UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

RODRIGO NEVES DOS SANTOS

Reconhecimento de gestos de mão humana em imagens de vídeo

São Carlos
2011

RODRIGO NEVES DOS SANTOS

RECONHECIMENTO DE GESTOS DE MÃO HUMANA EM IMAGENS DE VÍDEO

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica com ênfase
em Eletrônica

ORIENTADOR: Professor Doutor Adilson Gonzaga

São Carlos
2011

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

S237r

Santos, Rodrigo Neves dos.
Reconhecimento de gestos de mão humana em imagens de vídeo. / Rodrigo Neves dos Santos ; orientador Adilson Gonzaga –- São Carlos, 2011.

Monografia (Graduação em Engenharia Eétrica com Ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

1. Visão computacional. 2. Gestos de mão. 3. Rede neural. I. Titulo.

FOLHA DE APROVAÇÃO

Nome: Rodrigo Neves dos Santos

Título: "Reconhecimento de Gestos de Mão Humana em Imagens de Vídeo"

Trabalho de Conclusão de Curso defendido e aprovado em 28 / 11 / 2011,

com NOTA 9,7 (NOVE, SETE), pela comissão julgadora:

Prof. Dr. Ruy Barboza - EESC/USP

Profa. Dra. Maria Stela Veludo de Paiva - EESC/USP

Prof. Associado Homero Schiabel
Coordenador da CoC-Engenharia Elétrica
EESC/USP

Dedico este trabalho aos meus pais que me apoiaram em todos os momentos de minha graduação.

Agradecimentos

Agradeço aos meus pais pela educação, amor, apoio e por serem minha fonte de inspiração. As minhas irmãs e a todos de minha família por estarem sempre ao meu lado em qualquer etapa da minha vida.

Aos meus colegas de Universidade, em especial aos meus amigos Biondo, Michel, Nishizawa e Samir. A todos que estiveram comigo na Espanha, durante meu intercambio em Vigo, em especial aos amigos e companheiros da Urzaiz1 e à Susan pelo apoio e carinho.

A todos os professores da Universidade de São Paulo que contribuíram e se importaram com a minha formação. Ao Professor Dr. Adilson Gonzaga pela orientação e atenção neste trabalho e durante a graduação.

E a todos que contribuíram de alguma forma para a conclusão deste trabalho.

“A curiosidade é mais importante do que o conhecimento.”

Albert Einstein

Resumo

Este trabalho propõe um sistema de reconhecimento de gestos de uma mão humana em imagens de vídeo. Para que o sistema possa reconhecer os gestos são cumpridas as seguintes etapas: captura da imagem, processamento, extração de características e reconhecimento. A câmera ligada ao computador captura imagens da mão humana sem o auxílio de luvas ou *datagloves* que facilitem a extração das características. A imagem capturada é processada para que possa ser extraído um vetor de características a fim de modelar a postura da mão humana em um determinado instante de tempo. Este vetor de características é então utilizado em uma Rede Neural Artificial (RNA) treinada a fim de reconhecer o gesto executado. Todo o sistema é desenvolvido utilizando a linguagem de programação C, juntamente com a biblioteca OpenCV (Open Source Computer Vision Library), desenvolvida pela Intel para aplicativos na área de Visão Computacional. Os resultados mostram em que condições o algoritmo apresenta o melhor desempenho e que é possível utilizá-lo na aplicação de um sistema interação homem-computador.

Palavras-Chave: visão computacional, gestos de mão, rede neural.

Abstract

This paper develops a system for recognizing gestures of a human hand in video images. The following steps are fulfilled for the system can recognize gestures: image capture, processing, feature extraction and recognition. The camera connected to the computer capture images of the human hand without the aid of gloves or dataglove that facilitate the extraction of features. The captured image is processed, so that it can be extracted from an array of features to model the posture of the human hand in a given time. This feature vector is then used in an Artificial Neural Network (ANN) trained to recognize the gesture executed. The whole system is developed using the C programming language, along with a library OpenCV (Open Source Computer Vision Library), developed by Intel for applications in Computer Vision. The results are show under what conditions the algorithm performs best and that you can use it in the application of a human-computer interaction system.

Key Words: computer vision, hand gesture, neural network.

Índice de Figuras

Figura 1 – Cubo de cores RGB 26
Figura 2 – Exemplo de Subtração de Imagens 29
Figura 3 – Exemplo de Histograma e sua Imagem 29
Figura 4 – Histograma com Vale Assinalado 30
Figura 5 – Exemplo de Limiarização através do Método do Vale 30
Figura 6 – Exemplo de Erosão Binária 32
Figura 7 – Exemplo de Dilatação Binária 32
Figura 8 – Exemplo de Abertura 33
Figura 9 – Exemplo de Fechamento 34
Figura 10 – Aplicação de Filtro Morfológico. Fonte: *(GONZAGA, IRIS SEL, 2000).* 34
Figura 11 – Representação do Perceptron. Fonte: *(BARROS, 2009)* 35
Figura 12 – Representação de uma Rede Neural MLP. Fonte: *(FERRAMOLA, 2002)* 36
Figura 13 – Diagrama de blocos do projeto 39
Figura 14 – Código responsável pela Inicialização da Câmera. 41
Figura 15 – Exemplo de Quadro Capturado 42
Figura 16 – Etapas para o Processamento de Imagens 42
Figura 17 – Código responsável pela conversão para RGB normalizado 43
Figura 18 – Exemplo de Conversão para o RGB normalizado 43
Figura 19 – Exemplo do Canal r normalizado 44
Figura 20 – Exemplo de Subtração de Fundo 44
Figura 21 – Código responsável pelo Cálculo do Histograma 45
Figura 22 – Exemplo de Histograma para a Imagem da figura 15 45
Figura 23 – Exemplo de Binarização da Imagem da figura 20 46
Figura 24 – Código responsável pelos Filtros Morfológicos 46
Figura 25 – Exemplo de Aplicação de Filtro Morfológico 47
Figura 26 – Código responsável pelo Pós-Processamento da Imagem 47
Figura 27 – Exemplo de Remoção de Punho 48
Figura 28 – Código responsável por Encontrar os Limites da Imagem 48
Figura 29 – Código responsável por Extrair as Características 49
Figura 30 – Exemplo de Divisão de Imagem para Extração de Características 49
Figura 31 – Exemplo de Segmentação Manual (a) e pelo Algoritmo proposto (b) 52
Figura 32 – Imagens do Cenário 1A com Iluminação Ambiente e Fundo Branco 55
Figura 33 – Imagens do Cenário 1B com Iluminação Fluorescente e Fundo Branco 56
Figura 34 – Imagens do Cenário 1C com Iluminação Ambiente e Fundo Preto 56
Figura 35 – Imagens do Cenário 1D com Iluminação Fluorescente e Fundo Preto 57

Índice de Tabelas

Tabela 1 – Exemplo de representação cores no espaço RGB 27
Tabela 2 – Características da câmera *web* utilizada 40
Tabela 3 – Características do computador utilizado 40
Tabela 4 – *Softwares* e Biblioteca Utilizados 40
Tabela 5 – Parâmetros da Rede Neural utilizada 50
Tabela 6 – Cenários para Análise da Iluminação e Cor do Fundo 51
Tabela 7 – Distâncias Avaliadas entre a Mão Humana e a Câmera *Web* 53
Tabela 8 – Resultados para a Análise da Iluminação e Cor de Fundo 57
Tabela 9 – Resultados para a Distância entre a Mão e a Câmera 58
Tabela 10 – Parâmetros da Análise de Reconhecimento 59
Tabela B-1 – Classes de Gestos para a Análise da Distância entre a Mão e a Câmera 83
Tabela C-1 – Classes de Gestos para a Análise do Reconhecimento 84
Tabela D-1 – Tabela de Contingências para Análise do Reconhecimento 87

Sumário

Agradecimentos 9
Resumo 13
Abstract 15
Índice de Figuras 17
Índice de Tabelas 19
Sumário 21
Capítulo 1 – Introdução 23
1.1 Apresentação 23
1.2 Objetivos 23
1.3 Organização da Monografia 24
Capítulo 2 – Revisão de Literatura 25
2.1 Biblioteca OpenCV 25
2.2 Espaço de Cores 26
2.2.1 Espaço de cores RGB 26
2.2.2 RGB normalizado 27
2.3 Segmentação de Imagens 28
2.3.1 Subtração de Imagens de Fundo 28
2.3.2 Histograma 29
2.3.3 Limiarização 29
2.4 Morfologia matemática 31
2.4.1 Erosão binária 31
2.4.2 Dilatação binária 32
2.4.3 Abertura 33
2.4.4 Fechamento 33
2.4.5 Filtro morfológico – Abertura e Fechamento 34
2.5 Redes Neurais 35
2.5.1 Perceptron Multicamadas (MLP) 36
2.6 Gestos 37
2.7 Considerações Finais 37
Capítulo 3 – Materiais e Métodos 39
3.1 Materiais 40
3.2 Sistema Proposto 41
3.2.1 Captura de Imagens 41
3.2.2 Processamento de Imagens 42
3.2.3 Pós Processamento de Imagens 46
3.2.4 Extração de características 48
3.2.5 Treinamento da Rede Neural 50
3.2.6 Reconhecimento 50
3.3 Análises 51
3.3.1 Iluminação e Cor de Fundo 51
3.3.2 Distância entre Mão e Câmera 52
3.3.3 Reconhecimento 53

Capítulo 4 – Resultados e Conclusões 55
4.1 Resultados 55
4.1.1 Iluminação e Cor de Fundo 55
4.1.2 Distância entre Mão Humana e Câmera 58
4.1.3 Reconhecimento 59
4.2 Conclusões 60
4.3 Trabalhos Futuros 61
Referências Bibliográficas 63
APÊNDICE A – Códigos do Algoritmo Utilizado 65
A.1 Algoritmo para Análise da Iluminação e Cor de Fundo 65
A.2 Algoritmo de para Cálculo dos Parâmetros 70
A.3 Algoritmo de Captura de Quadros e Segmentação 72
A.4 Algoritmo de Extração de Características e Geração de Resultados 76
APÊNDICE B – Classes de Gestos para a Análise da Distância 83
APÊNDICE C – Classes de Gestos para a Análise do Reconhecimento 84
APÊNDICE D – Tabela de Contingências para Análise do Reconhecimento 87

Capítulo 1 – Introdução

1.1 Apresentação

Com o crescente aumento da influência dos computadores na sociedade, a Interação Humano-Computador (IHC) vem ganhando uma grande importância em nossa vida diária. A IHC tem se expandido rapidamente e de forma constante durante as últimas três décadas, atraindo profissionais de muitas outras disciplinas e incorporando diversos conceitos e abordagens (CARROL, 2009).

Dos esforços dedicados aos estudos desta área, surgiram diferentes métodos de IHC, como o reconhecimento de voz, dispositivos tácteis, reconhecimentos de faces, entre outros. Entretanto, apenas nos últimos anos é que tem aumentado o interesse em novas técnicas de IHC, como o estudo de movimento de mãos e braços.

Os gestos humanos são um meio de interação não verbal entre as pessoas, e vão desde ação simples para apontar e mover objetos até movimentos mais complexos que expressam os nossos sentidos e nos permitem nos comunicar (PAVLOVIC, 1997).

Como forma de comunicação entre usuário e máquina, a utilização de movimentos da mão como forma de interação com o computador é muito intuitiva e dá ao sistema uma grande usabilidade. Desta forma, diversos tipos de algoritmo para o reconhecimento de gestos de mão humana vêm sido estudados de forma a tornar esta interação mais simples e independente de dispositivos de auxilio, como luvas ou sensores de captura de movimento.

Assim, algoritmos de identificação de movimentos da mão humana, a partir de câmeras de captura de vídeo de baixo custo, são um grande desafio para a área de visão computacional. Este tipo de algoritmo pode ser aplicado em equipamentos que operam à distância e em tempo real, evitando o contato humano em locais inseguros e perigosos (GONZAGA, 2011).

1.2 Objetivos

Este trabalho tem por objetivo principal o desenvolvimento de um sistema de reconhecimento de gestos dinâmicos de mão humana situada à frente de uma *webcam*. Serão estudadas as melhores condições de ambiente para que o sistema seja preciso e eficiente, obtendo as melhores porcentagens no reconhecimento de gestos, para mãos de diferentes pessoas.

O projeto aborda todas as etapas de um problema de visão computacional, com a captura da imagem de vídeo, segmentação, extração de características, reconhecimentos de padrões treinados e interpretação dos resultados obtidos.

As imagens de vídeo devem ser obtidas através de câmeras de captura de vídeo de baixo custo, sendo analisadas as melhores condições de iluminação do ambiente e a distância entre a mão humana e a câmera que tornam o sistema mais eficiente.

A segmentação da imagem é direta, sem o auxilio de luvas ou dispositivos que facilitem a identificação da mão humana. Esta etapa foca na identificação dos melhores métodos de visão computacional que permitam a segmentação total da mão humana em um fundo simples e homogêneo.

As características da imagem são extraídas em quadros capturados em intervalos regulares para que a variação do posicionamento da mão seja reconhecida. O intervalo de tempo de captura dos quadros será analisado para que o sistema tenha o melhor desempenho.

1.3 Organização da Monografia

A monografia se estrutura de acordo com a segmentação em capítulos proposta a seguir:

Capítulo 1: Introdução – Apresenta o tema da monografia de modo a fornecer uma idéia total do trabalho.

Capítulo 2: Revisão de Literatura – Tem como objetivo apresentar os conceitos de visão computacional utilizados na segmentação de imagens de vídeo e extração de características; e a rede neural aplicada para o reconhecimento de gestos de mão humana.

Capítulo 3: Materiais e Métodos – Este capítulo trata a metodologia usada neste trabalho, bem como o material usado para que os objetivos propostos fossem alcançados.

Capítulo 4: Resultados e Conclusões – Este capítulo apresenta os resultados obtidos com a aplicação da metodologia e a análise destes resultados. Apresenta também as conclusões do estudo e suas possíveis aplicações futuras.

Capítulo 2 – Revisão de Literatura

2.1 Biblioteca OpenCV

OpenCV (*Open Source Computer Vision*) é uma biblioteca de código aberto para o desenvolvimento de aplicações na área de visão computacional. Originalmente criada pela Intel, em 2000, esta biblioteca é escrita nas linguagens de programação C e C++, sendo compatível nos sistemas operacionais Linux, Windows e Mac OS X.

A biblioteca OpenCV foi projetada com um forte foco em aplicações em tempo real, podendo tirar vantagens de processadores multicamadas. A biblioteca está disponível com o código fonte e os executáveis otimizados para os processadores Intel. Porém, um programa OpenCV ao ser executado, identifica automaticamente o tipo de processador que esta sendo usado e aciona a DLL (*Dynamic Link Library*) otimizada para este, permite que seus algoritmos sejam utilizados em diferentes sistemas de processamento de imagens de vídeo.

A OpenCV tem como objetivo fornecer uma infraestrutura simples que permita a construção de aplicações sofisticadas de visão computacional rapidamente, sendo totalmente livre para o uso acadêmico e comercial. A biblioteca OpenCV contém mais de 500 funções que contemplam diversas áreas da visão computacional, como: aprendizagem de máquina, processamento de imagens, entrada e saída de imagens e dispositivos de vídeo, entre outras (BRADSKI, 2008).

Sua utilização é constante em aplicações que envolvem a interação humano-computador em tempo real, devido às facilidades que a biblioteca fornece. Sistemas para o reconhecimento de gestos podem ser criados com o uso de funções da OpenCV em todas as etapas do projeto, como descrito a seguir:

- Na captura das imagens de vídeo, com funções que manipulam a entrada de vídeos de câmeras *web*;
- No processamento de imagens, com funções que permitem a segmentação de imagens;
- No pós-processamentos das imagens, na criação de filtros morfológicos que melhoram as características das imagens.
- Na aprendizagem de máquinas, com funções que permitem o treinamento de diversos tipos de redes neurais;
- No reconhecimento de padrões, utilizando funções que permitem o desenvolvimento de diversas aplicações.

2.2 Espaço de Cores

2.2.1 Espaço de cores RGB

O modelo RGB (*Red, Green, Blue*) é um sistema de cores aditivo formado pelas cores vermelho, verde e azul. Estas três cores primárias são combinadas de modo a formar as demais cores.

Cada cor no modelo de cores RGB é representada pela quantidade de vermelho, verde e azul, sendo que cada uma varia desde um valor mínimo (completamente escura) até um valor máximo (completamente intenso). O branco é representado por todas as cores primárias em seu valor máximo e o preto por todas em seu valor mínimo.

Um espaço de cor RGB é qualquer espaço de cor aditivo baseado no modelo de cor RGB. A representação mais utilizada para um espaço de cores RGB é a que associa um byte (8 bits) para cada uma das cores primárias, permitindo que seus valores variem de 0 a 255. Sendo assim podemos associar cada cor do espaço RGB a um ponto de um sistema de coordenadas de três eixos: R (vermelho), G (verde) e B (azul), com valores que variam de 0 a 255.

A figura 1 apresenta esta representação em forma de cubo de cores.

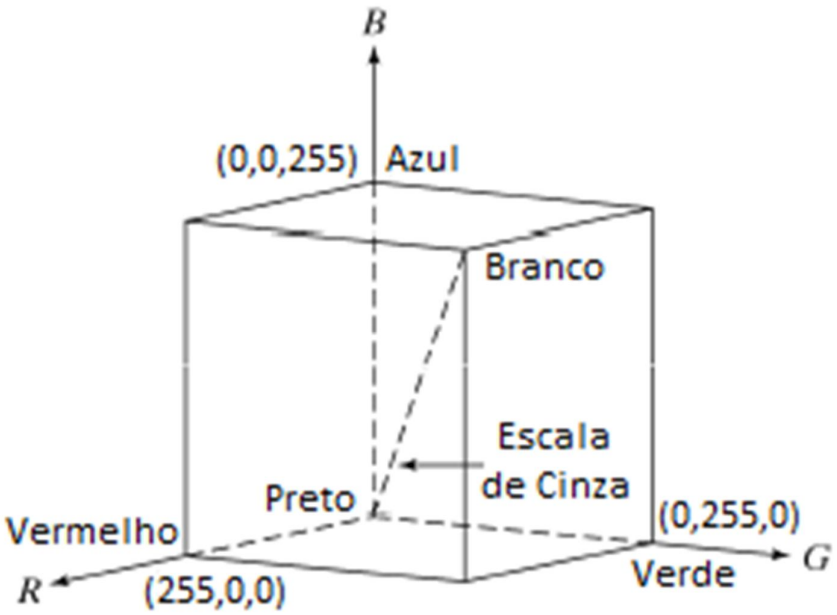


Figura 1 – Cubo de cores RGB

Assim, no espaço de cores RGB cada pixel é associado a três bytes que representam as intensidades de vermelho, verde e azul. Alguns exemplos de representações de cores no RGB podem ser vistos na tabela 1.

Tabela 1 – Exemplo de representação de cores no espaço RGB

Cores	R	G	B
Branco	255	255	255
Preto	0	0	0
Verde	0	255	0
Vermelho	255	0	0
Azul	0	0	255
Amarelo	255	255	0

A principal desvantagem do espaço RGB é a de que ele não é adequado para a representação de sistema baseados na percepção visual humana. Isto implica que cores próximas na percepção visual não representam cores no sistema RGB. Esta característica prejudica a segmentação de uma região de cores de interesse neste sistema (RIBEIRO H. L., 2006).

2.2.2 RGB normalizado

As componentes do espaço RGB normalizado são obtidas a partir do modelo de cores primárias RGB. A conversão entre estes dois modelos se dá através das equações (1), (2) e (3); que calculam os valores de r (vermelho normalizado), g (verde normalizado) e b (azul normalizado), respectivamente.

$$r = \frac{R}{R + G + B} \qquad (1)$$

$$g = \frac{G}{R + G + B} \qquad (2)$$

$$b = \frac{B}{R + G + B} \qquad (3)$$

Sabendo que a soma das três componentes deste espaço de cores é sempre conhecida (r+g+b=1), pode-se diminuir a dimensão espacial omitindo o terceiro componente b. As demais componentes são chamadas de cores puras visto que a dependência do brilho é diminuída em relação ao espaço RGB.

Uma propriedade importante deste espaço de cores é que o RGB normalizado é invariante às mudanças de orientação das fontes de luz na superfície, quando ignoramos a luz do ambiente em superfícies opacas. Aliado a simplicidade na transformação desde o espaço RGB, o RGB normalizado vem ganhando popularidade entre os pesquisadores (FIBIGER, 2004).

2.3 Segmentação de Imagens

A segmentação na área de visão computacional é o processo responsável por dividir uma imagem em grupos ou objetos. Seu objetivo é simplificar a representação para outra mais significativa e de mais fácil análise.

Para os seres humanos a tarefa de reconhecer objetos em uma imagem é trivial, independente do conhecimento prévio da cena visto que somos capazes de reconhecer as características similares que definem um objeto. Já para o processamento de imagens, o grande desafio é a extração rápida de características através da segmentação de imagens para a realização da análise (SALDANHA, 2009).

A segmentação de imagens é o passo inicial na área de processamento de imagens e deve parar quando o objeto de interesse da aplicação esteja isolado (GONZALES, 2003).

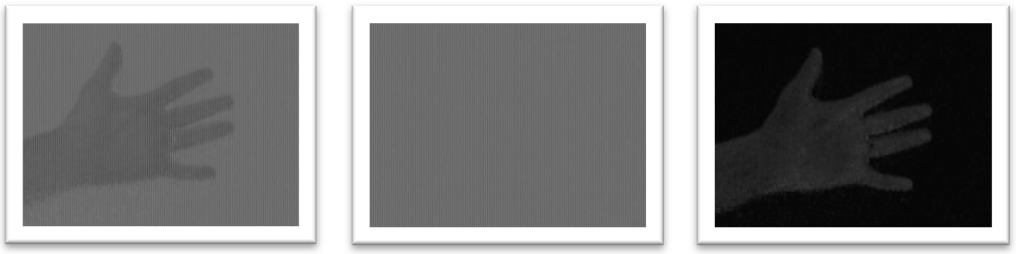
Os algoritmos para a segmentação de imagens normalmente utilizam características associadas aos níveis de cinza da imagem. As principais características são a descontinuidade, que consiste na divisão baseada na mudança brusca dos tons de cinza, como na detecção de pontos isolados, linhas e bordas; e a baseada em similaridades, baseada na separação de regiões que possuem tons de cinza com as mesmas características, como na limiarização (RIBEIRO J. M., 2007).

2.3.1 Subtração de Imagens de Fundo

A subtração de fundo tem como objetivo isolar objetos ou partes de um objeto em uma imagem. Atualmente existem diversas técnicas destinadas a este objetivo, algumas com resultados simples, outras com resultados mais robustos (SOUZA, 2009).

Um dos algoritmos mais simples que existem é o de subtração de quadros, baseando-se na diferença de imagem entre dois instantes de tempo diferentes. Cada pixel da imagem resultante é formado pela subtração do pixel corresponde no instante anterior pelo pixel no instante atual.

Em um cenário de fundo simples, é possível utilizar a subtração de fundo como parte da segmentação de imagens. Um exemplo desta técnica pode ser observado na figura 2.



(a) Imagem **(b) Fundo** **(c) Resultado da Subtração**

Figura 2 – Exemplo de Subtração de Imagens

A figura 2 mostra como é possível utilizar a subtração de imagens para segmentar um objeto em um cenário de fundo simples, em imagens em escala de cinza. O cenário de fundo é excluído da imagem, com os pixels desta região assumindo valores próximos a zero (cor preta).

2.3.2 Histograma

O histograma de uma imagem em tons de cinza é uma função H(k) que produz o número de ocorrências de cada nível de cinza k na imagem. Um exemplo de imagem e seu histograma podem ser observados na figura 3.

(a) Histograma **(b) Imagem**

Figura 3 – Exemplo de Histograma e sua Imagem

2.3.3 Limiarização

Uma das formas mais simples de segmentar uma imagem digital é através da limiarização. A idéia desta segmentação é a que um objeto pode ser definido por uma região formada por pixels que tenham em comum uma faixa de intensidades.

Assim é possível separar esta região das demais pela análise do histograma da imagem. Nessa situação o histograma apresenta um vale separando dois picos que

podem representar regiões distintas, como por exemplo, um fundo e um objeto (SALDANHA, 2009).

A segmentação através do histograma é uma técnica simples que pode ser usada quando o objeto apresenta uma quantidade de pixels com tonalidades de cinza similares e que contrastam com as demais áreas da imagem. A figura 4 apresenta um histograma para a figura 2.c, e nele é possível observar que há uma separação entre os pixels que representam o fundo (cor preta) e os que representam a mão.

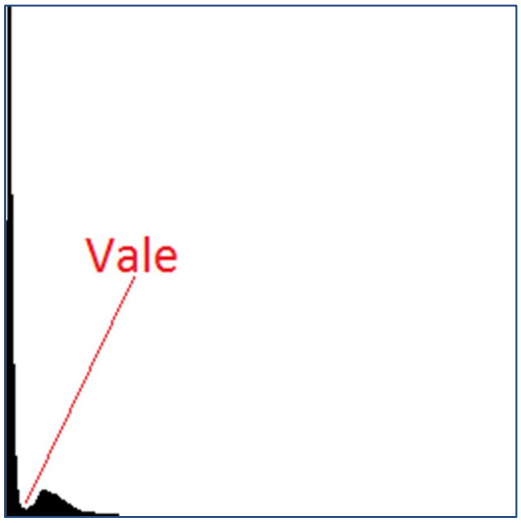


Figura 4 – Histograma com Vale Assinalado

A maior dificuldade da limiarização é na definição do valor que será o limite para a separação dos pixels. Uma das formas de definir este limite é através do método do vale, que busca este valor entre duas regiões de pico.

Após a fixação do valor limite, é possível limiarizar a imagem fazendo com que cada pixel assuma a intensidade máxima (branco) quando sua intensidade está acima deste limiar e que assuma o valor mínimo (preto) quando sua intensidade está abaixo deste limiar. A figura 5 traz a imagem resultante para a limiarização da figura 2.c através do método do vale.

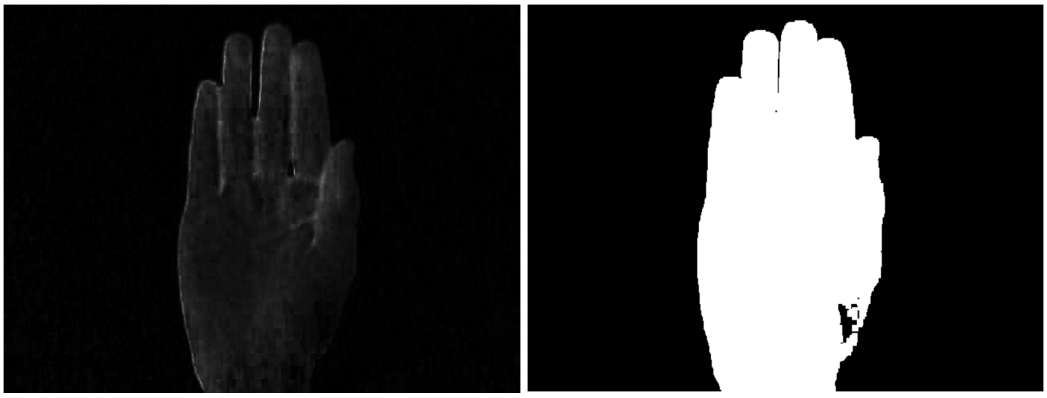


Figura 5 – Exemplo de Limiarização através do Método do Vale

2.4 Morfologia matemática

A morfologia matemática é baseada na teoria dos conjuntos, e é uma ferramenta para extração de componentes de imagens que sejam uteis na descrição da forma de uma região. Atualmente pode-se encontrar aplicações dos processos morfológicos na filtragem, segmentação, restauração, detecção de bordas, aumento de contraste, entre outros.

A análise morfológica permite extrair componentes da imagem que são uteis na representação e descrição da forma das regiões, como fronteiras e esqueletos. Também permite obter características importantes dos objetos na imagem como a forma e o tamanho.

As transformações morfológicas operam sobre conjuntos, mediante a utilização de outro conjunto de forma conhecida, denominado de elemento estruturante. O tamanho e a formato do elemento estruturante são escolhidos de acordo com a forma que se deseja obter.

As classes básicas de operadores da morfologia matemática são a erosão e a dilatação.

2.4.1 Erosão binária

A transformação de erosão binária é resultado de comprovar se o elemento estruturante B está completamente incluído dentro do conjunto A, sendo que quando não ocorre, o resultado da erosão é um conjunto vazio. A equação 4 mostra como a operação de erosão binária pode ser definida.

$$A \ominus B = \left\{ x \middle| (B)_x \subseteq A \right\} \qquad (4)$$

A equação 4 indica que a erosão de A por B é o conjunto de todos os pontos x tal que B, transladado por x, está contido em A (GONZAGA, 2000).

Quando os objetos da cena são menores que o elemento estruturante, eles desaparecem. Assim, a erosão se supõe como uma operação de degradação da imagem. A figura 6 mostra o resultado de uma operação de erosão binária.

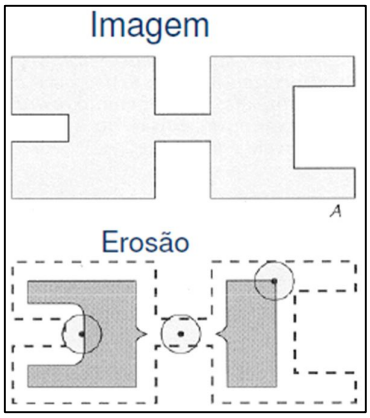


Figura 6 – Exemplo de Erosão Binária

2.4.2 Dilatação binária

A operação de dilatação binária e erosão binária são duais. As duas operações são duais, sendo que uma dilatação é o mesmo que uma erosão do complemento da imagem pelo elemento estruturante refletido.

O resultado da dilatação é o conjunto de elementos tal que pelo menos algum elemento do conjunto estruturante B pertence ao conjunto A, quando B se move sobre o conjunto A. A equação 5 mostra como a operação de dilatação binária pode ser definida.

$$A \oplus B = \left\{ x \middle| (\hat{B})_x \cap A \neq \varnothing \right\} \qquad (5)$$

A equação 5 indica que a dilatação de A por B é um conjunto de todo os deslocamentos de x tal que B' e A sobrepõem-se por pelo menos um elemento não nulo (GONZAGA, 2000).

Esta operação representa um crescimento progressivo do conjunto A, visto que quando o elemento estruturante passa sobre o conjunto, este se expandirá. A figura 7 mostra o resultado de uma operação de dilatação binária.

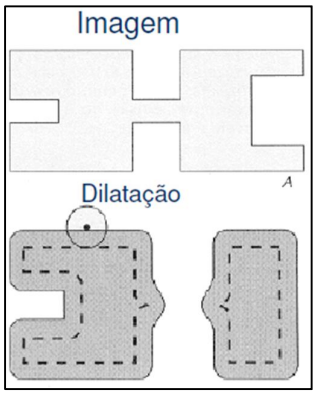


Figura 7 – Exemplo de Dilatação Binária

2.4.3 Abertura

A abertura de um conjunto A por um elemento estruturante B é definida como uma operação de erosão binária seguida de uma operação de dilatação. A equação 6 mostra como a operação de abertura pode ser definida.

$$A \circ B = (A \ominus B) \oplus B \qquad (6)$$

A abertura geralmente suaviza o contorno de uma imagem, quebra istmos estreitos e elimina protusões finas. Esta operação tende a abrir pequenos vazios ou espaços entre objetos próximos numa imagem, também sendo usada para remover ruídos (somente pontos pretos de ruído).

A figura 8 mostra o resultado de uma operação de abertura.

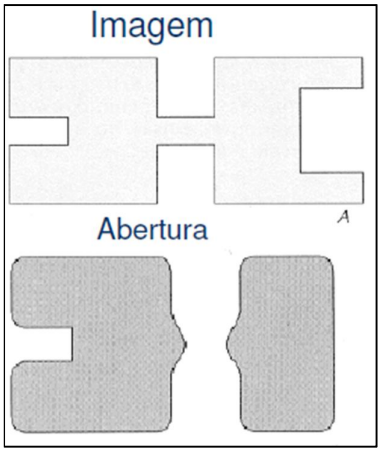


Figura 8 – Exemplo de Abertura

2.4.4 Fechamento

O fechamento de um conjunto A por um elemento estruturante B é definido como uma operação de dilatação binária seguida de uma operação de erosão. A equação 7 mostra como a operação de fechamento é definida.

$$A \bullet B = (A \oplus B) \ominus B \qquad (7)$$

O fechamento tende a suavizar os contornos, fundir partes, eliminar pequenos buracos e preencher fendas em um contorno. Esta operação tende a preencher ou fechar os vazios, podendo também remover muitos dos pixels brancos de ruído.

As operações de abertura e fechamento são duais relativamente à complementação e reflexão dos conjuntos.

A figura 9 mostra o resultado de uma operação de fechamento.

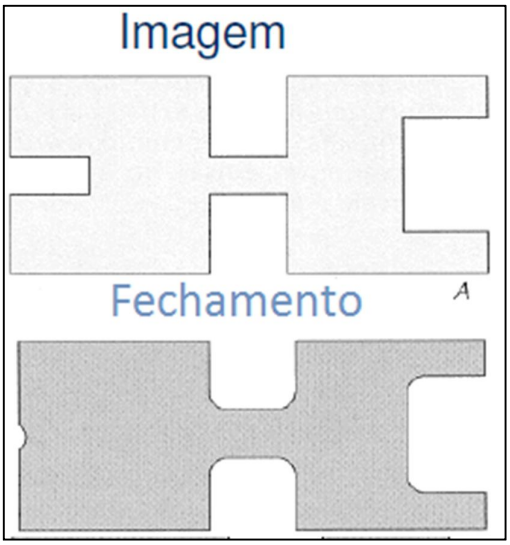


Figura 9 – Exemplo de Fechamento

2.4.5 Filtro morfológico – Abertura e Fechamento

Um filtro morfológico composto de uma operação de abertura seguida de uma operação de fechamento pode ser utilizado para a remoção de ruídos isolados, numa operação de pós-processamento de imagens. A figura 10 mostra o resultado da aplicação deste filtro sobre uma imagem.

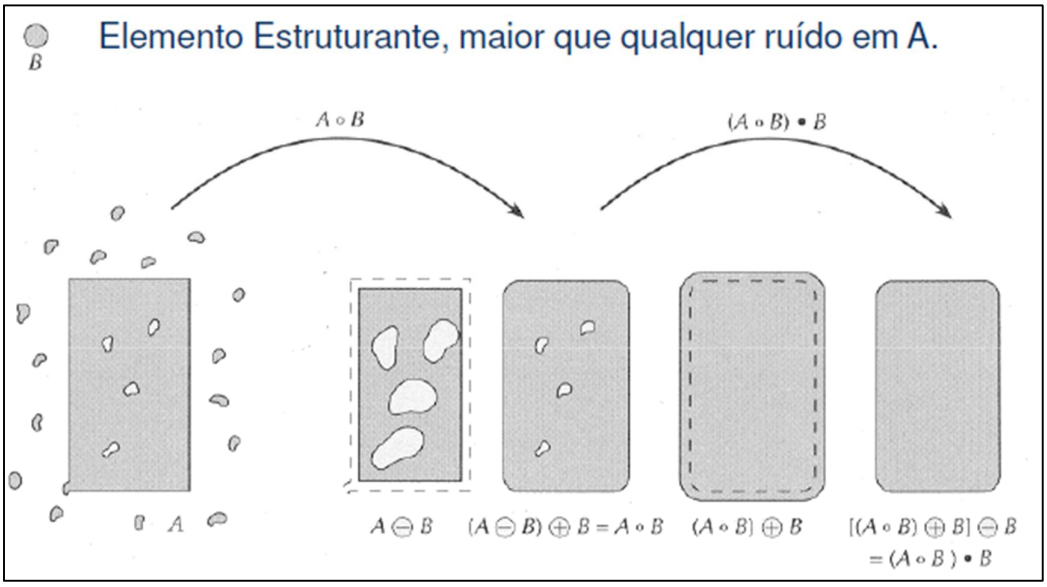


Figura 10 – Aplicação de Filtro Morfológico. Fonte: *(GONZAGA, IRIS SEL, 2000).*

2.5 Redes Neurais

As redes neurais são técnicas computacionais que oferecem um modelo matemático guiado na estrutura neural de organismos inteligentes, adquirindo conhecimento através da experiência (CARVALHO, 2000).

As redes neurais artificiais são compostas por unidades associadas a pesos, que são interconectadas através de canais de comunicação. Estas unidades realizam operações as suas entradas através de seus pesos, e sua saída esta interconectada a novas unidades de processamento.

O modelo matemático de uma unidade de processamento de uma rede neural é apresentado na equação 8. A figura 11 traz uma representação desta unidade.

$$\varphi(.) = x1.W1 + x2.W2 + \cdots + xpWp$$
$$y = 1, se\ \varphi(.) \geq b \qquad (8)$$
$$y = 0, se\ \varphi(.) < b$$

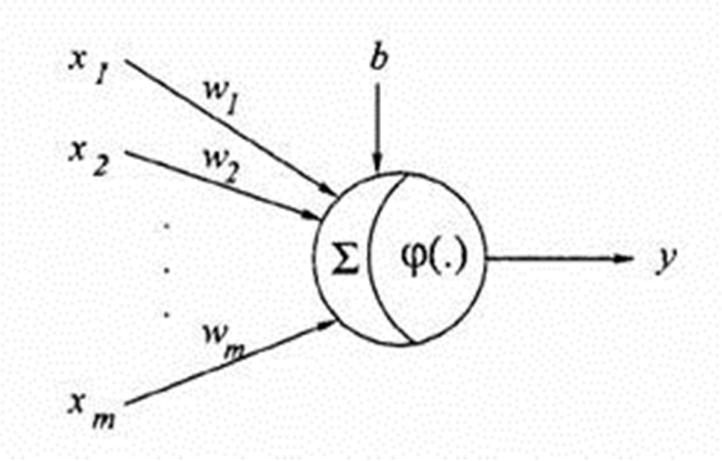


Figura 11 – Representação do Perceptron. Fonte: *(BARROS, 2009)*

Na equação 8, xi representa as entradas da unidade, Wi os pesos associados a cada entrada, b o limitador e y a saída da unidade. A saída da unidade assume o valor 1 somente quando a somatória da multiplicação da cada entrada por seu peso associado assume um valor acima do limitador.

As redes neurais são treinadas de modo que os pesos de cada unidade se adaptam, a partir de uma regra, aos exemplos utilizados para o treinamento. Normalmente as redes neurais são separadas em camadas, de modo que as unidades estejam conectadas às camadas superiores e/ou inferiores.

2.5.1 Perceptron Multicamadas (MLP)

As redes neurais MLP (*Multi-Layer Perceptron* – Percéptron Multicamadas) são formadas por diversas camadas, sendo que cada uma possui uma função específica. A camada de saída constrói a resposta a partir de estímulos da camada intermediária, que por sua vez é a responsável pela extração das características, sendo seus pesos uma codificação das características apresentadas pela camada de entrada (CARVALHO, 2000).

A figura 12 traz uma representação de uma rede neural MLP com uma camada intermediária.

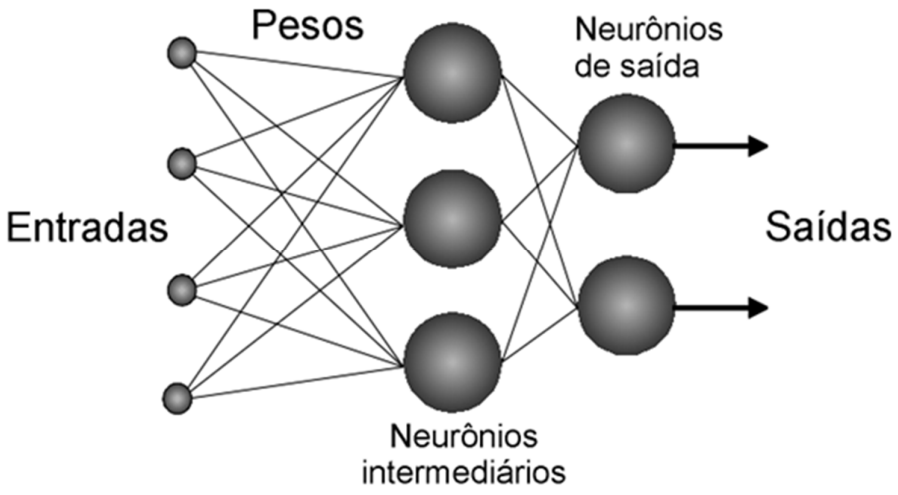


Figura 12 – Representação de uma Rede Neural MLP. Fonte: *(FERRAMOLA, 2002)*

Uma representação adequada para simular o mapeamento da entrada para a saída através das unidades intermediárias pode ser obtida se existirem conexões entre as unidades de entrada e um conjunto suficientemente grande de unidades intermediárias. A escolha adequada do número de unidades intermediárias que reproduz a saída com exatidão é o grande desafio das redes neurais MLP.

O treinamento da rede neural MLP é feita através do algoritmo de retro propagação de erro (*backpropagation)*. Neste algoritmo primeiramente um padrão é apresentado à camada de entrada da rede e passa por todas as camadas até que a camada de saída produza um valor.

Este resultado é comparado ao resultado esperado e um erro é calculado. Em seguida o erro obtido é propagado das camadas de saída até a camada de entrada, sendo que os pesos de cada camada vão sendo modificados conforme o erro se propaga. Desta forma os erros vão sendo diminuídos progressivamente.

Após o treinamento da rede MLP com o algoritmo *backpropagation*, ela pode ser usada para classificar novos dados. Neste caso, os dados são apresentados às camadas de entrada e são processados pelas camadas intermediárias até que os resultados sejam apresentados na saída.

2.6 Gestos

No contexto deste trabalho, gestos podem ser definidos como um movimento de braços e mãos com o objetivo de expressar sentimentos ou permitir a comunicação entre pessoas. Os gestos de mão possuem duas classificações básicas:

- Gesto estático, ou postura, que é definido por um momento particular da mão sendo caracterizado em uma imagem;
- Gesto dinâmico, que é definido por um movimento da mão ao longo de um tempo e caracterizado por um vídeo ou por uma sequencia de imagens.

Em particular, os gestos dinâmicos da mão podem ser caracterizados por uma configuração inicial e final da mão e pelos movimentos intermediários que caracterizam a trajetória do movimento.

Uma possível utilização dos gestos dinâmicos da mão na Interação Homem Computador é através da visão computacional. Esta abordagem se torna natural por não exigir que dispositivos mecânicos ou luvas sejam acoplados ao braço para que a interação seja realizada (RIBEIRO H. L., 2006).

2.7 Considerações Finais

Este capítulo apresentou alguns dos conceitos que serão utilizados para o desenvolvimento do sistema de reconhecimento de gestos dinâmicos de mão humana a partir de imagens de vídeo obtidas através de uma câmera *web*.

A biblioteca OpenCV foi apresentada, mostrando que pode ser utilizada para a implementação de algoritmos na área de visão computacional. O modelo e os espaços RGB e RGB normalizado também foram explicados, tendo sido dito que usar o espaço RGB normalizado é mais adequado para a segmentação de pele humana.

Alguns conceitos de segmentação de imagens foram apresentados, como subtração de fundo, histograma e limiarização, e estes conceitos podem ser utilizados no processamento de imagens para separação de área de interesse.

Foi também explicado como filtros morfológicos podem auxiliar na eliminação de ruídos no pós-processamento de imagens, através dos conceitos de morfologia

matemática. Também foi apresentada a rede neural MLP e seu algoritmo de retro propagação de erro (*backpropagation*).

Por último a definição de gestos dinâmicos de mão humana foi apresentada, bem como sua possível utilização na Interação Homem-Computador.

Capítulo 3 – Materiais e Métodos

O objetivo deste trabalho é desenvolver um algoritmo para o reconhecimento de gestos dinâmicos de mão humana que são capturados de imagens de vídeo por meio de uma *webcam*, analisando quais condições de ambiente que produzem os melhores índices de reconhecimento.

A figura 13 traz o diagrama de blocos da proposta de trabalho deste projeto.

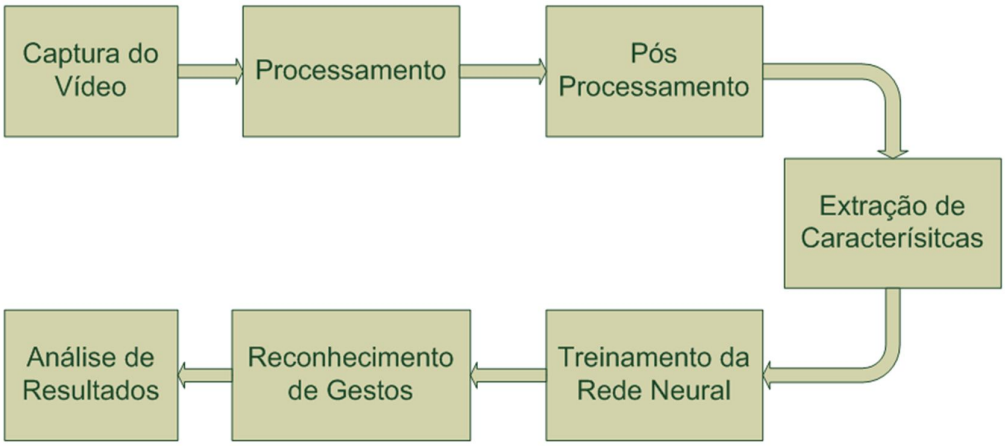


Figura 13 – Diagrama de blocos do projeto

Um determinado gesto da mão humana esquerda é executado em frente a uma *webcam*, a uma distância e iluminação fixas. As imagens de vídeos que representam os gestos são adquiridos por meio de uma *webcam*.

O processamento e pós-processamento das imagens de vídeo são realizados fazendo uso dos conceitos de visão computacional e implementados com o auxílio da biblioteca OpenCV, em linguagem de programação C.

A extração de características que são usadas na rede neural, bem como o seu treinamento e a sua aplicação são também realizadas com o auxilio da biblioteca OpenCV com as funções relacionadas à aprendizagem de máquina.

A análise de resultados identifica as condições de ambiente, referentes à distância da mão humana em relação à *webcam* e as condições de iluminação, que produzem os melhores resultados quanto ao reconhecimento dos gestos.

Na ultima etapa da análise, é determinado o algoritmo apresenta bons índices de desempenho quanto ao reconhecimento para diferentes tipos de mão humana, nas melhores condições de ambiente verificadas.

Assim, este projeto contempla todas as etapas de um sistema de visão computacional, e os materiais e métodos serão apresentados a seguir.

3.1 Materiais

A câmera *web* utilizada para a aquisição de imagens foi a *Genius Look 317*. As características desta câmera podem ser vistas na tabela 2.

Tabela 2 – Características da câmera *web* utilizada

Marca	Genius
Modelo	Look 317
Sensor de Imagem	VGA CMOS Image Sensor
Interface	USB 1.1/1.0
Tipo de Lente	Foco manual
Resolução de Imagem	640x480
Resolução de Vídeo	30 quadros/s
Sistemas Operacionais suportados	Windows 7, Vista, XP, 2000, Me, 98SE

O computador utilizado neste projeto possui as seguintes características, apresentadas na tabela 3.

Tabela 3 – Características do computador utilizado

Marca	Itautec
Modelo	INFOWAY NOTE W7645
Sistema Operacional	Windows 7 Professional
Processador	Intel Pentium Dual CPU T2330 1.6 GHz
Memória RAM	2 GB
Tipo de Sistema	Sistema Operacional 64 bits

Os programas (*softwares*) utilizados para o desenvolvimento do algoritmo e para a segmentação manual de imagens bem como a biblioteca que contém as funções de visão computacional e redes neurais são apresentados na tabela 4.

Tabela 4 – *Softwares* e Biblioteca Utilizados

Ambiente de Desenvolvimento	
Software	Microsoft Visual Studio C++ 2010
Versão	*Express Edition* v. 10.0.30319.0
Programa Usado para Segmentação Manual de Imagens	
Software	Adobe Photoshop CS5 Extended
Versão	12.1 x32
Funções de Visão Computacional e Aprendizagem de Máquina	
Biblioteca	OpenCV (Open Source Computer Vision Library)
Versão	2.0

3.2 Sistema Proposto

O algoritmo foi desenvolvido com o auxilio do software Microsoft Visual Studio C++, utilizando a linguagem de programação C e a biblioteca OpenCV. O Apêndice A traz os algoritmos completos, utilizado em cada uma das etapas deste sistema.

A metodologia utilizada para o desenvolvimento do sistema proposto na figura 13 será descrita a seguir.

3.2.1 Captura de Imagens

A *webcam* é posicionada por meio de um suporte, com suas lentes localizadas em frente a mão humana que executará os gestos. A distância entre a mão humana e a câmera é fixa para um grupo de imagens capturadas e sua influência em relação à taxa de reconhecimento dos gestos será fruto de análise.

A iluminação e a cor do fundo utilizados neste ambiente também são fixas, e sua influência no processo de segmentação da imagem também será analisada.

A câmera *web* está conectada ao computador por meio de uma conexão USB. O trecho do programa responsável pela captura do vídeo da *webcam* e pela separação dos quadros que serão analisado são mostrados na figura 14.

```
CvCapture* capture;
IplImage* frame = 0;

capture = cvCaptureFromCAM(0);
frame = cvQueryFrame( capture );
```

Figura 14 – Código responsável pela Inicialização da Câmera.

A função *cvCaptureFromCAM(0)* captura os quadros da câmera *web* a uma taxa máxima de 30 quadros por segundo. A função cvQueryFrame(capture) é a responsável por associar o quadro extraído a variável *frame*, do tipo imagem.

A câmera web utilizada (Genius Look 317), integrada a este sistema, fornece a variável *frame* uma imagem de três canais de oito bits cada, no espaço RGB, de resolução 640x480 pixels.

A variável *frame* é capturada a uma taxa de 15 quadros por segundo quando o programa está sendo executado, porém apenas alguns destes quadros serão processados e representarão o gesto dinâmico executado. O intervalo de tempo de captura dos quadros analisados é fixo, e definido de acordo com a velocidade de processamento do sistema.

Um exemplo de frame capturado pode ser visto na figura 15.

Figura 15 – Exemplo de Quadro Capturado

3.2.2 Processamento de Imagens

O objetivo desta etapa é segmentar a mão humana em relação ao fundo da imagem. Cada quadro capturado no intervalo de tempo definido passa por este processo de segmentação. As etapas deste processo podem ser observadas na figura 16.

Figura 16 – Etapas para o Processamento de Imagens

Primeiramente a imagem capturada é transformada do espaço de cores RGB para o espaço RGB normalizado, visto que é mais adequado para a representação da pele humana (FIBIGER, 2004). A figura 17 traz o código que trata desta conversão.

```
        cvCopy( frame, image, 0 );

        // Dados da Imagem
        height = image->height;
        width = image->width;
        step = image->widthStep;
        channels = image->nChannels;
        data = (uchar *)image->imageData;

        //Conversão para rgb normalizado
        for(i=0;i<height;i++) for(j=0;j<width;j++)
        {
                B=data[i*step+j*channels+0];
                G=data[i*step+j*channels+1];
                R=data[i*step+j*channels+2];
                L=B+G+R;

                if(L==0) { b=0; g=0; r=0; }
                else { b=(float)B/L; g=(float)G/L; r=(float)R/L; }

                data[i*step+j*channels+0]=(uchar)255*b;
                data[i*step+j*channels+1]=(uchar)255*g;
                data[i*step+j*channels+2]=(uchar)255*r;
        }
```

Figura 17 – Código responsável pela conversão para RGB normalizado

A variável *frame* é copiada para a variável *image,* que representará a imagem no espaço RGB normalizado. A altura (*height*) e largura (*height*) da imagem, o número de canais da imagem (*channels*) e o vetor de representação da imagem (*data*) são capturados inicialmente. Em seguida, todos os pixels da imagem são convertidos para o RGB normalizado, conforme a descrição teórica do item 2.2.2.

Um exemplo desta conversão pode ser visto na figura 18.

Figura 18 – Exemplo de Conversão para o RGB normalizado

O terceiro canal da imagem que representa o vermelho normalizado é isolado através da função *cvCvtPixToPlane(imagem, 0, 0, plano3, 0).* Um exemplo do canal r normalizado pode ser observado na figura 17.

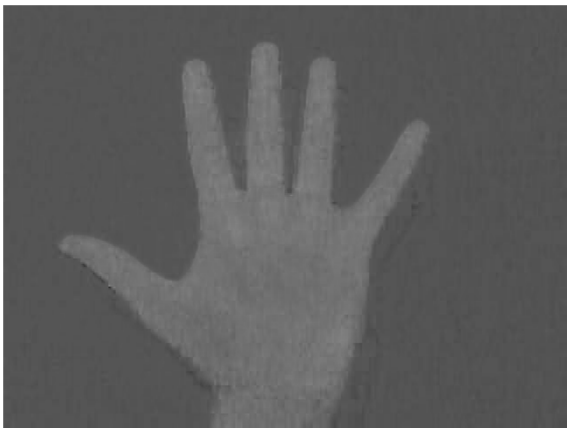
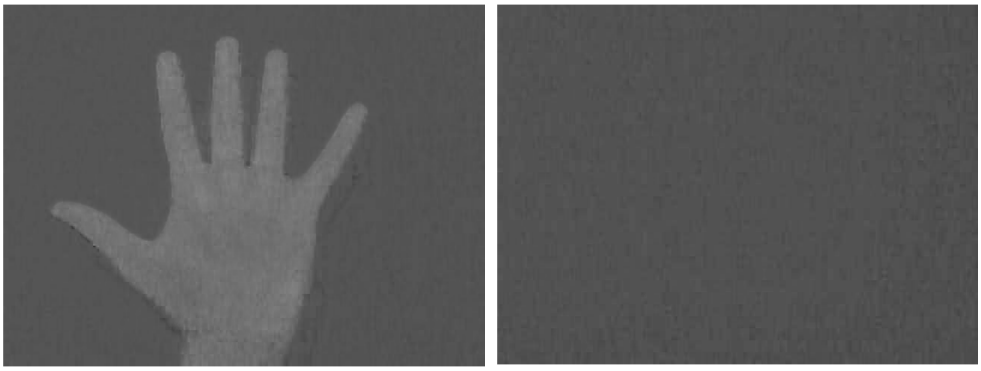
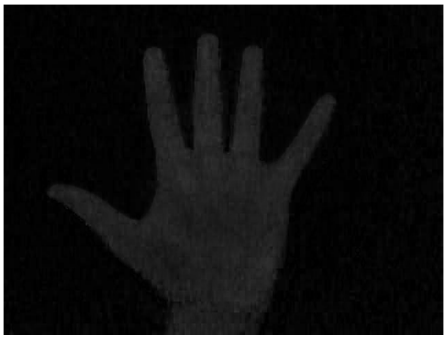


Figura 19 – Exemplo do Canal r normalizado

Em seguida o fundo da cena capturado previamente por meio da variável *img_fundo*, é subtraído da imagem no espaço RGB normalizado, representada pela variável *plano3*, pela função *cvAbsDiff(img_fundo,plano3,plano3)*. A figura 20 traz as imagens presentes nesta função em um exemplo.



(a) Imagem **(b) Fundo**



(c) Subtração de Fundo

Figura 20 – Exemplo de Subtração de Fundo

O histograma (*hist*) desta imagem é calculado e um valor para limiarização (*thr*) da imagem é definido utilizando-se o método do vale. O trecho de programa da figura 21 trata destas etapas.

```
cvCalcHist( &plano3, hist, 0, 0 );

// Encontra o Vale
float *valor, *valorAnt;
if( !para)
{
	thr=0;
	valorAnt=cvGetHistValue_1D(hist,1);
	for(i=2;(i<256)&&(!para);i++)
	{
		valor=cvGetHistValue_1D(hist,i);
		if(*valor < *valorAnt)
			para=true;
		*valorAnt=*valor;
	}
	para=false;
	for(;(i<256)&&(!para);i++)
	{
		valor=cvGetHistValue_1D(hist,i);
		if(*valor > *valorAnt)
			para=true;
		*valorAnt=*valor;
	}
	thr=i;
}
```

Figura 21 – Código responsável pelo Cálculo do Histograma

A figura 22 traz o histograma gerado para o exemplo da figura 15 com o limiar estabelecido pelo método do vale.

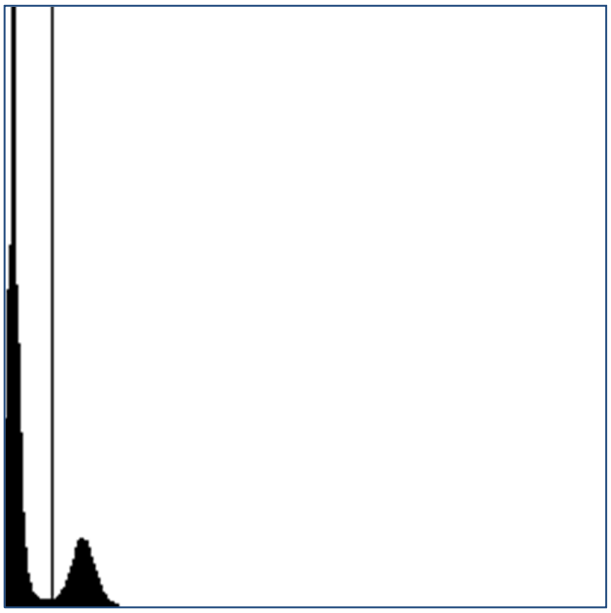


Figura 22 – Exemplo de Histograma para a Imagem da figura 20

Por ultimo a imagem é binarizada com o valor de limiarização encontrado (thr), utilizando-se a função *cvThreshold(plano3,plano3,thr,255,CV_THRESH_BINARY)*. A figura 23 traz a figura 20 binarizada.

Figura 23 – Exemplo de Binarização da Imagem da figura 20

3.2.3 Pós Processamento de Imagens

Após a obtenção da imagem segmentada, são utilizados dois filtros morfológicos, de abertura e de fechamento, para a remoção de ruídos ainda presentes na imagem. As funções da figura 24 executam estes filtros sobre a imagem binária resultante do processamento de imagens.

```
//Filtros Morfologicos
cvMorphologyEx(img_rgb_cinza,img_rgb_cinza,NULL,0,CV_MOP_OPEN,1);
cvMorphologyEx(img_rgb_cinza,img_rgb_cinza,NULL,0,CV_MOP_CLOSE,1);
```

Figura 24 – Código responsável pelos Filtros Morfológicos

A figura 25 traz o resultado da imagem após a aplicação dos filtros morfológicos.

Figura 25 – Exemplo de Aplicação de Filtro Morfológico

Outra etapa deste pós-processamento de imagens é a remoção do punho que executa o gesto dinâmico, que não traz informações significativas quanto à representação do gesto. O trecho de programa da figura 26 aplica este procedimento.

```
	//Centro da Imagem
	cvMoments(g_gray,&momento,1);
	x=(int)(momento.m10/momento.m00);
	y=(int)(momento.m01/momento.m00);

	//Encontra Raio
	step = imagem->widthStep;
	channels = imagem->nChannels;
	data = (uchar *)imagem->imageData;
	raio=0;
	for(j=x;j>=0;j--)
		if(data[y*step+j*channels]==0)
		{	raio=x-j; break; }

	//Imagem sem Punho
	for(i=0;i<(imagem->width);i++) for(j=y+raio;j<(imagem->height);j++)
	{
		data[j*step+i*channels+0]=0;
		data[j*step+i*channels+1]=0;
		data[j*step+i*channels+2]=0;
	}
```

Figura 26 – Código responsável pelo Pós-Processamento da Imagem

Primeiramente o centro da imagem que representa a mão é localizado através do momento de inércia dos pixels brancos da imagem binarizada (valor=1). Em seguida, é calculada a distância euclidiana entre o centro e o seu extremo esquerdo máximo, mantendo o valor de y que representa o centro constante.

Esta distância é usada para definir o extremo inferior da nova imagem. Nesta nova imagem, os pixels que estão localizados abaixo desta distância em relação ao centro da imagem são definidos com valor zero. A figura 27 traz o resultado da imagem após a remoção do punho.

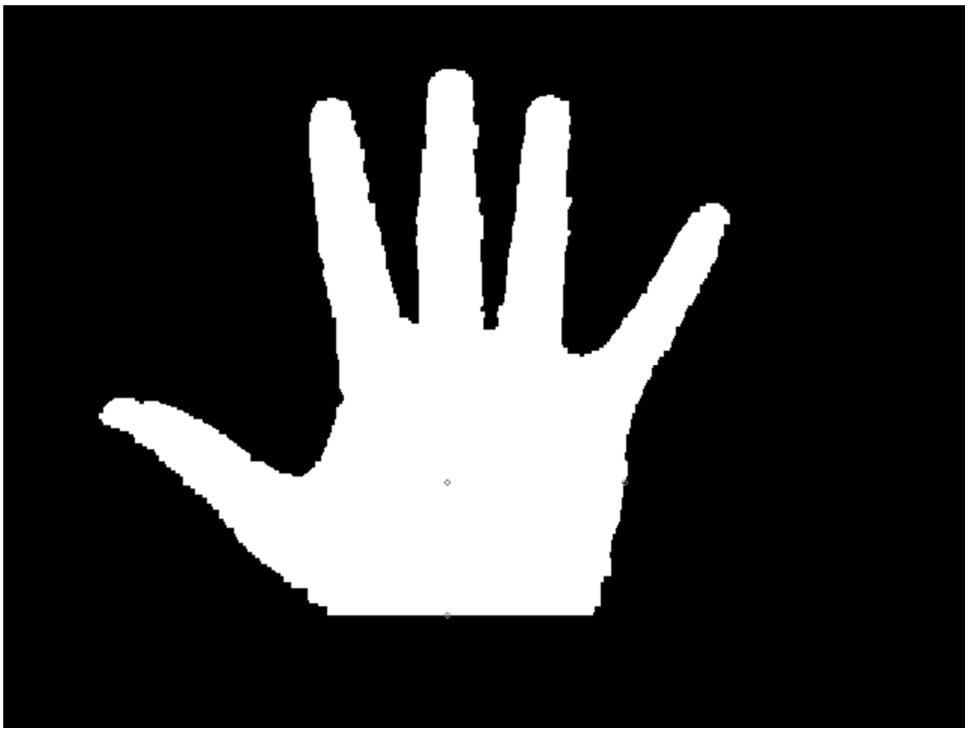


Figura 27 – Exemplo de Remoção de Punho

3.2.4 Extração de características

Primeiramente se buscam os limites que definem a nova imagem, sendo os extremos definidos pelos pontos que tem pixel branco na escala de cinza (valor=255). O trecho do código da figura 28 busca estes extremos.

```
int imin=imagem->width, imax=0, jmin=imagem->height, jmax=0;
step = g_gray->widthStep;
data = (uchar *)g_gray->imageData;
for(i=0;i<(g_gray->width);i++)
       for(j=0;j<(g_gray->height);j++)
              if(data[j*step+i]==255)
              {
                     if(i<imin) imin=i;
                     if(j<jmin) jmin=j;
                     if(i>imax) imax=i;
                     if(j>jmax) jmax=j;
              }
```

Figura 28 – Código responsável por Encontrar os Limites da Imagem

Em seguida a imagem é dividida em 16 quadrantes de igual área e é calculada a porcentagem de pixels brancos de cada quadrante. Estes valores representam o vetor de características de um dos quadros capturados do vídeo.

O trecho de código da figura 29 executa estes dois procedimentos.

```
//Calcula N Pixels Branco
step = imagem->widthStep;
channels = imagem->nChannels;
data = (uchar *)imagem->imageData;
for(i=imin;i<imax;i++) for(j=jmin;j<jmax;j++)
       if(data[j*step+i*channels+0]==255)
       {
              N_PIXELS[0]++;
              iL=(int)(((float)(i-imin)/(imax-imin))*PX_LIN);
              jC=(int)(((float)(j-jmin)/(jmax-jmin))*PX_COL);
              N_PIXELS[1+iL+jC*PX_LIN]++;
       }

//Vetor de Caracteristicas
for(i=0;i<N_CARACT;i++)
       Caract[i] =
   (int) ((float)N_PIXELS[i+1]/((imax-imin)*(jmax-jmin)/(N_CARACT))*100);
```

Figura 29 – Código responsável por Extrair as Características

A imagem da figura 30 traz um exemplo de imagem segmentada e dividida em 16 quadrantes de tamanhos iguais para o cálculo da porcentagem de pixels brancos.

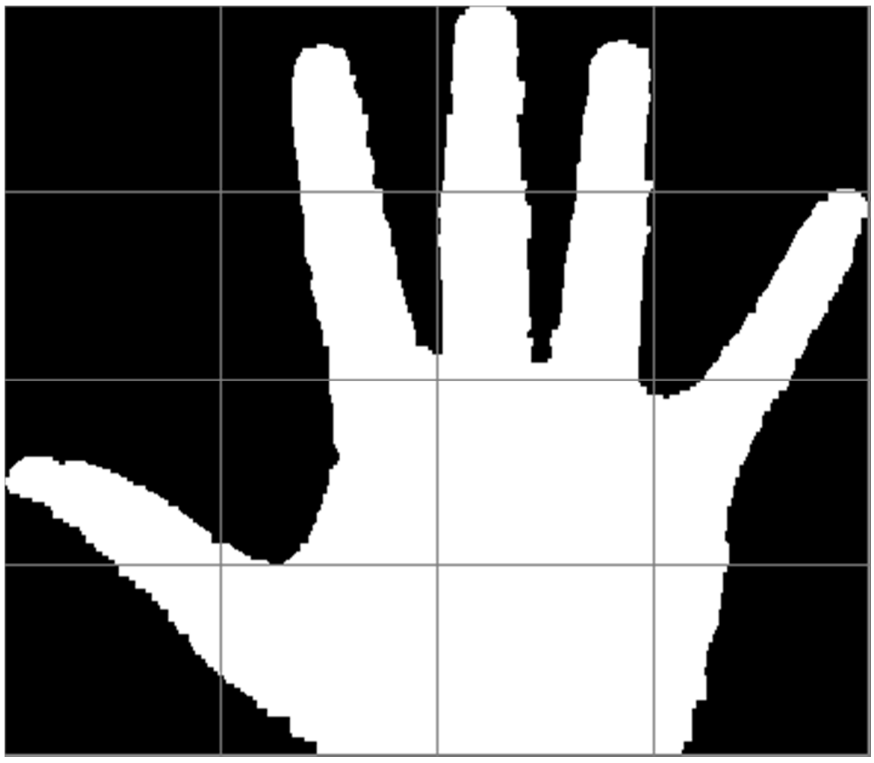


Figura 30 – Exemplo de Divisão de Imagem para Extração de Características

Em seguida, os vetores de características que representam cada quadro de um vídeo são agrupados sequencialmente, de forma que o novo vetor represente as características do vídeo. A função armazena, encontrada no Apêndice A, agrupa este vetor de característica em um arquivo de dados.

3.2.5 Treinamento da Rede Neural

Newman et al (1998) propuseram um algoritmo de rede neural MLP para o reconhecimento de letras. Este algoritmo está preparado para o treinamento e o reconhecimento de uma rede neural a partir de um vetor de características (NEWMAN, 1998).

Um algoritmo com parâmetros similares será utilizado para o treinamento da rede neural MLP a partir do vetor de característica que define cada imagem de cada vídeo que representa um gesto de mão humana. O código do programa desenvolvido pode ser encontrado no Apêndice A.

Os parâmetros e critérios que definem a rede neural utilizada são mostrados na tabela 5.

Tabela 5 – Parâmetros da Rede Neural utilizada

Característica	Valor Associado
Rede Neural	MLP – Perceptron Multicamadas
Algoritmo	Retro propagação (*Backpropagation*)
Número de Camadas	4
Neurônios por Camada	(16*Nº de Quadros) / 100 / 100 / (Nº de Classes)
Critério para Treino	Máximo de 300 iterações com erro 0,01

Como amostras para o treinamento desta rede neural são utilizadas os vetores de características que representam um gesto associado à classe que o representa. O número de classes e a quantidade de amostras para cada classe durante o treinamento da rede neural MLP variam de acordo com a análise realizada no item 3.3 deste trabalho.

3.2.6 Reconhecimento

As funções desenvolvidas para o reconhecimento da rede neural foram similares às utilizadas por Newman et al (1998). Elas podem ser encontradas no código do programa, no Apêndice A.

A partir de um vetor de dados de entrada e da rede neural treinada, o programa fornece como resultado um indicador que representa a probabilidade de pertencer a cada uma das classes (NEWMAN, 1998).

A interpretação destes resultados será realizada para cada análise no capítulo 4 deste trabalho.

3.3 Análises

As seguintes análises e critérios serão utilizados para atingir os objetivos deste trabalho.

3.3.1 Iluminação e Cor de Fundo

Primeiramente serão avaliadas as condições de ambiente da captura de imagens, como o tipo de iluminação e a cor do cenário de fundo utilizado. A tabela 6 traz os cenários que serão analisados.

Tabela 6 – Cenários para Análise da Iluminação e Cor do Fundo

Cenário	Tipo de Iluminação	Cor de Fundo
1A	Ambiente	Branco
1B	Fluorescente Superior	Branco
1C	Ambiente	Preto
1D	Fluorescente Superior	Preto

O fundo utilizado para captura de imagens é sempre simples e as cores variam entre branco e preto. As iluminações são a ambiente, a luz natural do dia, e a obtida por uma lâmpada incandescente localizada superiormente à cena.

Para cada cenário são capturadas 50 imagens, a uma distância entre a câmera web e mão humana tal que se posicione entre o limite superior e inferior da imagem. Essas imagens são submetidas a uma segmentação manual através do programa *Adobe Photoshop*, e ao algoritmo de processamento e pós-processamento de imagens deste trabalho.

Como classificador para este sistema foi criada uma tabela de contingências. Foi definida como imagem padrão a obtida através da segmentação manual e a imagem a ser analisada a obtida por meio da segmentação pelo algoritmo proposto. A figura 31 traz um exemplo destas imagens.

(a) (b)

Figura 31 – Exemplo de Segmentação Manual (a) e pelo Algoritmo proposto (b)

Para cada imagem são calculados os seguintes índices:

- VP (Verdadeiro Positivo) – O pixel é branco na imagem padrão e na analisada;
- VN (Verdadeiro Negativo) – O pixel é preto na imagem padrão e na analisada;
- FP (Falso Positivo) – O pixel é branco na imagem analisada e preto na padrão;
- FN (Falso Negativo) – O pixel é preto na imagem analisada e branco na padrão.

Como métricas para esta análise são usadas:

- Sensibilidade – $TPR = \frac{VP}{VP+FN}$
- Acurácia – $A = \frac{VP+VN}{VP+VN+FP+FN}$
- Precisão – P= $\frac{VP}{VP+FP}$

Em cada cenário é calculada a média destes parâmetros para todas as imagens capturadas. O melhor cenário será aquele que apresentar os maiores valores para cada parâmetro.

Este cenário é utilizado em todas as análises seguintes.

3.3.2 Distância entre Mão e Câmera

Nesta etapa foi analisada a distância entre a mão humana e câmera *web* que captura as imagens e produz os melhores resultados no reconhecimento de gestos. As distâncias analisadas se encontram na tabela 7.

Tabela 7 – Distâncias Avaliadas entre a Mão Humana e a Câmera *Web*

Cenário	Distancia
2A	35 cm
2B	45 cm
2C	50 cm
2D	55 cm
2E	60 cm
2F	70 cm
2G	80 cm
2H	85 cm
2I	90 cm
2J	100 cm
2K	110 cm

Para cada cenário são capturadas 5 classes de gestos dinâmicos de mão humana, sendo que em cada classe são capturadas 40 amostras. As classes de gestos podem ser visualizadas no Apêndice B.

Em seguida, cada grupo de vídeos capturado é submetido ao algoritmo de reconhecimento de gestos proposto neste trabalho, sendo que metade das imagens é utilizada para treinar a rede neural e a outra metade é submetida à rede treinada.

A porcentagem de acerto no reconhecimento de gestos para cada cenário é calculada e também a probabilidade média que o algoritmo encontrou para a classe correta.

A melhor distância será a que apresentar melhores resultados e será utilizada nas etapas de análise seguintes.

3.3.3 Reconhecimento

Nesta etapa, pretende se verificar se o algoritmo apresenta bons índices de desempenho quanto ao reconhecimento para diferentes tipos de mão humana, nas melhores condições de ambiente.

São analisadas 25 classes de gestos, sendo capturadas 50 amostras de cada classe executadas por 5 mãos humanas distintas. As 25 classes utilizadas podem ser vistas no Apêndice C deste trabalho.

Os vídeos capturados são submetidos ao algoritmo de reconhecimento de gestos, sendo metade utilizada para treinar a rede neural e a outra metade submetida à rede treinada.

Calcula-se, então, a porcentagem de acerto do reconhecimento de gestos, a probabilidade média que o algoritmo estipulou para a classe correta e a matriz de confusão para este sistema. Na matriz de contingência estará disponível o número de classes caracterizadas como cada classe e os seguintes parâmetros:

- Falsa Aceitação - $FPR = \frac{VP}{VP+TN}$;
- Falsa Rejeição - $FNR = \frac{VN}{VN+TP}$

Todos os resultados obtidos serão analisados no capítulo 4 deste trabalho.

Capítulo 4 – Resultados e Conclusões

Este capítulo apresenta os resultados encontrados para as análises descritas no capítulo 3 e as conclusões dos resultados obtidos.

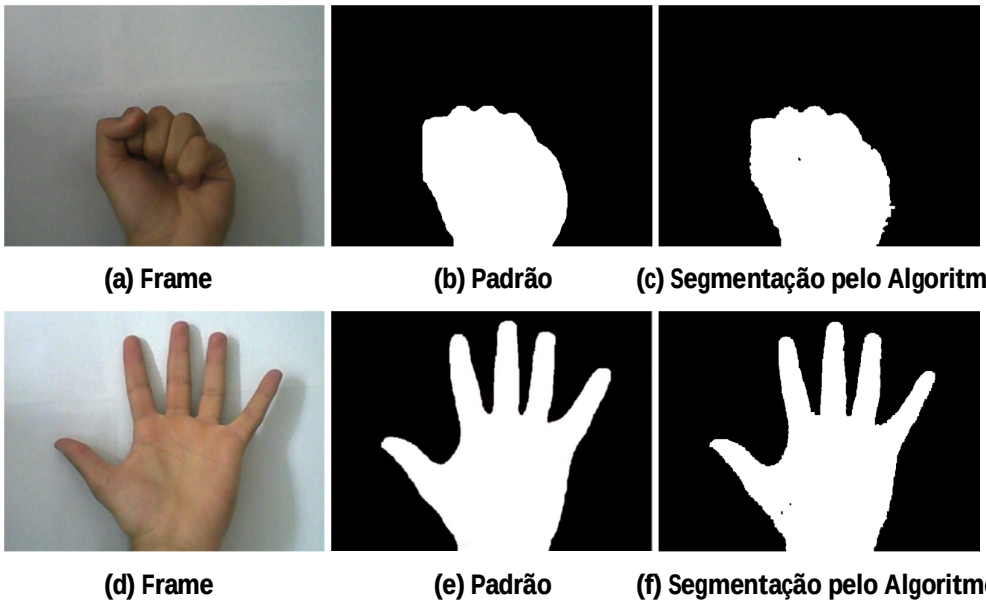
4.1 Resultados

Os resultados apresentados referem-se à análise das melhores condições de iluminação e a cor de fundo do cenário; da distância entre a mão humana e a câmera *web*; e a análise do algoritmo proposto nas melhores condições encontradas.

4.1.1 Iluminação e Cor de Fundo

As figuras 32, 33, 34 e 35 trazem dois exemplos de quadros que foram segmentados manualmente e usando o algoritmo proposto, para cada um dos cenários analisados quanto à iluminação do ambiente e a cor do cenário de fundo utilizado.

A figura 32 traz os resultados obtidos para dois quadros capturados com a iluminação ambiente e com o cenário de fundo branco.



(a) Frame **(b) Padrão** **(c) Segmentação pelo Algoritmo**

(d) Frame **(e) Padrão** **(f) Segmentação pelo Algoritmo**

Figura 32 – Imagens do Cenário 1A com Iluminação Ambiente e Fundo Branco

A figura 33 traz os resultados obtidos para dois quadros capturados com a iluminação fluorescente superior e com o cenário de fundo branco.

(a) Frame **(b) Padrão** **(c) Segmentação pelo Algoritmo**

(d) Frame **(e) Padrão** **(f) Segmentação pelo Algoritmo**

Figura 33 – Imagens do Cenário 1B com Iluminação Fluorescente e Fundo Branco

A figura 34 traz os resultados obtidos para dois quadros capturados com a iluminação ambiente e com o cenário de fundo preto.

(a) Frame **(b) Padrão** **(c) Segmentação pelo Algoritmo**

(d) Frame **(e) Padrão** **(f) Segmentação pelo Algoritmo**

Figura 34 – Imagens do Cenário 1C com Iluminação Ambiente e Fundo Preto

A figura 35 traz os resultados obtidos para dois quadros capturados com a iluminação fluorescente superior e com o cenário de fundo preto.

(a) Frame (b) Padrão (c) Segmentação pelo Algoritmo

(d) Frame (e) Padrão (f) Segmentação pelo Algoritmo

Figura 35 – Imagens do Cenário 1D com Iluminação Fluorescente e Fundo Preto

A comparação visual entre as imagens capturadas permite concluir que a segmentação com o cenário de fundo branco apresenta os melhores resultados.

A segmentação pelo algoritmo proposto para os cenários 1A e 1B são as mais similares à segmentação padrão e são as que possuem a menor quantidade de ruído.

A análise visual destes dois cenários não permite encontrar grandes diferenças para a segmentação da imagem com a variação da iluminação no cenário de fundo branco. Os quadros segmentados com a iluminação ambiente e com a iluminação fluorescente superior apresentam resultados similares.

A tabela 8 mostra os resultados quantitativos das análises dos quatro cenários e a sensibilidade, a acurácia e a precisão médias para o grupo de imagens capturado.

Tabela 8 – Resultados para a Análise da Iluminação e Cor de Fundo

Cenário	Iluminação	Fundo	TPR	A	P
1A	Ambiente	Branco	95.280%	98.507%	99.849%
1B	Fluorescente Superior	Branco	95.340%	98.534%	99.933%
1C	Ambiente	Preto	57.992%	91.914%	99.987%
1D	Fluorescente Superior	Preto	59.549%	86.773%	96.150%

Os cenários com os melhores resultados foram os de fundo simples de cor branca. Assim como a análise visual permitiu inferir, a variação das condições de iluminação para este cenário não alterou de forma significativa os parâmetros de desempenho da segmentação analisados, sendo que a iluminação fluorescente superior apresentou resultados levemente superiores para os três parâmetros avaliados.

A sensibilidade e a acurácia apresentaram alterações significativas quando a cor do cenário de fundo foi alterada de branco para preto, podendo caracterizar um método de segmentação que não é robusto à variação das condições do cenário de fundo.

Assim, foi escolhido o cenário 1B com o fundo de cor branco e com a iluminação fluorescente superior para a realização das demais etapas deste trabalho, visto que foi o cenário que apresentou os melhores resultados na análise visual e para os três parâmetros calculados: sensibilidade, acurácia e precisão.

4.1.2 Distância entre Mão Humana e Câmera

A tabela 9 apresenta a avaliação quantitativa da influencia da distância entre a mão humana e a câmera *web* na porcentagem de reconhecimento de gestos dinâmicos e na porcentagem média definida para a classe correta no grupo de gestos em cada cenário.

Tabela 9 – Resultados para a Distância entre a Mão e a Câmera

Cenário	Distância	% de Reconhecimento		% Média para a Classe Correta
		Treinamento	Testes	
2A	35 cm	100 %	96 %	87,523 %
2B	45 cm	100 %	96 %	88,236 %
2C	50 cm	100 %	95 %	87,531 %
2D	55 cm	100 %	96 %	87,250 %
2E	60 cm	100 %	95 %	86,762 %
2F	70 cm	100 %	95 %	86,120 %
2G	80 cm	100 %	94 %	85,341 %
2H	85 cm	100 %	89 %	79,128 %
2I	90 cm	100 %	73 %	64,523 %
2J	100 cm	100 %	63 %	54,871 %
2K	110 cm	100 %	54 %	47,234 %

Os resultados obtidos permitem inferir que para valores de distâncias entre 35 cm e 80 cm o algoritmo proposto apresenta valores satisfatórios para a porcentagem de reconhecimento e para a porcentagem média de a classe reconhecida ser a classe correta.

É possível observar que a partir de valores de distância acima de 80 cm os valores dos parâmetros utilizados passam a diminuir de forma significativa com o aumento da distância entre a câmera *web* e a mão que executa o gesto.

Uma possível explicação para esta diminuição é que com o aumento da distância a imagem passa a ser muito pequena e isto impede que a imagem represente o gesto de uma mão humana, mesmo com o uso de características proporcionais que são independentes do tamanho da imagem.

Assim se estabelece como valores de distância entre 35 cm e 80 cm como os valores que apresentam resultados ótimos de reconhecimento.

A distância de 45 cm entre a mão humana e a câmera *web* utilizada para a captura de vídeos foi utilizada para a análise da ultima etapa deste trabalho, por apresentar os melhores resultados.

4.1.3 Reconhecimento

A tabela 10 traz o resultado da execução do algoritmo de reconhecimento de gestos dinâmicos de mão humana para uma amostra de 25 classes, com 50 imagens para cada uma das classes. A matriz de confusão para esta análise pode ser encontrada no Apêndice D.

Tabela 10 – Parâmetros da Análise de Reconhecimento

% de Reconhecimento		% Média para a Classe Correta	FNR	FPR
Treinamento	Testes			
100 %	94,72 %	84,634 %	5,28%	0,223%

O valor de 5,12% para a taxa de rejeição média (FNR) indica que poucas classes de entrada foram classificadas erroneamente. O valor máximo para esta taxa para uma classe, observado na tabela de contingências, foi de 12%.

O valor de 0,223% para a taxa de aceitação média (FPR) indica a razão de amostras que foram classificados em uma classe sem pertencer a ela em relação ao total de elementos que não pertencem a esta classe. O valor máximo para esta taxa para uma classe, observado na tabela de contingências, foi de 0,518%.

Para cada gesto analisado pelo algoritmo proposto ofereceu como resposta a probabilidade de a amostra de entrada pertencer a cada uma das classes. A probabilidade média do sistema oferecer a classe correta foi de 84,634%.

O sistema apresentou ainda uma alta precisão (94,72%) no reconhecimento de gestos visto que o grupo total de amostras utilizado para o reconhecimento é de 625 gestos dinâmicos de mão humana.

4.2 Conclusões

A análise das condições de iluminação e cor do cenário de fundo permitiu concluir que o método de segmentação de imagens desenvolvido apresenta melhores resultados com a cor de cenário de fundo simples branco. A variação da iluminação de ambiente para fluorescente superior com o cenário simples branco não alterou de forma significativa o bom desempenho da segmentação, sendo que a iluminação fluorescente superior propiciou resultados levemente melhores.

A variação do cenário para a cor de fundo preta influenciou negativamente no desempenho da segmentação, indicando um método que não é robusto a variação da cor do cenário de fundo simples utilizado.

Em relação à variação da distância entre a câmera *web* utilizada para a captura de imagens e a mão humana que executa os gestos, o algoritmo mostrou se eficiente no reconhecimento dos gestos para valores entre 35 cm e 80 cm. Este desempenho se explica pela escolha de características proporcionais ao tamanho da mão humana com o corte do punho, que independem do tamanho da imagem.

A distância de 45 cm para um cenário de fundo simples com a cor branca e iluminação florescente superior foram as condições de ambiente para o sistema que resultaram em melhores porcentagens no reconhecimento de gestos.

Nestas condições o sistema foi analisado para um conjunto de 25 classes de gestos dinâmicos distintos de mão humana, sendo que foram capturadas 50 amostras de 5 mãos humanas diferentes. O sistema mostrou se preciso e eficiente para este grupo fechado de classes, com alta porcentagem de reconhecimento de gestos e alta probabilidade média associada à classe de resposta correta.

Assim, em condições de ambiente limitadas, o sistema proposto mostrou-se apto a ser utilizado em aplicações de Interação Humano-Computador (IHC).

4.3 Trabalhos Futuros

O método de segmentação utilizado não se mostrou robusto com a variação das condições do cenário de fundo. Assim, podem ser estudados e utilizados em projetos futuros outros métodos que respondam melhor não só a variação da cor como a presença de objetos dinâmicos no cenário de captura dos gestos. Algoritmos de segmentação que utilizam redes neurais são exemplos de métodos que respondem melhor a estas condições.

O número de classes utilizadas deve ser aumentado para garantir uma análise estatística mais eficiente.

O desenvolvimento de aplicações que utilizem este algoritmo também pode ser objeto de trabalhos futuros. Programas que utilizam o reconhecimento de gestos de mão humana para o acionamento de aplicativos em um computador são exemplos de aplicações. Neste cenário a mão humana, executando um gesto dinâmico previamente estabelecido, seria uma interface de entrada para programas executados em um computador, sem o auxilio do teclado ou do mouse para tal função.

Referências Bibliográficas[1]

BARROS, R. S. **Trabalho de Pesquisa Acadêmica - Redes Neurais (I. A.),** 2009. Disponível em: <http://rbarrosx.blogspot.com/2009/01/trabalho-de-pesquisa-acadmica-redes.html>. Acesso em: 14 de setembro de 2011.

BRADSKI, G.; KABLER, A. **Learning OpenCV: Computer Vision with OpenCV Library**. O'Reilly Media.[s.n.] Sebastopol, CA - EUA, 2008.

CARROL, J. M. **Human Computer Interaction (HCI),** 2009. Disponível em: <http://www.interactiondesign.org/encyclopedia/human_computer_interaction_hci.html>. Acesso em 24 de agosto de 2011.

CARVALHO, A. P. L. F. **Redes Neurais Artificiais**, 2000. Disponível em: <http://www.icmc.usp.br/~andre/research/neural/index.htm#intro>. Acesso em 27 de agosto de 2011.

FERRAMOLA, M. L. **Introdução a Redes Neurais**, 2002. Disponível em: <http://www.cin.ufpe.br/~if114/Monografias/Redes%20Neurais/Com%20Pesos/introducao.htm>. Acesso em 12 de setembro de 2011

FIBIGER, R. S. **Estudo Comparativo de Técnicas de Visão Computacional para Detecção de Pele Humana,** 68p. Campo Grande: UCDB, 2004.

GONZAGA, A. **Material Didatico - Disciplina SEL5886**, 2000. Disponível em: <http://iris.sel.eesc.usp.br/sel886/index_arquivos/Page750.htm>. Acesso em 12 de setembro de 2011.

GONZAGA, A. **Kanguera: Olho Local Mão Distante**, 2011. Disponível em: <http://iris.sel.eesc.usp.br/weblab/default.html>. Acesso em 1 de Setembro de 2011

GONZALES, R. C.; WOODS, R. E.; EDDINS, S. E. **Digital Image Processing using MATLAB,** 624 p. New York: Pretince Hall, 2003.

NEWMAN, D. J.; HETTICH, S.; BLAKE, C. L.; MERZ, C. J. **UCI Repository of machine learning databases**, 1998. Disponível em:<http://www.ics.uci.edu/~mlearn/MLRepository.html>. Acesso em 10 de setembro de 2011.

PAVLOVIC, V. I.; SHARMA, R.; HUAN, T. S. **Visual Interpretation of Hand Gestures for Human-Computer Interaction.** IEEE Transactions on Pattern Analysis and Machine Intelligence, vol 19, no 7, 1997.

RIBEIRO, H. L. 2006. 144p. **Reconhecimento de gestos utilizando segmentação de imagens dinâmicas de mãos baseada no modelo de misturas Gaussianas e na cor da pele.** Dissertação (Mestrado) - Escola de Engenharia de São Carlos, São Carlos, SP, Brasil.

[1] De acordo com a Associação Brasileira de Normas Técnicas. NBR 6023.

RIBEIRO, J. M. 2007. 101p. **Segmentação da pele humana em imagens coloridas baseada em valores das médias da vizinhança em subimagens.** Dissertação (Mestrado) - Escola de Engenharia de São Carlos, São Carlos, SP, Brasil.

SALDANHA, M. F. S.; FREITAS, C. C. Segmentação de Imagens Digitais: Uma Revisão, In: **XI Workshop do Curso de Computação Aplicada.** LIT/INPE, 2009.

SOUZA, G. S. **Visão Computacional - INF2604 / Prof. Marcelo Gattass**, 2009., Disponível em: <http://www.tecgraf.puc-rio.br/~mgattass/ra/trb09/Guilherme/Visao Computacional%20-%20Trabalho%202.htm> Acesso em 26 de agosto de 2011.

APÊNDICE A – Códigos do Algoritmo Utilizado

A.1 Algoritmo para Análise da Iluminação e Cor de Fundo

```
//     BIBLIOTECAS
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

//     DEFINICOES
// Define o Numero de Imagens a Capturar
#define N_IMAGENS               50

// Define Cenário
#define CENA                      3

int Captura=1, i, j, k;
char NomeImg[50];
bool para=true;

int hcc=1;

int c;
int thr=5, armazenou=0, salva=0;

IplImage *image = 0, *img_rgb_cinza=0, *img_fundo=0, *img_hist=0;

void Histograma (IplImage * imagem, IplImage * hist_img);

void callback (IplImage* frame)
{
       int step,height,width,channels;
       unsigned char *data;
       int R, G, B, L;
       float r, g, b;

       if( (char) c == 's')
              salva=1;

       cvCopy( frame, image, 0 );

       cvNamedWindow( "Video", 1);
       cvShowImage( "Video", frame);


       // Dados da Imagem
       height = image->height;
       width = image->width;
       step = image->widthStep;
       channels = image->nChannels;
       data = (uchar *)image->imageData;

       //Identica a Faixa e Zera o Resto
       for(i=0;i<height;i++) for(j=0;j<width;j++)
       {
```

```
                B=data[i*step+j*channels+0];
                G=data[i*step+j*channels+1];
                R=data[i*step+j*channels+2];
                L=B+G+R;

                if(L==0) { b=0; g=0; r=0; }
                else { b=(float)B/L; g=(float)G/L; r=(float)R/L; }

                data[i*step+j*channels+0]=(uchar)255*b;
                data[i*step+j*channels+1]=(uchar)255*g;
                data[i*step+j*channels+2]=(uchar)255*r;
        }



        cvNamedWindow( "bin", 1 );
        Histograma(image,img_hist);

        if(salva==1)
        {
                if( Captura > N_IMAGENS)
                        salva=0;
                else
                {
                        sprintf(NomeImg,"./Imagem/%d/Img%d -
Padrao.bmp",CENA,Captura);
                        cvSaveImage(NomeImg,img_rgb_cinza);
                }
        }

        cvMorphologyEx(img_rgb_cinza,img_rgb_cinza,NULL,0,CV_MOP_OPEN,1);
        cvMorphologyEx(img_rgb_cinza,img_rgb_cinza,NULL,0,CV_MOP_CLOSE,1);

        cvShowImage("bin",img_rgb_cinza);

        if(salva==1)
        {
                if( Captura > N_IMAGENS)
                        salva=0;
                else
                {
                        printf("\n Capturando a Imagem na Cena %d - Img %d",CENA,
Captura);

                        sprintf(NomeImg,"./Imagem/%d/Img%d -
Frame.bmp",CENA,Captura);
                        cvSaveImage(NomeImg,frame);

                        sprintf(NomeImg,"./Imagem/%d/Img%d -
Binaria.bmp",CENA,Captura);
                        cvSaveImage(NomeImg,img_rgb_cinza);

                        Captura++;

                        salva=0;
                }
        }

        c=cvWaitKey(10);
}

void Histograma (IplImage * imagem, IplImage * hist_img)
```

```
{
	IplImage* plano3 = cvCreateImage( cvGetSize(imagem), 8, 1 );
	int hist_size = 256, bin_w, i;
	float max_value=0;

	cvCvtPixToPlane( imagem, 0, 0, plano3, 0 );


	plano3->origin = imagem->origin;

	if((char) c == 'f' )
	{
		cvCopy(plano3, img_fundo, 0 );
		//capt_max=5;
	}
	if((char) c == 'z')
		img_fundo=0;
	if( ((char) c == '=') && thr<255)
		thr++;
	if( ((char) c == '-') && thr>0)
		thr--;
	if((char) c == 't')
		para=false;


	if(img_fundo!=0)
	{
		cvAbsDiff(img_fundo,plano3,plano3);
	}



	CvHistogram* hist;
	{
		float _ranges[] = { 0, 255 };
		float* ranges[] = { _ranges};
		hist = cvCreateHist(1,&hist_size,CV_HIST_ARRAY,ranges,1);
	}

	cvCalcHist( &plano3, hist, 0, 0 );

	cvZero( hist_img );
	cvGetMinMaxHistValue( hist, 0, &max_value, 0, 0 );
    cvScale( hist->bins, hist->bins, ((double)hist_img->height)/max_value, 0 );
	cvSet( hist_img, cvScalarAll(255), 0 );

	bin_w = cvRound((double)hist_img->width/hist_size);
    for( i = 0; i < hist_size; i++ )
	    cvRectangle( hist_img, cvPoint(i*bin_w, hist_img->height),
                     cvPoint((i+1)*bin_w,hist_img->height -
cvRound(cvGetReal1D(hist->bins,i))),
                     cvScalarAll(0), -1, 8, 0 );

	// Encontra o Vale
	float *valor, *valorAnt;
	if( !para)
	{
		thr=0;
		valorAnt=cvGetHistValue_1D(hist,1);
		for(i=2;(i<256)&&(!para);i++)
		{
			valor=cvGetHistValue_1D(hist,i);
```

```
                        if(*valor < *valorAnt)
                                para=true;
                        *valorAnt=*valor;
                }
                para=false;
                for(;(i<256)&&(!para);i++)
                {
                        valor=cvGetHistValue_1D(hist,i);
                        if(*valor > *valorAnt)
                                para=true;
                        *valorAnt=*valor;
                }
                thr=i;
        }

        cvRectangle(hist_img,cvPoint(thr,0),cvPoint(thr,hist_img-
>height),cvScalarAll(0),-1,8,0);

        cvNamedWindow( "hist", 1 );
        cvShowImage("hist",hist_img);

        if(img_fundo!=0)
        {
                cvThreshold(plano3,plano3,thr,255,CV_THRESH_BINARY);
        }

        cvCopy(plano3,img_rgb_cinza);

        cvReleaseImage(&plano3);
        cvReleaseHist(&hist);
}


int main( int argc, char** argv )
{

        CvCapture* capture;
        IplImage* frame = 0;
        capture = cvCaptureFromCAM(0);
        frame = cvQueryFrame( capture );

        if( !frame )
            exit(0);

        if( !image )
        {
                        image = cvCreateImage( cvGetSize(frame), 8, 3 );
                        image->origin = frame->origin;
                        img_fundo = cvCreateImage( cvGetSize(frame), 8, 1 );
                        img_fundo->origin = frame->origin;
                        img_rgb_cinza = cvCreateImage( cvGetSize(frame), 8, 1 );
                        img_rgb_cinza->origin = frame->origin;
                        img_hist = cvCreateImage( cvSize(256,256), 8, 1 );
        }

        cvCopy( frame, image, 0 );

        while(1)
        {
                frame = cvQueryFrame( capture );
                callback(frame);
        }
```

```
        // Fecha as Janelas Criadas
        cvDestroyWindow("Video");
        cvDestroyWindow("rgb");
        cvReleaseImage( &image );
        return 0;
}
```

A.2 Algoritmo de para Cálculo dos Parâmetros

```
//     BILIOTECAS
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

//     DEFINICOES
// Define o Numero de Imagens a Capturar
#define N_IMAGENS                2

// Define Cenário
#define N_CENAS                              4
#define ARQUIVO_DATA      "./analise.data"

int main( int argc, char** argv )
{
       // Variáveis
       char NomeImg[50];
       int Img=1, Cena=1, i, j;

       int heightP, widthP, stepP;
    unsigned char *dataP;

       int heightA, widthA, stepA;
    unsigned char *dataA;

       IplImage* imageA;
       IplImage* imageP;

       int TP, FP, TN, FN;

       FILE *fp;

       //Arquivo de dados
       fp=fopen(ARQUIVO_DATA,"w");

       //dados
       fprintf(fp,"CENA IMG TP FP TN FN \n");

       for(;Cena<=N_CENAS;Cena++)
              for(Img=1;Img<=N_IMAGENS;Img++)
              {
              // Carrega a Imagem a Padrão
              sprintf(NomeImg,"./Imagem/%d/Img%d - Padrao.bmp",Cena,Img);
              imageP = cvLoadImage(NomeImg,CV_LOAD_IMAGE_GRAYSCALE);

              // Carrega a Imagem a ser Analisada
              sprintf(NomeImg,"./Imagem/%d/Img%d - Binaria.bmp",Cena,Img);
              imageA = cvLoadImage(NomeImg,CV_LOAD_IMAGE_GRAYSCALE);

              // Dados da Imagem Padrão
              heightP = imageP->height;
              widthP = imageP->width;
              stepP = imageP->widthStep;
              dataP = (uchar *)imageP->imageData;
```

```
		// Dados da Imagem a ser Analisada
		heightA = imageA->height;
		widthA = imageA->width;
		stepA = imageA->widthStep;
		dataA = (uchar *)imageA->imageData;

		//Variáveis de Análise
		TP=0; FP=0;
		TN=0; FN=0;

		//Calcula TP, FP, TN e FN
		for(i=0;i<heightP;i++) for(j=0;j<widthP;j++)
		{
			if(dataA[i*stepA+j]>125) //Amostra Positiva
			{
				if(dataP[i*stepP+j]>125) // Padrão Positivo
					TP++;
				else // Padrão Negativo
					FP++;
			}
			else // Amostra Negativa
			{
				if(dataP[i*stepP+j]>125) // Padrão Positivo
					FN++;
				else // Padrão Negativo
					TN++;
			}

		}

		//Salva dados
		fprintf(fp,"%d %d %d %d %d %d \n",Cena,Img,TP,FP,TN,FN);

	}

	fclose(fp);

	return 0;
}
```

A.3 Algoritmo de Captura de Quadros e Segmentação

```
//       BIBLIOTECAS
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

//    DEFINICOES
// Define o Numero de Classes e de Capturas em cada Classe
#define N_CAPTURAS               40
#define N_CLASSES                5
#define N_CAP_TEMPO                 5
#define WAIT                        1

//       VARIÁVEIS
int Classe=1, Captura=1,i, j, k, tempo=0,wt=0;
char NomeImg[50];
bool para=true;

int hcc=1;

int c;
int thr=5, capt_max=0, treinou=0, armazenou=0, salva=0;
IplImage *image = 0, *img_rgb_cinza=0, *img_fundo=0, *img_hist=0;

void Histograma (IplImage * imagem, IplImage * hist_img);

void callback (IplImage* frame)
{
       int step,height,width,channels;
       unsigned char *data;
       int R, G, B, L;
       float r, g, b;

       if( (char) c == 's')
              salva=1;

       cvCopy( frame, image, 0 );

       cvNamedWindow( "Video", 1);
       cvShowImage( "Video", frame);

       // Dados da Imagem
       height = image->height;
       width = image->width;
       step = image->widthStep;
       channels = image->nChannels;
       data = (uchar *)image->imageData;

       //Conversão para rgb normalizado
       for(i=0;i<height;i++) for(j=0;j<width;j++)
       {
              B=data[i*step+j*channels+0];
              G=data[i*step+j*channels+1];
              R=data[i*step+j*channels+2];
              L=B+G+R;
```

```
                if(L==0) { b=0; g=0; r=0; }
                else { b=(float)B/L; g=(float)G/L; r=(float)R/L; }

                data[i*step+j*channels+0]=(uchar)255*b;
                data[i*step+j*channels+1]=(uchar)255*g;
                data[i*step+j*channels+2]=(uchar)255*r;
        }

        cvNamedWindow( "bin", 1 );
        Histograma(image,img_hist);

        //Filtros Morfologicos
        cvMorphologyEx(img_rgb_cinza,img_rgb_cinza,NULL,0,CV_MOP_OPEN,1);
        cvMorphologyEx(img_rgb_cinza,img_rgb_cinza,NULL,0,CV_MOP_CLOSE,1);

        cvShowImage("bin",img_rgb_cinza);

        if((salva==1) && (( tempo % WAIT) == 0) )
        {
                if( (Classe >= N_CLASSES) && (Captura > N_CAPTURAS))
                        salva=0;
                else
                {
                        printf("\n Capturando Classe %d, Imagem %d, Tempo
%d",Classe,Captura,tempo/WAIT+1);


        sprintf(NomeImg,"./Imagem/C%d_I%d_T%d.bmp",Classe,Captura,tempo/WAIT+1);
                        cvSaveImage(NomeImg,img_rgb_cinza);

                        tempo++;

                        if( (tempo/WAIT+1) > N_CAP_TEMPO )
                        {
                                Captura++;
                                tempo=0;
                                salva=0;

                                if(Captura > N_CAPTURAS)
                                {
                                        Classe++;
                                        Captura=1;
                                }
                        }

                }


        }
        else if (salva == 1)
                tempo++;


        c=cvWaitKey(10);

}

void Histograma (IplImage * imagem, IplImage * hist_img)
{
        IplImage* plano3 = cvCreateImage( cvGetSize(imagem), 8, 1 );
        int hist_size = 256, bin_w, i;
        float max_value=0;
```

```
      cvCvtPixToPlane( imagem, 0, 0, plano3, 0 );


      plano3->origin = imagem->origin;

      if((char) c == 'f' )
      {
            cvCopy(plano3, img_fundo, 0 );
            //capt_max=5;
      }
      if((char) c == 'z')
            img_fundo=0;
      if( ((char) c == '=') && thr<255)
            thr++;
      if( ((char) c == '-') && thr>0)
            thr--;
      if((char) c == 't')
            para=false;


      if(img_fundo!=0)
      {
            cvAbsDiff(img_fundo,plano3,plano3);
      }


      CvHistogram* hist;
      {
            float _ranges[] = { 0, 255 };
            float* ranges[] = { _ranges};
            hist = cvCreateHist(1,&hist_size,CV_HIST_ARRAY,ranges,1);
      }

      cvCalcHist( &plano3, hist, 0, 0 );

      cvZero( hist_img );
      cvGetMinMaxHistValue( hist, 0, &max_value, 0, 0 );
    cvScale( hist->bins, hist->bins, ((double)hist_img->height)/max_value, 0 );
      cvSet( hist_img, cvScalarAll(255), 0 );

      bin_w = cvRound((double)hist_img->width/hist_size);
    for( i = 0; i < hist_size; i++ )
        cvRectangle( hist_img, cvPoint(i*bin_w, hist_img->height),
                     cvPoint((i+1)*bin_w,hist_img->height -
cvRound(cvGetReal1D(hist->bins,i))),
                     cvScalarAll(0), -1, 8, 0 );

      // Encontra o Vale
      float *valor, *valorAnt;
      if( !para)
      {
            thr=0;
            valorAnt=cvGetHistValue_1D(hist,1);
            for(i=2;(i<256)&&(!para);i++)
            {
                  valor=cvGetHistValue_1D(hist,i);
                  if(*valor < *valorAnt)
                        para=true;
                  *valorAnt=*valor;
            }
            para=false;
```

```
		for(;(i<256)&&(!para);i++)
		{
			valor=cvGetHistValue_1D(hist,i);
			if(*valor > *valorAnt)
				para=true;
			*valorAnt=*valor;
		}
		thr=i;
	}

	cvRectangle(hist_img,cvPoint(thr,0),cvPoint(thr,hist_img-
>height),cvScalarAll(0),-1,8,0);

	if(img_fundo!=0)
	{
		cvThreshold(plano3,plano3,thr,255,CV_THRESH_BINARY);
	}

	cvCopy(plano3,img_rgb_cinza);
	cvReleaseImage(&plano3);
	cvReleaseHist(&hist);
}


int main( int argc, char** argv )
{

	CvCapture* capture;
	IplImage* frame = 0;
	capture = cvCaptureFromCAM(0);
	frame = cvQueryFrame( capture );

	if( !frame )
	    exit(0);

	if( !image )
	{
			image = cvCreateImage( cvGetSize(frame), 8, 3 );
			image->origin = frame->origin;
			img_fundo = cvCreateImage( cvGetSize(frame), 8, 1 );
			img_fundo->origin = frame->origin;
			img_rgb_cinza = cvCreateImage( cvGetSize(frame), 8, 1 );
			img_rgb_cinza->origin = frame->origin;
			img_hist = cvCreateImage( cvSize(256,256), 8, 1 );
	}

	cvCopy( frame, image, 0 );

	while(1)
	{
		frame = cvQueryFrame( capture );
		callback(frame);
	}

	// Fecha as Janelas Criadas
	cvDestroyWindow("Video");
	cvDestroyWindow("rgb");
	cvReleaseImage( &image );
	return 0;
}
```

A.4 Algoritmo de Extração de Características e Geração de Resultados

```
//       BIBLIOTECAS
#include <math.h>
#include <cv.h>
#include <highgui.h>
#include <ml.h>
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include <string.h>

//     DEFINICOES
// Define o Numero de Classes e de Capturas em cada Classe
#define N_CAPTURAS               40
#define N_CLASSES                5
#define N_CARACT                 16
#define N_CAP_TEMPO                 5

#define PX_LIN   4
#define PX_COL   4

#define PORCENTAGEM_TREINAM   75

#define XML                              "./sinais.xml"
#define ARQUIVO_DATA      "./sinais.data"
#define ESTATISTICA_DATA     "./estatistica.data"

//     VARIAVEIS
int Classe=1, Captura=1, Tempo=1;
char NomeImg[50];
CvMemStorage* g_storage = NULL;

int Aleatorio[N_CLASSES*(N_CAPTURAS*PORCENTAGEM_TREINAM/100)];
int Dados[N_CLASSES][N_CAPTURAS][N_CARACT*N_CAP_TEMPO];
int Caract[N_CARACT];
int i, j, k;
int c;
int thr=5, capt_max=0, treinou=0, armazenou=0, salva=0;

IplImage *image = 0, *img_rgb_cinza=0, *img_fundo=0, *img_hist=0;

static int
read_num_class_data( const char* filename, int var_count,
                     CvMat** data, CvMat** responses )
{
    const int M = 1024;
    FILE* f = fopen( filename, "rt" );
    CvMemStorage* storage;
    CvSeq* seq;
    char buf[M+2];
    float* el_ptr;
    CvSeqReader reader;
    int i, j;

    if( !f )
        return 0;

    el_ptr = new float[var_count+1];
    storage = cvCreateMemStorage();
```

```
    seq = cvCreateSeq( 0, sizeof(*seq), (var_count+1)*sizeof(float), storage );

    for(;;)
    {
        char* ptr;
        if( !fgets( buf, M, f ) || !strchr( buf, ',' ) )
            break;
        el_ptr[0] = buf[0];
        ptr = buf+2;
        for( i = 1; i <= var_count; i++ )
        {
            int n = 0;
            sscanf( ptr, "%f%n", el_ptr + i, &n );
            ptr += n + 1;
        }
        if( i <= var_count )
            break;
        cvSeqPush( seq, el_ptr );
    }
    fclose(f);

    *data = cvCreateMat( seq->total, var_count, CV_32F );
    *responses = cvCreateMat( seq->total, 1, CV_32F );

    cvStartReadSeq( seq, &reader );

    for( i = 0; i < seq->total; i++ )
    {
        const float* sdata = (float*)reader.ptr + 1;
        float* ddata = data[0]->data.fl + var_count*i;
        float* dr = responses[0]->data.fl + i;

        for( j = 0; j < var_count; j++ )
            ddata[j] = sdata[j];
        *dr = sdata[-1];
        CV_NEXT_SEQ_ELEM( seq->elem_size, reader );
    }

    cvReleaseMemStorage( &storage );
    delete el_ptr;
    return 1;
}

void armazena(void)
{
       FILE *fp;
       float sort;

       printf("\n Armazenando. Aguarde....");

       // Inicializa os Enderecos
       for(i=0;i<(N_CLASSES*(N_CAPTURAS*PORCENTAGEM_TREINAM/100));i++)
         Aleatorio[i]=i;

       // Embaralha
       for(i=0;i<(N_CLASSES*(N_CAPTURAS*PORCENTAGEM_TREINAM/100));i++)
       {
              // Sorteia e muda pra inteiro
              sort=((rand()%(N_CLASSES*(N_CAPTURAS*PORCENTAGEM_TREINAM/100))));
              j=(int)sort;

              //Troca posicoes
```

```
                k=Aleatorio[j];
                Aleatorio[j]=Aleatorio[i];
                Aleatorio[i]=k;
        }

        //Armazena todos os Dados no .data
        fp=fopen(ARQUIVO_DATA,"w");
        for(i=0;i<(N_CLASSES*(N_CAPTURAS*PORCENTAGEM_TREINAM/100));i++)
        {

        fprintf(fp,"%c",Aleatorio[i]/(N_CAPTURAS*PORCENTAGEM_TREINAM/100)+'A');
                //fprintf(fp,"%c",i/N_CAPTURAS+'A');
                for(j=0;j<N_CARACT*N_CAP_TEMPO;j++)

        fprintf(fp,",%d",Dados[Aleatorio[i]/(N_CAPTURAS*PORCENTAGEM_TREINAM/100)][
Aleatorio[i]%(N_CAPTURAS*PORCENTAGEM_TREINAM/100)][j]);
                        //fprintf(fp,",%d",Dados[i/N_CAPTURAS][i%N_CAPTURAS][j]);
                fprintf(fp,"\n");
        }
        for(i=0;i<N_CLASSES;i++)
for(j=(N_CAPTURAS*PORCENTAGEM_TREINAM/100);j<N_CAPTURAS;j++)
        {
                fprintf(fp,"%c",i+'A');
                for(k=0;k<(N_CARACT*N_CAP_TEMPO);k++)
                        fprintf(fp,",%d",Dados[i][j][k]);
                fprintf(fp,"\n");
        }
        fclose(fp);

        printf("\n Armazenou!!!");
        armazenou=1;
}

void Treina(void)
{
    char v_filename_to_save[] = XML;
    char* filename_to_save = v_filename_to_save;
       char  v_data_filename[] = ARQUIVO_DATA;
    char* data_filename = v_data_filename;
       FILE *fp;

       const int class_count = N_CLASSES;
    CvMat* data = 0;
    CvMat train_data;
    CvMat* responses = 0;
    CvMat* mlp_response = 0;

       armazena();

    int ok = read_num_class_data( data_filename, N_CARACT*N_CAP_TEMPO, &data,
&responses );
    int nsamples_all = 0, ntrain_samples = 0;
    int i, j;
    double train_hr = 0, test_hr = 0;
    CvANN_MLP mlp;

       if( !ok )
    {
        printf( "Could not read the database %s\n", data_filename );
    }

    printf( "The database %s is loaded.\n", data_filename );
```

```
    nsamples_all = data->rows;
    ntrain_samples = (int)(nsamples_all*PORCENTAGEM_TREINAM/100);


    CvMat* new_responses = cvCreateMat( ntrain_samples, class_count, CV_32F );

    printf( "Unrolling the responses...\n");
    for( i = 0; i < ntrain_samples; i++ )
    {
        int cls_label = cvRound(responses->data.fl[i]) - 'A';
        float* bit_vec = (float*)(new_responses->data.ptr + i*new_responses-
>step);
        for( j = 0; j < class_count; j++ )
            bit_vec[j] = 0.f;
        bit_vec[cls_label] = 1.f;
    }
    cvGetRows( data, &train_data, 0, ntrain_samples );

    int layer_sz[] = { data->cols, 100, 100, class_count };
    CvMat layer_sizes =
        cvMat( 1, (int)(sizeof(layer_sz)/sizeof(layer_sz[0])), CV_32S, layer_sz
);
    mlp.create( &layer_sizes );
    printf( "Training the classifier (may take a few minutes)...");
    mlp.train( &train_data, new_responses, 0, 0,
        CvANN_MLP_TrainParams(cvTermCriteria(CV_TERMCRIT_ITER,300,0.01),
             CvANN_MLP_TrainParams::RPROP,0.01));
    cvReleaseMat( &new_responses );
    printf("\n");

       mlp_response = cvCreateMat( 1, class_count, CV_32F );

    fp=fopen(ESTATISTICA_DATA,"w");
       for( i = 0; i < nsamples_all; i++ )
    {
        int best_class;
        CvMat sample;
        cvGetRow( data, &sample, i );
        CvPoint max_loc = {0,0};
        mlp.predict( &sample, mlp_response );
        double minV, maxV;
             cvMinMaxLoc( mlp_response, &minV, &maxV, 0, &max_loc, 0 );
        best_class = max_loc.x + 'A';

             fprintf(fp,"%c",(int)responses->data.fl[i]);
             for(k=0;k<N_CLASSES;k++)
                    fprintf(fp,",%f",(mlp_response->data.fl[k]-
(float)minV)/(float)(maxV-minV));
             fprintf(fp,"\n");

             int r = fabs((double)best_class - responses->data.fl[i]) <
FLT_EPSILON ? 1 : 0;

        if( i < ntrain_samples )
            train_hr += r;
        else
            test_hr += r;
    }
       fclose(fp);

    test_hr /= (double)(nsamples_all-ntrain_samples);
    train_hr /= (double)ntrain_samples;
```

```
    printf( "Recognition rate: train = %.1f%%, test =
%.1f%%\n",train_hr*100,test_hr*100);

    mlp.save(filename_to_save);

    cvReleaseMat(&mlp_response);
    cvReleaseMat(&data);
    cvReleaseMat(&responses);

       printf("\n\n Treinou!!!");
}

void caract(void)
{
       int N_PIXELS[N_CARACT+1],iL,jC, DIST;

       for(i=0;i<=N_CARACT;i++)
              N_PIXELS[i]=0;

       IplImage  *imagem = NULL, *g_gray = NULL, *gray=NULL;
       CvMoments momento;
       CvSeq* contorno = NULL, *circulos=NULL,*dedos=NULL;
       CvSeq *maior = NULL, *next = NULL;

       int  x=0,y=0,step,channels,raio=0,j,i;
       unsigned char *data;

       //Carrega Imagem
       sprintf(NomeImg,"./Imagem/C%d_I%d_T%d.bmp",Classe,Captura,Tempo);
       imagem=cvLoadImage(NomeImg,1);

       g_gray = cvCreateImage( cvGetSize(imagem), 8, 1 );
       gray = cvCreateImage( cvGetSize(imagem), 8, 1 );
       cvCvtColor( imagem, g_gray, CV_BGR2GRAY );
       cvThreshold( g_gray, g_gray, 100, 255, CV_THRESH_BINARY );

       //Centro da Imagem
       cvMoments(g_gray,&momento,1);
       x=(int)(momento.m10/momento.m00);
       y=(int)(momento.m01/momento.m00);

       //Encontra Raio
       step = imagem->widthStep;
       channels = imagem->nChannels;
       data = (uchar *)imagem->imageData;
       raio=0;
       for(j=x;j>=0;j--)
              if(data[y*step+j*channels]==0)
              {      raio=x-j; break; }

       //Imagem sem Punho
       for(i=0;i<(imagem->width);i++) for(j=y+raio;j<(imagem->height);j++)
       {
              data[j*step+i*channels+0]=0;
              data[j*step+i*channels+1]=0;
              data[j*step+i*channels+2]=0;
       }

       //Contorno Certo
       cvCvtColor( imagem, g_gray, CV_BGR2GRAY );
       cvThreshold( g_gray, g_gray, 100, 255, CV_THRESH_BINARY );
       cvFindContours( g_gray, g_storage, &contorno );
```

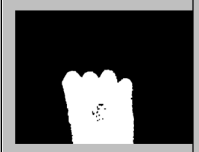
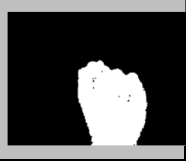
```
	next=contorno->h_next;
	maior=contorno;
	while(next)
	{
		if(next->total > maior->total)
			maior=next;
		next=next->h_next;
	}
	contorno=maior;
	contorno->h_next=NULL;
	contorno->h_prev=NULL;
	contorno->v_next=NULL;
	contorno->v_prev=NULL;

	cvZero( g_gray );
	if( contorno )

	cvDrawContours(g_gray,contorno,cvScalarAll(255),cvScalarAll(255),100);

	int imin=imagem->width, imax=0, jmin=imagem->height, jmax=0;
	step = g_gray->widthStep;
	data = (uchar *)g_gray->imageData;
	for(i=0;i<(g_gray->width);i++)
		for(j=0;j<(g_gray->height);j++)
			if(data[j*step+i]==255)
			{
				if(i<imin) imin=i;
				if(j<jmin) jmin=j;
				if(i>imax) imax=i;
				if(j>jmax) jmax=j;
			}

	//Calcula N Pixels Branco
	step = imagem->widthStep;
	channels = imagem->nChannels;
	data = (uchar *)imagem->imageData;
	for(i=imin;i<imax;i++) for(j=jmin;j<jmax;j++)
		if(data[j*step+i*channels+0]==255)
		{
			N_PIXELS[0]++;
			iL=(int)(((float)(i-imin)/(imax-imin))*PX_LIN);
			jC=(int)(((float)(j-jmin)/(jmax-jmin))*PX_COL);
			N_PIXELS[1+iL+jC*PX_LIN]++;
		}

	//Caracteristicas
	for(i=0;i<N_CARACT;i++)
		Caract[i] = (int) ((float)N_PIXELS[i+1]/((imax-imin)*(jmax-
jmin)/(N_CARACT))*100);


}

void Extrai(void)
{
	//Caracteristicas
	caract();

	// Preenche
	for(k=0;k<N_CARACT;k++)
		Dados[Classe-1][Captura-1][(Tempo-1)*N_CARACT+k]=Caract[k];
```

```
}


int main( int argc, char** argv )
{
	g_storage = cvCreateMemStorage(0);

	//Dados 0
	for(i=0;i<N_CLASSES;i++) for(j=0;j<N_CAPTURAS;j++)
for(k=0;k<N_CARACT*N_CAP_TEMPO;k++)
		Dados[i][j][k]=0;

	//PROCESSOS
	for(Classe=1;Classe<=N_CLASSES;Classe++)
	{
		printf("\n Captura da Classe %d. Aguarde....", Classe);
		for(Captura=1;Captura<=N_CAPTURAS;Captura++)
		for(Tempo=1;Tempo<=N_CAP_TEMPO;Tempo++)
		{
			Extrai();
			//cvWaitKey();
		}
	}

	Treina();

	c=cvWaitKey();

	return 0;
}
```

APÊNDICE B – Classes de Gestos para a Análise da Distância

A tabela B-1 traz as 5 classes utilizadas para a análise da distância entre a mão humana e a câmera.

Tabela B-1 – Classes de Gestos para a Análise da Distância entre a Mão e a Câmera

Classe	Quadro 1	Quadro 2	Quadro 3	Quadro 4	Quadro 5
1					
2					
3					
4					
5					

APÊNDICE C – Classes de Gestos para a Análise do Reconhecimento

A tabela C-1 traz as 25 classes utilizadas para a análise da distância entre a mão humana e a câmera.

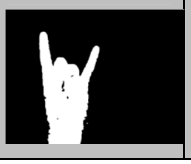
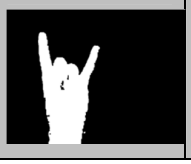
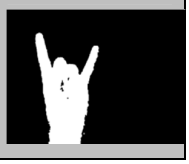
Tabela C-1 – Classes de Gestos para a Análise do Reconhecimento

Classe	Quadro 1	Quadro 2	Quadro 3	Quadro 4	Quadro 5
1					
2					
3					
4					
5					
6					
7					
8					

Tabela C-1 – Classes de Gestos para a Análise do Reconhecimento

Classe	Quadro 1	Quadro 2	Quadro 3	Quadro 4	Quadro 5
9					
10					
11					
12					
13					
14					
15					
16					
17					
18					

Tabela C-1 – Classes de Gestos para a Análise do Reconhecimento

Classe	Quadro 1	Quadro 2	Quadro 3	Quadro 4	Quadro 5
19					
20					
21					
22					
23					
24					
25					

APÊNDICE D – Tabela de Contingências para Análise do Reconhecimento

A tabela D-1 traz as tabela de contingências para a análise de reconhecimento do capítulo 4. A tabela traz a precisão (94,72%) da análise, a falsa aceitação (FPR) e a falsa rejeição (FNR) para cada classe analisada.

Tabela D-1 – Tabela de Contingências para Análise do Reconhecimento

	SAÍDAS								
Ent	**A**	**B**	**C**	**D**	**E**	**F**	**G**	**H**	**I**
A	24	-	-	-	-	-	-	-	-
B	-	25	-	-	-	-	-	-	-
C	-	-	24	-	-	-	-	-	-
D	-	-	-	24	-	-	-	-	-
E	-	-	-	-	25	-	-	-	-
F	-	-	1	-	-	22	-	-	-
G	-	-	-	-	-	-	23	-	-
H	-	-	-	-	-	-	-	25	-
I	-	1	-	-	-	-	-	-	22
J	-	-	-	-	-	-	-	-	1
K	-	-	-	-	-	-	-	-	-
L	-	-	-	-	-	-	-	-	-
M	-	-	-	-	-	-	-	-	-
N	-	-	-	-	-	-	-	-	-
O	-	-	-	-	-	-	-	-	-
P	-	-	-	-	-	-	-	-	-
Q	-	-	-	-	-	1	-	1	-
R	-	-	-	-	-	-	-	-	-
S	-	-	-	-	1	-	-	-	-
T	-	-	-	-	-	-	-	-	1
U	-	-	-	-	-	-	-	-	-
V	-	-	-	-	-	-	-	-	-
W	-	-	-	-	-	-	-	1	-
X	-	-	-	-	-	-	-	-	-
Y	-	-	-	-	-	-	-	-	-
FPR	0%	0.166%	0.166%	0%	0.166%	0.166%	0%	0.332%	0.332%

Tabela 13 – Tabela de Contingências para Análise do Reconhecimento

	SAÍDAS								
Ent	**J**	**K**	**L**	**M**	**N**	**O**	**P**	**Q**	**R**
A	-	-	-	-	-	-	-	-	-
B	-	-	-	-	-	-	-	-	-
C	-	-	-	-	-	-	-	-	-
D	-	-	-	-	-	-	-	-	-
E	-	-	-	-	-	-	-	-	-
F	-	-	-	-	-	-	-	2	-
G	-	1	-	-	-	1	-	-	-
H	-	-	-	-	-	-	-	-	-
I	1	-	-	1	-	-	-	-	-
J	24	-	-	-	-	-	-	-	-
K	-	23	-	-	-	-	1	-	-
L	-	-	24	-	-	-	-	1	-
M	-	-	-	25	-	-	-	-	-
N	-	-	-	1	23	-	-	-	-
O	-	-	-	-	-	24	-	-	-
P	-	-	-	-	-	-	25	-	-
Q	-	-	-	-	-	-	-	23	-
R	-	1	-	-	1	-	-	-	23
S	-	-	-	-	-	-	-	-	-
T	-	1	-	-	-	-	-	-	-
U	-	-	-	-	-	-	1	-	-
V	-	-	-	-	1	-	-	-	-
W	-	-	-	-	-	-	-	-	-
X	-	-	-	-	-	-	-	-	-
Y	-	-	-	-	-	-	100%	-	-
FPR	0.166%	0.518%	0%	0.332%	0.332%	0.166%	0.518%	0.518%	0%

Tabela 13 – Tabela de Contingências para Análise do Reconhecimento

	SAÍDAS							
Ent	**S**	**T**	**U**	**V**	**W**	**X**	**Y**	**FNR**
A	-	-	-	1	-	-	-	4%
B	-	-	-	-	-	-	-	0%
C	-	-	-	1	-	-	-	4%
D	1	-	-	-	-	-	-	4%
E	-	-	-	-	-	-	-	0%
F	-	-	-	-	-	-	-	12%
G	-	-	-	-	-	-	-	8%
H	-	-	-	-	-	-	-	0%
I	-	-	-	-	-	-	-	8%
J	-	-	-	-	-	-	-	4%
K	1	-	-	-	-	-	-	8%
L	-	-	-	-	-	-	-	4%
M	-	-	-	-	-	-	-	0%
N	-	-	1	-	-	-	-	8%
O	-	1	-	-	-	-	-	4%
P	-	-	-	-	-	-	-	0%
Q	-	-	-	-	-	-	-	8%
R	-	-	-	-	-	-	-	8%
S	24	-	-	-	-	-	-	4%
T	-	23	-	-	-	-	-	8%
U	-	-	23	-	1	-	-	8%
V	-	-	-	24	-	-	-	4%
W	1	-	-	-	22	1	-	12%
X	-	-	-	-	-	25	-	0%
Y	-	-	-	-	-	1	23	8%
FPR	0.518%	0.166%	0.166%	0.332%	0.166%	0.332%	0%	**94.72%**