o seu valor em 0. Entretanto, ela reduz a estabilidade do sistema(MOORE, 1999). A equação 2.6 representa o cálculo da ação de controle integral, onde K_i é a constante integral e dt é o tempo do processo.

$$u(t) = K_i . \int_0^t e(t)dt \tag{2.6}$$

2.4.3 Controle Derivativo

A ação de controle derivativo é proporcional à taxa de variação do sinal de erro, ou seja, a derivada do erro(FUENTES, 2005). É uma estimativa sobre a tendência de aumento ou diminuição de erro no futuro. Assim, o controle derivativo acelera o processo de correção do erro, pois atua de forma antecipatória quando são detectadas variações no sinal de erro(FACCIN, 2004). A equação 2.7 representa o cálculo da ação de controle derivativo, onde K_d é a constante derivativa e $de(t)$ é o mesmo que $de(t) = e(t) - e(t-1)$.

$$u(t) = K_d . \frac{de(t)}{dt} \tag{2.7}$$

2.4.4 Controle Proporcional-Integral-Derivativo

A união destes três modos básicos de controle contínuo produz um dos mais eficientes algoritmos de controle já desenvolvidos, o controlador PID, pois ele concilia simplicidade e atendimento às necessidades de controle para a grande maioria dos casos industriais(FACCIN, 2004).

Segundo Ogata (2010) uma vantagem deste controlador é quando na ausência de uma abordagem analítica(modelo matemático da planta desconhecido), ainda sim é possível a obtenção de sintonia dos parâmetros por meio de abordagens experimentais.

A estrutura do PID pode ser vista no diagrama da figura 11.

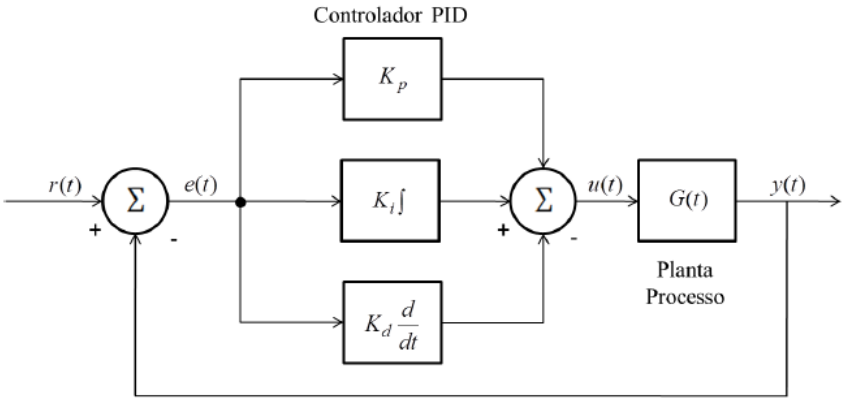

Figura 11 – Diagrama do controle PID

Fonte**:** Adaptado de Ogata (2010)

A combinação dos modos de controle Proporcional, Integral e Derivativo resulta na equação 2.8.

$$u(t) = K_p e(t) + K_i . \int_0^t e(t)dt + K_d . \frac{de(t)}{dt} \tag{2.8}$$

Na área da robótica e Visão Computacional o PID é largamente utilizado em sistemas que utilizam a visão como um rastreador de posição de objetos. Pode ser utilizado para controlar a navegação de um robô, como em (SILVA et al., 2008), onde é aplicado o controle PID em sistema integrado de controle de time de futebol de robôs. Para controlar distância de veículos, como em (HEINEN et al., 2004), onde o trabalho apresenta um método de navegação de veículos de carga autônomos utilizando Visão Computacional com algoritmo de segmentação de cores. E também para controlar o posicionamento de uma câmera *Pan-Tilt*(tem dois graus de liberdade, horizontal e vertical) que segue um determinado objeto rastreado, como em (KIKUCHI, 2007) e a aplicação prática nesse trabalho.

2.5 Dispositivos Robóticos

Segundo Pio, Castro e Júnior (2006) a robótica é definida da seguinte maneira:

> A robótica é a ligação inteligente entre a percepção e a ação, ou seja, através da percepção do ambiente em que o sistema (ou dispositivo) está inserido, é possível realizar tarefas pré-definidas ou não, para interagir fisicamente com este ambiente(PIO; CASTRO; JÚNIOR, 2006).

No esquema da figura 12, esse conceito apresentado pelo autor é bem representado.

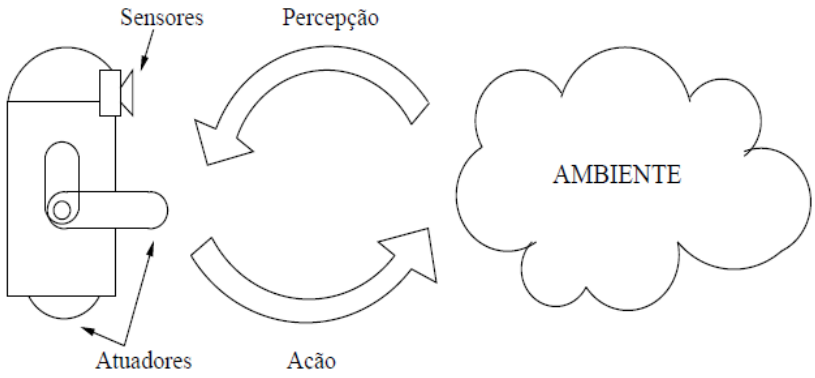

Figura 12 – Sistema Robótico

Fonte**:** Pieri (2002)

Na figura 12, pode-se perceber que o agente que realiza a percepção e a atuação sobre o ambiente é um robô. O robô é um mecanismo automático, em geral com aspecto semelhante ao de um homem, e que realiza trabalhos e movimentos humanos(AURÉLIO, 2014), para realizar isso, o robô dispõe das principais características(BUDIHARTO; GROUP, 2014):

Sensoriamento Da mesma forma que um humano realiza a percepção do ambiente, o robô emula essa percepção através de sensores, que podem ser sensores visuais, de movimento e presença, distância, luz, químicos, sonoros, entre outros.

Movimento O robô pode mover-se no ambiente, seja por rodas, por pernas mecânicas ou por qualquer tipo de mecanismos capazes de alterar sua posição de origem.

Energia O robô precisa de energia para poder realizar as ações, podendo ser energia solar, elétrica, bateria, dependendo de quais atividades ele irá realizar.

Programabilidade O robô pode ser programado para fazer várias atividades, aí então ele pode operar automaticamente.

Capacidade Mecânica Além da percepção, o robô também pode agir em um ambiente, modificando-o de acordo com sua programação, como representado na figura 12.

Inteligência O programador é a pessoa que faz o robô ser inteligente, ou seja, o robô é programado para saber o que está fazendo quando receber os dados do ambiente.

Dentre essas características, os sensores e atuadores são os responsáveis pela interação com o ambiente. Pelo sensoriamento é possível a percepção do sistema robótico acerca do ambiente e pela atuação é possível interagir fisicamente com este ambiente. Nesse projeto, foi utilizado uma câmera como sensor de posição de objetos através da Visão Computacional, e servomotores para realizar os movimentos necessários.

2.5.1 Sensores

Os sensores são transdutores, ou seja, conversores de grandezas físicas em sinais elétricos correspondentes(MORAES; JÚNIOR; PECEGUEIRO, 2003). Os sensores são utilizados para medir grandezas como temperatura, distância, intensidade de luz, presença, velocidade dentre outras grandezas. Há dois tipos de sensores, os internos e os externos(PIERI, 2002).

Os sensores internos são usados para medir a posição e velocidade ou aceleração das juntas de um robô manipulador, ou seja, qual a posição atual do robô. Exemplo: Potenciômetros.

Os sensores externos são utilizados para monitorar o próprio robô e a sua relação com o ambiente ao seu redor, bem como a tarefa que lhe foi destinada.

2.5.1.1 Câmera

A utilização de câmeras em sistema robóticos vem se tornando cada vez mais frequente(KIKUCHI, 2007), muito em razão do crescimento da Visão Computacional, que tem como uma das principais consequências a disponibilidade de câmeras com maior resolução e com baixo custo(BRADSKI; KAEHLER, 2013).

A câmera utilizada para este projeto possui as seguintes especificações básicas:

- **Resolução**: 5.0 *Mega Pixel*
- **Formato de vídeo**: 24 *bit true color*
- **Taxa de quadros**: 30 fps
- **Conexão**: USB 2.0

- **Compatibilidade**: *Windows* 2000/NT/ME/XP/Vista e *Linux*

Na robótica, as câmeras podem ser aplicadas como sensores de posição, que podem determinar as localização de objetos(KIKUCHI, 2007). Tendo a informação da localização de um objeto, o sistema de controle envia sinais aos atuadores para realizarem as operações no ambiente.

2.5.2 Atuadores

Atuadores são dispositivos responsáveis por dar movimento à uma máquina-ferramenta ou robô(PIERI, 2002). Um comando dado pelo sistema de controle é enviado para o dispositivo e este realiza a ação no meio físico. Um exemplo de atuador ou um conjunto de atuadores é o braço robótico, representado na figura 13.

Figura 13 – Exemplo de Atuador - Braço Robótico
Fonte**:** Multilógica (2014)

Existem três tipo de atuadores: os pneumáticos, os hidráulicos e os elétricos. Os pneumáticos e hidráulicos, tem como fonte de energia ar comprimido e fluídos, respectivamente. Entretanto, estes necessitam de bombas, compressores, filtros, acumuladores, equipamentos para refrigeração, válvulas, dentre outros acessórios, fazendo com que requeiram muita manutenção(PIERI, 2002).

Neste projeto foram utilizados atuadores elétricos, que sem dúvida são os mais utilizados em projetos de robótica(PIERI, 2002), esse tipo de transmissão é feita utilizando sinais elétricos. Possui as seguintes vantagens(CARRARA, 2007):

- Transmissão em longas distâncias sem perdas.
- Alimentação pode ser feita pelos mesmos fios que são utilizados para transmissão de dados.

- Não necessita de equipamentos auxiliares.
- Fácil conexão com computadores.
- Fácil instalação.
- Mais fácil para realizar operações matemáticas.
- Permite que o mesmo sinal elétrico seja compartilhado entre diversos dispositivos (obedecendo os limites da resistência).

Um dos atuadores elétricos mais comuns na robótica são os chamados servomotores (CARRARA, 2007).

2.5.2.1 Servomotores

(PIERI, 2002) define os servomotores da seguinte maneira:

> Servomotores são motores de corrente contínua que incorporam alguns dispositivos para tornar seu uso mais simples. Ele é utilizado principalmente para controle de posição. Incluem engrenagens, limitadores de eixo, potenciômetro para a malha de realimentação e pequenos circuitos integrados para o controle de posição(PIERI, 2002).

Na figura 14 é apresentado os componentes internos de um servomotor.

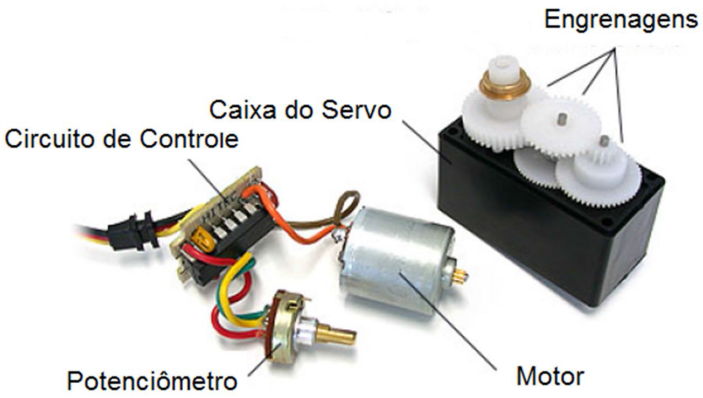

Figura 14 – Componentes internos de um Servomotor

Fonte**:** Santos (2007a)

Nas partes apresentadas na figura 14, são apresentados os componentes(SANTOS, 2007a):

Circuito de Controle Responsável pelo monitoramento do potenciômetro e acionamento do motor, visando obter uma posição pré-determinada.

Potenciômetro Ligado ao eixo de saída do servo, serve para monitorar em qual posição se encontra o servo.

Motor Movimenta as engrenagens e o eixo principal do servo.

Engrenagens Reduzem a rotação do motor, transferem mais torque ao eixo principal de saída e movimentam o potenciômetro junto do servo.

Caixa do servo Caixa para acondicionar as diversas partes do servo.

Sua maior vantagem é o controle de posição e a precisão, e ser capaz de manter a posição mesmo sofrendo forças contrárias. Ele tem sua angulação limitação de 0º a 180º(SANTOS, 2007a).

Nesse trabalho, foram utilizados dois servomotores *Tower Pro 9g*. As especificações são apresentadas no Quadro 1.

O controle do servomotor é feito através de sinais do tipo PWM (*Pulse Width Modulation*), onde a posição angular irá depender da largura de um pulso enviado(CARRARA, 2007). Este sinal pode ter 0V (0 lógico) ou 5V (1 lógico), logo 0V significa nenhum sinal e 5V significa um sinal existente.

O circuito de controle do servo fica monitorando este sinal em intervalos de 20ms, se dentro deste intervalo ele percebe uma alteração do sinal de 0V para 5V durante 1ms até 2ms ele altera a posição do seu eixo para coincidir com o sinal que recebeu.

No esquema apresentado na Figura 15, é apresentado um exemplo de como o servomotor é controlado.

Especificação	**Valor**
Voltagem de Operação	4.8V
Ângulo de rotação	180 graus
Velocidade	0,1 seg/60 graus
Torque	1,8 kgf.cm
Temperatura de Operação	0ºC - 55ºC
Dimensões	32.2 x 31 x 11.8 mm
Peso	9g

Quadro 1 – Especificações do servomotor Tower Pro

Fonte**:** Sparkgo (2014)

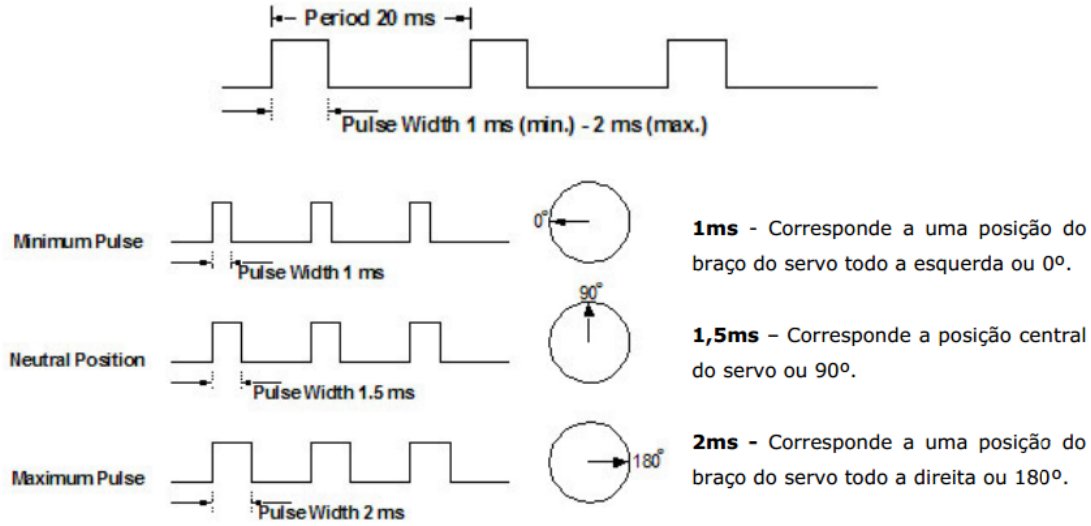

Figura 15 – Exemplo de controle por PWM em Servomotor

Fonte**:** Santos (2007a)

Santos (2007a) apresenta um exemplo do funcionamento e comportamento de um servomotor:

> Se, por exemplo, o servomotor recebe um sinal de 1,5ms, ele verifica se o potenciômetro está no meio, se estiver ele não faz nada. Se o potenciômetro não estiver naquela posição, o circuito de controle aciona o motor até o potenciômetro estar na posição certa. A direção da rotação do motor do servomotor vai depender também da posição do potenciômetro, o motor vai girar na direção que mais rápido levar o potenciômetro até a posição certa(SANTOS, 2007a).

2.6 *Arduino*

O *Arduino* uma plataforma de computação de *hardware* livre baseado em um simples microcontrolador e um ambiente de desenvolvimento para programar no *hardware*(ARDUINO, 2014c), ou seja, são sistemas digitais ligados a sensores e atuadores, que permitem construir sistemas que tenham a percepção do ambiente e respondem com ações físicas. Tem acessos de I/O (Input/Output), sobre a qual foram desenvolvidas bibliotecas com funções que simplificam a sua programação, por meio de uma sintaxe similar à das linguagens C e C++(RENNA, 2013). Na figura 16 pode-se observar uma representação do *Arduino Uno*, que foi utilizado neste projeto.

O *Arduino Uno* tem as seguintes especificações, conforme o Quadro 2:

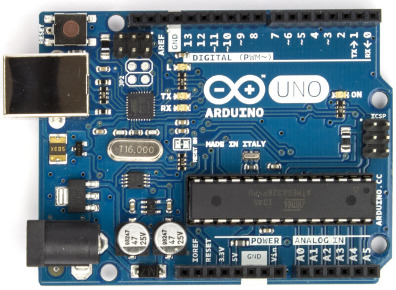

Figura 16 – *Arduino Uno*

Fonte**:** Arduino (2014b)

Especificação	**Valor**
Microcontrolador	ATmega328
Voltagem operacional	5V
Voltagem de Entrada (Recomendado)	7-12V
Voltagem de Entrada (Limites)	6-20V
Pinos de I/O Digitais	14 (onde 6 podem ser usados para PWM)
Pinos de entrada analógica	6
Corrente DC por pino de I/O	40mA
Corrente DC por pino de 3.3V	50mA
Memória Flash	32KB (ATmega328) onde 0.5KB são usados para o carregamento do *boot*
SRAM	2KB (ATmega328)
EEPROM	1KB (ATmega328)
Velocidade do Clock	16MHz

Quadro 2 – Especificações do *Arduino Uno*

Fonte**:** Arduino (2014b)

2.6.1 Comunicação Serial

O *Arduino* é programável através de um IDE (*Integrated Development Environment*) próprio. IDE é um *software* usado para aumentar a produtividade dos desenvolvedores e a qualidade dos produtos(SANTOS, 2007b). A linguagem utilizada também é própria, muito similar à linguagem C e C++(SILVA, 2011).

Uma das funções encontradas nessa IDE é o Monitor Serial, que é uma interface de comunicação serial feita entre o *Arduino* e um computador ou outros dispositivos(ARDUINO, 2014e).

O Monitor Serial (figura 17) é uma ferramenta muito útil, especialmente para depuração de código, é possível receber e enviar os dados do computador ao *Arduino*.

Os dados que são recebidos e enviados trafegam sobre uma taxa de Transmissão(*Baud Rate*) que é taxa por segundo em que alterações de estado ou bits(dados) são enviados de/para a placa *Arduino*. A

configuração padrão é 9600 *baud*, ou seja, 9.600 bits por segundo ou 1.200 caracteres por segundo (9.600 / 8 bits por caractere = 1.200 *bytes* ou caracteres)(MCROBERTS, 2011).

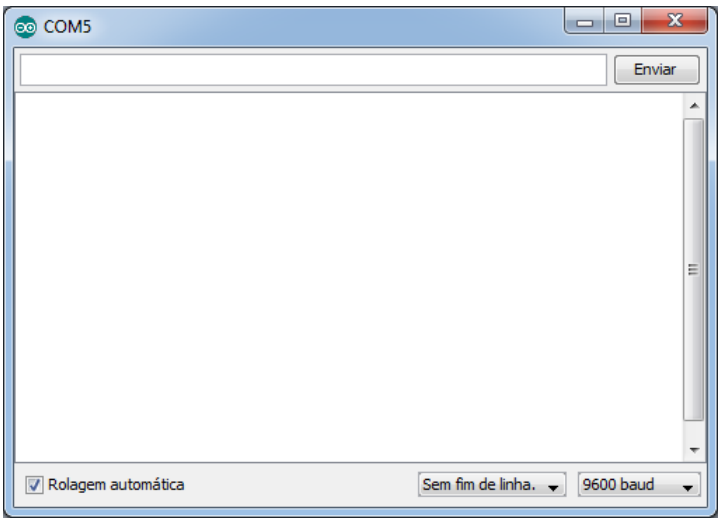

Figura 17 – Exemplo de Monitor Serial

Fonte**:** Elaborado pelo autor

2.6.2 Comunicação com Dispositivos Robóticos

A comunicação com computador se dá através da comunicação Serial via cabo USB. Com dispositivos robóticos o *Arduino* se comunica através de seus pinos de Entrada/-Saída.

O *Arduino* conta com 14 pinos digitais que podem ser usados para entrada e saída de dados para controlar dispositivos. Eles operam em 5V e cada pino fornece ou recebe, no máximo, 40mA de corrente, e tem um resistor interno de 20-50kOhms(ARDUINO, 2014b).

Possui também 6 pinos de entradas analógicas, que são identificadas por A0 ao A5. que são mais utilizados para ler dados enviados de sensores, mas também podem ser utilizados como pinos digitais(ARDUINO, 2014a).

Na figura 18 é representado um exemplo de conexão e dispositivo robótico, no caso uma conexão de um *Arduino Uno* com um servomotor.

2.7 *Raspberry Pi*

O *Raspberry Pi* é um computador de baixo custo que possui o tamanho de um cartão de crédito(FOUNDATION, 2014b). Pode ser utilizado para diversos projetos pelo seu tamanho e por ter diversos tipos de conexões, inclusive para dispositivos robóticos. Foi desenvolvido pela *Raspberry Pi Foundation* no Reino Unido, com fins educativos, tanto para adultos, como para crianças(FOUNDATION, 2014b). Tem os modelos A, B e B+

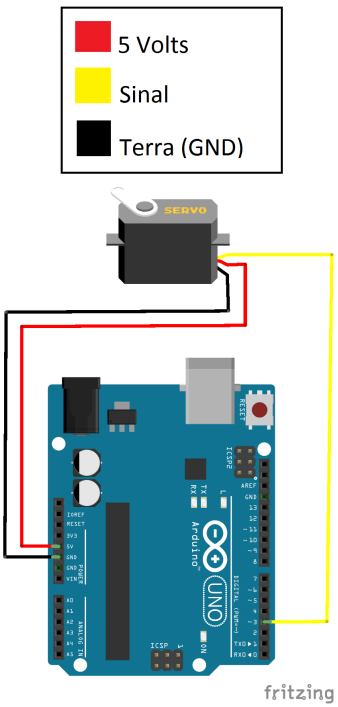

Figura 18 – Esquema de *Arduino* com Servomotor

Fonte**:** Elaborado pelo autor

disponíveis, sendo escolhida para este projeto o modelo B, pois era a versão disponível para o projeto. Na figura 19, é apresentado o modelo B em sua imagem real.

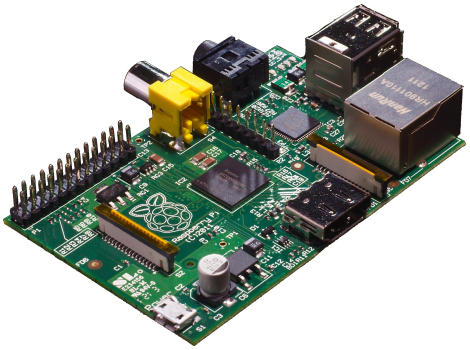

Figura 19 – *Raspberry Pi* Modelo B

Fonte**:** Wikipedia (2014)

O *Raspberry Pi* Modelo B, possui as seguintes especificações técnicas(ELINUX, 2014):

- SOC (*System on a chip*) Broadcom BCM2835.
- Processador ARM (*Advanced RISC Machine*) 11 com frequência de *clock* a 700MHz.
- GPU (*Graphics Processing Unit*) VideoCore IV a 250MHz com suporte à decodificação de vídeos 1080p.

- Memória RAM (*Randomic Access Memory*) de 512MB.
- 2 Portas USB.
- 1 Conexão *Ethernet* 10/100 *Registered Jack* (RJ)45.
- 1 Slot para SD (*Secure Digital*).
- 1 Conexão HDMI (*High-Definition Multimedia Interface*).
- 1 Conexão TRS (*Tip-Ring-Sleeve*) de 3,5mm.
- 1 Conexão RCA (Radio Corporation America).
- 1 Conexão micro USB para alimentação 5v.

Além desses componentes, o Modelo B possui um barramento de 26 pinos para conexão com diversos periféricos de baixo nível, identificados na lista abaixo(FOUNDATION, 2014a)(PEIXOTO, 2012):

- 8 GPIO's (*General Purpose Input/Output*) a 3.3V;
- 2 pinos UART (*Universal Asynchronus Receiver/Transmitter*), 3.3V (debug); ou 2 GPIOs a 3.3V;
- I^2C (*Inter-Integrated Circuit interface*) (3.3V); ou 2 GPIOs at 3.3V;
- SPI (*Serial Peripheral Interface*) *interface* (3.3V); ou 5-3.3V;
- 3.3V, 5V e GND de alimentação;
- *Camera Serial Interface 2* (CSI-2) *interface*);
- Segunda *interface* I^2C (3.3V) (se os pinos forem reconfigurados por *software*)
- I^2S (*Integrated Interchip Sound*) *interface* (se os pinos forem reconfigurados por *software*);
- 6 pinos reservados para uso futuro;

Tem ainda disponível dois conectores DSI (*Display Serial Interface*) de 15 pinos para *displays*, um conector CSI-2 para câmera e acesso a GPU JTAG (*Joint Test Action Group*)(PEIXOTO, 2012). Um esquema é apresentado na figura 20 demonstrando as possibilidades de conexões para o funcionamento básico do *Raspberry Pi*.

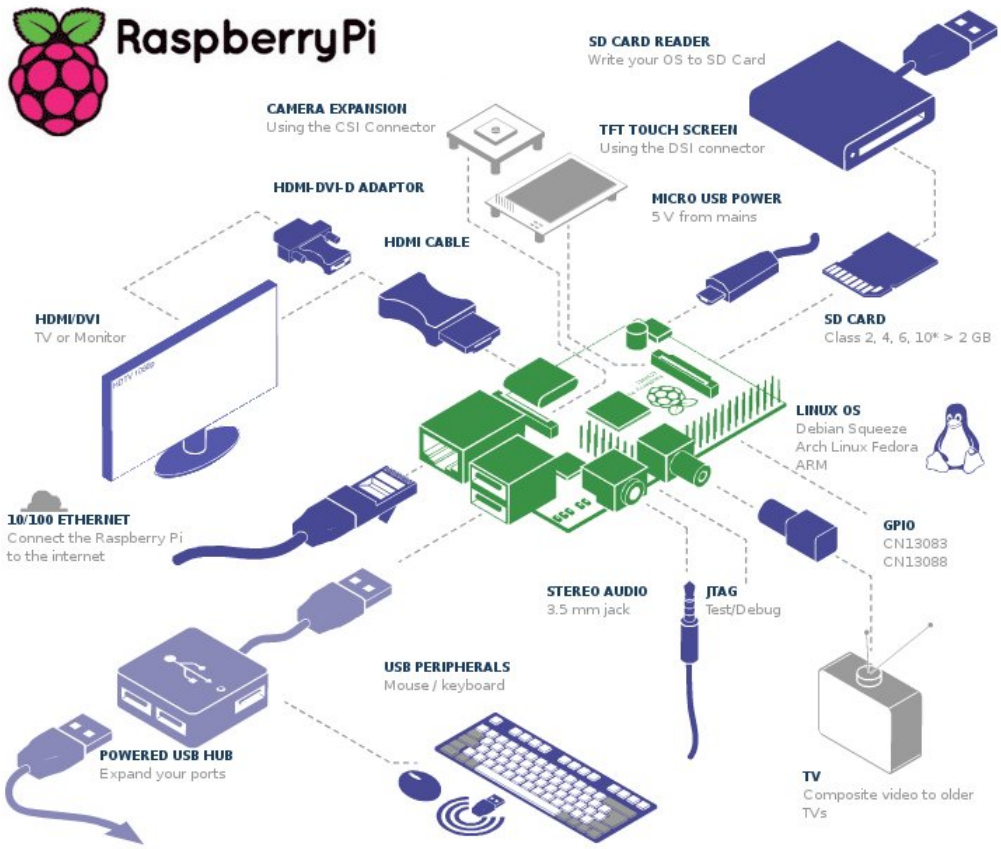

Figura 20 – Esquema completo *Raspberry Pi* Modelo B

Fonte**:** Wvshare (2014)

2.7.1 Raspbian

Como todo computador, o *Raspberry Pi* também necessita de um sistema operacional e este utiliza o *Linux*. O *Linux* é tecnicamente apenas o *Kernel*, e um sistema operacional possui muito mais do que isso como a coleção total de *drivers*, serviços e aplicações(RICHARDSON; WALLACE, 2013). Diversas distribuições de sistemas operacionais baseadas em *Kernel Linux* são disponibilizadas de forma gratuita para computadores pessoais como *Ubuntu* e *Debian*.

Para o *Raspberry Pi* foram criadas algumas distribuições, visando atender aos seus requisitos de *software*, uma vez que suas características são limitadas. Umas dessas distribuições é a *Raspbian*.

Raspbian é a distribuição baseada no *Debian*, e é oficialmente recomendada pela *Raspberry Pi Foundation*(RICHARDSON; WALLACE, 2013). É totalmente compatível com o *hardware* do *Raspberry Pi*, tem uma área de trabalho completa (Figura 21) e contém aplicações muito utilizadas como *web browser*.

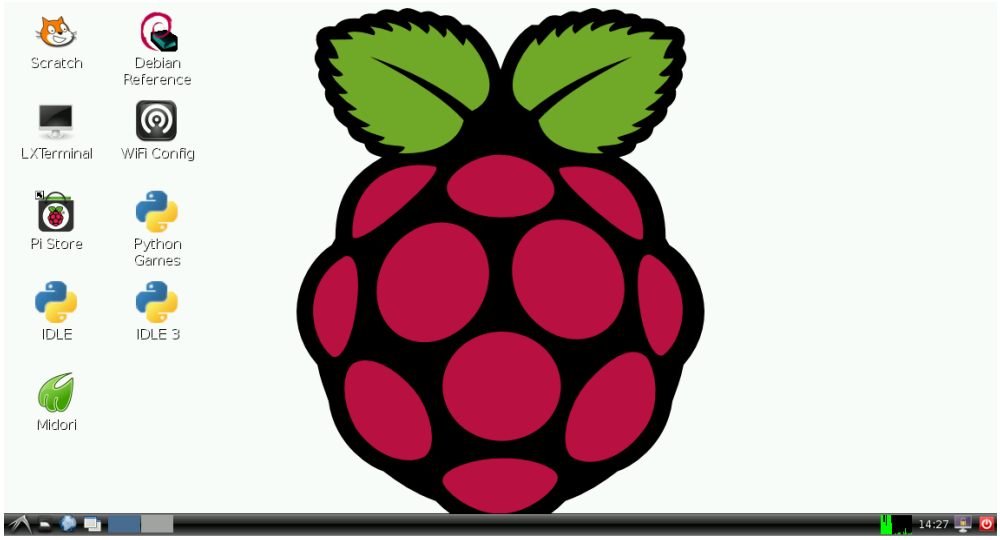

Figura 21 – Área de Trabalho do Raspibian

Fonte**:** Elaborado pelo autor

2.8 Trabalhos Relacionados

Nesta seção são apresentados três trabalhos relacionados ao tema proposto nesse trabalho, as aplicações práticas destes também se assemelham com a aplicação prática do trabalho presente. Destes trabalhos foram retirados alguns conceitos e técnicas utilizadas.

2.8.1 Detecção e Acompanhamento de Movimento Através de uma Câmara de Vídeo

No trabalho de Peixoto (2012), a abordagem utilizada pelo autor para detecção e acompanhamento de um determinado objeto (no caso, uma face) foi por algoritmos separados, sendo utilizado o *Viola-Jones* para detecção e o *Camshift* para o acompanhamento.

Além disso, a câmera utilizada dispensa demais acessórios, pois é uma câmera com *Pan-Tilt-Zoom*, ou seja, tem a função de movimentação desta com dois graus de liberdade (horizontal e vertical), além de ser IP (*Internet Protocol*), sendo utilizada em uma LAN (*Local Area Network*). No trabalho presente, é utilizada uma câmera convencional conectada a dois servomotores, controlados por um Arduino, que servem para a rotação Pan-Tilt.

Para processamento, o autor utiliza o *Raspberry Pi*, contudo o minicomputador se mostrou inviável para o projeto, pois o poder de processamento limitado dele não foi capaz de suportar a demanda exigida para o processamento da Visão Computacional e de funções de servidor *web* (para receber as imagens pela LAN).

2.8.2 Rastreamento em Tempo Real com Câmera de Pan-Tilt

Silva (2011) propõe um sistema escrito na linguagem *Processing* que realiza o rastreamento e acompanhamento de faces em tempo real utilizando o algoritmo *Viola-Jones* e uma câmera com *Pan-Tilt*.

O controle do sistema não utiliza o controle PID, entretanto utiliza uma margem de erro que minimiza os tremores nas imagens capturadas. No trabalho presente, além de utilizar PID, é utilizado também uma margem de erro para maior precisão da correção.

Além dessa abordagem, o autor utiliza a comunicação serial com *Arduino*, passando informações para movimentação de servomotores via serial em formato de caracteres para serem convertidos de acordo com a tabela ASCII (*American Standard Code for Information Interchange*), para números.

Uma análise que também foi verificada nesse trabalho é que a linguagem *Processing* atende aos requisitos do trabalho, porém foi considerada não ideal, pois ela utiliza uma linguagem de mais alto nível do que a utilizada pelas interfaces do OpenCV gerando uma ligeira lentidão, sendo recomendado uma linguagem de mais baixo nível como a linguagem C. No trabalho presente foi utilizado C++ que tem compatibilidade semelhante à linguagem C, assim possui um desempenho mais elevado.

2.8.3 Sistema de Captura de Movimentos para Comandar Equipamentos Eletrônicos

No trabalho de Júnior (2010), o autor propõe um sistema que captura gestos da mão, e de acordo com cada gesto já pré-definido no sistema são enviados comandos para equipamentos eletrônicos executarem ações correspondentes, diferente do trabalho presente que o padrão é capturado em tempo real.

O autor utiliza a *EmguCV*, que é uma plataforma *.NET* para a biblioteca OpenCV(EMGU, 2014), com a finalidade de utilizar uma interface mais amigável para o usuário do sistema.

Outro item do trabalho é a utilização de um *kit* de robótica educacional. Esse *kit* tem uma grande possibilidade de movimentos, a montagem é simples, bem como uma aplicação de comunicação com o robô montado.

3 DESENVOLVIMENTO

Neste capítulo é apresentada uma aplicação prática de controle de dispositivos robóticos usando Visão Computacional, utilizando os conceitos dos principais tópicos apresentados na seção anterior.

A aplicação prática é denominada câmera seguidora, ou seja, uma câmera que fará o acompanhamento de um determinado objeto rastreado, buscando deixá-lo mais no centro possível da imagem.

3.1 Estrutura de *Software*

O sistema foi executado em duas plataformas, a plataforma *Linux*, *Raspbian*(seção 2.7.1), que é executada no *Raspberry Pi* (seção 2.7), e na plataforma *Windows*, em sua versão 7 *Ultimate*, executada em um *notebook* Asus, modelo A45A, que tem as seguintes especificações, apresentadas no Quadro 3.

Foi escolhido as duas plataformas para verificar se há diferenças significativas entre uma e outra, e também por serem de uso comum em computadores pessoais.

Como citado nas seções anteriores, a biblioteca de Visão Computacional utilizada é a OpenCV. As instruções de instalação do OpenCV no *Linux* podem ser vistas no ANEXO A(CASTLE, 2014), e as instruções de instalação do OpenCV no *Windows 7* podem ser encontradas no ANEXO B(OPENCV, 2014b).

O sistema foi escrito na linguagem de programação C++, que é um conjunto de regras usadas para escrever programas de computador e que mostram ao computador que tarefas ele precisa realizar(MEDINA; FERTIG, 2005). Essa linguagem foi escolhida, pois é totalmente compatível com a biblioteca OpenCV(seção 2.3).

Especificação	**Valor**
Processador	Processador Intel Core i5 3210M
Chipset	Intel Chief River Chipset HM76
Memória	6GB
Gráfico	*Integrated Intel High Definition Graphics* 4000
HD (*Hard Disk*)	750GB

Quadro 3 – Especificações ASUS A45A

Fonte**:** ASUSTek (2014)

Para traduzir essa linguagem para uma forma no qual o computador possa compreender e executar, foi necessário utilizar um compilador(SANTEE, 2005). Para a plataforma *Linux* foi escolhido o *CMake*, em sua versão 2.4.9, que é um *software* que ajuda a simplificar o processo de compilação(compilador *GNU Compiler Collection*) para desenvolvimento de projetos em diferentes linguagens(PROJECT, 2014). A escolha deste programa foi pelo fácil uso com a biblioteca OpenCV e por ser gratuito(CMAKE, 2014).

Para a plataforma *Windows* foi escolhido a IDE *Visual Studio 2012 Professional*, que é pago, porém o IFSP (Instituto Federal de São Paulo) providenciou uma licença gratuita através de uma conta no *Dreamspark*, que é um programa da *Microsoft* que dá suporte a educação técnica fornecendo acesso a *software* da *Microsoft* para fins de aprendizado, ensino e pesquisa(MICROSOFT, 2014b). A escolha desse IDE foi por conter o compilador *Visual C++* para a execução do sistema, sua compatibilidade com a biblioteca OpenCV(OPENCV, 2014a) e por ter vários materiais disponíveis na internet.

3.1.1 Configurações no Linux

Para compilar e executar o sistema no CMake foi necessário criar um arquivo de configurações *CMakeLists.txt*, cujo conteúdo são as seguintes informações(OPENCV, 2014d):

```
1 cmake_minimum_required(VERSION 2.8)
2 project( CameraSeguidora )
3 find_package( OpenCV REQUIRED )
4 add_executable( CameraSeguidora CameraSeguidora.cpp )
5 target_link_libraries( CameraSeguidora \${OpenCV_LIBS} )
```

Após a criação desse arquivo no mesmo diretório do código fonte, para executar o sistema foi necessário utilizar os seguintes comandos no Terminal(OPENCV, 2014d):

```
1 cd <CameraSeguidora_diretorio>
2 cmake .
3 make
4 sudo ./CameraSeguidora
```

3.1.2 Configurações no Windows

Com as variáveis de ambiente já criadas na instalação do OpenCV conforme ANEXO B, foram seguidos os passos das configurações do OpenCV no *Visual Studio 2012 Professional*(OPENCV, 2014a):

- Após a criação do projeto, clicar com o botão direito do mouse sobre este projeto na aba *Solution Explorer* e clicar com o botão esquerdo na opção *Properties*.
- Na tela de configuração, no canto superior esquerdo selecionar a opção *Release* no campo *Configuration*.
- Clicar na opção *C/C++ > General* no canto esquerdo.
- No campo *Additional Include Libraries*, adicionar o caminho *$(OPENCV_DIR)/../../include* e clicar em *OK*.
- Clicar na opção *Linker > General* no canto esquerdo
- No campo *Additional Libraries Directories*, adicionar o caminho *$(OPENCV_DIR)/lib* e clicar em *OK*.
- Clicar na opção *Linker > Input*
- No campo Additional Dependencies, adicionar os seguintes valores:
  - opencv_core249.lib
  - opencv_imgproc249.lib
  - opencv_highgui249.lib
  - opencv_ml249.lib
  - opencv_video249.lib
  - opencv_features2d249.lib
  - opencv_calib3d249.lib
  - opencv_objdetect249.lib
  - opencv_contrib249.lib
  - opencv_legacy249.lib
  - opencv_flann249.lib

3.2 Estrutura de Hardware

A montagem dos equipamentos para realizar tal aplicação prática é apresentada na figura 22.

A descrição da função de cada item na figura são as seguintes:

Câmera Para captura das imagens.

Raspberry Pi Para realizar o processamento do sistema de Visão Computacional.

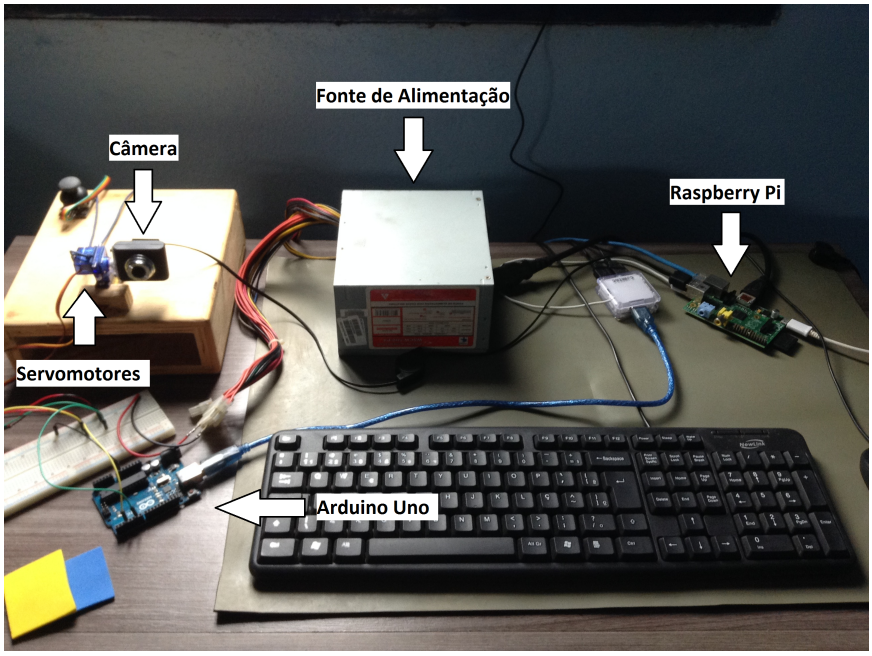

Figura 22 – Montagem dos Equipamentos

Fonte**:** Elaborado pelo autor

Arduino Uno Para processar os dados enviados via serial do *Raspberry Pi* e, com esses dados, mover os servomotores que estão conectados nas portas três e seis digitais que também podem ser usadas para conexões PWM.

Servomotores Um servomotor tem a função de mover a câmera horizontalmente e o outro verticalmente.

Fonte de Alimentação Utilização de 5V para alimentação elétrica dos servomotores.

3.3 Fluxo do sistema

Na figura 23 é representado um fluxo de execução do sistema. Como pode-se observar, é possível escolher entre realizar a detecção de faces para rastreá-la, ou selecionar um determinado objeto com o mouse. A região da face ou a região de seleção do mouse, chamada região de interesse, é enviada para as funções de pré-processamento que preparam os objetos necessários para executar o algoritmo *Camshift*. Este rastreia o objeto, retornando sua coordenada (x, y) na imagem capturada, e com essa informação é então feito o Controle PID que resulta em um valor, e este valor é enviado ao *Arduino Uno* via serial, para que, com esses dados, sejam controlados os servomotores corretamente.

3.3.1 Captura da Imagem

A captura da imagem é feita partir de uma câmera, cuja especificações podem ser encontradas na seção 2.5.1.1. Para ter acesso às imagens capturadas, é necessário

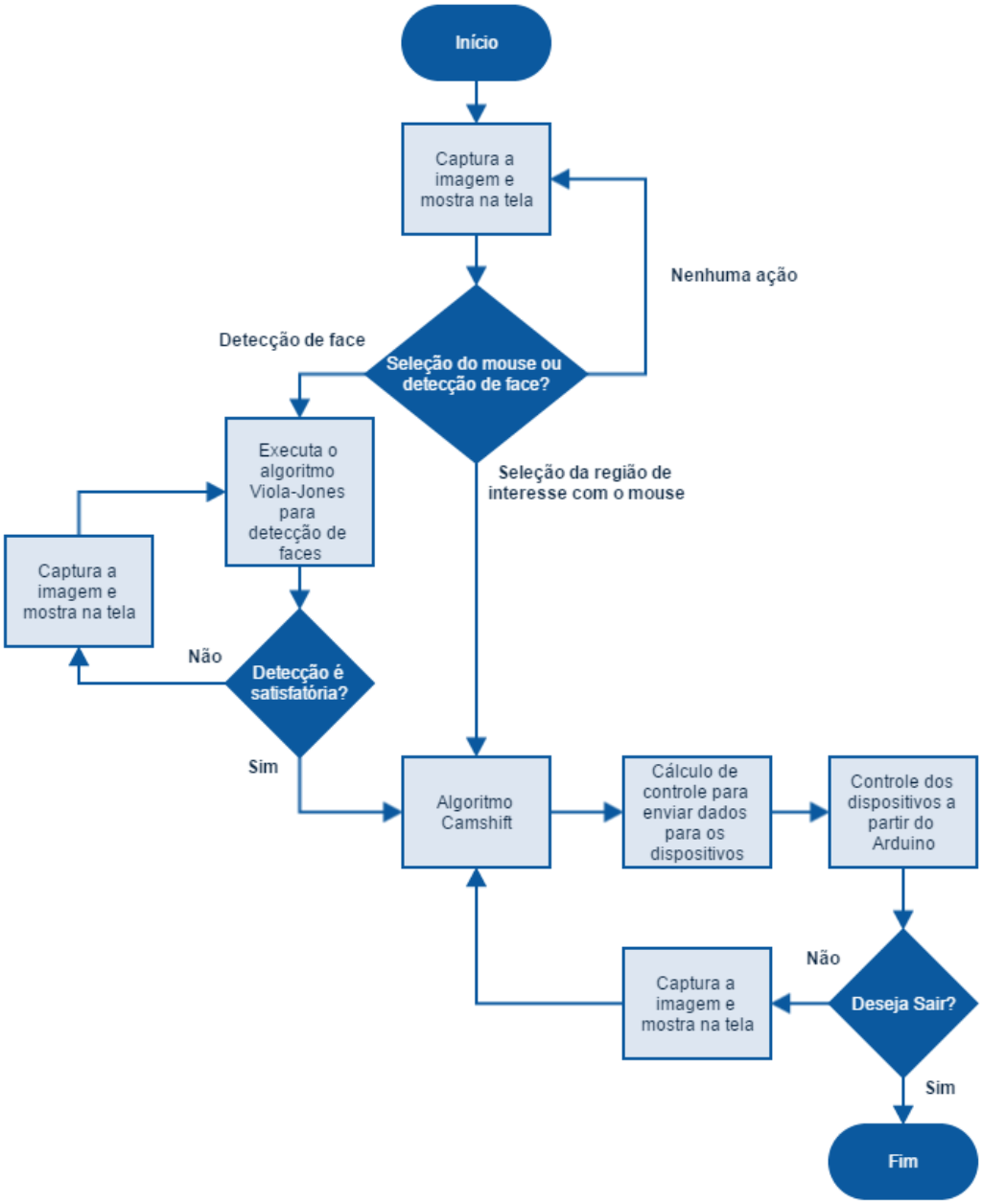

Figura 23 – Diagrama das partes do Desenvolvimento

Fonte**:** Elaborado pelo autor

que haja uma conexão via USB entre a câmera e o *Raspberry Pi*. O êxito da conexão e a disponibilidade da câmera são verificados por métodos da biblioteca OpenCV(seção 2.3).

A imagem capturada é uma imagem digital, ou seja, uma matriz de números (seção 2.2), por essa razão, no OpenCV um objeto de imagem é um objeto da classe *Mat* que representa uma matriz. O objeto de captura, responsável pela verificação da disponibilidade da câmera, pela captura da imagem e por atribuir as propriedades de altura e largura da imagem capturada, pertence à classe *VideoCapture*.

A resolução escolhida para captura de imagem no *Raspberry Pi* foi 320 *pixels* de largura por 240 *pixels* de altura (ou 320 colunas por 240 linhas em matriz), uma resolução baixa, pois o poder de processamento do *Raspberry Pi* se mostrou insuficiente para resoluções maiores, gerando travamentos constantes no sistema.

Outra característica sobre a imagem é o seu espaço de cores, o BGR (seção 2.2.2), este utilizado pela biblioteca *OpenCV* por razões de melhor desempenho de memória, sendo similar ao RGB em seu uso.

Como o sistema é executado em tempo real, as imagens são capturadas dentro de um *loop* e apresentadas, caracterizando uma captura de vídeo.

Para serem apresentadas, as imagens são colocadas em interfaces gráficas (janelas), que são declaradas pela função *cvNamedWindow*, estas também recebem uma função *setMouseCallback*, que direcionam eventos do mouse a uma determinada função. A figura 24 mostra uma janela apresentando uma imagem.

Figura 24 – Tela de Captura de Imagem

Fonte**:** Elaborado pelo autor

3.3.2 Seleção da Região de Interesse

A seleção da região de interesse a ser rastreada pode ser feita de duas formas, ou por rastreamento de faces, ou por seleção com o mouse. Qualquer uma das duas opções resultará uma área de interesse que é utilizada pelo algoritmo *Camshift*.

3.3.2.1 Detecção de Face

Na opção por detecção de face, é executado o algoritmo de *Viola-Jones* (seção 2.3.1.2), cujo objetivo é percorrer a imagem capturada identificando um padrão já estabelecido. Esse padrão é configurado anteriormente em um arquivo de formato XML.

Um objeto da classe *CascadeClassifier* é criado e este recebe o arquivo XML de configuração. O OpenCV já tem em seus projetos de exemplos alguns classificadores treinados

para detecção de face, um desse tem nome de *haarcascade_frontalface_default.xml*, e segundo (ROUBEROL, 2012), dentre todos os arquivos de configuração que vem por padrão, este é o que tem melhor média de detecções.

A imagem tem que ser convertida primeiramente para a escala de cinza para se obter uma melhor detecção, através da função *cvtColor* da biblioteca OpenCV e após essa operação, é chamada a função *detectMultiScale* que armazenará em um vetor de objetos *Rect* (retângulos), as faces detectadas.

Importante ressaltar que a detecção de faces é apenas frontal, tendo que ser utilizado outro arquivo de configuração XML para detectar faces em perfil (figura 25).

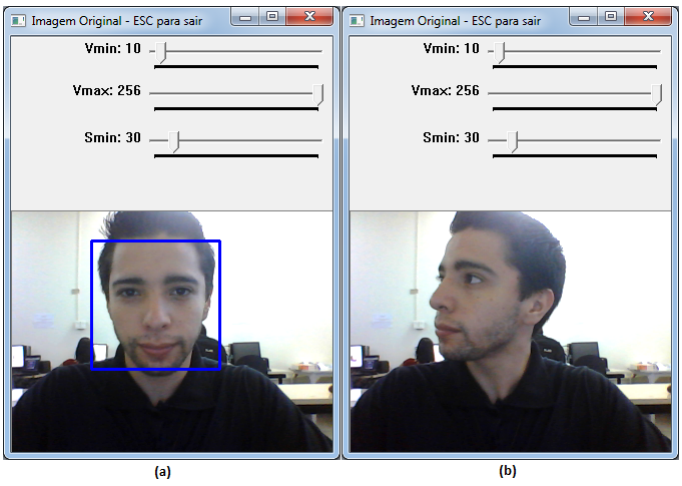

Figura 25 – (a) Detecção da face frontal (b) Não detecção em perfil

Fonte**:** Elaborado pelo autor

Quando a detecção for satisfatória, ou seja, não havendo nenhum tipo de interferência ou erro de detecção, é então capturado a região da face onde o retângulo está localizado e enviado ao algoritmo de rastreamento.

3.3.2.2 Seleção com o Mouse

Uma vez posicionado o objeto de interesse no campo de visão da câmera, pode-se selecionar o objeto com o mouse, clicando e arrastando sobre ele (figura 26). Assim que finalizado a seleção, é dado início às etapas do rastreamento.

3.3.3 Rastreamento

Tanto pela escolha da detecção de face quando pela escolha pela seleção do objeto a ser rastreado com o mouse, uma região de interesse será fornecida para ser utilizada pelo algoritmo de rastreamento *Camshift*(seção 2.3.1.1).

Figura 26 – Seleção da Região de Interesse

Fonte**:** Elaborado pelo autor

Para se obter a localização do objeto ou face na imagem, são necessárias algumas operações de pré-processamento(seção 2.2.1) para criar condições para o algoritmo obter sucesso(seção 2.3.1.1).

Uma dessas operações de pré-processamento é a conversão do espaço de cores BGR para HSV (seção 2.2.3) com a função *cvtColor* da biblioteca OpenCV. O espaço de cores HSV se mostrou mais viável do que o espaço de cores BGR para rastreamento, além de ser melhor de se manipular e também mais preciso(IKEDA, 2003), pois o HSV possui apenas um canal de cores, enquanto o RGB possui três, tornando a manipulação mais complexa. Essa conversão é necessária para extrair o canal de Matiz(H) da imagem e com essa informação criar o histograma.

A criação do histograma (seção 2.2.4)da região de interesse se dá pela função *calcHist*, obtendo o gráfico da frequência de cada grupo de cores determinados. Na figura 27 é apresentado uma exemplo de um histograma criado de uma determinada região de interesse.

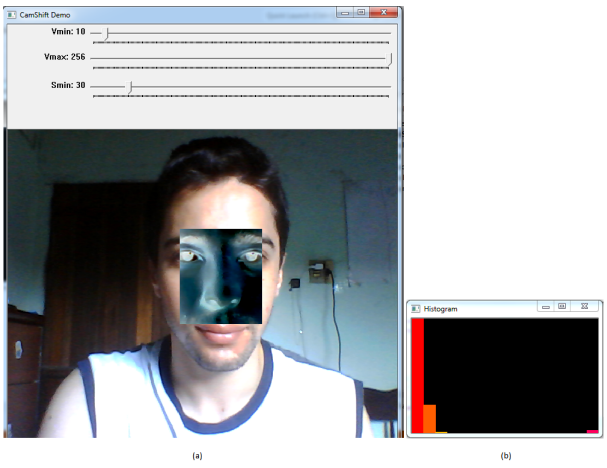

Figura 27 – Seleção da região de interesse e Histograma

Fonte**:** Elaborado pelo autor

Com o histograma criado, o próximo passo é a função de *backprojection* (seção 2.3.1.1), onde esse histograma é comparado ao histograma do canal Matiz da imagem inteira. A imagem obtida por essa função é uma imagem em tons de cinza, onde a região de interesse corresponde às cores mais claras e o que não é da região de interesse à cor preta, ou seja, aonde for mais claro, há mais probabilidade de ser um pixel da região de interesse, aonde não for, não há probabilidade. Na figura 28, um exemplo de imagem em *backprojection* com uma face de região de interesse.

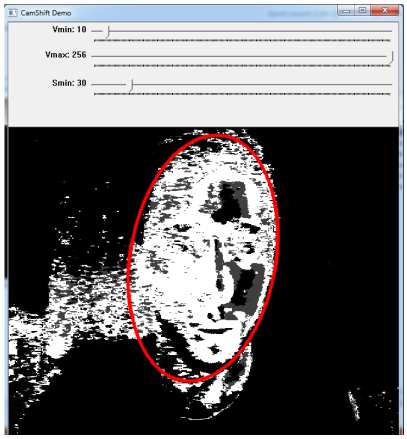

Figura 28 – Imagem resultante da função *backprojection*

Fonte**:** Elaborado pelo autor

A função *Camshift* recebe como parâmetro essa imagem resultante da função *backprojection*, que pode ser chamada de **imagem de probabilidades**, onde são feitos os cálculos (seção 2.3.1.1) para que se resulte um objeto *RotatedRect*. Esse objeto é adaptado com uma função *boundingRect*, que resulta em um retângulo perfeito e este é desenhado sobre a região de interesse na imagem e apresentado, conforme ilustra a figura 29.

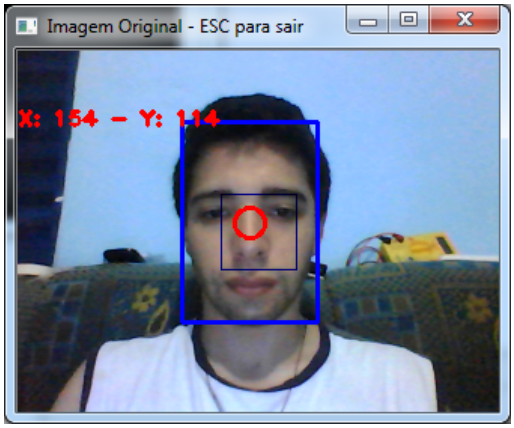

Figura 29 – Retângulo formado sobre objeto ratreado

Fonte**:** Elaborado pelo autor

3.3.4 Controle

O objeto da classe *RotatedRect* gerado pelo algoritmo *Camshift*, tem propriedades que armazenam suas características, um exemplo disso é a propriedade *center*, que guarda os atributos x e y, que são as coordenadas do centro da posição do objeto rastreado.

As variáveis de controle dos servomotores denominadas mvX e mvY, já iniciam no chamado "ponto morto", ou seja, na posição em que o sistema ainda não está executando, onde os servomotores estão direcionados para frente.

Quando a posição do objeto rastreado ultrapassa os limites da chamada margem de erro, que é uma região na forma quadrada de 30 *pixels* de margem em torno do ponto central da imagem (*setpoint*), as funções de controle são acionadas para calcular os valores de ação dos servomotores.

No projeto as funções que realizam esse controle são denominadas *controlServoX* que controla a posição horizontal da câmera, e a *controlServoY* que controla a posição vertical da câmera. Essas funções realizam o cálculo PID (seção 2.4.4), onde se recebe a posição do objeto rastreado, calcula-se o erro entre a região da margem de erro e a posição recebida e se realiza os cálculos proporcional (seção 2.4.1), integral (seção 2.4.2) e derivativo (seção 2.4.3).

As constantes de controle proporcional, integral e derivativo(Kp, Ki e Kd, respectivamente) foram ajustadas pela sintonia manual, utilizando os critérios do Quadro 4.

Parâmetro	**Ao aumentar, o processo ...**	**Ao diminuir, o processo ...**
Kp	Torna-se mais rápido.	Torna-se mais lento.
	Fica mais instável ou mais oscilante.	Geralmente se torna mais estável
		ou menos oscilante.
	Tem mais sobressinal.	Tem menos sobressinal.
Ki	Torna-se mais rápido, atingindo	Torna-se mais lento, demorando para
	rapidamente o *setpoint*.	atingir o *setpoint*.
	Fica mais instável ou mais oscilante.	Fica mais estável ou menos oscilante.
	Tem mais sobressinal.	Tem menos sobressinal.
Kd	Torna-se mais rápido.	Torna-se mais lento.
	Tem mais sobressinal.	Tem menos sobressinal.

Quadro 4 – Efeito de cada parâmetro PID sobre o processo

Fonte**:** SENAI (2012)

3.3.5 Envio da dados para o *Arduino* via Serial

Segundo (NEWARK, 2013), o *Raspberry Pi* tem apenas um pino para PWM, então para o projeto um dispositivo como o *Arduino Uno* que tem 6 pinos digitais que podem ser usados como PWM (seção 2.6) foi a solução viável para realizar esse controle do servomotores.

3.3.5.1 Comunicação no Raspbian

A comunicação do sistema executando na plataforma *Raspbian* com o *Arduino Uno* é feita via Serial através da conexão USB, utilizando a biblioteca *ESerial.h*, que pode ser vista no ANEXO D(EMBECOSM, 2013).

Um objeto da classe *ESerial* é criado e iniciado com a função *connect*, passando como parâmetro o nome da porta de conexão com o *Arduino Uno*(geralmente */dev/ttyACM0*), a taxa de transmissão *baud* (seção 2.6.1) e a paridade de uso.

Se a conexão for satisfatória, a função *sendChar* pode ser utilizada para o envio de dados, passando um *byte* via Serial.

3.3.5.2 Comunicação no Microsoft Windows

A comunicação do sistema executando na plataforma *Windows* com o *Arduino Uno* é feita via Serial através de uma conexão USB(seção 2.6.1), utilizando as bibliotecas *Serial.cpp* e *SerialClass.h*, que podem ser vistas no ANEXO E e ANEXO F(ANDROID, 2014), respectivamente.

Para enviar dados para o *Arduino*, é preciso criar um objeto da classe *Serial*. Este objeto irá, através de sua função *WriteData*, estabelecer a comunicação com uma determinada porta Serial, que no caso desse projeto é a *COM5*.

3.3.5.3 Método de envio dos dados

Como os dados que trafegam na comunicação via Serial são *bytes*, e cada caractere corresponde a um *byte*, os valores obtidos pelo cálculo PID para o servomotor que movimenta horizontalmente e o que movimenta verticalmente, são convertidos de inteiros para caracteres, obtendo o caractere correspondente ao valor original, de acordo com a tabela ASCII. Essa tabela pode ser encontrada no APÊNDICE C(UNINOVE, 2014).

Os dois caracteres obtidos são atribuídos a um vetor de caracteres e enviados para o *Arduino* via comunicação Serial, através da função *WriteData* no caso da plataforma ser *Microsoft Windows* e a função *sendChar* no caso da plataforma ser *Raspbian*.

O código fonte do sistema câmera seguidora pode ser visto no APÊNDICE A

3.3.6 Ação dos Dispositivos Robóticos

No *Arduino Uno*, os dados são recebidos através da classe *Serial*, esta iniciada atribuindo o valor da taxa de transmissão (*baud rate*), conforme visto na seção 2.6.1. O valor passado é 9.600 *baud*.

Os servos são declarados pela biblioteca *Servo.h* e são relacionados com os correspondentes pinos através da função *attach*, passando como parâmetro o número do pino. No caso, o pino 3 pertence à conexão com o objeto da classe Servo, *servoX*, que corresponde ao servomotor que movimenta a câmera horizontalmente, e o pino 6 pertence à conexão com o *servoY*, o servomotor que movimenta a câmera verticalmente.

Todos os procedimentos acima estão dentro da função *setup()*, que é executada assim que o *Arduino* inicia. Essa função é geralmente utilizada para inicialização de variáveis, estados dos pinos, objetos utilizados por bibliotecas, etc(ARDUINO, 2014f). A função *loop()* se comporta exatamente como o nome sugere, ela executa em *loop* permitindo respostas imediatas através de sensores e atuadores(ARDUINO, 2014d)(seção 2.5).

Dentro da função *loop*, é verificado o estado da conexão Serial através da função *available*. A partir da resposta satisfatória dessa função, é feita a leitura dos dados no Serial, esses das são obtidos pela função *read*, e são atribuídos a um vetor de inteiros, convertendo novamente os caracteres recebidos em seus valores originais. Concluído o recebimento dos dados, são movidos os servos através da função da classe *Servo*, *write*, passando os valores recebidos, sendo que o primeiro valor corresponde ao *servoX* e o segundo ao *servoY*.

Como esse código está na função *loop*, o *Arduino*, enquanto ativo, verifica a todo momento se há algum dado enviado pela Serial, caso tenha, movimenta os servomotores de acordo com os dados recebidos. A figura 30 mostra a conexão feita fisicamente entre o *Arduino* e os servomotores.

O código fonte do *Arduino* pode ser visto no APÊNDICE B.

3.4 Resultados

A partir da estrutura apresentada nas seções anteriores, foram realizados testes com o sistema para se obter um melhor controle possível.

Os testes de controle foram baseados na sintonia manual de PID apresentada no quadro 4, no qual foram alterados os valores das constantes proporcional, integral e

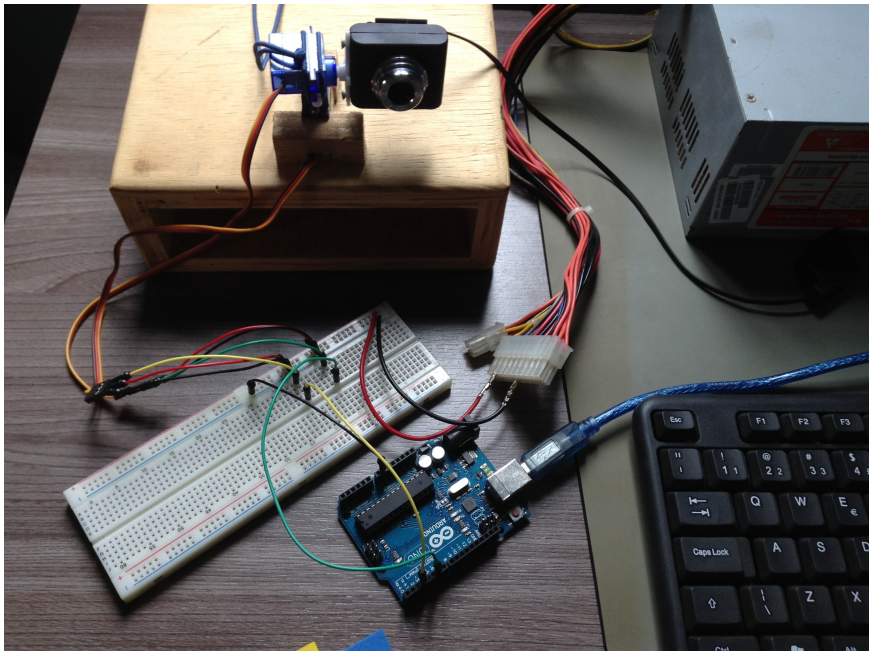

Figura 30 – Conexão do *Arduino* com Servomotores

Fonte**:** Elaborado pelo autor

derivativa, com base em observações de comportamento do processo de rastreamento, até que se obtivesse um resultado satisfatório, ou seja, um processo rápido, estável e sem sobressinal.

3.4.1 Testes de controle – *Raspberry Pi*

Os primeiros testes foram realizados no *Raspberry Pi*, porém foram encontradas algumas dificuldades quanto à capacidade de processamento deste.

A primeira dificuldade encontrada foram com erros causados quando era dado o comando de execução do sistema. Alguns erros eram apresentados, sendo o mais recorrente o erro *"Initial capture error: Unable to load initial memory buffers"*. Esse erro acontecia de forma aleatória. Não foi encontrado uma solução para esse problema.

A segunda dificuldade encontrada foi na detecção de faces. O algoritmo fazia com que o sistema travasse, causando atrasos de cerca de 6 segundos, ou seja, alguma movimentação externa da face a ser rastreada, demorava cerca de 6 segundos para ser reproduzida na tela. Não foi encontrado uma solução para esse problema.

A terceira dificuldade encontrada foi na comunicação Serial com o *Arduino*. A câmera seguia o objeto rastreado, porém com um atraso considerável, movimentos rápidos do objeto rastreado, por exemplo, faziam o sistema se perder. Não foi encontrado uma solução para esse problema.

Pelos problemas apresentados nos testes com o *Raspberry Pi*, não foram encontrados

resultados satisfatórios.

3.4.2 Testes de controle – Notebook

Nos testes feitos no *notebook* não foram encontrados problemas quanto a processamento. Foram realizados dois testes, sendo um capturando imagens com resolução de 640x480 e outro com imagens com resolução de 320x240.

Os resultados do teste com a resolução em 640x480 são apresentados na tabela 1.

Tabela 1 – Testes de Rastreamento com imagens de resolução 640x480

Teste	Kp	Ki	Kd	Velocidade	Estabilidade	Sobressinal
1	0.1	0	0	Rápido	Instável	Com sobressinal
2	0.001	0	0	Lento	Estável	Sem sobressinal
3	0.003	0	0	Rápido	Estável	Com sobressinal
4	0.0025	0	0	Rápido	Estável	Sem sobressinal
5	0.0025	0.001	0	Rápido	Estável	Sem sobressinal
6	0.0025	0.001	0.001	Rápido	Estável	Com sobressinal

Fonte**:** Elaborado pelo autor

Primeiro iniciou-se com um valor qualquer na constante proporcional, no caso 0.1. Esse valor foi considerado um valor alto, pois causou instabilidade no sistema. Então foi diminuído o valor da constante para 0.001, onde se obteve um processo lento, mas estável. Nos próximos testes foram sendo ajustados os valores conforme o comportamento do processo até que no teste 4, foi apresentado um resultado satisfatório apenas com valor na constante proporcional. No testes 5 foi colocado o valor 0.001 na constante integral, onde se obteve um resultado satisfatório também. No teste 6, ao tentar colocar um valor na constante derivativa, o processo voltou a ter sobressinal. Os testes foram concluídos, sendo os testes 4 e 5 sendo como os que se obtiveram resultados satisfatórios.

Nos testes realizados com imagens capturadas de resolução 320x240, os resultados foram diferentes, devido à maior área de captura. Na tabela 2 são apresentados os resultados dos testes.

Primeiramente o valor 0.01 foi lançado na constante proporcional para ser verificado o comportamento do processo. Com este valor o processo foi lento. Então foi elevado o valor para 0.3, que fez com que o processo ficasse rápido, porém com instabilidade.

Tabela 2 – Testes de Rastreamento com imagens de resolução 320x240

Teste	Kp	Ki	Kd	Velocidade	Estabilidade	Sobressinal
1	0.01	0	0	Lento	Estável	Sem sobressinal
2	0.03	0	0	Rápido	Instável	Com sobressinal
3	0.02	0	0	Lento	Estável	Sem sobressinal
4	0.0175	0.001	0	Rápido	Estável	Com sobressinal
5	0.015	0.001	0	Rápido	Estável	Com sobressinal
6	0.015	0.0005	0.001	Rápido	Estável	Com sobressinal
7	0.015	0.0005	0	Rápido	Estável	Com sobressinal
8	0.01	0.0005	0	Rápido	Estável	Com sobressinal
9	0.0125	0.0005	0.0001	Rápido	Estável	Com sobressinal
10	0.012	0.0005	0.0001	Rápido	Estável	Com sobressinal
11	0.012	0.00025	0.0001	Rápido	Estável	Com sobressinal
12	0.011	0.0002	0.0001	Rápido	Estável	Sem sobressinal

Fonte**:** Elaborado pelo autor

Esse valor foi diminuído para 0.2 e então o processo ficou rápido novamente, porém co sobressinal, aí então foram feitos vários ajustes em cada teste para tirar o sobressinal do processo. Enfim, no teste 12 se obteve um processo satisfatório.

3.4.3 Testes de detecção de faces

Como foi mencionado na seção 3.3.2.1, apenas faces frontais são detectadas pelo algoritmo, de acordo com o arquivo de configuração XML passado. A detecção somente falha quando é coberto uma parte da face, como na figura 31 ou quando a face é virada para o lado como apresentado na figura 25 na seção 3.3.2.1.

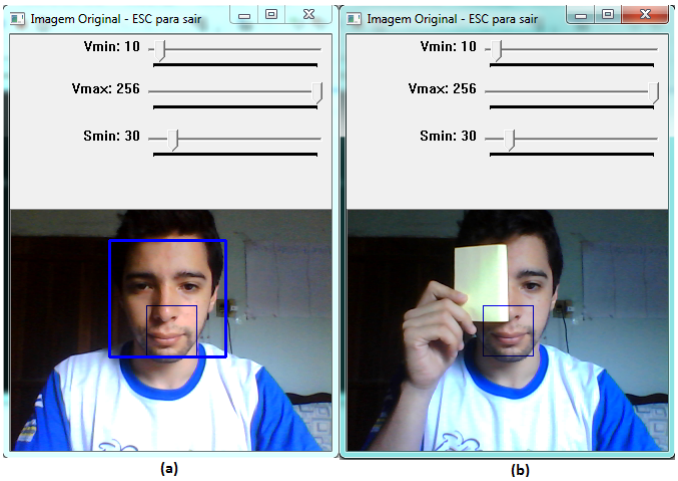

Figura 31 – (a) Detecção de faces (b)Interferência de Detecção de Faces

Fonte**:** Elaborado pelo autor

3.4.4 Testes de rastreamento

Visto que o *Camshift* é um algoritmo de rastreamento baseado em cores, foram feitos testes com objetos com a mesma cor do objetos rastreado ou a face rastreada. Quando um objeto da mesma cor entra no campo de captura da câmera, ele só interfere no rastreamento se este se aproxima da área de rastreamento do objeto. Se colocamos o objeto de interferência ao fundo, sem entrar no campo de captura, não é prejudicado o processo. Na figura 32, em (a) tem-se um objeto da mesma cor do objeto rastreado ao lado, sem interferir no rastreamento. Em (b) esse objeto entra no campo de captura do objeto rastreado, o que causa a interferência.

Figura 32 – (a) Camshift sem interferência (b) Camshift com interferência

Fonte**:** Elaborado pelo autor

3.4.5 Testes de perda de rastreamento

Foram feitos testas para as situações de perda de rastreamento. Percebeu-se que quando o rastreamento era perdido, as coordenadas de rastreamento ficavam com valores zerados, ou seja, $(x = 0)$ e $(y = 0)$. Para corrigir esse erro, foi criado uma função denominada *lost*, que era chamada quando o objeto era perdido ou quando os valores das coordenadas eram zerados. Essa função reseta o sistema, fazendo com que o sistema fique em seu estado inicial, com o rastreamento desativado, capturando as imagens e solicitando um método de detecção, e os servos voltam para a posição inicial, colocando a câmera voltada para frente.

3.4.6 Testes de estabilidade dos servos

Percebeu-se também que os servos ficando, muitas vezes, instáveis, pois como *setpoint* era apenas uma coordenada na imagem, com o mínimo de erro o sistema movia os servomotores para corrigi-lo. Isso acontecia em tempo real, fazendo com que houvessem algumas instabilidades na imagem com os servomotores movendo-se a

todo instante. A solução para este problema foi criar uma margem de erro, como citada na seção 3.3.4.

Essa margem de erro, um quadrado de 60 *pixels* de altura e 60 *pixels* de largura com o *setpoint* no centro, é uma área onde o objeto rastreado tem que ficar posicionado, a correção a partir dos servomotores só ocorre quando o objeto sai dessa área, então os erros de menos expressão são ignorados pelo sistema. A margem de erro é apresentada na figura 33 com o *setpoint* no centro.

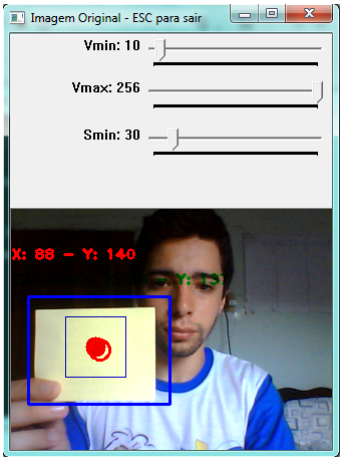

Figura 33 – Exemplo de Margem de erro com Setpoint

Fonte**:** Elaborado pelo autor

4 CONCLUSÃO

Neste trabalho foi desenvolvido um sistema de visão computacional utilizando a biblioteca OpenCV para controlar dispositivos robóticos. A aplicação prática foi a câmera seguidora que tem o objetivo de detectar e rastrear um objeto através de uma câmera convencional, e mover servomotores que controlam a posição da câmera, de forma que a câmera acompanhe os movimentos do objeto de interesse, deixando-o sempre no centro da imagem capturada.

Foram realizados estudos sobre a biblioteca de Visão Computacional OpenCV, da qual foram retirados os principais algoritmos do sistema.

Os algoritmos mais complexos como *Viola-Jones* e *Camshift* apresentaram grande eficiência, possibilitando a execução do sistema de forma a não ocorrerem perdas e falhas constantes.

A plataforma *Arduino* possibilitou a comunicação entre sistema e dispositivos robóticos de forma simples e prática e da mesma forma que foram enviados dados para o *Arduino*, pode-se capturar dados se o objetivo da aplicação for o uso de sensores.

Com o desenvolvimento de uma versão do sistema para a plataforma *Linux*, o sistema demonstrou que pode ser executado utilizando apenas bibliotecas, compiladores e plataforma gratuitos sendo o baixo custo da aplicação prática uma grande contribuição para estudos básicos de Visão Computacional.

A estrutura do sistema permite que o código seja reaproveitado para uso, não só para aplicações de controle de dispositivos robóticos, mas em aplicações simples de Interação Homem-Computador, como sistemas de detecção de movimentos ou sistemas de representação gráfica de objetos (*paint* utilizando visão computacional).

A partir dos resultados obtidos pela aplicação prática, pode-se observar que o sistema atende aos objetivos propostos para este trabalho, obtendo apenas uma limitação, que foi a utilização do *Raspberry Pi*.

4.1 Limitações

A primeira limitação encontrada foi a execução do sistema no *Raspberry Pi*, onde os algoritmos mais avançados de visão computacional causaram travamentos no sistema.

Além dessa limitação relacionada ao processamento, o fato de haver atrasos entre a movimentação do objeto e a correção dos servomotores também contribuíram para que os resultados não fossem satisfatórios com o *Raspberry Pi*, ou seja, não foi possível

executar o sistema de forma correta no dispositivo.

No sistema, uma limitação encontrada foi o fato de a detecção de faces ser somente frontal, não detectando a lateral da face ou em perfil, não possibilitando que o rastreamento fosse feito somente pelo algoritmo *Viola-Jones*, pois qualquer movimento mais significativo com a face a detecção era perdida, perdendo consequentemente o rastreamento se este fosse feito pelo algoritmo *Viola-Jones*.

No algoritmo *Camshift*, a limitação encontrada foi quando houve interferência de luminosidade no objeto rastreado, alterando sua cor e, consequentemente, causando perdas de rastreamento.

4.2 Trabalhos Futuros

Os erros causados no *Raspberry Pi*, podem ter soluções simples, como a utilização da *picamera* feita para uso do próprio *Raspberry Pi* com conexão CSI(HUGHES, 2014). Por ter compatibilidade com o *Raspberry Pi*, e por não ser de conexão USB, essa câmera pode ser a solução para os problemas na inicialização da execução do sistema.

Outra possibilidade para o *Raspberry Pi* seria utilizar seus pinos para conexão direta com os servomotores, eliminando a comunicação serial com *Arduino*. Existem bibliotecas como a *wiringPi* que possibilitam a conexão de servomotores no *Raspberry Pi*, mesmo este não tendo pinos próprios para a função de PWM(HENDERSON, 2014).

Existem câmeras que tem em seu próprio *hardware* as funções de movimentação vertical e horizontal como a *Logitech QuickCam Orbit AF*(LOGITECH, 2014), que podem ser utilizadas para produzir os mesmos resultados desse trabalho, sem a necessidade de utilizar o *Arduino*, apenas conectando a câmera via USB e a controlando via código.

A sintonia PID foi feita de forma manual, porém existem métodos de sintonia como o de *Ziegler-Nichols*(ZIEGLER; NICHOLS, 1942) que obtém resultados bem mais precisos e confiáveis do que os resultados obtidos de forma manual.

Quando é perdido o rastreamento do objeto, o sistema reseta, ou seja, volta a sua posição inicial. Pode-se desenvolver alternativas para esse comportamento como, por exemplo, fazer com que a câmera procure o objeto rastreado depois de perdido.

REFERÊNCIAS

ANDROID. *Android Playground – CPPWindows*. 2014. Disponível em: <http://playground.arduino.cc/Interfacing/CPPWindows>. Acesso em: 16 nov. 2014.

ARDUINO. *Arduino – AnalogInputPins*. 2014. Disponível em: <http://arduino.cc/en/Guide/Introduction>. Acesso em: 02 nov. 2014.

ARDUINO. *Arduino – ArduinoBoardUno*. 2014. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 02 nov. 2014.

ARDUINO. *Arduino – Introduction*. 2014. Disponível em: <http://arduino.cc/en/Guide/Introduction>. Acesso em: 02 nov. 2014.

ARDUINO. *Arduino – loop*. 2014. Disponível em: <http://arduino.cc/en/Reference/loop>. Acesso em: 16 nov. 2014.

ARDUINO. *Arduino – Serial*. 2014. Disponível em: <http://arduino.cc/en/reference/serial>. Acesso em: 02 nov. 2014.

ARDUINO. *Arduino – Setup*. 2014. Disponível em: <http://arduino.cc/en/Reference/Setup>. Acesso em: 16 nov. 2014.

ASUSTEK. *Notebooks and Ultrabooks - A45A - ASUS*. 2014. Disponível em: <http://www.asus.com/br/Notebooks_Ultrabooks/A45A/specifications/>. Acesso em: 21 nov. 2014.

AURÉLIO, D. *Significado de Robô, Dicionário de Português*. 2014. Disponível em: <http://www.dicionariodoaurelio.com/Robo>. Acesso em: 02 nov. 2014.

BECKER, D. T. Graduação, *Visão Computacional Aplicada em um Braço Robótico Antropomórfico Didático*. Cascavel: [s.n.], 2013.

BRADSKI, G.; KAEHLER, A. *Learning OpenCV: computer vision in c++ with the opencv library*. Cambridge: O'Reilly Media, Inc., 2013.

BRADSKI, G. R. Computer vision face tracking for use in a perceptual user interface. *Conference on Computer Vision and Pattern Recognition*, Citeseer, 1998.

BUDIHARTO, W.; GROUP, S. P. *Modern Robotics with OpenCV*. Science Publishing Group, 2014. ISBN 9781940366128. Disponível em: <http://books.google.com.br/books?id=mbFUAwAAQBAJ>. Acesso em: 02 nov. 2014.

CAMPOS, F. M. S. *Deteção e Rastreamento de Faces em Vídeos: como rastrear faces em vídeos.* 2011. Disponível em: <http://www.bitabit.eng.br/2011/02/28/como-rastrear-faces-em-videos/>. Acesso em: 02 nov. 2014.

CARRARA, V. *Apostila de Robótica*. [S.l.], 2007.

CASTLE, R. *Installing OpenCV on a Raspberry Pi*. 2014. Disponível em: <http://robertcastle.com/2014/02/installing-opencv-on-a-raspberry-pi/>. Acesso em: 21 nov. 2014.

CMAKE. *Overview - CMake*. 2014. Disponível em: <http://www.cmake.org/overview>. Acesso em: 21 nov. 2014.

ELINUX. *RPi Hardware – eLinux.org*. 2014. Disponível em: <http://elinux.org/RPi_Hardware>. Acesso em: 02 nov. 2014.

EMBECOSM. *esp9-arduino-facetracker – GitHub*. 2013. Disponível em: <https://github.com/embecosm/esp9-arduino-facetracker>. Acesso em: 17 nov. 2014.

EMGU. *EmguCV – OpenCV in .NET (C#, VB, C++ and more)*. 2014. Disponível em: <http://www.emgu.com/wiki/index.php/Main_Page>. Acesso em: 08 nov. 2014.

FACCIN, F. *Abordagem Inovadora no Projeto de Controladores PID*. Dissertação (Mestrado) — Universidade Federal do Rio Grande do Sul. Escola de Engenharia. Departamento de Engenharia Química., Porto Alegre, 2004.

FACON, J. *Processamento e Análise de Imagens*. Paraná: [s.n.], 2005.

FERNANDES, L. S. Interação humano–computador: visão computacional como meio de interação. Guaxupé, 2012.

FOUNDATION, R. P. *FAQs – Raspiberry Pi*. 2014. Disponível em: <http://www.raspberrypi.org/help/faqs/>. Acesso em: 02 nov. 2014.

FOUNDATION, R. P. *Raspberry Pi*. 2014. Disponível em: <http://www.raspberrypi.org/>. Acesso em: 02 nov. 2014.

FUENTES, R. C. *Apostila de Automação Industrial*. Santa Maria, 2005.

GARBIN, S. M. Estudo das evolução das interfaces homem-computador. *Trabalho de conclusão de curso (Engenharia Elétrica com ênfase em Eletrônica)-Escola de engenharia de São Carlos, Universidade de São Paulo, São Carlos, SP*, 2010.

GONZALEZ, R. C.; WOODS, R. E. *Digital Image Processing*. 3. ed. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2006. ISBN 013168728X.

HEINEN, F. et al. Navegação de veículos de carga autônomos utilizando visão computacional com algoritmo de segmentação por cores. In: *IV Congresso Internacional de Automação, Sistemas e Instrumentação*. [S.l.: s.n.], 2004. p. 1–10.

HENDERSON, G. *Software PWM Library*. 2014. Disponível em: <https://projects.drogon.net/raspberry-pi/wiringpi/software-pwm-library/>. Acesso em: 21 nov. 2014.

HUGHES, D. *picamera*. 2014. Disponível em: <http://picamera.readthedocs.org/en/release-1.8/>. Acesso em: 21 nov. 2014.

IKEDA, O. Segmentation of faces in video footage using hsv color for face detection and image retrieval. In: *Image Processing, 2003. ICIP 2003. Proceedings. 2003 International Conference on*. [S.l.: s.n.], 2003. v. 3, p. III–913–6 vol.2. ISSN 1522-4880.

ITSEEZ. *OpenCV*. 2014. Disponível em: <http://itseez.com/OpenCV/>. Acesso em: 02 nov. 2014.

JÚNIOR, J. M. d. S. *Sistema de Captura de Movimentos para Comandar Equipamentos Eletrônicos*. Gravataí: [s.n.], 2010.

KIKUCHI, D. Y. *Sistema de Controle Servo Visual de uma Câmera Pan-Tilt com Rastreamento de uma Região de Referência*. Dissertação (Mestrado) — Universidade de São Paulo, 2007.

LEPARMENTIER, G. *Manipulating colors in NET Part I*. 2014. Disponível em: <http://www.codeproject.com/Articles/19045/Manipulating-colors-in-NET-Part>. Acesso em: 16 nov. 2014.

LIENHART, R.; KURANOV, A.; PISAREVSKY, V. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. In: *Pattern Recognition*. [S.l.]: Springer, 2003. p. 297–304.

LOGITECH. *QuickCam Orbit AF*. 2014. Disponível em: <http://support.logitech.com/product/3480>. Acesso em: 21 nov. 2014.

MANZI, F. A. Graduação, *Aplicação de Visão Computacional para Extração de Características em Imagens do Olho Humano*. São Carlos: [s.n.], 2007.

MARENGONI, M.; STRINGHINI, S. Tutorial: introdução à visão computacional usando opencv. *Revista de Informática Teórica e Aplicada*, v. 16, n. 1, p. 125–160, 2009.

MCROBERTS, M. *Arduino Básico*. [S.l.]: Novatec Editora, 2011.

MEDINA, M.; FERTIG, C. Algoritmos e programação. *São Paulo, Novatec Editora*, 2005.

MICROSOFT. *Kinect*. 2014. Disponível em: <http://www.xbox.com/pt-BR/Kinect/Home-new>. Acesso em: 21 nov. 2014.

MICROSOFT. *Microsoft Dreamspark – O que é Dreamspark*. 2014. Disponível em: <https://www.dreamspark.com/What-Is-Dreamspark.aspx>. Acesso em: 21 nov. 2014.

MILANO, D. d.; HONORATO, L. B. Visão computacional. 2010.

MOLZ, R. F. *Uma Metodologia para o Desenvolvimento de Aplicações de Visão Computacional Utilizando um Projeto Conjunto de Hardware e Software*. Tese (Doutorado) — Universidade Federal do Rio Grande do Sul. Instituto de Informática. Programa de Pós-Graduação em Computação., 2001.

MOORE, C. F. *Instrument Engineer's Handbook: process control.* 3. ed. [S.l.]: CRC Press, 1999.

MORAES, A. A.; JÚNIOR, N. M.; PECEGUEIRO, J. J. *Robótica*. [S.l.], 2003.

MULTILÓGICA. *Braço robótico Edge OWI-535*. 2014. Disponível em: <http://multilogica-shop.com/Bra%C3%A7o-rob%C3%B3tico-Edge-OWI-535>. Acesso em: 16 nov. 2014.

NEWARK, C. O. *PWM Control with the Raspberry Pi*. 2013. Disponível em: <http://embedded-computing.com/guest-blogs/pwm-control-with-the-raspberry-pi/>. Acesso em: 16 nov. 2014.

OGATA, K. *Engenharia de Controle Moderno*. 5. ed. [S.l.]: Prentice-Hall, Inc., 2010.

OLIVEIRA, G. C. *Algoritmo PID*. São Carlos, 2013.

OPENCV. *How to build applications with OpenCV inside the Microsoft Visual Studio*. 2014. Disponível em: <http://docs.opencv.org/doc/tutorials/introduction/windows_visual_studio_Opencv/windows_visual_studio_Opencv.html>. Acesso em: 21 nov. 2014.

OPENCV. *Instalation on Windows*. 2014. Disponível em: <http://docs.opencv.org/doc/tutorials/introduction/windows_install/windows_install.html>. Acesso em: 21 nov. 2014.

OPENCV. *Meanshift and Camshift*. 2014. Disponível em: <http://docs.opencv.org/trunk/doc/py_tutorials/py_video/py_meanshift/py_meanshift.html>. Acesso em: 16 nov. 2014.

OPENCV, D. T. *Using OpenCV with gcc and CMake*. 2014. Disponível em: <http://docs.opencv.org/doc/tutorials/introduction/linux_gcc_cmake/linux_gcc_cmake.html>. Acesso em: 16 nov. 2014.

OSHIRO, S. H.; GOLDSCHMIDT, R. R. Monografia, *Processamento de Imagens*. Rio de Janeiro: [s.n.], 2008. Disponível em: <http://www2.comp.ime.eb.br/techreports/repositorio/2008_05.pdf>. Acesso em: 02 nov. 2014.

PEIXOTO, P. T. d. C. *Deteção e Acompanhamento de Movimento Através de uma Câmara de Vídeo*. Dissertação (Mestrado) — Instituto Politécnico do Porto. Instituto Superior de Engenharia do Porto, 2012.

PIERI, E. R. d. Curso de robótica móvel. *UFSC–Universidade Federal de Santa Catarina*, 2002.

PIO, J. L. d. S.; CASTRO, T. H. C. d.; JÚNIOR, A. N. d. C. A robótica móvel como instrumento de apoio à aprendizagem de computação. In: *Anais do Simpósio Brasileiro de Informática na Educação*. [S.l.: s.n.], 2006. v. 1, n. 1, p. 497–506.

PROJECT, Q. *CMake Manual*. 2014. Disponível em: <http://www.qt-project.org/doc/qt-5/cmake-manual.html>.

RENATO, F. *Como desbloquear o Android 4.0 pelo reconhecimento facial*. 2014. Disponível em: <http://www.techtudo.com.br/dicas-e-tutoriais/noticia/2012/08/como-desbloquear-o-android-40-pelo-reconhecimento-facial.html>. Acesso em: 21 nov. 2014.

RENNA, R. B. e. a. D. Introdução ao kit de desenvolvimento arduino. *Tutoriais PET-Tele (Universidade Federal Fluminense–UFF)*, Niterói, 2013.

RÉZIO, A. C. C. Graduação, *Reconhecimento e Rastreamento de Objetos*. Goiás: [s.n.], 2008.

RICHARDSON, M.; WALLACE, S. *Primeiros Passos com o Raspberry Pi*. São Paulo: Editora Novatec, 2013.

ROUBEROL, B. *Comparision of Face Detection Tools (face.com, opencv, libccv)*. 2012. Disponível em: <http://balthazar-rouberol.com/blog/2012/11/25/comparison-of-face-detection-tools-face-com-opencv-libccv/>. Acesso em: 16 nov. 2014.

SANTEE, A. Programação de jogos com c++ e directx. *São Paulo: Novatec*, 2005.

SANTOS, A. *Servomotores*. [S.l.], 2007. Disponível em: <www.pictronics.com.br/downloads/apostilas/servomotores.pdf>. Acesso em: 02 nov. 2014.

SANTOS, A. K. d. Os ides (ambiente de desenvolvimento integrado) como ferramentas de trabalho de informática. 2007.

SENAI, U. E. de Desenvolvimento Educacional do. *Técnico em Automação Industrial*. Rio Grande do Sul, 2012.

SILVA, M. O. et al. O sistema integrado do time de futebol de robôs uspdroids. *Team Description Paper on Latin American Robots Competition*, 2008.

SILVA, R. C. d. *Rastreamento em Tempo Real com Câmera de Pan-Tilt*. Bauru: [s.n.], 2011.

SOUSA, K. A. O. *Uso de Visão Computacional em Dispositivos Móveis para Auxílio à Travessia de Pedestres com Deficiência Visual*. Dissertação (Mestrado) — Universidade Presbiteriana Mackenzie, São Paulo, 2013.

SOUZA, L. R. d. Graduação, *Algoritmo para Reconhecimento e Acompanhamento de Trajetória de Padrões em Vídeos*. Juazeiro: [s.n.], 2011.

SPARKGO. *Datasheet Micro Servo SG90*. 2014. Disponível em: <http://datasheet.sparkgo.com.br/SG90Servo.pdf>. Acesso em: 16 nov. 2014.

UNINOVE. *Tabela ASCII*. 2014. Disponível em: <http://files.uninove720.webnode.com.br/200000039-1ec291f0ba/00%20Tabela_ASCII.pdf>. Acesso em: 21 nov. 2014.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: IEEE. *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*. [S.l.], 2001. v. 1, p. I–511.

WIKIMEDIA. *Colors – Wikimedia Commons*. 2014. Disponível em: <http://commons.wikimedia.org/wiki/Colors>. Acesso em: 16 nov. 2014.

WIKIPEDIA. *Raspberry Pi photo*. 2014. Disponível em: <http://pt.wikipedia.org/wiki/Ficheiro:Raspberry_Pi_Photo.jpg>. Acesso em: 16 nov. 2014.

WVSHARE. *Raspberry Pi model B*. 2014. Disponível em: <http://www.wvshare.com/product/RPi-B-CN.htm>. Acesso em: 16 nov. 2014.

ZIEGLER, J.; NICHOLS, N. Optimum settings for automatic controllers. *trans. ASME*, v. 64, n. 11, 1942.

ANEXO A – INSTRUÇÕES DE INSTALAÇÃO DO OPENCV NAS DISTRIBUIÇÕES LINUX BASEADOS EM DEBIAN

As instruções abaixo devem ser executados no *Terminal* do *Linux*.

Atualizar o sistema Primeiro passo é a atualização do sistema executando os seguintes comandos:

```
1 sudo apt-get update
2 sudo apt-get upgrade
```

Instalação das dependências 1 Primeiro as seguintes dependências:

```
1 sudo apt-get -y install build-essential cmake cmake-curses-
    gui pkg-config libpng12-0 libpng12-dev libpng++-dev
    libpng3 libpnglite-dev zlib1g-dbg zlib1g zlib1g-dev
    pngtools libtiff4-dev libtiff4 libtiffxx0c2 libtiff-tools
     libeigen3-dev
```

Instalação das dependências 2 Depois essas:

```
1 sudo apt-get -y install libjpeg8 libjpeg8-dev libjpeg8-dbg
    libjpeg-progs ffmpeg libavcodec-dev libavcodec53
    libavformat53 libavformat-dev libgstreamer0.10-0-dbg
    libgstreamer0.10-0 libgstreamer0.10-dev libxine1-ffmpeg
    libxine-dev libxine1-bin libunicap2 libunicap2-dev swig
    libv4l-0 libv4l-dev python-numpy libpython2.6 python-dev
    python2.6-dev libgtk2.0-dev
```

Instalação do OpenCV Fazer download do OpenCV pelo comando:

```
1 wget http://ufpr.dl.sourceforge.net/project/opencvlibrary/
    opencv-unix/2.4.9/opencv-2.4.9.zip
```

Descompactar e preparar para construção Executar os seguintes comandos:

```
1 unzip opencv-2.4.9.zip
2 cd opencv-2.4.9
3 mkdir release
4 cd release
5 ccmake ../
```

Configuração Para configurar é necessário pressionar a tecla ‘c’ e colocar as opções de configuração conforme a necessidade do projeto. Para terminar a configuração, pressionar ‘c’ novamente e ‘g’ para criação do *Makefile*.

Construção Com as instruções abaixo o OpenCV será compilado e instalado no *Linux*:

```
1 make
2 sudo make install
```

ANEXO B – INSTRUÇÕES DE INSTALAÇÃO DO OPENCV NO WINDOWS 7

Download do OpenCV Fazer download do opencv através do link: http://sourceforge.net/projects/opencvlibrary/files/opencv-win/

Descompactação Descompactar o arquivo em uma pasta de preferência, por exemplo C:/opencv/.

Criar variáveis de ambiente Para a criação das variáveis de ambiente, abrir o Prompt de comando e executar um dos seguintes comandos:

```
1 REM Para Windows 32 bit
2 setx -m OPENCV_DIR C:/opencv/Build/x86/vc11
3 REM Para Windows 64 bit
4 setx -m OPENCV_DIR C:/opencv/Build/x64/vc11
```

ANEXO C – TABELA ASCII

Tabela ASCII (American Standard Code for Information Interchange)

A tabela ASCII pode ser dividida em três partes:

- **conjunto de controle**: códigos de 00h até 1Fh (0 a 31 decimal). Usados para controle de dispositivos. Por exemplo, o código 0Ch (*form feed*) recebido por uma impressora determinada gera um avanço de uma página. O código 0Dh (*carriage return*) é enviado pelo teclado quando a tecla ENTER é pressionada. Tais códigos podem mudar de acordo com o equipamento.
- **conjunto padrão**: códigos de 20h a 7Fh (32 a 127 decimal). Código padrão para qualquer computador. Representam os caracteres usados na manipulação de textos: códigos-fonte, documentos, mensagens de correio eletrônico, etc. São constituídos das letras do alfabeto latino (minúsculo e maiúsculo), números e os símbolos mais usados (como +, -, > etc.).
- **conjunto estendido**: códigos de 80h até FFh (128 a 255 decimal). São caracteres imprimíveis de livre uso por cada fabricante. Pertencem a esta parte do código os caracteres especiais: é, ç, ã, ü ...

Dec.	Hex.	Controle		Dec.	Hex.	Controle
0	00h	NUL (Null)		24	18h	CAN (Cancel)
1	01h	SOH (Start of Heading)		25	19h	EM (End of Media)
2	02h	STX (Start of Text)		26	1Ah	SUB (Substitute)
3	03h	ETX (End of Text)		27	1Bh	ESC (Escape)
4	04h	EOT (End of Transmision)		28	1Ch	FS (File Separator)
5	05h	ENQ (Enquiry)		29	1Dh	GS (Group Separator)
6	06h	ACK (Acknowledge)		30	1Eh	RS (Record Separator)
7	07h	BEL (Bell)		31	1Fh	US (Unit Separator)
8	08h	BS (Backspace)				
9	09h	HT (Horizontal Tab)				
10	0Ah	LF (Line Feed)				
11	0Bh	VT (Vertical Tab)				
12	0Ch	FF (Form Feed)				
13	0Dh	CR (Carriage Return)				
14	0Eh	SO (Shift Out)				
15	0Fh	SI (Shift In)				
16	10h	DLE (Data Link Escape)				
17	11h	DC1 (Device control 1)				
18	12h	DC2 (Device control 2)				
19	13h	DC3 (Device control 3)				
20	14h	DC4 (Device control 4)				
21	15h	NAK (Negative Acknowledge)				
22	16h	SYN (Synchronous Idle)				
23	17h	ETB (End Transmission Block)				

Dec.	Hex.	Caractere		Dec.	Hex.	Caracter		Dec.	Hex.	Caractere
32	20h	<espaco>		81	51h	Q		130	82h	é
33	21h	!		82	52h	R		131	83h	â
34	22h	"		83	53h	S		132	84h	ä
35	23h	#		84	54h	T		133	85h	à
36	24h	$		85	55h	U		134	86h	å
37	25h	%		86	56h	V		135	87h	c
38	26h	&		87	57h	W		136	88h	ê
39	27h	'		88	58h	X		137	89h	ë
40	28h	(		89	59h	Y		138	8Ah	è
41	29h	)		90	5Ah	Z		139	8Bh	ï
42	2Ah	*		91	5Bh	[		140	8Ch	î
43	2Bh	+		92	5Ch	\		141	8Dh	ì
44	2Ch	,		93	5Dh	]		142	8Eh	Ä
45	2Dh	-		94	5Eh	^		143	8Fh	Å
46	2Eh	.		95	5Fh	_		144	90h	É
47	2Fh	/		96	60h	`		145	91h	æ
48	30h	0		97	61h	a		146	92h	Æ
49	31h	1		98	62h	b		147	93h	ô
50	32h	2		99	63h	c		148	94h	ö
51	33h	3		100	64h	d		149	95h	ò
52	34h	4		101	65h	e		150	96h	û
53	35h	5		102	66h	f		151	97h	ù
54	36h	6		103	67h	g		152	98h	ÿ
55	37h	7		104	68h	h		153	99h	Ö
56	38h	8		105	69h	i		154	9Ah	Ü
57	39h	9		106	6Ah	j		155	9Bh	¢
58	3Ah	:		107	6Bh	k		156	9Ch	£
59	3Bh	;		108	6Ch	l		157	9Dh	¥
60	3Ch	<		109	6Dh	m		158	9Eh	₧
61	3Dh	=		110	6Eh	n		159	9Fh	ƒ
62	3Eh	>		111	6Fh	o		160	A0h	á
63	3Fh	?		112	70h	p		161	A1h	í
64	40h	@		113	71h	q		162	A2h	ó
65	41h	A		114	72h	r		163	A3h	ú
66	42h	B		115	73h	s		164	A4h	ñ
67	43h	C		116	74h	t		165	A5h	Ñ
68	44h	D		117	75h	u		166	A6h	ª
69	45h	E		118	76h	v		167	A7h	º
70	46h	F		119	77h	w		168	A8h	¿
71	47h	G		120	78h	x		169	A9h	⌐
72	48h	H		121	79h	y		170	AAh	¬
73	49h	I		122	7Ah	z		171	ABh	½
74	4Ah	J		123	7Bh	{		172	ACh	¼
75	4Bh	K		124	7Ch	|		173	ADh	¡
76	4Ch	L		125	7Dh	}		174	AEh	«
77	4Dh	M		126	7Eh	~		175	AFh	»
78	4Eh	N		127	7Fh	<delete>		176	B0h	░
79	4Fh	O		128	80h	Ç		177	B1h	▒
80	50h	P		129	81h	ü		178	B2h	▓

Dec.	Hex.	Caractere		Dec.	Hex.	Caractere
179	B3h	┃		226	E2h	Γ
180	B4h	┤		227	E3h	π
181	B5h	╡		228	E4h	Σ
182	B6h	╢		229	E5h	σ
183	B7h	╖		230	E6h	µ
184	B8h	╕		231	E7h	T
185	B9h	╣		232	E8h	Φ
186	BAh	║		233	E9h	Θ
187	BBh	╗		234	EAh	Ω
188	BCh	╝		235	EBh	δ
189	BDh	╜		236	ECh	∞
190	BEh	╛		237	EDh	φ
191	BFh	┐		238	EEh	∈
192	C0h	└		239	EFh	∩
193	C1h	┴		240	F0h	≡
194	C2h	┬		241	F1h	±
195	C3h	├		242	F2h	≥
196	C4h	─		243	F3h	≤
197	C5h	┼		244	F4h	⌠
198	C6h	╞		245	F5h	⌡
199	C7h	╟		246	F6h	÷
200	C8h	╚		247	F7h	≈
201	C9h	╔		248	F8h	°
202	CAh	╩		249	F9h	∙
203	CBh	╦		250	FAh	·
204	CCh	╠		251	FBh	√
205	CDh	═		252	FCh	ⁿ
206	CEh	╬		253	FDh	²
207	CFh	╧		254	FEh	∙
208	DOh	╨		255	FFh	
209	D1h	╤				
210	D2h	╥				
211	D3h	╙				
212	D4h	╘				
213	D5h	╒				
214	D6h	╓				
215	D7h	╫				
216	D8h	╪				
217	D9h	┘				
218	DAh	┌				
219	DBh					
220	DCh					
221	DDh					
222	DEh					
223	DFh					
224	E0h	A				
225	E1h	ß				

ANEXO D – CÓDIGO FONTE DA BIBLIOTECA ESERIAL.H - LINUX

```
1  //
   ------------------------------------------------------------------------

2  // Trivial Linux serial driver class
3
4  // Copyright (C) 2013  Embecosm Limited <info@embecosm.com>
5
6  // Contributor Jeremy Bennett <jeremy.bennett@embecosm.com>
7
8  // This program is free software: you can redistribute it and/or
       modify it
9  // under the terms of the GNU Lesser General Public License as
       published by
10 // the Free Software Foundation, either version 3 of the License, or
       (at your
11 // option) any later version.
12
13 // This program is distributed in the hope that it will be useful,
       but WITHOUT
14 // ANY WARRANTY; without even the implied warranty of MERCHANTABILITY
        or
15 // FITNESS FOR A PARTICULAR PURPOSE.  See the GNU Lesser General
       Public
16 // License for more details.
17
18 // You should have received a copy of the GNU Lesser General Public
       License
19 // along with this program.  If not, see <http://www.gnu.org/licenses
       />.
20 //
   ------------------------------------------------------------------------

21
22 // Based on a subset of the Tserial interface proposed by Thierry
       Schneider
23 // and inspired by the ideas on Arduino playground about serial
       connections
24 // between Linux and Arduino. This only provides the functionality
       needed to
25 // run the Instructables face tracker example
```

```
26 #ifndef ESERIAL_H
27 #define ESERIAL_H
28
29 #include <cstdlib>
30 #include <cstring>
31 #include <errno.h>
32 #include <sys/types.h>
33 #include <sys/stat.h>
34 #include <fcntl.h>
35 //#include <unistd.h>
36 #include <iostream>
37 #include <sstream>
38
39 using std::cerr;
40 using std::endl;
41 using std::ostringstream;
42 using std::string;
43
44 //! Enumeration for parity types
45 enum serial_parity {
46   spNONE,
47   spODD,
48   spEVEN };
49
50 class Eserial
51 {
52  private:
53   int  handle;                          //!< File handle to serial
       line.
54
55  public:
56   //! Constructor
57   Eserial()
58     {
59       handle = -1;
60     }
61
62   //! Destructor
63   ~Eserial()
64     {
65       disconnect();
66     }
```

```
67
68  //! Close the connection
69  void disconnect(void)
70  {
71    // Don't care if it fails
72    if (-1 != handle)
73      {
74        close (handle);
75        handle = -1;
76    }
77  }
78  //! Open the connection.
79
80  //! @param[in] portname  The name of the port to use
81  //! @param[in] rate      The bit rate to set
82  //! @param[in] parity    The parity to use - ignored
83  int  connect (const char    *portname,
84                int           rate,
85                serial_parity  parity)
86  {
87    string portstr = portname;
88    string ratestr =
89      static_cast<ostringstream*>(&(ostringstream() << rate))->str();
90    string comm = "stty -F " + portstr + " cs8 " + ratestr
91      + " ignbrk -brkint -icrnl -imaxbel -opost -onlcr -isig"
92      + " -icanon -iexten -echo -echoe -echok -echoctl -echoke"
93      + " noflsh -ixon -crtscts";
94    std::cout << comm << endl;
95
96    if (-1 == system (comm.c_str()))
97      {
98        cerr << "ERROR: stty failed: " << strerror (errno) << endl;
99      }
100
101   handle = open (portname, O_RDWR);
102   if (-1 == handle)
103     {
104       cerr << "ERROR: open failed: " << strerror (errno) << endl;
105     }
106 }
107 //! Write a byte
108 //! @param[in] data  The byte to write.
```

```
109     void sendChar(char data)
110     {
111       if (-1 == write (handle, &data, 1))
112         {
113           cerr << "ERROR: write failed: " << strerror (errno) << endl;
114         }
115     }
116 };
117
118 #endif  // ESERIAL_H
```

ANEXO E – CÓDIGO FONTE DA BIBLIOTECA SERIAL.CPP - WINDOWS

```
1  #include "SerialClass.h"
2  #include <string.h>
3
4  using namespace std;
5
6  Serial::Serial(char* portName)
7  {
8      //We're not yet connected
9      this->connected = false;
10
11     //Try to connect to the giv0en port throuh CreateFile
12     this->hSerial = CreateFileA(portName,
13             GENERIC_READ | GENERIC_WRITE,
14             0,
15             NULL,
16             OPEN_EXISTING,
17             FILE_ATTRIBUTE_NORMAL,
18             NULL);
19
20     //Check if the connection was successfull
21     if(this->hSerial==INVALID_HANDLE_VALUE)
22     {
23         //If not success full display an Error
24         if(GetLastError()==ERROR_FILE_NOT_FOUND){
25
26             //Print Error if neccessary
27             printf("ERROR: Handle was not attached. Reason: %s not
                   available.\n", portName);
28
29         }
30         else
31         {
32             printf("ERROR!!!");
33         }
34     }
35     else
36     {
37         //If connected we try to set the comm parameters
38         DCB dcbSerialParams = {0};
```

```
39
40          //Try to get the current
41          if (!GetCommState(this->hSerial, &dcbSerialParams))
42          {
43              //If impossible, show an error
44              printf("failed to get current serial parameters!");
45          }
46          else
47          {
48              //Define serial connection parameters for the arduino
                    board
49              dcbSerialParams.BaudRate=CBR_9600;
50              dcbSerialParams.ByteSize=8;
51              dcbSerialParams.StopBits=ONESTOPBIT;
52              dcbSerialParams.Parity=NOPARITY;
53
54               //Set the parameters and check for their proper
                     application
55              if(!SetCommState(hSerial, &dcbSerialParams))
56              {
57                  printf("ALERT: Could not set Serial Port parameters")
                        ;
58              }
59              else
60              {
61                   //If everything went fine we're connected
62                  this->connected = true;
63                  //We wait 2s as the arduino board will be reseting
64                  Sleep(ARDUINO_WAIT_TIME);
65              }
66          }
67      }
68
69  }
70
71  Serial::~Serial()
72  {
73      //Check if we are connected before trying to disconnect
74      if(this->connected)
75      {
76          //We're no longer connected
77          this->connected = false;
```

```
78          //Close the serial handler
79          CloseHandle(this->hSerial);
80      }
81  }
82
83  int Serial::ReadData(char *buffer, unsigned int nbChar)
84  {
85      //Number of bytes we'll have read
86      DWORD bytesRead;
87      //Number of bytes we'll really ask to read
88      unsigned int toRead;
89
90      //Use the ClearCommError function to get status info on the
            Serial port
91      ClearCommError(this->hSerial, &this->errors, &this->status);
92
93      //Check if there is something to read
94      if(this->status.cbInQue>0)
95      {
96          //If there is we check if there is enough data to read the
                required number
97          //of characters, if not we'll read only the available
                characters to prevent
98          //locking of the application.
99          if(this->status.cbInQue>nbChar)
100         {
101             toRead = nbChar;
102         }
103         else
104         {
105             toRead = this->status.cbInQue;
106         }
107
108         //Try to read the require number of chars, and return the
                number of read bytes on success
109         if(ReadFile(this->hSerial, buffer, toRead, &bytesRead, NULL)
                && bytesRead != 0)
110         {
111             return bytesRead;
112         }
113
114     }
```

```
115
116     //If nothing has been read, or that an error was detected return
            -1
117     return -1;
118
119 }
120
121 bool Serial::WriteData(char *buffer, unsigned int nbChar)
122 {
123     DWORD bytesSend;
124
125     //Try to write the buffer on the Serial port
126     if(!WriteFile(this->hSerial, (void *)buffer, nbChar, &bytesSend,
            0))
127     {
128         //In case it don't work get comm error and return false
129         ClearCommError(this->hSerial, &this->errors, &this->status);
130
131         return false;
132     }
133     else
134         return true;
135 }
136
137 bool Serial::IsConnected()
138 {
139     //Simply return the connection status
140     return this->connected;
141 }
```

ANEXO F – CÓDIGO FONTE DA BIBLIOTECA SERIAL.H - WINDOWS

```
1  #ifndef SERIALCLASS_H_INCLUDED
2  #define SERIALCLASS_H_INCLUDED
3  #define ARDUINO_WAIT_TIME 2000
4  #include <windows.h>
5  #include <stdio.h>
6  #include <stdlib.h>
7
8  class Serial
9  {
10     private:
11         //Serial comm handler
12         HANDLE hSerial;
13         //Connection status
14         bool connected;
15         //Get various information about the connection
16         COMSTAT status;
17         //Keep track of last error
18         DWORD errors;
19
20     public:
21         //Initialize Serial communication with the given COM port
22         Serial(char* portName);
23         //Close the connection
24         //NOTA: for some reason you can't connect again before
               exiting
25         //the program and running it again
26         ~Serial();
27         //Read data in a buffer, if nbChar is greater than the
28         //maximum number of bytes available, it will return only the
29         //bytes available. The function return -1 when nothing could
30         //be read, the number of bytes actually read.
31         int ReadData(char *buffer, unsigned int nbChar);
32         //Writes data from a buffer through the Serial connection
33         //return true on success.
34         bool WriteData(char *buffer, unsigned int nbChar);
35         //Check if we are actually connected
36         bool IsConnected();
37 };
38
```

```
39 #endif // SERIALCLASS_H_INCLUDED
```

APÊNDICE A – CÓDIGO FONTE DO SISTEMA DA CÂMERA SEGUIDORA - WINDOWS(OPÇÕES PARA LINUX EM COMENTÁRIOS)

```
1  #include "opencv2/imgproc/imgproc.hpp"
2  #include "opencv2/highgui/highgui.hpp"
3  #include "opencv2/video/tracking.hpp"
4  #include "opencv/cv.h"
5  /* (CASO SEJA LINUX)
6  #include "Eserial.h"*/
7  #include "SerialClass.h"
8  #include <stdlib.h>
9
10 using namespace cv;
11 using namespace std;
12
13 //VARIAVEIS GLOBAIS
14 const int FRAME_WIDTH = 320;
15 const int FRAME_HEIGHT = 240;
16 const string WINDOW_ORIGINAL_NAME = "Imagem Original - ESC para sair"
       ;
17 const int tempo = 1;
18 const int idCamera = 0;
19
20 //VARIAVEIS DE COTROLE DOS VALORES HSV
21 int vmin = 10, vmax = 256, smin = 30;
22
23 //VARIAVEIS DE CONTROLE DA SELECAO
24 bool selectObject = false;
25 bool mouseMode = false;
26 Point origin;
27 Rect selection;
28
29 //VARIAVEIS DE CONTROLE DA FACE
30 bool faceMode = false;
31 bool faceSelected = false;
32
33 //VARIAVEIS DE CONTROLE DE CALIBRACAO
34 bool calibrate = false;
35 bool controlCamera = false;
36
37 //IMAGENS PRINCIPAIS
```

```
38 Mat roi;
39 Mat mask = Mat(Size(FRAME_WIDTH, FRAME_HEIGHT), CV_8UC3, Scalar
       (255,255,255));
40 Mat originalImage, hsv, maskThresh, hue, backproj, hist,
       grayscaleFrame = Mat::zeros(FRAME_HEIGHT, FRAME_WIDTH, CV_8UC3);
41
42 //VARIAVEIS DE CONTROLE
43 int trackObject = 0;
44 bool showBackProj = false;
45
46 //COORDENADAS
47 int calibX, calibY;
48 Point pontoOneAreaDeteccao((FRAME_WIDTH/2)-25, (FRAME_HEIGHT/2)-25);
49 Point pontoTwoAreaDeteccao((FRAME_WIDTH/2)+25, (FRAME_HEIGHT/2)+25);
50
51 //PID
52 int err, prevErr;
53 double dt = 0;
54 double p = 0;
55 double i = 0;
56 double d = 0;
57 double mvX = 110;
58 double mvY = 70;
59
60 //ENTRADA DO TECLADO
61 char c;
62
63 //CONEXAO SERIAL
64 Serial* ser;
65 char portaSerial[5] = "COM5";
66 /* (CASO SEJA LINUX)
67 Eserial * arduino_com; */
68
69 //TEXTOS DE AJUDA
70 static void help()
71 {
72     cout << "\n\n Menu: \n"
73             "\t ESC - SAIR\n"
74                         "\t r - RESET\n"
75                         "\t c - CALIBRAR (Caso nao calibre, a
                               referencia sera o centro da imagem)\n"
76             "\t s - INICIALIZAR CONTROLE\n"
```

```
77                          "\t o - VER BACKPROJECT\n\n"
78              "Selecione o objeto com o MOUSE ou PRESSIONE F para
                    rastreamento de Faces!\n\n";
79  }
80  static void helpFace(){
81          cout << "\n\n Opcoes para Rastreamento de Faces: \n"
82              "\t ESC - SAIR\n"
83                          "\t r - RESET\n"
84              "\t i - INICIALIZAR RASTREAMENTO DA FACE\n"
85                          "\t s - INICIALIZAR CONTROLE\n"
86                          "\t o - VER BACKPROJECT(Depois de iniciado o
                                rastreamento)\n\n";
87  }
88
89  //CONVERTER INTEIRO PARA STRING
90  String intToString(int numero){
91          stringstream ss;
92          ss << numero;
93          return ss.str();
94  }
95
96  //EVENTO DO MOUSE ONDE SE SELECIONA A PARTE A SER RASTREADA
97  void onMouse(int event, int x, int y, int, void*){
98          if(selectObject){
99                  selection.x = MIN(x, origin.x);
100                 selection.y = MIN(y, origin.y);
101                 selection.width = abs(x - origin.x);
102                 selection.height = abs(y - origin.y);
103                 selection &= Rect(0, 0, FRAME_WIDTH, FRAME_HEIGHT);
104         }
105         switch(event){
106         case CV_EVENT_LBUTTONDOWN:
107                 origin = Point(x, y);
108                 selection = Rect(x, y, 0, 0);
109                 selectObject = true;
110                 break;
111         case CV_EVENT_LBUTTONUP:
112                 selectObject = false;
113                 if( selection.width > 0 && selection.height > 0 )
114             trackObject = -1;
115                 break;
116         }
```

```
117 }
118
119 //METODO QUE CRIA OS TRACKBARS NA TELA DE CONFIGURACAO
120 void createAllTrackbars(void){
121     createTrackbar( "Vmin", WINDOW_ORIGINAL_NAME, &vmin, 256, 0 );
122     createTrackbar( "Vmax", WINDOW_ORIGINAL_NAME, &vmax, 256, 0 );
123     createTrackbar( "Smin", WINDOW_ORIGINAL_NAME, &smin, 256, 0 );
124 }
125
126 //MOSTRAR AS COORDENADAS DO RASTREAMENTO ATUAL
127 void showCoordinate(Point center){
128         circle(originalImage, center, 10, Scalar(0, 0, 255), 2, 8, 0)
              ;
129         putText(originalImage, "X: "+ intToString(center.x) +" - Y: "
              +intToString(center.y), Point(0, 50), 1, 1, Scalar
              (0,0,255), 2);
130 }
131
132 //CALIBRA A POSICAO ATUAL DO RASTREAMENTO
133 void calibrateCam(Point center){
134         mask = Mat(Size(FRAME_WIDTH, FRAME_HEIGHT), CV_8UC3, Scalar
              (255,255,255));
135         calibX = center.x;
136         calibY = center.y;
137         rectangle(mask, Point(calibX-30,calibY-30), Point(calibX+30,
              calibY+30), Scalar(255,0,0), 1, 8, 0);
138         circle(mask, center, 10, Scalar(0, 0, 255), -1, 8, 0);
139         putText(mask, "CALIBRADO X: "+ intToString(calibX) +" - Y: "+
              intToString(calibY), Point(0, 75), 1, 1, Scalar(0,255,0),
              2);
140 }
141
142 //PARA 640x480
143 //double kp = 0.011;
144 //double ki = 0.0002;
145 //double kd = 0.0001;
146
147 //PARA 320x240
148 double kp = 0.025;
149 double ki = 0.001;
150 double kd = 0.001;
151 double calculoPID(int erro){
```

```
152         //CALCULO PROPORCIONAL
153         p = kp * err;
154
155         //CALCULO INTEGRAL
156         i = i + (err*ki);
157
158         //CALCULO DERIVATIVO
159         d = kd * (err - prevErr);
160
161         return (p + i + d);
162 }
163
164 void controlServoX(Point center){
165         if(calibX == 0)
166                 calibX = FRAME_WIDTH / 2;
167
168         err = center.x - calibX;
169         mvX += calculoPID(err);
170
171         if(mvX <= 0)
172                 mvX = 0;
173         if(mvX > 180)
174                 mvX = 180;
175
176         prevErr = err;
177 }
178
179 void controlServoY(Point center){
180         if(calibY == 0)
181                 calibY = FRAME_HEIGHT / 2;
182
183         //ERRO = SETPOINT - PROCESSVALUE
184         err = center.y - calibY;
185
186         //CALCULO PID
187         mvY += calculoPID(err);
188
189         if(mvY < 0)
190                 mvY = 0;
191         if(mvY > 180)
192                 mvY = 180;
193
```

```
194         prevErr = err;
195 }
196
197 //ENVIA OS DADOS PARA O ARDUINO MOVER OS SERVOS
198 char coordXY[2];
199 int iX;
200 int iY;
201 void enviaSerial(){
202         iX = (int) mvX;
203         iY = (int) mvY;
204         coordXY[0] = (char) iX;
205         coordXY[1] = (char) iY;
206         ser->WriteData(coordXY, 2);
207         /* (CASO SEJA LINUX)
208         arduino_com->sendChar (coordXY[0]);
209     arduino_com->sendChar (coordXY[1]);*/
210 }
211
212 //FUNCAO PARA TRATAR A PERDA DO OBJETO RASTREADO
213 void lost(){
214         char perdido[2];
215         perdido[0] = (char) 110;
216         perdido[1] = (char) 70;
217         ser->WriteData(perdido, 2);
218         /* CASO SEJA LINUX
219         arduino_com->sendChar (coordXY[0]);
220     arduino_com->sendChar (coordXY[1]);*/
221         mvX = 110;
222         mvY = 70;
223 }
224
225 //METODO PARA RESETAR TUDO, FICAR A CONFIGURACAO DE INICIO
226 void reset(){
227         mask = Mat(Size(FRAME_WIDTH, FRAME_HEIGHT), CV_8UC3, Scalar
            (255,255,255));
228         //MARGEM DE ERRO
229         rectangle(mask, pontoOneAreaDeteccao, pontoTwoAreaDeteccao,
            Scalar(255,0,0), 1, 8, 0);
230         calibX = FRAME_WIDTH / 2;
231         calibY = FRAME_HEIGHT / 2;
232         trackObject = 0;
233         faceMode = false;
```

```
234         faceSelected = false;
235         controlCamera = false;
236         lost();
237 }
238
239 //METODO MAIN
240 int main(){
241         //MENSAGEM DE AJUDA
242         help();
243
244         //VERIFICA A CONECTIVIDADE DA PORTA SERIAL E COLOCA A CAMERA
                EM "PONTO MORTO"
245         ser = new Serial(portaSerial);
246         if(ser->IsConnected()){
247                 cout << "Estamos conectados!" << endl;
248                 char primeiraConexao[2];
249                 primeiraConexao[0] = (char) mvX;
250                 primeiraConexao[1] = (char) mvY;
251                 ser->WriteData(primeiraConexao, 2);
252         }
253         /* (CASO SEJA LINUX)
254         arduino_com = new Eserial ();
255         if (arduino_com != 0)
256     {
257       arduino_com->connect("/dev/ttyACM0", 9600, spNONE);
258           reset();
259           cout << "Conectado!";
260     }else{
261                 cout << "Nao conectado!";
262         }*/
263
264         //INICIALIZA AS VARIAVEIS DE CONTROLE
265         Rect trackWindow;
266         RotatedRect trackBox;
267         int hsize = 16; //QUANTAS SUBDIVISOES DO HISTOGRAMA
268         float hranges[] = {0,180}; //RANGE DE VALORES H PARA O
              HISTOGRAMA
269         const float* phranges = hranges;
270
271         //VARIAVEIS PARA DETECCAO DA FACE
272         CascadeClassifier face_cascade;
273         if( !face_cascade.load("haarcascade_frontalface_default.xml")
```

```
        ){ cout << "--(!)Error loading\n";};
274
275     //ABRINDO E TESTANDO CAMERA
276     VideoCapture cap(idCamera);
277     if(!cap.isOpened()){
278             cout << "Falha na captura do video!" << endl;
279             exit(-1);
280     }
281
282     //SETANDO RESOLUCAO DAS TELAS PARA 320x240
283     cap.set(CV_CAP_PROP_FRAME_WIDTH, FRAME_WIDTH);
284     cap.set(CV_CAP_PROP_FRAME_HEIGHT, FRAME_HEIGHT);
285
286     //DADOS DA IMAGEM E EVENTO DE MOUSE
287     namedWindow(WINDOW_ORIGINAL_NAME, 1);
288     setMouseCallback(WINDOW_ORIGINAL_NAME, onMouse, 0);
289
290     //CRIANDO TRACKBARS
291     createAllTrackbars();
292
293     //MARGEM DE ERRO
294     rectangle(mask, pontoOneAreaDeteccao, pontoTwoAreaDeteccao,
            Scalar(255,0,0), 1, 8, 0);
295
296     for(;;){
297             //LENDO IMAGEM E INVERTENDO HORIONTALMENTE
298             cap.read(originalImage);
299             flip(originalImage, originalImage, 1);
300
301             //SE QUISER DETECTAR FACES
302             if(faceMode){
303
304                     //CRIANDO UM VETOR PARA GUARDAR AS FACES
                            ENCONTRADAS
305                     std::vector<Rect> faces;
306
307                     //CONVERTENDO A IMAGEM PARA ESCALA DE CINZA E
                            EQUILIZANDO O HISTOGRAMA
308                     cvtColor(originalImage, grayscaleFrame,
                            CV_BGR2GRAY);
309                     equalizeHist(grayscaleFrame, grayscaleFrame);
310
```

```
311                         //ENCONTRANDO AS FACES NA IMAGEM CAPTURADA E
                                GUARDANDO NO VETOR
312                         face_cascade.detectMultiScale(grayscaleFrame,
                                faces, 1.1, 3,
                                CV_HAAR_FIND_BIGGEST_OBJECT|
                                CV_HAAR_SCALE_IMAGE, Size(30,30));
313
314                         //DESENHANDO UM RETANGULO PARA TODAS AS FACES
                                ENCONTRADAS
315                         for (int i = 0; i < faces.size(); i++)
316                         {
317                                 Point pt1(faces[0].x + faces[i].width
                                        , faces[i].y + faces[i].height);
318                                 Point pt2(faces[i].x, faces[i].y);
319                                 rectangle(originalImage, pt1, pt2,
                                        Scalar(255,0,0), 2, 8, 0);
320                         }
321
322                         //SE A FACE ESTA SELECIONADA COMECA O
                                RASTREAMENTO
323                         if(faceSelected){
324                                 selection = faces.at(0);
325                                 Mat roi(originalImage, selection);
326                                 trackObject = -1;
327                                 faceMode = false;
328                                 faceSelected = false;
329                         }
330                 }
331
332                 //SELECIONANDO NA JANELA
333                 if(selectObject && selection.width > 0 && selection.
                        height > 0 ){
334                         Mat roi(originalImage, selection);
335                         bitwise_not(roi, roi);
336                 }
337
338                 //SE ESTA RASTREANDO
339                 if(trackObject){
340                         int _vmin = vmin;
341                         int _vmax = vmax;
342
343                         //CRIANDO IMAGEM HSV
```

```
344                     cvtColor(originalImage, hsv, COLOR_BGR2HSV);
345
346                     //CRIANDO UMA "MASCARA" THRESHOLD DA IMAGEM
                            HSV
347                     inRange(hsv, Scalar(0, smin, MIN(_vmin,_vmax)
                            ),
348                     Scalar(180, 255, MAX(_vmin, _vmax)),
                            maskThresh);
349
350                     //PREPARANDO VARIAVEIS PARA CRIACAO DO
                            HISTOGRAMA
351                     int ch[] = {0, 0};
352                     hue.create(hsv.size(), hsv.depth()); //
                            CRIANDO A IMAGEM HUE PARA SER COPIADO O
                            CANAL H DA IMAGEM HSV PARA A IMAGEM HUE
353                     mixChannels(&hsv, 1, &hue, 1, ch, 1);
354
355                     //SE E A PRIMEIRA ITERACAO DO RASTREAMENTO
356             if( trackObject < 0 ){
357                 Mat roi(hue, selection);
358                                 Mat maskroi(maskThresh, selection);
359                 calcHist(&roi, 1, 0, maskroi, hist, 1, &hsize, &
                        phranges);
360                 normalize(hist, hist, 0, 255, CV_MINMAX);
361                                 trackWindow = selection;
362                                 trackObject = 1;
363                     }
364
365                     //BACKPROJECTION E CAMSHIFT
366                     calcBackProject(&hue, 1, 0, hist, backproj, &
                            phranges);
367                     backproj &= maskThresh;
368                     trackBox = CamShift(backproj, trackWindow,
                            TermCriteria( CV_TERMCRIT_EPS |
                            CV_TERMCRIT_ITER, 10, 1 ));
369
370                     //PEGANDO A TRACKBOX(ROTATEDRECT) E
                            TRANSFORMANDO NUM RETANGULO E DESENHANDO
                            ESSE RETANGULO NA TELA
371                     trackWindow = trackBox.boundingRect();
372                     rectangle(originalImage, trackWindow, Scalar
                            (255, 0, 0), 2, 8, 0);
```

```
373
374                         //MOSTRA AS COORDENADAS DO OBJETO RASTREADO
375                         showCoordinate(trackBox.center);
376
377                         //FUNCAO QUE MOSTRA A IMAGEM EM BACKPROJETION
378                         if(showBackProj){
379                                 cvtColor( backproj, originalImage,
                                        COLOR_GRAY2BGR );
380                         }
381
382                         //SE FOR ATIVADO O CONTROLE DAS CAMERAS
383                         if(controlCamera){
384                                 //VERIFICA SE ESTA DENTRO DA MARGEM
                                        DE ERRO
385                                 if(trackBox.center.x <
                                        pontoOneAreaDeteccao.x || trackBox
                                        .center.x > pontoTwoAreaDeteccao.x
                                        ){
386                                         controlServoX(trackBox.center
                                                );
387                                 }
388                                 if(trackBox.center.y <
                                        pontoOneAreaDeteccao.y || trackBox
                                        .center.y > pontoTwoAreaDeteccao.y
                                        ){
389                                         controlServoY(trackBox.center
                                                );
390                                 }
391
392                                 //SE ESTIVER PERDIDO VOLTA PRO CENTRO
393                                 if(trackBox.center.x == 0 && trackBox
                                        .center.y == 0){
394                                         lost();
395                                         trackObject = 0;
396                                         cout << "Perdeu o
                                                rastreamento! Selecione
                                                novamente o objeto!" <<
                                                endl;
397                                 }else{
398                                         enviaSerial();
399                                 }
400                         }
```

```
401                     }
402
403                     //MOSTRA A IMAGEM
404                     imshow(WINDOW_ORIGINAL_NAME, originalImage &= mask);
405
406                     //SAIDA COM ESC E DEMAIS OPCOES DO TECLADO
407                     c = (char)waitKey(10);
408                     if( c == 27 ) break;
409                     switch(c)
410                     {
411                             case 's' : controlCamera = true;
412                                     break;
413                             case 'c' : calibrateCam(trackBox.center);
414                                     break;
415                             case 'o': showBackProj = !showBackProj;
416                                     break;
417                             case 'f' : faceMode = true; system("cls");
                                    helpFace();
418                                     break;
419                             case 'i' : faceSelected = true;
420                                     break;
421                             case 'r' : reset(); system("cls"); help();
422                                     break;
423                             default: ;
424                     }
425             }
426             //SAI DO PROGRAMA
427             destroyAllWindows();
428             return 0;
429     }
```

APÊNDICE B – CÓDIGO DO *ARDUINO*

```
1  #include <Servo.h>
2
3  //DECLARA AS VERIAVEIS DE CONTROLE DO SERVO
4  Servo servoX, servoY;
5  int pinX = 3;
6  int pinY = 6;
7  int x, y;
8  int coord[2];
9
10 //INICIALIZA OS SERVOS E A CONEXAO SERIAL
11 void setup(){
12   servoX.attach(pinX);
13   servoY.attach(pinY);
14   servoX.write(110);
15   servoY.write(70);
16   Serial.begin(9600);
17 }
18
19 void loop(){
20   //CAPTURA OS CARACTERES ENVIADOS POR SERIAL, CONVERTE-OS PARA
       INTEIRO
21   // E OS ARMAZENAM EM UM VETOR DE INTEIROS
22   if(Serial.available() > 0){
23     for(int i = 0; i < 2; i++){
24         do{
25           coord[i] = Serial.read();
26         }while(coord[i] < 0);
27     }
28
29     //O PRIMEIRO CARACTERE RECEBIDO E O DO SERVO X E O SEGUNDO DO
         SERVO Y
30     x = coord[0];
31     y = coord[1];
32     //MANDA O VALOR PARA MOVIMENTAR OS SERVOS
33     servoX.write(x);
34     servoY.write(y);
35   }
36   delay(1);
37 }
```