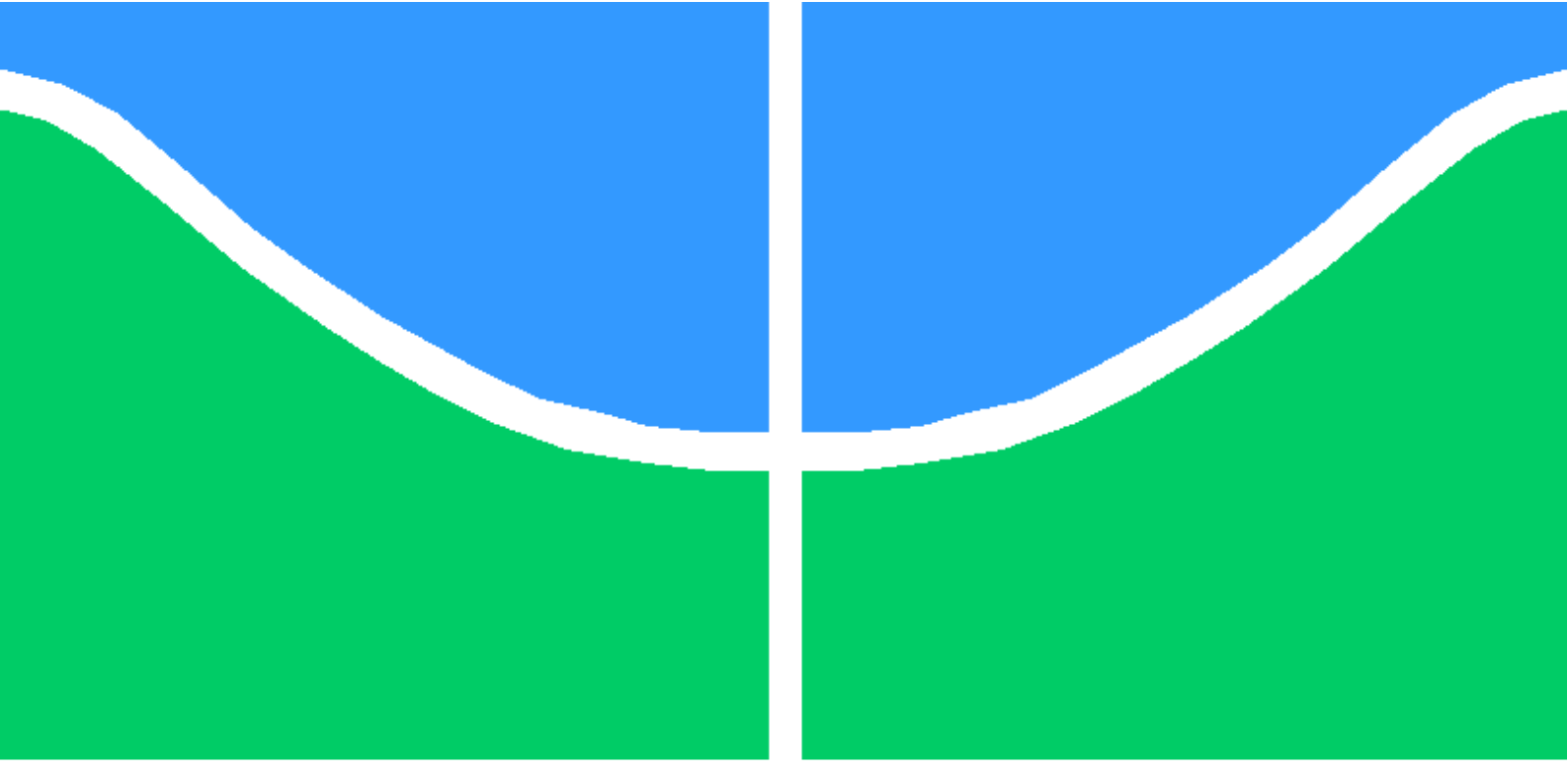


TRABALHO DE GRADUAÇÃO

RECONHECIMENTO ÓPTICO DE MOVIMENTOS E POSTURAS DA MÃO COMO INTERFACE PARA COMPUTADORES

Fernanda Brandi da Silva

Brasília, dezembro de 2006

UNIVERSIDADE DE BRASÍLIA

FACULDADE DE TECNOLOGIA

UNIVERSIDADE DE BRASÍLIA
Faculdade de Tecnologia

TRABALHO DE GRADUAÇÃO

RECONHECIMENTO ÓPTICO DE MOVIMENTOS E POSTURAS DA MÃO COMO INTERFACE PARA COMPUTADORES

Fernanda Brandi da Silva

Relatório submetido como requisito parcial para obtenção
do grau de Engenheiro Eletricista

Banca Examinadora

Prof. Ricardo Lopes de Queiroz, Ph.D., UnB/ENE - Orientador _____

Prof. Alexandre Zaghetto, Mestre, UnB/ENE - Examinador _____

Prof. Geovany Araújo Borges, Docteur, UnB/ENE - Examinador _____

FICHA CATALOGRÁFICA

DA SILVA, FERNANDA BRANDI
Reconhecimento Óptico de Movimentos e Posturas de Mão como Interface para Computadores. [Distrito Federal] 2006.
77p. (ENE/FT/UnB, Engenheiro Eletricista, 2006)
Monografia de Graduação - Universidade de Brasília.
Faculdade de Tecnologia.
Departamento de Engenharia Elétrica.

1. Visão Computacional	2. Estimação de Movimento
3. Reconhecimento de Cores	4. Rastreamento de Objetos em Imagens
5. Interação Humano-Computador	6. Reconhecimento de Gestos
I. ENE/FT/UnB	II. Título (série)

REFERÊNCIA BIBLIOGRÁFICA

DA SILVA, F. B. (2006). Reconhecimento Óptico de Movimentos e Posturas de Mão como Interface para Computadores. Monografia de Graduação, Publicação ENE 02/2006, Departamento de Engenharia Elétrica, Universidade de Brasília, Brasília, DF, 76p.

CESSÃO DE DIREITOS

NOME DO AUTOR: Fernanda Brandi da Silva

TÍTULO: Reconhecimento Óptico de Movimentos e Posturas de Mão como Interface para Computadores.

GRAU / ANO: Engenheiro Eletricista / 2006

Fernanda Brandi da Silva
SQSW 300 - Bloco O - Apartamento 411 - Sudoeste
70673-052 - Brasília - DF - Brasil.

Dedicatória

Dedico o presente trabalho a meus pais e irmãs que principalmente em momentos difíceis, dando broncas, conselhos, ou me deixando descobrir meus próprios rumos, sempre me deram suporte para seguir em frente nessa caminhada tortuosa.

Fernanda Brandi da Silva

Agradecimentos

À minha mãe, Valéria, que desde sempre respondia minhas milhares de perguntas com a maior paciência, exceto quando eu abusava da boa vontade dela na hora de dormir. Ela que nunca repreendeu seriamente meu espírito curioso e questionador que desde criança tomou conta de mim e que acredito, de algum modo, me levou para o caminho da engenharia.

Ao meu pai, Ruy César, que mesmo com seu jeito reservado e mais distante sempre me ajudou especialmente nas situações mais sérias e decisivas. Que ele continue sendo a pessoa responsável e justa que sempre foi, mas, torço eu, que ele possa curtir mais a vida e desista de ver aqueles filmes de gosto duvidoso para que possamos conversar também sobre outras amenidades.

Às minhas irmãs, Patrícia e Cristiane, que em todo momento sempre foram parte essencial da família e que sempre me apresentaram pontos de vista diferentes para as questões da vida. Que elas alcancem felicidades e realizações na vida acadêmica, profissional e pessoal tanto quanto eu acho que muito merecem.

Aos meus grandes amigos da UnB que espero levar para vida... Chico, Bianchi, Roques, Sasaki, Ana Ravena, Izumi, Clara, Lulis, Sumaia, Otávio, Pombo e tantos outros que sempre estiveram presentes e unidos nessa vida acadêmica. Agradeço em especial ao Chico e Bianchi por muitas madrugadas quase viradas de muito trabalho e divertimento no laboratório, pessoas que nunca me deixaram esquecer que quando gostamos do que fazemos todo trabalho é prazeroso.

Ao pessoal do GPDS e do grupo Image pelos ensinamentos sempre valiosos... Mintsu, Capim, Alves, Zaghetto, Grilo, Karen, Buraco, Ortiz, por propiciarem um ambiente de estudo e pesquisa tão agradável, eclético e enriquecedor. Uma menção especial ao meu orientador, prof. Queiroz, por ter me convidado e me incentivado a trabalhar com ele em momento tão oportuno. Ao Mintsu um agradecimento especial por muitas conversas, ensinamentos, idéias e madrugadas de companheirismo e trabalho, uma real fonte zen de inspiração.

Aos professores que serviram de inspiração por mostrarem gosto e talento na transmissão de conhecimentos e que de fato poderiam ser chamados de mestres. A eles minha sincera admiração por seguirem uma profissão por vezes tão desgastante e sem o devido reconhecimento que tanto lhes é de direito.

Aos amigos distantes que mesmo com a distância geográfica não são menos amigos. Em especial, Raquel Portugal, uma amiga que descobri quase que acidentalmente e que desejo muito bem. Obrigada pelas muitas horas de conversa e ombro amigo. E, por último mas nunca menos especial, Maíra Lioi, pela segunda chance de uma amizade que espero que perdure por toda a vida. Mesmo.

Fernanda Brandi da Silva

RESUMO

O presente trabalho apresenta uma alternativa viável de interface entre usuário e computador. Esta interface, baseda em imagens, tem o objetivo de reconhecer a mão, seus movimentos e posturas em imagens dinâmicas possibilitando, em tempo real, o controle do *mouse*. Foi idealizado que esta interface visual possuísse plena funcionalidade sem artifícios ou sensores externos, sendo necessário somente uma simples *webcam* e o programa aqui projetado.

ABSTRACT

The present work presents a viable alternative for an interface between user and computer. This interface, based on images, has the goal of recognizing the hand, its motion and gestures in dynamic images making mouse control possible, in real time. This visual interface was idealized to be fully functional without any other gadget or external sensors, it is only necessary a simple webcam and the program developed here.

SUMÁRIO

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SÍMBOLOS

Siglas

SAD	*sum of absolute differences*
RGB	*red, green, blue*
YCbCr	*luminance, blue chrominance, red chrominance*
HSI	*hue, saturation, intensity*
HSL	*hue, saturation, lightness*
HSV	*hue, saturation, value*
DP	desvio padrão

Unidades

nm	nanômetro	10^{-9} metro
THz	terahertz	10^{12} hertz
ms	milisegundo	10^{-3} segundo

1 INTRODUÇÃO

Porque pequenas caminhadas ou grandes jornadas começam sempre com o primeiro passo.

1.1 HISTÓRIA DA INTERAÇÃO HUMANO-MÁQUINA

Desde o surgimento das primeiras máquinas se procurou sempre diversificar e aprimorar os modos de o usuário interagir com esses aparelhos. Os propósitos dessa diversificação iam desde possibilitar a imposição de novos comandos à máquina quanto facilitar a vida de quem manejava esses instrumentos.

Quando o computador começou a ser utilizado pelo público em geral não foi diferente. Primeiramente, o meio mais comum de se entrar dados em um computador era o teclado. Como a interface gráfica ainda não era utilizada, o teclado supria toda a necessidade dessa interação humano-computador.

Com o passar dos anos, novas tecnologias de *software* foram surgindo e com elas a interface gráfica. Junto a essa evolução foi sentida a precisão de um instrumento de ”navegação” mais intuitivo. Era então criado o *mouse*, periférico capaz de traduzir movimentos mecânicos em movimento de um ponteiro na tela, e cliques em alguma ação para os programas.

Desde então, os possíveis modos de se interagir com o computador não têm sofrido grandes alterações. Algumas alternativas já foram apresentadas porém poucas delas tiveram algum apelo significativo junto ao público.

A Figura 1.1 expõe justamente alguns dos periféricos mais comuns atualmente. O teclado e o *mouse*, como dito anteriormente, são os grandes dominantes no quesito popularidade e robustez. O microfone e a câmera foram introduzidos no mercado porém ainda estão longe de serem largamente vistos como alternativa para essa interação homem-máquina.

Nos últimos anos, vários estudos estão sendo realizados em diversos institutos e universidades do mundo para mudar esse quadro. Há pesquisas que desenvolvem programas para reconhecimento e entendimento de comandos via voz, ou ainda se empenham na utilização de interfaces que possam distinguir e interpretar objetos ou ações em imagens capturadas por uma *webcam*.

No que concerne às pesquisas em relação a imagens, há tentativas de interação homem-computador

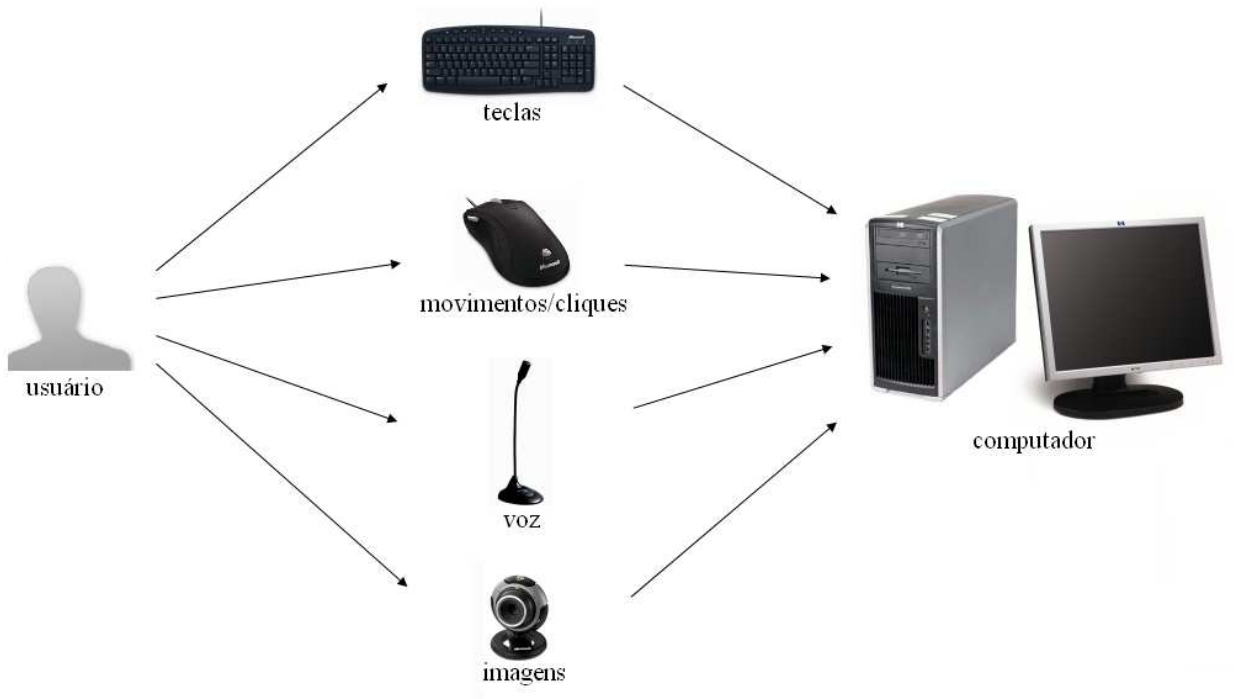


Figura 1.1: Periféricos de interação humano-computador

via reconhecimento e acompanhamento (*tracking*) da cabeça[1], mãos[2], olhar[3], objetos, pessoas, etc. O campo da visão computacional tem larga aplicabilidade podendo ser utilizada desde interface para jogos até ferramenta de auxílio a pessoas com alguma dificuldade motora.

Pessoas que porventura não tenham plena mobilidade dos membros superiores e inferiores poderiam, por exemplo, fazer uso de um computador ”navegando” com a cabeça ou até mesmo pelo acompanhamento do olhar.

Outro exemplo de utilização de uma interface visual seria no caso de necessidade de proteção da máquina na existência de um computador em ambiente público. Há a possibilidade do uso desse computador sem ter proximidade física com ele e, consequentemente, sem qualquer tipo de exposição de periféricos.

É possível citar outros empregos como ambientes de realidade virtual onde seria possível interagir com o cenário ou objetos em cena através de movimentos e ações do usuário sem a necessidade de artifícios como *mouse*, luvas ou sensores.

Finalmente, mais uma utilização dentre inúmeras outras, seria ainda o uso de imagens como interface

[1]Demonstração disponível em http://www.hhi.fraunhofer.de/english/im/products/headtracking/index.html
[2]Demonstração disponível em http://www.hhi.fraunhofer.de/english/im/products/handtracking/index.html
[3]Demonstração disponível em http://www.hhi.fraunhofer.de/english/im/products/gazetracking/index.html

para jogos eletrônicos[4] em que se mostre imperativo de alguma forma a movimentação corporal real do jogador para que tais movimentos sejam traduzidos para o aplicativo como jogadas.

1.2 MOTIVAÇÃO E OBJETIVOS

Primeiramente, a motivação deste projeto foi dar continuidade a um trabalho anterior [1] de discentes do curso de graduação em Engenharia Elétrica[5] na Universidade de Brasília[6]. Este trabalho prévio gerou um sistema de interface visual capaz de determinar o movimento resultante a cada quadro capturado por uma câmera e com isso mover o ponteiro do *mouse* na tela. Foi obtido também na interface supracitada a associação de movimentos em certas regiões da imagem para acionamento de algumas teclas.

A partir dessa interface visual e aspirando a essas novas tecnologias de visão computacional, o objetivo desse projeto é tentar melhorar o programa de modo que fosse possível distinguir entre os diversos objetos que aparecessem na imagem e, posteriormente, focar a atenção em certos objetos de acordo com o interesse do usuário.

A idéia, nesse caso, é atingir um bom grau de reconhecimento de cor de pele de modo que seja possível reconhecer e seguir a mão e, posteriormente, identificar e ainda interpretar certos gestos para que se possa clicar os vários cliques do *mouse* dependendo da postura manual que o usuário assumir.

A proposta ainda é não ser necessário nenhum tipo de artifício externo como luvas ou algum tipo de sensor que auxilie na localização da mão quando se analisa a imagem. Outro desafio é trabalhar o mais perto possível do tempo real, ou seja, um sistema de respostas imediatas.

1.3 POSSÍVEIS DIFICULDADES E POSSÍVEIS SOLUÇÕES

Definida então a proposta do projeto, serão agora explanadas algumas das dificuldades previstas que possam ser encontradas no decorrer do trabalho.

Um primeiro fato a se observar se trata justamente da essência do projeto e da visão computacional em geral. Como é possível impor algum tipo de inteligência ao programa para que ele seja capaz de discernir os diferentes objetos em cena? E mais adiante, com que critério de seleção é razoável optar por certo

[4] Mais informações disponíveis em http://www.eyetoy.com/index.asp?pageID=18

[5] Endereço eletrônico oficial: http://www.ene.unb.br/

[6] Endereço eletrônico oficial: http://www.unb.br/

objeto, preterindo outro?

O maior problema que se pode constatar inicialmente é a questão de que uma imagem capturada por uma câmera ser somente um aglomerado de dados matemáticos com valores que representam informações espaciais e ponderações entre mínimos e máximos de uma ou outra cor.

Nesse momento, o desafio envolve observar essa matriz de dados numéricos e distinguir de algum modo diferentes características na imagem. Algumas soluções pensadas para resolver essa questão se baseiam em algoritmos de separação de elementos baseada em padrões (cores, movimentos...), detecção de bordas, classificação por separação espacial entre os diversos movimentos, etc.

Algumas dessas idéias são postas em prática no projeto e seus resultados, dificuldades e sucessos expostos ao longo deste trabalho, principalmente no Capítulo 4.

Outra dificuldade que se pode enxergar agora é que mesmo que se consiga separar os objetos na imagem, como então escolher entre um ou outro? Qual critério utilizar? Como foi dito anteriormente, o computador não faz distinção entre os elementos da imagem a menos que forneçam mais informações sobre o que se queira ou não queira.

Uma saída idealizada para essa questão envolve classificar como válidos ou inválidos os objetos de acordo com cores de interesse, como cor de pele ou alguma outra cor a escolha. Novamente, algumas dessas soluções são apresentadas no decorrer do presente trabalho.

Com o desenvolvimento do projeto novas dificuldades foram surgindo e outras soluções se apresentando. Os Capítulos 2 e 3 fazem uma introdução teórica sobre os assuntos abordados mais adiante. Todo o desenvolvimento do projeto foi retratado com mais detalhes no Capítulo 4. O Capítulo 5 expõe as conclusões, dificuldades de fato encontradas e propostas para trabalhos futuros.

2 MOVIMENTO

Movimento é o termo utilizado para descrever um corpo que, no tempo, altera sua posição em relação a um referencial espacial.

2.1 ESTIMAÇÃO DE MOVIMENTO

Estimação de movimento é o processo realizado para encontrar os movimentos resultantes que ocorrem entre pelo menos dois quadros. Esse processo consiste em procurar o lugar mais provável onde um objeto presente num quadro (geralmente chamado atual) se localizava em outro quadro (anterior).

Essa procura poderia ser feita pixel a pixel porém o processamento envolvido seria extremamente custoso. Uma alternativa para essa limitação é a divisão de cada quadro em macroblocos, que nada mais são que conjuntos de NxN pixels, onde N é um número inteiro. Multiplicando N^2 pelo número de macroblocos da imagem, o resultado é exatamente o número total de pixels da mesma.

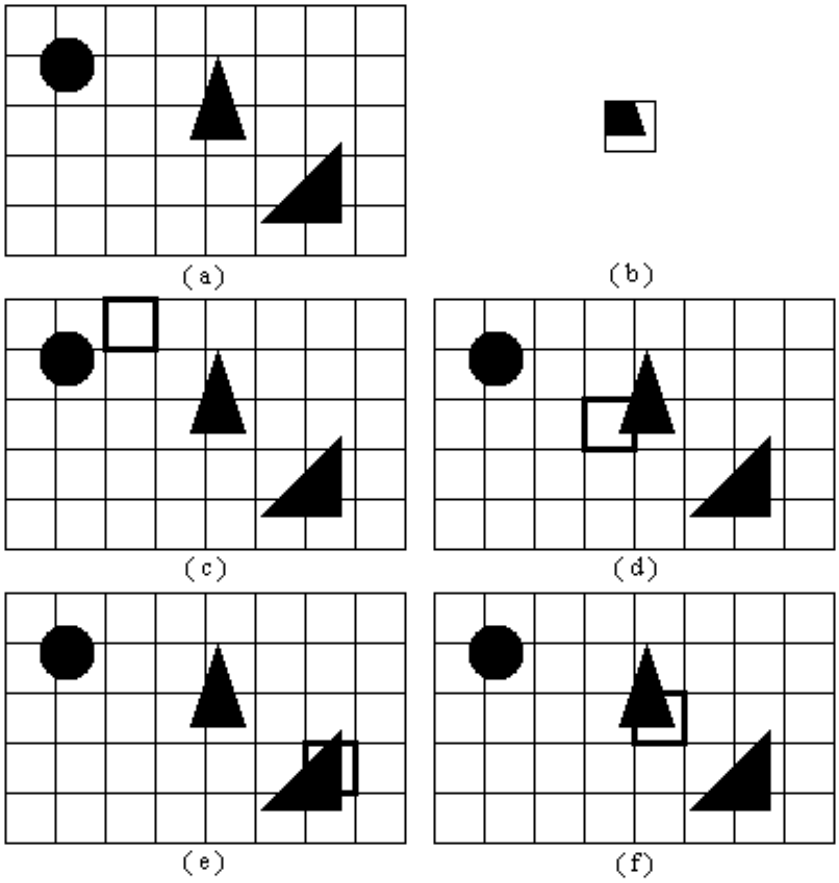


Figura 2.1: Exemplo de procura por macrobloco em uma imagem

Com essa grade de macroblocos formada então trabalha-se em nível de conjuntos de pixels e não mais pixels individuais. As procuras são realizadas analisando macroblocos e sendo geralmente o critério de parada a melhor combinação encontrada no macrobloco central. Um exemplo de procura em macroblocos é citada na Figura 2.1.

A Figura 2.1(a) representa um quadro já dividido em macroblocos. O macrobloco destacado na Figura 2.1(b) é o que se quer encontrar na imagem. As Figuras 2.1(a-c) representam más combinações, ou seja, esses macroblocos não se assemelham ao macrobloco em 2.1(b). Por fim, a Figura 2.1(f) é uma boa combinação, coincidindo toda ou maior parte da informação contida no macrobloco procurado.

Porém, esse último exemplo mostrou a busca dentro de uma mesma imagem sem movimento. Neste momento será mostrado como é feita uma procura entre dois quadros diferentes.

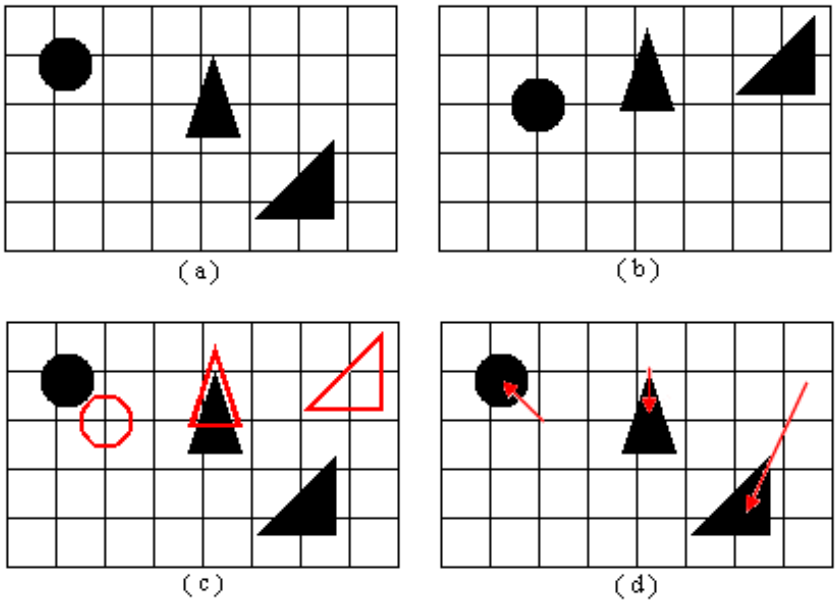


Figura 2.2: Exemplo de estimação de movimento

A Figura 2.2(a) representa o quadro atual, já na 2.2(b) é mostrado o quadro anterior. Percebe-se que os objetos de 2.2(b) se deslocaram de tal modo que estão em disposições diferentes em relação a 2.2(a). A Figura 2.2(c) indica essa translação dos macroblocos entre os dois quadros e em 2.2(d) seus respectivos vetores resultantes de movimento.

2.2 SOMA DAS DIFERENÇAS ABSOLUTAS

A soma das diferenças absolutas, ou SAD (em inglês, *Sum of Absolute Differences*) como é também conhecida, é um critério largamente utilizado em estimação de movimento. Este processo é de suma importância e se baseia na soma das diferenças de valores de pixels. Pode-se realizar essa subtração tanto em se tratando de procura pixel a pixel quanto na questão de procura de macroblocos. A idéia da SAD é exemplificada na Figura 2.3.

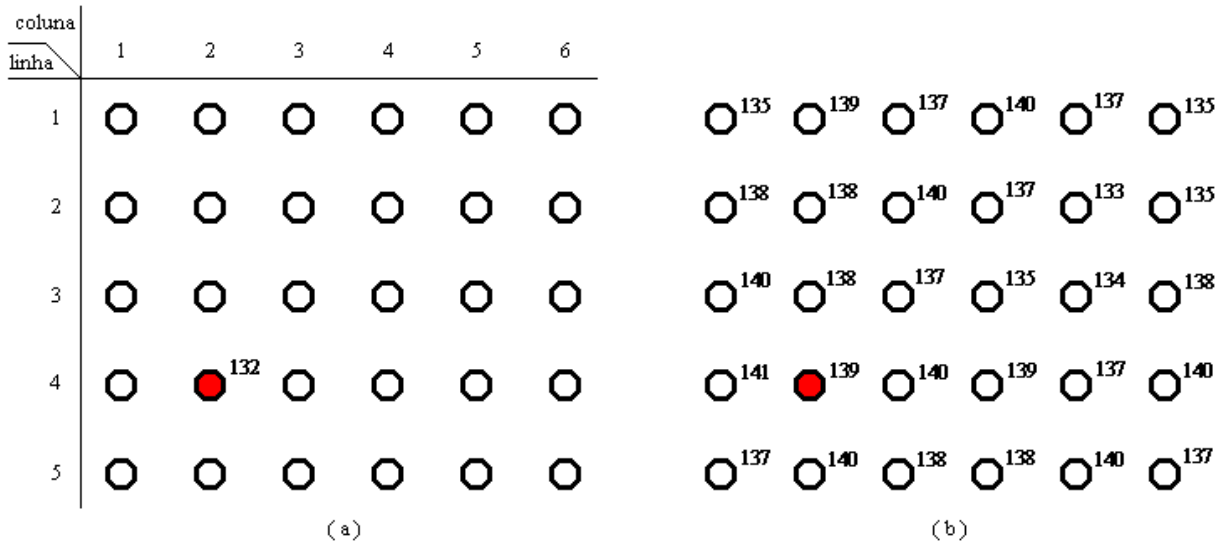


Figura 2.3: Exemplo de dois quadros diferentes, atual e anterior, respectivamente

A Figura 2.3(a) representa a grade da imagem atual onde cada um desses espaços equivale a um pixel. O mesmo ocorre da Figura 2.3(b) porém para o quadro anterior. O pixel destacado da imagem atual corresponde ao pixel que se quer encontrar na imagem anterior. Para isso, é iniciada a procura exatamente nas mesmas coordenadas na imagem anterior, ou seja, linha 4, coluna 2.

Para calcular a diferença entre os pixels basta subtrair o valor do pixel que se quer procurar, no caso o pixel atual tem valor 132, e o pixel correspondente no quadro anterior. Ou seja:

$$\Delta(x, y) = |f_{atual}(x, y) - f_{anterior}(x + m, y + n)| \tag{2.1}$$

Em que *m* e *n* representam os deslocamentos, em pixels, da procura no quadro anterior.

Os tipos de procura serão mais detalhados na Seção 2.3 porém iremos nos ater aqui a um dos procedimentos. Esse algoritmo de procura consiste então em subtrair o pixel anterior do pixel atual e repetir o procedimento para os vizinhos do pixel anterior (vizinhança de quadro, N_4, nesse caso). O deslocamento da procura no quadro anterior segue a menor diferença encontrada na vizinhança mais próxima. O critério

de parada dessa procura é quando a menor diferença entre o pixel atual e o pixel anterior se encontra no pixel central da respectiva vizinhança. Um exemplo dessa procura é mostrado na Figura 2.4.

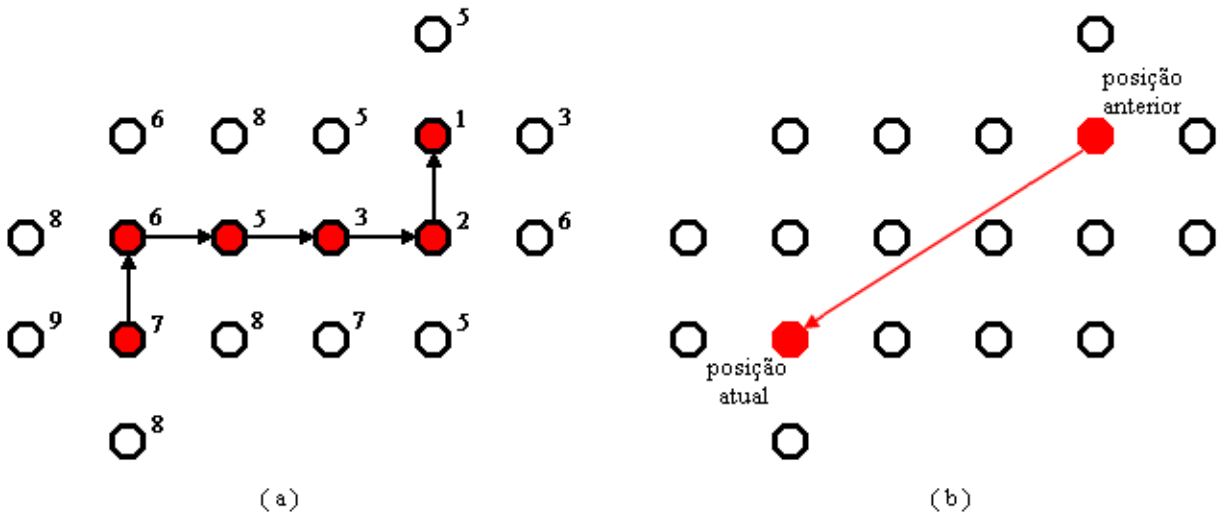


Figura 2.4: Exemplo da procura por pixel entre dois quadros

Pode-se observar que os números ao lado dos pixels da Figura 2.4(a) representam a diferença entre o valor do pixel atual e o respectivo pixel do quadro anterior. O deslocamento da procura segue sempre para o lugar onde, na vizinhança, houve a menor diferença como é mostrada nessa Figura. Vemos que o critério de parada é obedecido quando a menor diferença entre os pixels se localiza no pixel central da vizinhança, onde no caso exposto ocorre o valor 1 (central) e seus vizinhos têm diferenças de valor 2, 3 e 5.

O vetor resultante desse deslocamento é explicitado na Figura 2.4(b). Observa-se que o vetor tem sentido contrário ao que seria deslocamento final mostrado na Figura 2.4(a) porém isso é justificado pela lógica de que procuramos em 2.4(a) de onde o pixel veio no quadro anterior e não para onde ele foi nesse mesmo quadro.

Para procurar por macroblocos ao invés de pixels a lógica é semelhante, é só lembrar que macroblocos são conjuntos de NxN pixels. Observe a Figura 2.5.

A Figura 2.5(a) representa o quadro atual e 2.5(b) o anterior. Observe que o objeto contido no macrobloco em 2.5(a) foi deslocado a partir da posição de onde estava em 2.5(b). Para encontrar esse vetor de movimento é necessário subtrair cada pixel do macrobloco atual (Figura 2.6(a)) pelo respectivo pixel do macrobloco do quadro anterior (Figura 2.6(b)). Porém, vale observar que para recalcular as diferenças na vizinhança, os deslocamentos realizados do macrobloco anterior - no caso de N_4 - são de um pixel para esquerda (Figura 2.6(c)), um para cima (Figura 2.6(d)), um para a direita (Figura 2.6(e)) e um para baixo (Figura 2.6(f)) e não de um macrobloco para baixo, um para cima e assim por diante.

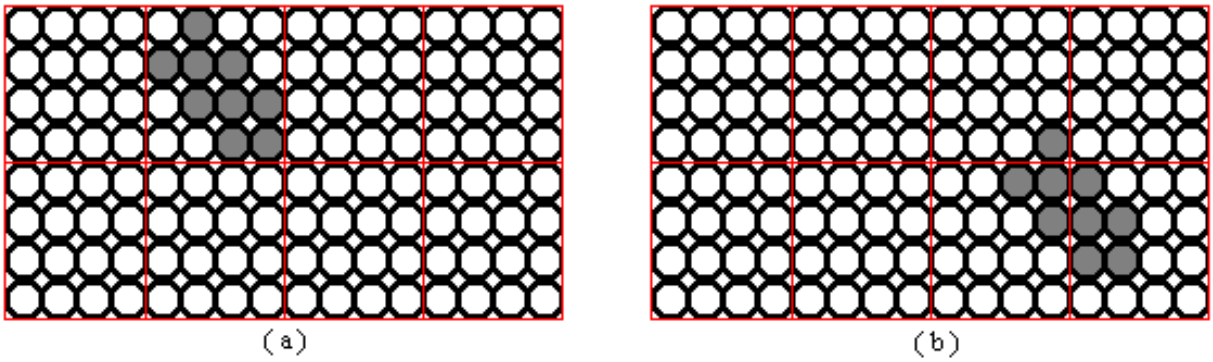


Figura 2.5: Exemplo de dois quadros explicitando os pixels e macroblocos

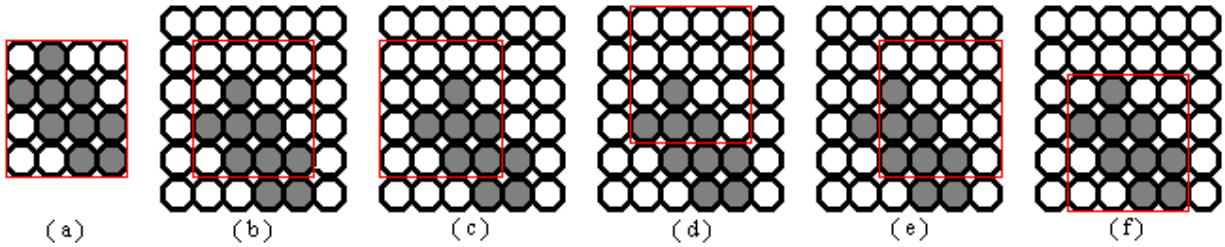


Figura 2.6: Procura de macroblocos em vizinhança N_4

No exemplo da Figura 2.6 a melhor combinação entre o macrobloco atual procurado e os macroblocos verificados no quadro anterior é o indicado na Figura 2.6(f).

Ao contrário da procura pixel a pixel onde o próprio resultado da subtração representava já o valor a se comparar com os valores vizinhos, na busca por macroblocos se deve somar todos os módulos das N^2 diferenças (dentro do macrobloco) de modo que ao final do procedimento dentro de cada macrobloco se obtenha somente um valor representando todas as diferenças, chamado SAD. Ao se comparar a SAD de cada macrobloco e de seus vizinhos é que se toma a decisão de seguir a menor delas para prosseguir com a busca.

$$SAD(x,y) = \sum_{m=0}^{N-1}\sum_{n=0}^{N-1} \Delta(x+m, y+n) \tag{2.2}$$

Ao final da procura é calculado o vetor de movimento resultante do macrobloco subtraindo as posições final (quadro atual) e inicial (quadro anterior). Desse modo é encontrado tanto o deslocamento (linha e coluna) em x quanto em y.

$$\delta_l(x,y) = l_{atual}(x,y) - l_{anterior}(x+m, y+n) \tag{2.3}$$

$$\delta_c(x,y) = c_{atual}(x,y) - c_{anterior}(x+m, y+n) \tag{2.4}$$

Onde, novamente, *m* representa o número de pixels verticais (linhas) e *n* representa o número de

pixels horizontais (colunas) deslocados até se encontrar a melhor combinação entre os macroblocos atual e anterior.

2.3 TIPOS DE BUSCA

Para estimar movimento é possível utilizar diversas técnicas de procura. Nessa Seção serão apontadas algumas delas indicando superficialmente as diferenças, o processamento envolvido e a eficiência de cada uma delas.

2.3.1 Busca Completa

A busca completa (*full search*, em inglês) consiste em, como o próprio nome já induz, procurar na imagem inteira. Esse tipo de busca demanda muito tempo e processamento pois cada macrobloco é procurado da imagem inteira para somente depois se utilizar o critério de escolha da menor SAD.

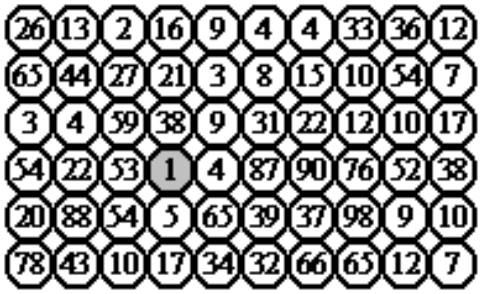


Figura 2.7: Exemplo de busca completa

Na Figura 2.7 são mostrados todos os valores de SAD da imagem anterior em relação ao pixel procurado no quadro atual. Após todos os cálculos é que se determina que a melhor combinação seja aquela que tem menor SAD dentre todas as outras (valor 1, destacado na Figura 2.7).

Com intuito de diminuir o esforço computacional de se buscar um quadro inteiro, é utilizado o artifício de se procurar somente dentro de uma janela e a partir dos resultados da SAD dentro desse intervalo é que então se decide qual a melhor combinação. Essa janela geralmente é limitada a N (ou N/2) pixels para cima, para baixo e para os lados, onde N é o número de pixels em cada linha e coluna de um macrobloco.

Para exemplificar a quantidade de procuras realizadas numa busca completa imaginamos uma janela de 16 pixels de largura por 16 de altura. É considerado que a janela é do tamanho da própria imagem e que a procura será feita buscando pixels (e não macroblocos). O número mínimo de SADs para cada pixel

procurado é então:

$$Quant_{min,SADs,completa} = 16\ linhas \cdot 16\ colunas = 256\ iteracoes \tag{2.5}$$

Existe maior eficiência nesse tipo de busca pois são testadas todas as possibilidades para a partir daí tomar uma decisão de qual a melhor combinação. O custo de ter em mãos essa eficiência é o alto processamento envolvido que dependendo da aplicação, torna inviável sua utilização.

2.3.2 Busca Espiral

A busca espiral é realizada de forma que se procura primeiramente no pixel central e em sua vizinhança dos oito pixels mais próximos, N_8 (Figura 2.8(a)). Se a menor SAD se encontrar no pixel central então o processo acaba, do contrário parte-se a procurar no quadrado externo à N_8, como mostrado na Figura 2.8(b)).

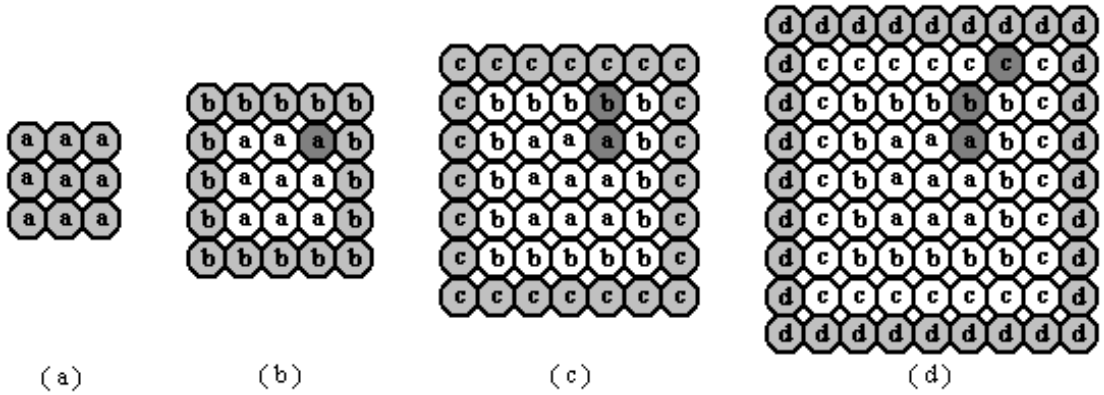


Figura 2.8: Exemplo de busca espiral

Observe que a cada nova busca, apesar de ser calculado somente as SADs do quadrado exterior ao último, o número de iterações vai crescendo à medida que se afasta do pixel central. A busca é terminada quando o quadrado exterior não possuir uma SAD menor que a menor SAD do quadrado imediatamente interior a ele.

A Figura 2.9 mostra o resultado final das iterações e o vetor resultante do deslocamento do pixel.

O número mínimo de iterações que é possível fazer em um exemplo qualquer seria a procura no pixel central e em sua vizinhança de oito pixels, no melhor dos casos onde a menor SAD é a do pixel central, ou seja:

$$Quant_{min,SADs,espiral} = 9\ iteracoes \tag{2.6}$$

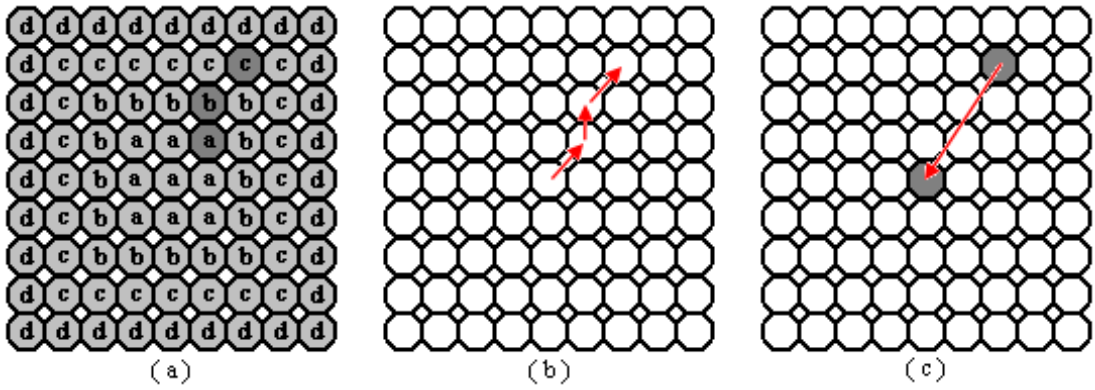


Figura 2.9: Resultado da busca espiral

2.3.3 Busca Circular

A busca circular é muito similar à busca espiral, a única diferença são os vizinhos procurados quando se vai afastando do pixel central. Em vez de abrir a procura com quadrados em volta do pixel central, a forma de procura se assemelha a círculos com centro no pixel central. A Figura 2.10 ilustra esse tipo de busca.

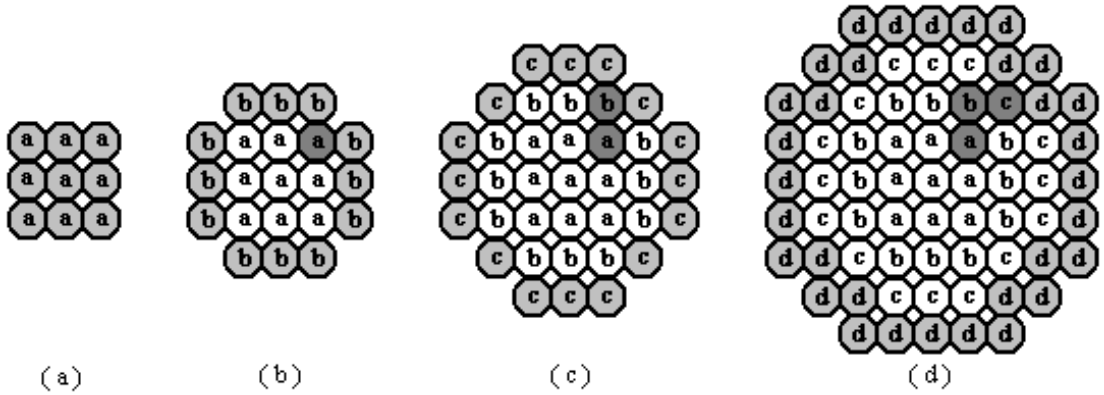


Figura 2.10: Exemplo de busca circular

Semelhante à busca espiral, o número de iterações vai crescendo quando se afasta do pixel central e o critério de parada é quando não houver SAD menor no círculo mais externo em relação à menor SAD do círculo imediatamente interior a ele.

A Figura 2.11 mostra o resultado final das iterações e o vetor resultante do deslocamento do pixel entre dois quadros.

A quantidade mínima de iterações é também semelhante ao método da busca em espiral. A diferenciação se dá ao crescer o raio de procura quando a busca circular tende a buscar menos pixels

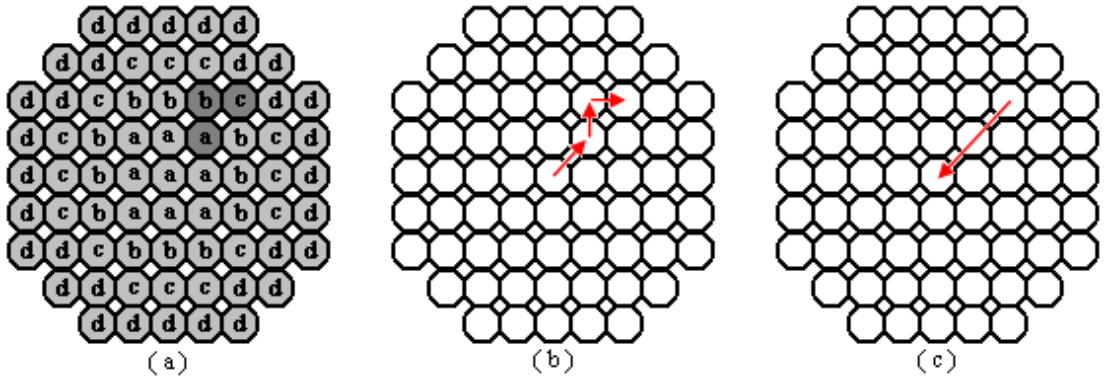


Figura 2.11: Resultado da busca circular

em relação à busca espiral. De qualquer forma, no melhor dos casos o número de buscas mínimo é:

$$Quant_{min,SADs,circular} = 9\ iteracoes \tag{2.7}$$

2.3.4 Busca Telescópica

A busca telescópica consiste em procurar na vizinhança mais distante dentro de uma janela pré-determinada. Este método tem um número único de iterações (quando definido o tamanho da janela) pois vai sempre convergindo para um pixel como mostrado na Figura 2.12.

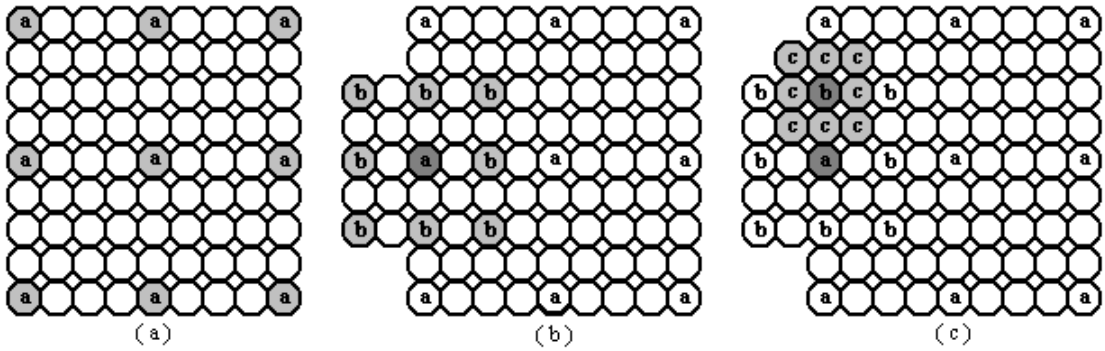


Figura 2.12: Exemplo de busca telescópica

A cada nova iteração são calculados sempre oito novas SADs pois a do pixel central após cada deslocamento já foi previamente determinada. O critério de parada, como dito anteriormente, é sempre o final do processo de iterações (convergência). No exemplo da Figura 2.12 se tem um máximo de três deslocamentos. A Figura 2.13 mostra o resultado final das iterações e o vetor resultante do processo.

O número mínimo de iterações não depende de o pixel central ser o candidato com menor SAD pois de qualquer forma se deve completar o ciclo de iterações. Então, para o caso onde há no máximo três

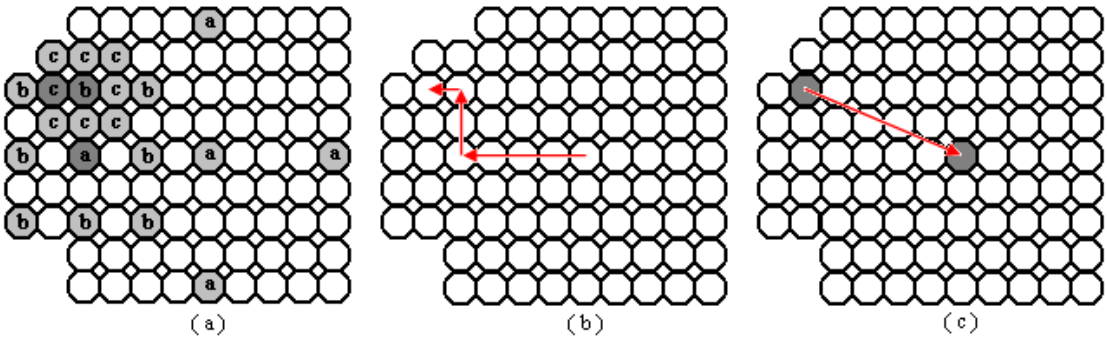


Figura 2.13: Resultado da busca telescópica

deslocamentos (janela de 15x15 pixels):

$$Quant_{min,SADs,telescopica} = 25\ iteracoes \tag{2.8}$$

A vantagem desse método é que se torna possível realizar buscas mais amplas com relativamente um número menor de iterações em relação aos outros métodos aqui apresentados.

2.3.5 Busca Hexagonal

A busca hexagonal consiste em fazer a SAD no pixel central e mais seis vizinhos. O intuito é fazer menos procuras tentando sempre manter a eficiência do processo. A Figura 2.14 mostra o estilo de busca desse método.

Observe que a cada nova iteração é feito somente mais três cálculos de SAD pois as outras quatro já foram previamente calculadas. Ao se chegar à situação onde a SAD do pixel central é a menor, calcula-se a SAD da vizinhança N_4. Essa última iteração define então a melhor combinação entre o pixel anterior e o atual.

A Figura 2.15 mostra o resultado final das iterações e o vetor resultante do deslocamento do pixel.

Realizando agora a contagem do número de cálculos de SAD para um exemplo qualquer, onde, na melhor das hipóteses a menor SAD seria a do pixel central logo na primeira iteração. O número de procuras seria:

$$Quant_{min,SADs,hexagonal} = 7\ iteracoes \tag{2.9}$$

Percebe-se a diminuição considerável no número de buscas em relação às buscas anteriores.

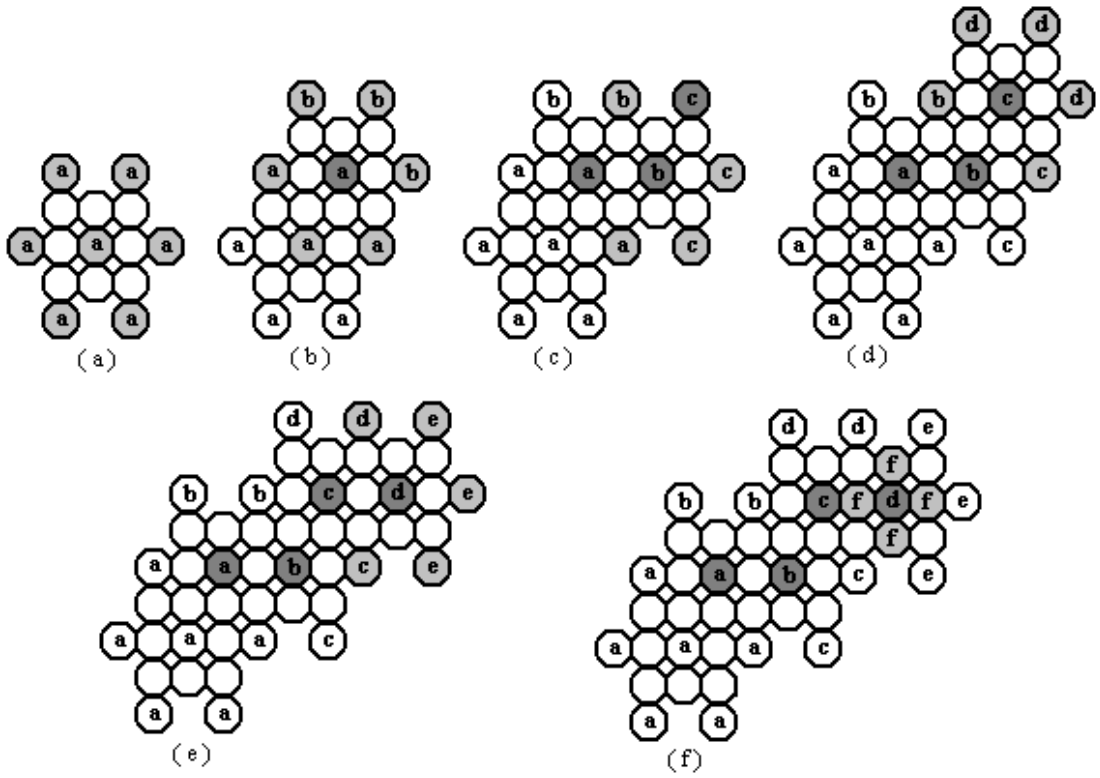


Figura 2.14: Exemplo de busca hexagonal

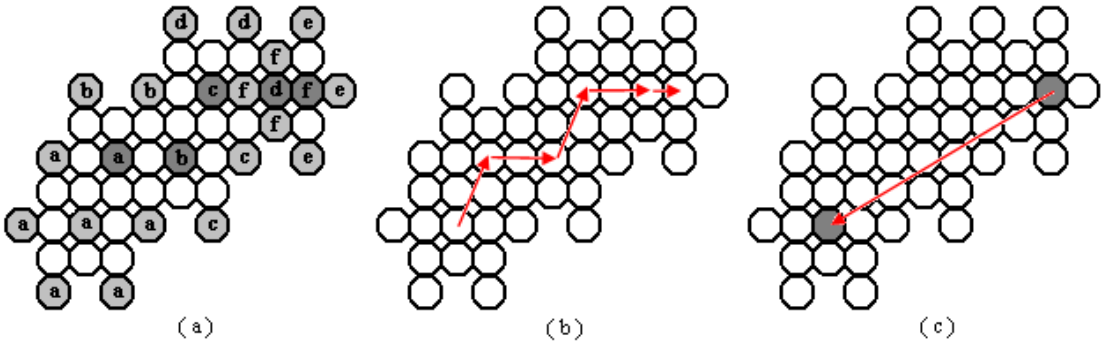


Figura 2.15: Resultado da busca hexagonal

2.3.6 Busca Diamante

A busca diamante é similar à busca hexagonal porém com uma vizinhança de procura ainda menor. Esse método é mostrado na Figura 2.16.

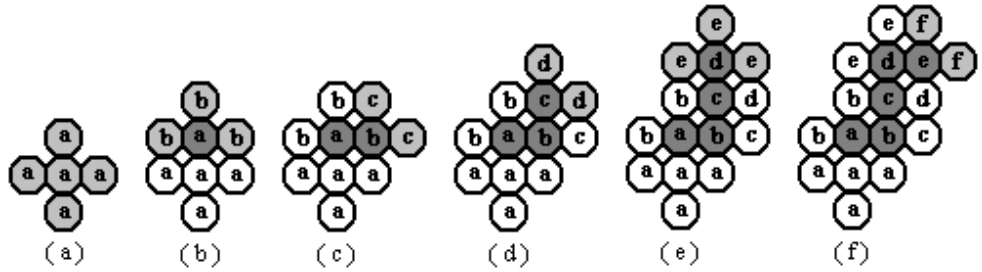


Figura 2.16: Exemplo de busca diamante

Em cada nova iteração são calculadas somente mais duas ou no máximo três SADs pois as outras já foram previamente determinadas. O critério de parada do método é a SAD do pixel central ser a menor entre ela e seus quatro vizinhos (vizinhança N_4). O resultado desse tipo de busca é explicitado na Figura 2.17.

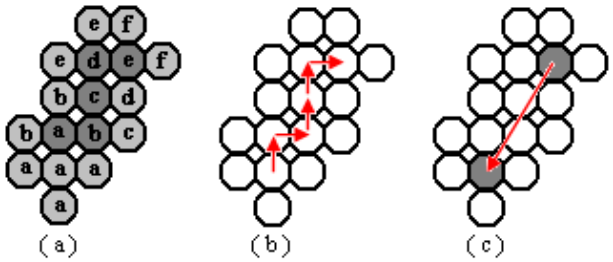


Figura 2.17: Resultado da busca diamante

O número mínimo de iterações nesse tipo de busca acontece quando o pixel central é a melhor combinação, ou seja, a menor SAD ali se encontra. Esse número é:

$$Quant_{min,SADs,diamante} = 5\ iteracoes \tag{2.10}$$

Dependendo da aplicação da busca, o método escolhido pode variar pois em alguns casos pode se ter o benefício de uma melhor procura em detrimento do tempo de processamento, seria então o caso de se utilizar uma busca completa que é ótima. Já há situações em que se é necessária uma busca mais leve e localizada, sendo esse o caso é recomendado escolher alguma das buscas sub-ótimas como a hexagonal ou diamante. Já no caso de se necessitar uma busca sem tanto esforço computacional mas que possa procurar em espaços mais amplos seria interessante utilizar a busca, também sub-ótima, telescópica. Como se pode

ver, não há resposta única para o uso de buscas, deve-se analisar cada caso e a partir daí tomar uma decisão de qual método utilizar.

3 CORES

O termo cor é empregado para referir-se à sensação consciente de um observador cuja retina se acha estimulada por energia radiante.

Cores são sensações visuais causadas por feixes de luz que estimulam nossa retina. No mundo real, analógico, se tem um espectro de cores visualmente perceptíveis caracterizadas fisicamente pela Tabela 3.1 [2].

Cor	Comprimento de onda	Freqüência
vermelho	625-740 nm	480-405 THz
laranja	590-625 nm	510-480 THz
amarelo	565-590 nm	530-510 THz
verde	500-565 nm	600-530 THz
ciano	485-500 nm	620-600 THz
azul	440-485 nm	680-620 THz
violeta	380-450 nm	790-680 THz

Tabela 3.1: Tabela de cores do espectro visível

Para ilustrar esse espectro basta observar a Figura 3.1 (escala de comprimento de onda, em nm). Abaixo do azul se encontra o ultravioleta e acima do vermelho, o infravermelho que já são ambos imperceptíveis aos nossos olhos.

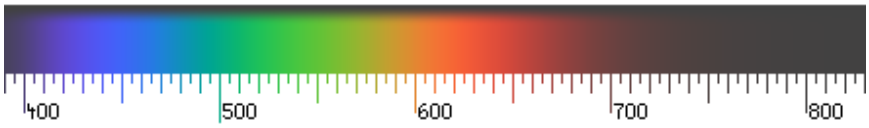


Figura 3.1: Espectro de cores visualmente perceptíveis

Para digitalizar as cores foram criados vários sistemas de representação. Na Seção 3.1 alguns desses modelos são mais bem exemplificados.

3.1 SISTEMA DE REPRESENTAÇÃO DE CORES

3.1.1 Modelo RGB

O modelo de representação RGB (do inglês, *Red Green Blue*) se baseia num sistema de coordenadas cartesianas onde cada um dos eixos representa uma das cores primárias aditivas: vermelho, verde e azul [3]. Imaginando um cubo de lado unitário e com um de seus vértices na origem do sistema, tem-se o cubo que representa todas as cores RGB como mostrado na Figura 3.2. Na Figura 3.2(a) é mostrada a vista interior do cubo e em 3.2(b) a vista exterior.

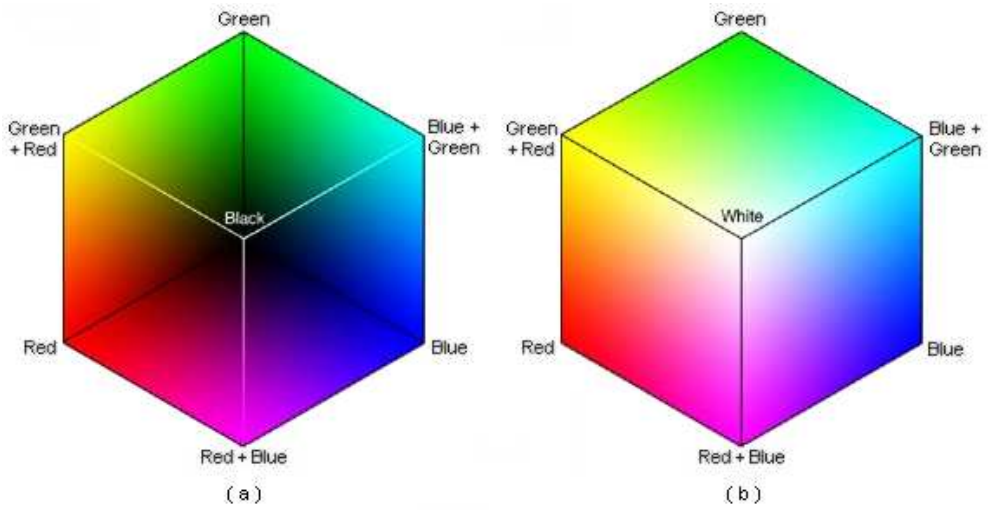


Figura 3.2: Cubo RGB

O cubo RGB da Figura 3.2 pode ser visto aberto na imagem 3.3.

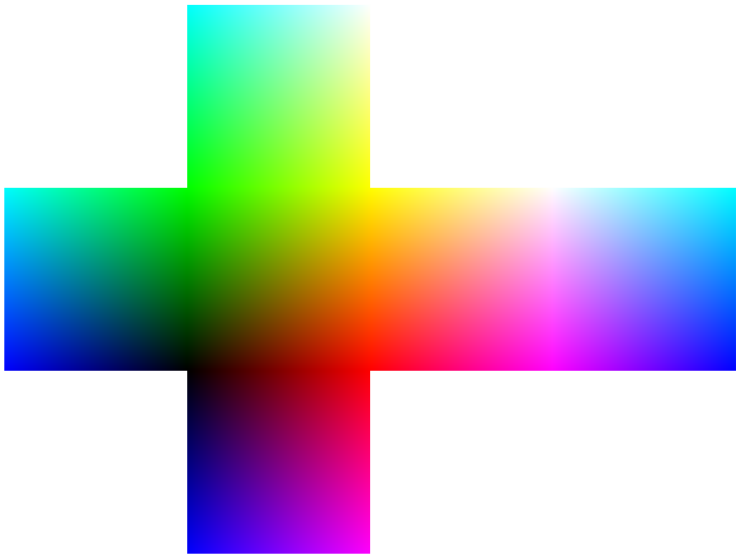


Figura 3.3: Faces do cubo RGB aberto

O sistema RGB é usualmente utilizado em monitores, televisões e câmeras digitais. Ao se misturar diferentes intensidades de cada uma das componentes primárias, em teoria, são obtidas todas as outras cores do cubo.

Figura 3.4: Exemplo de imagem real e suas componentes RGB

A imagem real da Figura 3.4(a) foi dividida em suas três componentes: vermelho (Figura 3.4(b)), verde (3.4(c)) e azul (3.4(d)).

3.1.2 Modelo YCbCr

O modelo de representação YCbCr confere um espaço de cores composto de luminância (Y, em inglês *luminance*), crominância azul (Cb, em inglês *blue chrominance*) e crominância vermelha (Cr, em inglês *red chrominance*). Esse sistema é comumente utilizado na transmissão de vídeos. Suas três componentes podem ser obtidas a partir do modelo RGB, como visto na Figura 3.5 [4], e representados pelas igualdades a seguir [5]:

$$Y = k_r R + (1 - k_b - k_r)G + k_b B \tag{3.1}$$

$$C_b = \frac{0,5}{1 - k_b}(B - Y) \tag{3.2}$$

$$C_r = \frac{0,5}{1 - k_r}(R - Y) \tag{3.3}$$

O processo inverso é descrito por:

$$R = Y + \frac{1 - k_r}{0,5}C_r \tag{3.4}$$

$$G = Y - \frac{2k_b(1 - k_b)}{1 - k_b - k_r}C_b - \frac{2k_r(1 - k_r)}{1 - k_b - k_r}C_r \tag{3.5}$$

$$B = Y + \frac{1 - k_b}{0,5}C_b \tag{3.6}$$

Pela recomendação da ITU-R BT.601, definem-se os valores das constantes k_b e k_r:

$$k_b = 0,144 \tag{3.7}$$

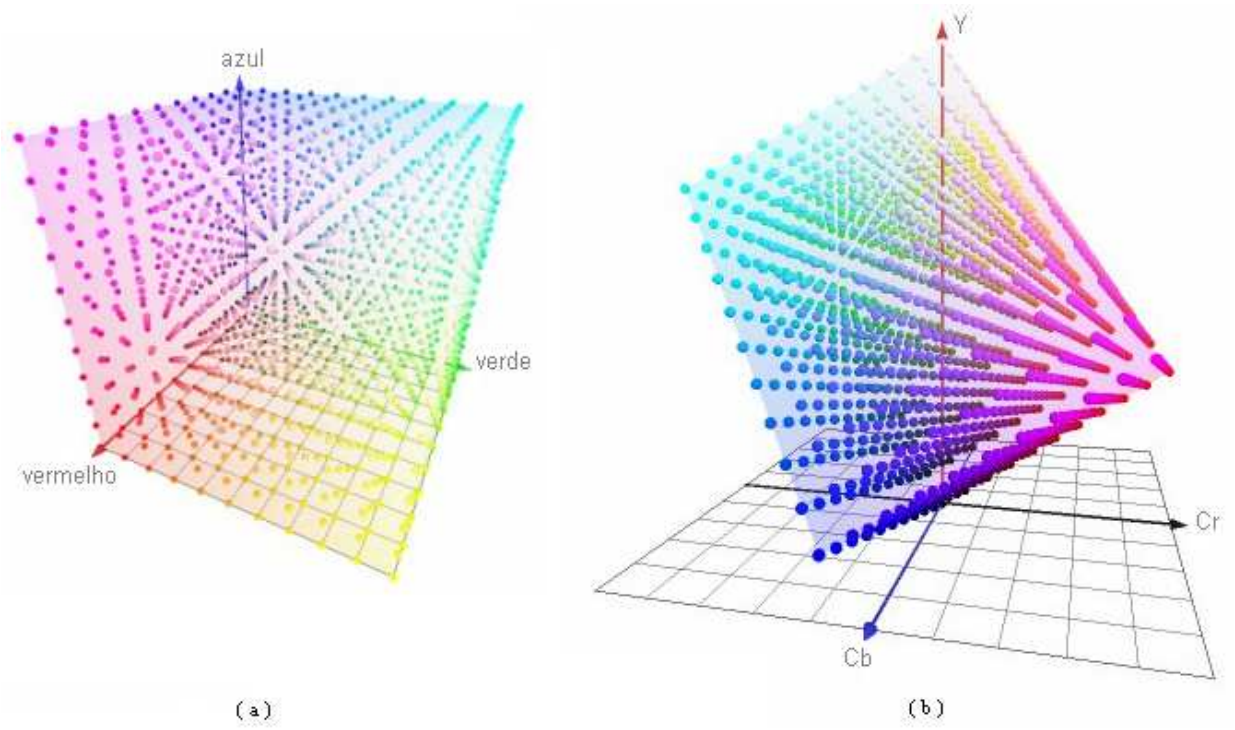


Figura 3.5: Cubo RGB e a representação YCbCr

$$k_r = 0,299 \tag{3.8}$$

Substituindo o valor das constantes nas equações e simplificando, obtemos:

$$Y = 0,299R + 0,587G + 0,114B \tag{3.9}$$

$$C_b = 0,564(B - Y) \tag{3.10}$$

$$C_r = 0,713(R - Y) \tag{3.11}$$

E, voltando ao modelo RGB:

$$R = Y + 1,402C_r \tag{3.12}$$

$$G = Y - 0,344C_b - 0,714C_r \tag{3.13}$$

$$B = Y + 1,772C_b \tag{3.14}$$

Um exemplo de imagem dividida em suas componentes Y, Cb e Cr é mostrado na Figura 3.6.

É possível perceber que a luminância (Figura 3.6(b)) é essencialmente a imagem original em escala de cinza. É também a imagem que mais se aproxima do verde (Figura 3.4(c)) do sistema RGB pois ambas são as componentes mais nítidas da composição. Dependendo da aplicação pode-se até fazer aproximações grosseiras entre essas duas componentes.

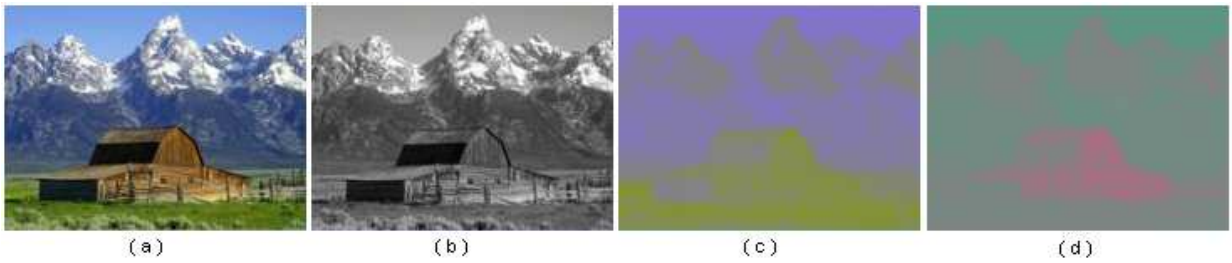


Figura 3.6: Exemplo de imagem real e suas componentes YCbCr

As Figuras 3.6(c) e (d) retratam, respectivamente, as componentes Cb e Cr. Observa-se que a informação contida nessas duas imagens, ao olho humano, não contribui tanto para a interpretação objetiva e clara da imagem. Por essa razão, freqüentemente essas duas componentes são sub-amostradas em transmissões de vídeo pois a maior parte da informação visual da imagem fica concentrada na luminância.

3.1.3 Modelo HSI

Outro modelo de representação de cores é chamado HSI, do inglês *Hue Saturation Intensity*, ainda conhecido também como HSL ou HSV (do inglês, *Lightness* e *Value*, respectivamente). Suas componentes são: matiz, saturação e intensidade.

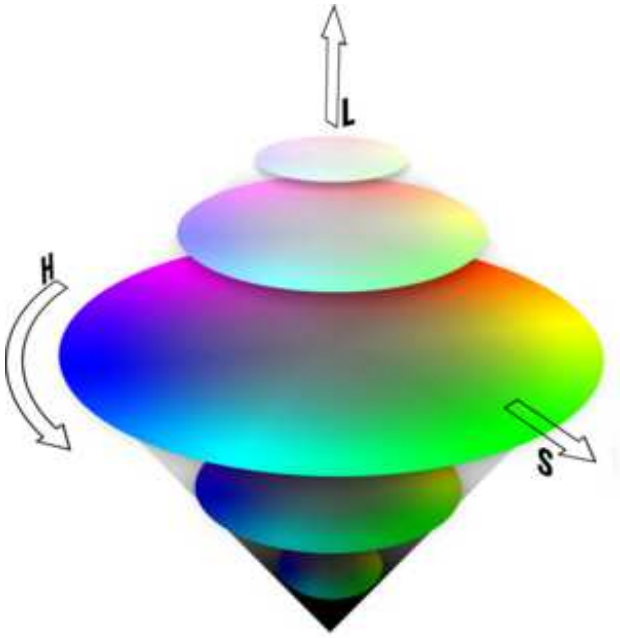


Figura 3.7: Representação gráfica do modelo HSI

As componentes desse sistema estão representadas na Figura 3.7. Percebe-se que as referências desse modelo são polares, diferentemente dos modelos anteriores (RGB e YCbCr) pois não se trata mais de um

cubo e sim de um duplo cone.

O matiz (*hue*) é determinado pelo ângulo que percorre a Figura numa volta inteira de 360° determinando o que poderia se chamar de cor pura dominante que é percebida pelo observador, sendo 0° o vermelho, 120° o verde e, por fim, 240° o azul. Os ângulos de 60°, 180° e 300° representam, respectivamente, amarelo, ciano e magenta.

A saturação é o raio do círculo representando a quantidade de branco diluída no matiz. Quanto mais alta a saturação mais viva é a cor, quanto mais baixa, mais esbranquiçada ela se apresenta. A intensidade é a quantidade de luz refletida pelo objeto, variando do valor mais baixo que seria totalmente escuro até seu valor mais alto que seria o mais claro.

As três componentes desse sistema HSI podem também ser obtidas a partir do modelo RGB [6]. Pode-se representar essas componentes pelas igualdades a seguir:

$$H = arccos\left\{\frac{\frac{1}{2}\left[(R-G)+(R-B)\right]}{\sqrt{(R-G)^2+(R-B)(G-B)}}\right\} \tag{3.15}$$

$$S = 1 - \frac{3}{(R+G+B)}\left[min(R,G,B)\right] \tag{3.16}$$

$$I = \frac{R+G+B}{3} \tag{3.17}$$

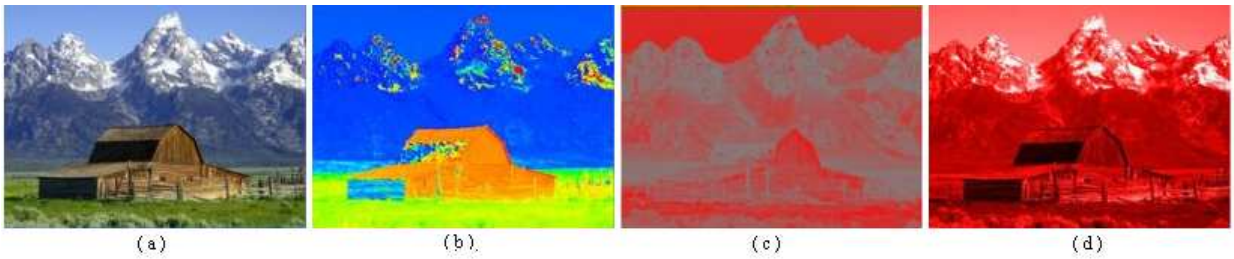


Figura 3.8: Exemplo de imagem real e suas componentes HSI

A Figura 3.8 apresenta uma imagem original e as componentes de matiz (3.8(b)), saturação (3.8(c)) e intensidade (3.8(d)) separadamente. Assim como os sistemas anteriores, percebe-se que há uma componente que destaca visualmente melhor os detalhes da imagem, sendo nesse modelo a componente de intensidade (d) responsável por essa característica.

3.2 COR DE PELE

Um dos desafios na detecção de cores é determinar um modelo que se aplique bem ao reconhecimento de cor de pele. Vários estudos foram e ainda são realizados a cada dia para elaborar algoritmos que acusem

precisamente cor de pele.

Dependendo da aplicação é possível se apoiar em um ou mais modelos de representação de cores, sendo os mais comuns RGB, YCbCr e HSI.

Há teorias que afirmam que mesmo que cor de pele não possua um só tom, ou seja, mesmo havendo pessoas negras, brancas, mulatas, pardas, amarelas etc., a dita cor de pele pode ser identificada como sendo a mesma ”cor”. No caso do sistema YCbCr é sustentado o fato que as diversas cores de pele se encontrem numa mesma faixa de crominância (Cb e Cr) - como visto na Figura 3.9 - alterando de forma mais significante somente a luminância (Y) de acordo com o tom de cada pele [7].

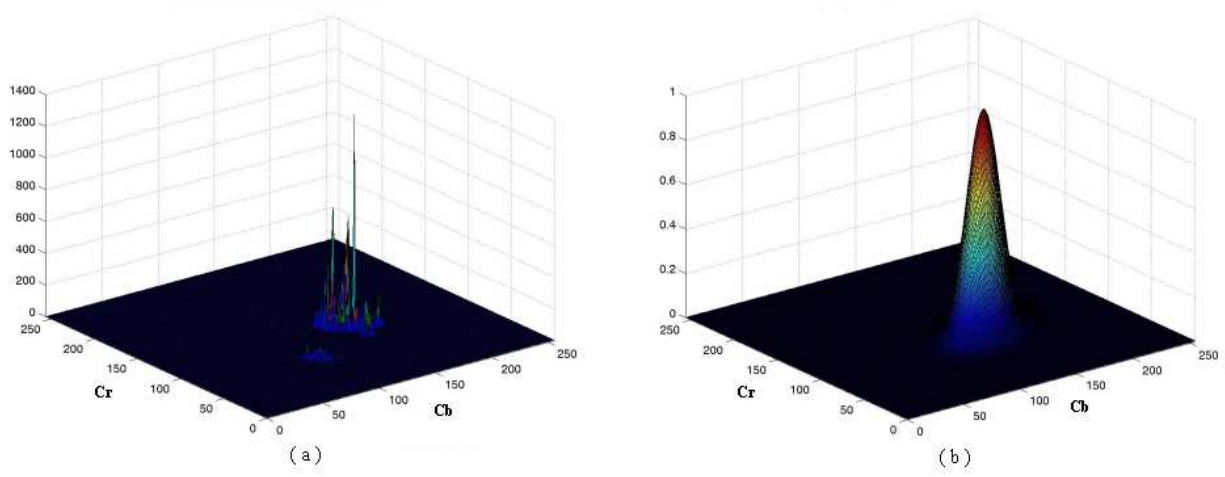


Figura 3.9: Gráfico com distribuição de cor de diversas amostras de cor de pele

A Figura 3.9 [8] acusa a crominância de diversas amostras de cor de pele de distintos tons, por assim dizer. É constatada a concentração mais intensa em certa área do gráfico ratificando a teoria que independente do tom de pele, seja negro, branco ou qualquer outra variação, a crominância é praticamente a mesma. A Figura 3.9(b) traça uma distribuição gaussiana sobre as amostras encontradas em 3.9(a).

Vários artigos apresentam diversas soluções para identificação cada vez mais robusta de cor de pele. A dificuldade não se encontra somente em descobrir que regiões na imagem têm cor de pele porém em também saber que regiões não a possuem. É dito isso pois há muitas regiões que dependendo da eficiência do modelo que se utilize podem ser confundidas com pele, causando eventuais problemas para a aplicação.

Vale lembrar que não há ainda nenhum modelo de identificação de pele que registre com plena exatidão tudo o que for pele e que ignore com mesma precisão tudo na imagem que não seja.

Um exemplo de identificação de cor de pele se encontra na Figura 3.10 [7](modificado). Na Figura são expostos três casos de reconhecimento de pele, cada um baseado em um modelo de representação distinto:

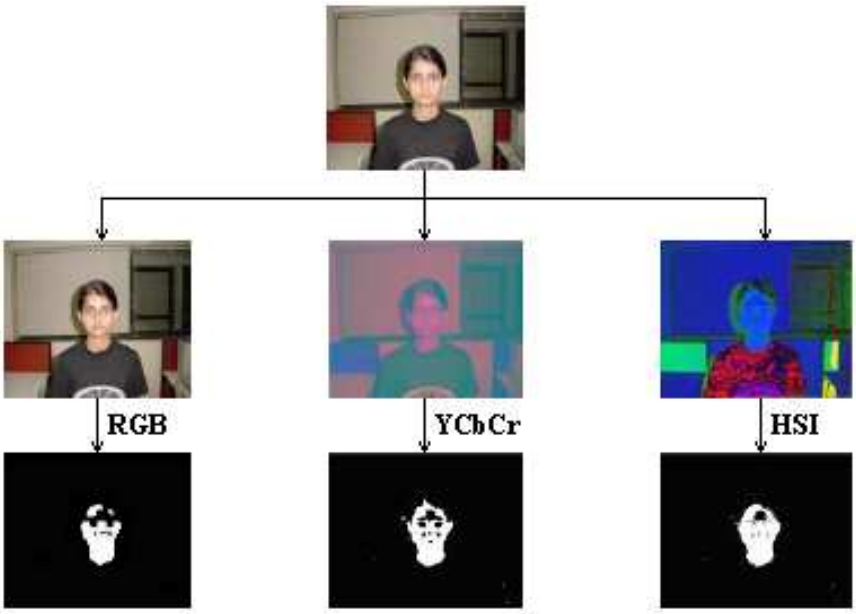


Figura 3.10: Exemplo de identificação de cor de pele baseada em RGB, YCbCr e HSI

RGB, YCbCr e HSI, respectivamente.

Não só a eficácia desses algoritmos de reconhecimento é medida mas também seu tempo de processamento. Há casos em que, dependendo da aplicação, é possível abdicar de certa exatidão na identificação da pele para ter, em contrapartida, menos esforço computacional e resultados mais imediatos como, por exemplo, é demandado em situações onde se deseje trabalhar o mais próximo possível do tempo real.

4 APLICATIVO E RESULTADOS

- Podia-me dizer por favor, qual é o caminho para sair daqui? - Perguntou Alice.

- Isso depende muito do lugar para onde você quer ir. - Respondeu o Gato.

- Não me importa muito onde... - Falou Alice.

- Nesse caso não importa por onde você vá. - Disse o Gato.

(Lewis Carroll, Alice no País das Maravilhas)

Este Capítulo visa apresentar o que foi realizado no projeto explicitando em cada Subseção alguns dos passos seguidos. O resultado gerado neste projeto foi um aplicativo que detecta movimentos e também cor de pele a cada quadro, e que ao se associar essas duas informações é possível reconhecer os movimentos e postura da mão para tanto mover o *mouse* quanto fazer os cliques direito e esquerdo do mesmo.

4.1 INICIANDO A CAPTURA DE IMAGENS

O programa desenvolvido tem uma interface gráfica simples com o menu principal mostrado na Figura 4.1.

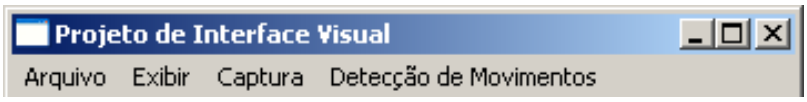


Figura 4.1: Menu principal do programa de interface visual

Para iniciar o uso do programa se deve clicar no menu *Captura* e depois em *Iniciar* como na Figura 4.2.

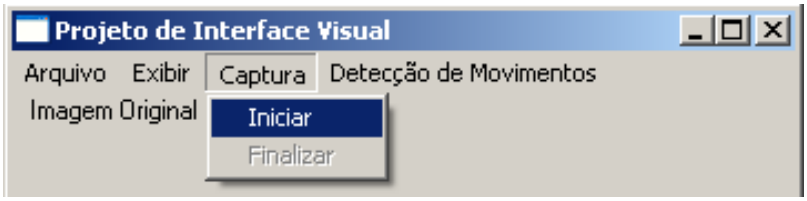


Figura 4.2: Menu responsável por iniciar a captura de imagens

Ao clicar em *Iniciar* o programa se conecta ao driver da *webcam* que estiver conectada ao computador. Caso não haja uma conectada será exibido um aviso para que algum dispositivo de captura seja conectado. Quando a câmera estiver conectada então a imagem da mesma será mostrada na janela do programa, chamada *Imagem original*.

Figura 4.3: Janela mostrando a imagem capturada pela *webcam*

A Figura 4.3 retrata esta janela. A imagem capturada é colorida e tem o formato RGB explicado na Subseção 3.1.1, e como já foi explanado, possui três componentes: vermelho, verde e azul. O programa é configurado para capturar e exibir 30 quadros por segundo, o que é suficiente para criar a ilusão de imagem em movimento contínuo que os olhos humanos podem normalmente perceber.

A resolução da imagem depende da resolução disponível na própria *webcam*. A maioria dos testes realizados com o programa de interface visual fizeram uso de imagens na resolução de 352 *pixels* de largura por 288 *pixels* de altura. A alteração da resolução utilizada em nada interfere o pleno funcionamento do programa.

4.2 VETORES DE MOVIMENTO

A partir da imagem capturada supracitada, o próximo passo é realizar a estimação de movimento. A estimação explicada na Seção 2.1 é realizada ao se clicar no menu *Detecção de Movimentos* e

posteriormente em *Detectar movimentos* como mostrado na Figura 4.4.

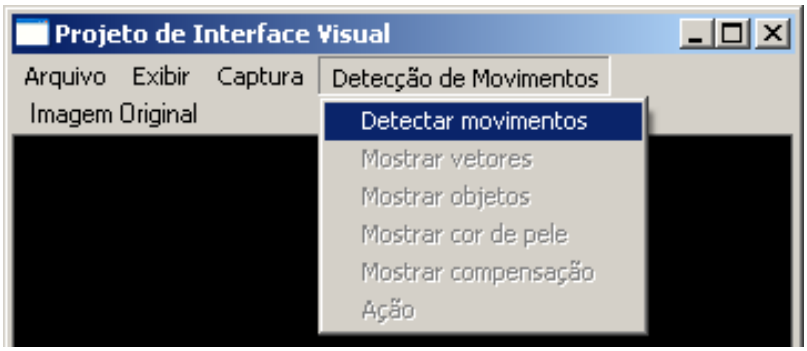


Figura 4.4: Menu responsável por iniciar a estimação de movimento

Ao se iniciar a estimação de movimento as opções de mostrar vetores, objetos, cor de pele e compensação são habilitados. Para mostrar o resultado da estimação de movimento é possível exibir outra janela no programa que mostra os vetores de movimento resultante de cada macrobloco. Para mostrar essa janela deve-se clicar no menu *Detecção de Movimentos* e depois em *Mostrar vetores* (Figura 4.5).

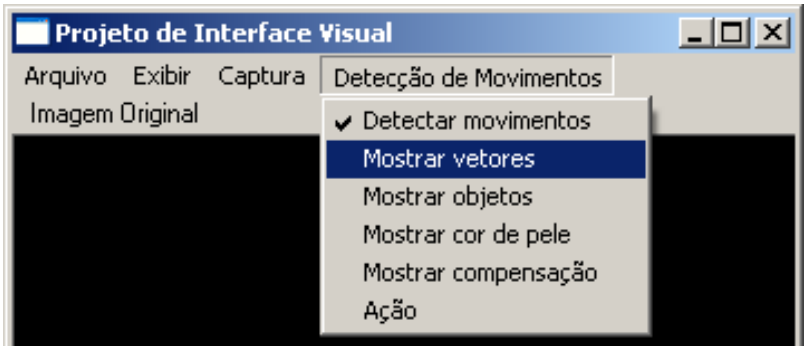


Figura 4.5: Menu responsável por mostrar os vetores de movimento

A Figura 4.6 apresenta justamente os vetores de uma imagem dinâmica.

O tamanho do macrobloco foi estipulado em 8x8 *pixels*. Testes com blocos de 16x16, 4x4 e 2x2 *pixels* foram realizados também porém esses outros tamanhos não geraram resultados satisfatórios como os de 8x8 *pixels*.

Ao se utilizar macroblocos de 16x16 *pixels* (Figura 4.7) é percebida uma perda na qualidade da estimação pelo macrobloco ser considerado muito grande para uma resolução baixa como a de uma *webcam* comum. Já macroblocos de 2x2 *pixels* (Figura 4.9) geraram dois problemas, primeiramente o esforço computacional de estimar movimento em todos os macroblocos ser bem maior que no caso de macroblocos de 8x8 *pixels*, e o outro problema constatado foi ainda a perda de contexto do macrobloco ocasionando mais ruído que nos casos de macroblocos maiores, mesmo limitando os vetores válidos como

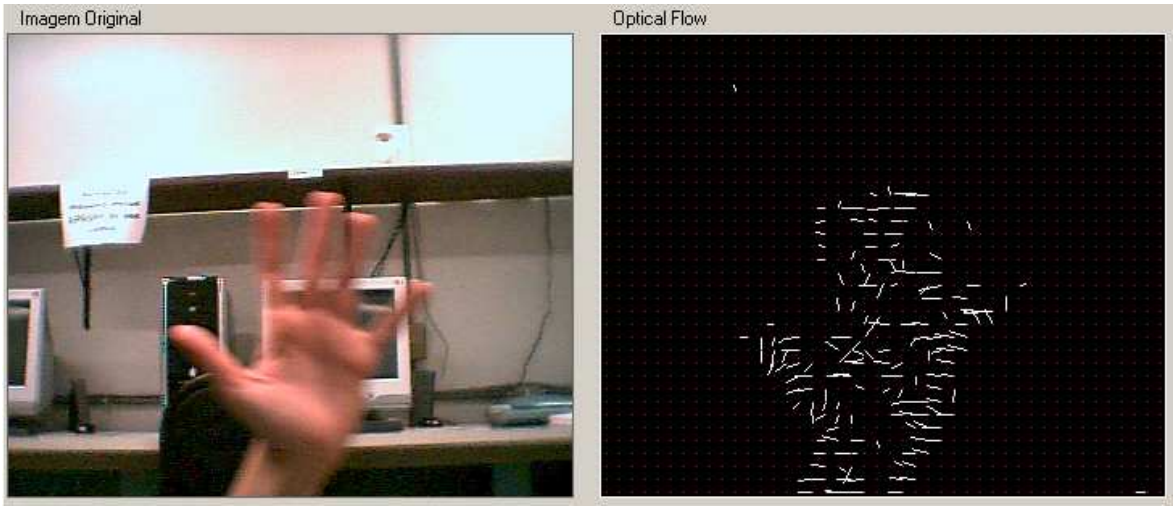


Figura 4.6: Janela mostrando os vetores de movimento em macroblocos de 8x8 *pixels*

vetores de módulo acima de quatro *pixels* (limite esse utilizado em qualquer um dos casos de tamanho de macrobloco).

Por fim, macroblocos de 4x4 *pixels* (Figura 4.8) se mostraram de boa qualidade na hora de estimar o movimento porém consumiam ainda um processamento além do esperado para uma aplicação que propõe trabalhar o máximo possível próximo de tempo real. Ao final dos testes, os macroblocos de 8x8 *pixels* foram os que se mostraram com a melhor relação custo/benefício (qualidade *versus* processamento).

Para efeito ilustrativo será calculado o número de macroblocos presentes em uma imagem de 352x288 *pixels* com tamanhos de macroblocos distintos para que seja percebida a diferença de processamento envolvida na estimação de movimento.

Macroblocos de 16x16 *pixels*:

$$M16_{linhas} = 288/16 = 18\ macroblocos \tag{4.1}$$

$$M16_{colunas} = 352/16 = 22\ macroblocos \tag{4.2}$$

$$M16 = 18 \cdot 22 = 396\ macroblocos \tag{4.3}$$

Macroblocos de 8x8 *pixels*:

$$M8_{linhas} = 288/8 = 36\ macroblocos \tag{4.4}$$

$$M8_{colunas} = 352/8 = 44\ macroblocos \tag{4.5}$$

$$M8 = 36 \cdot 44 = 1584\ macroblocos \tag{4.6}$$

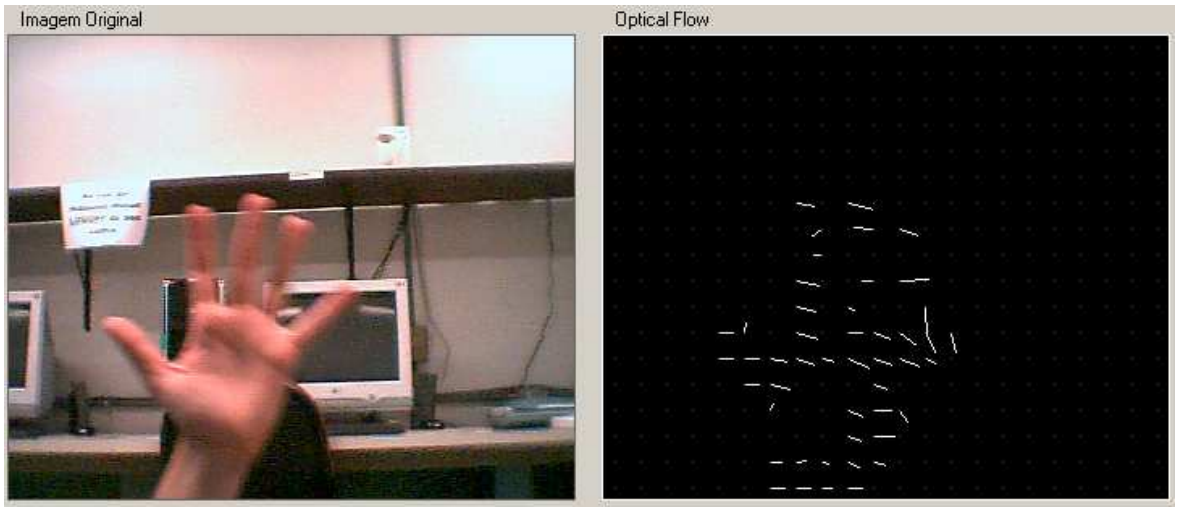


Figura 4.7: Janela mostrando os vetores de movimento em macroblocos de 16x16 *pixels*

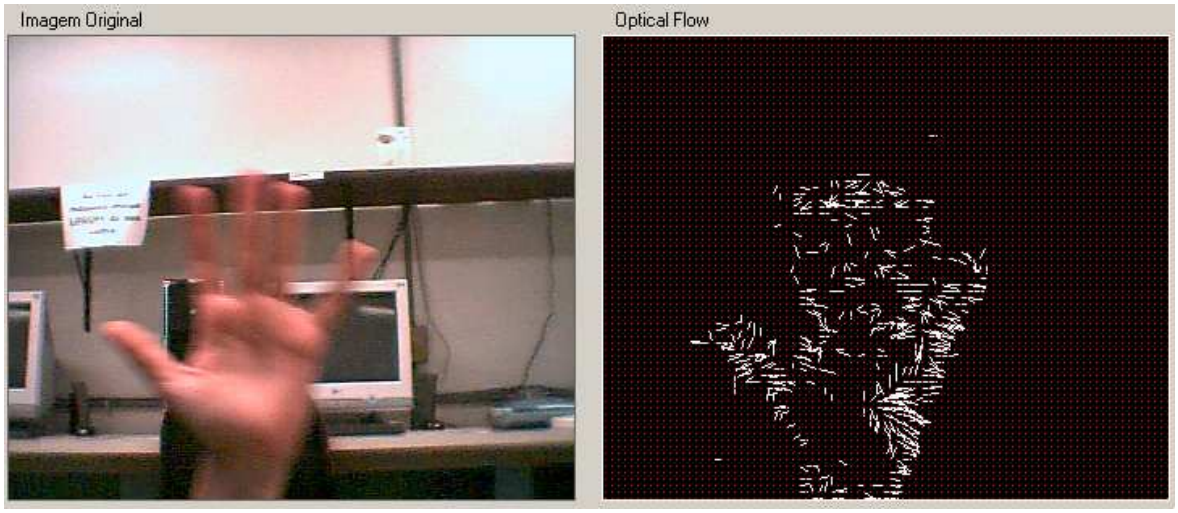


Figura 4.8: Janela mostrando os vetores de movimento em macroblocos de 4x4 *pixels*

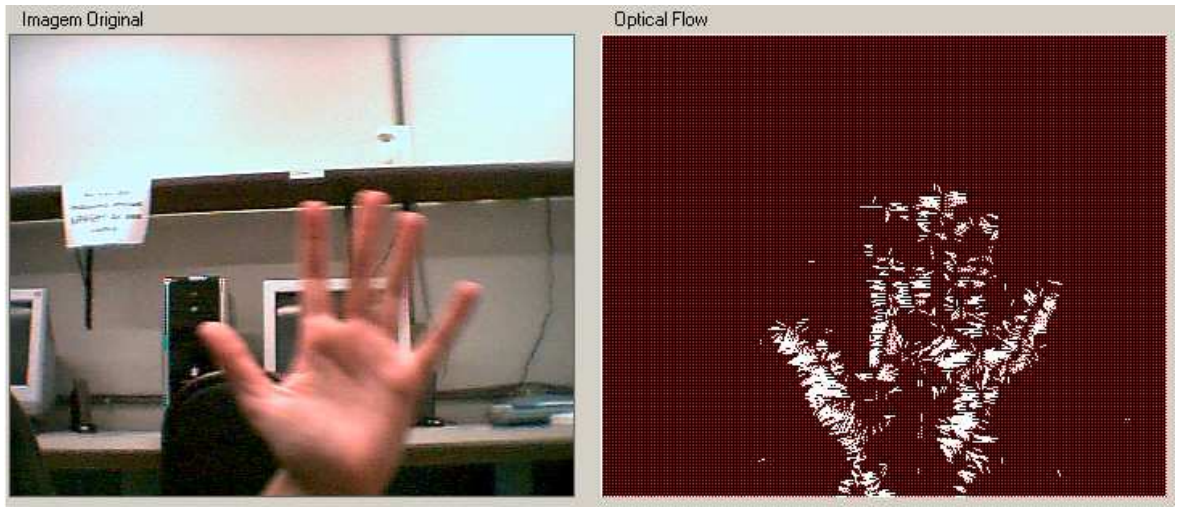


Figura 4.9: Janela mostrando os vetores de movimento em macroblocos de 2x2 *pixels*

Macroblocos de 4x4 *pixels*:

$$M4_{linhas} = 288/4 = 72\ macroblocos \tag{4.7}$$

$$M4_{colunas} = 352/4 = 88\ macroblocos \tag{4.8}$$

$$M4 = 72 \cdot 88 = 6336\ macroblocos = 4 \cdot M8 \tag{4.9}$$

Macroblocos de 2x2 *pixels*:

$$M2_{linhas} = 288/2 = 144\ macroblocos \tag{4.10}$$

$$M2_{colunas} = 352/2 = 176\ macroblocos \tag{4.11}$$

$$M2 = 144 \cdot 176 = 25344\ macroblocos = 16 \cdot M8 \tag{4.12}$$

Como pode ser visto pelas equações, a quantidade de buscas nos casos de macroblocos de 4x4 e 2x2 *pixels* começa a se tornar pouco praticável se for desejado obter também menor tempo na estimação total de cada quadro.

Lembrando ainda que deve ser feita a estimação de movimento de todos os macroblocos de cada um dos 30 quadros capturados por segundo e que a estimação de cada macrobloco demanda várias operações de SAD (Seção 2.2), portanto o esforço pode se tornar algo em torno de 480 vezes maior considerando um segundo (30 quadros) se forem utilizados macroblocos de 2x2 *pixels* (em comparação com macroblocos de 8x8 *pixels*).

O programa de interface visual realiza a estimação de movimento via busca diamante (Subseção 2.3.6) pois a mesma tem boa qualidade para as necessidades do programa e se mostrou ainda a busca com menor esforço computacional dentre as alternativas. Mais uma vez foi escolhido um método com melhor custo/benefício sobre as opções existentes.

4.3 OBJETOS BASEADOS EM VETORES DE MOVIMENTO

Baseado em vetores de movimento, algumas abordagens para identificar diferentes objetos foram cogitadas. A idéia inicial foi utilizar algoritmos não supervisionados de classificação de dados. Essa classificação é apoiada em certos atributos que geram *n* partições de uma imagem. Os atributos, nesse caso poderiam ser a presença e proximidade espacial de vetores de movimento.

Um dos algoritmos testados para esta tarefa foi o algoritmo *K-Means* [9]. Este algoritmo analisa todos os dados presentes e indica uma categoria para eles. O usuário deve previamente indicar o número *k* de categorias para que o algoritmo classifique os dados dentro de uma dessas *k* classes.

O *K-Means* funciona de modo que ao se iniciar uma classificação, o algoritmo, já sabendo a quantidade *k* de classificações, distribui centróides aleatoriamente pela imagem e começa a calcular a distância, geralmente euclidiana, entre cada dado e esses centróides. Os dados então são agrupados de forma que sua classificação recaia na categoria onde houver a menor distância com o respectivo centróide.

Terminada esta primeira iteração, o centróide de cada um dos *k* objetos (classes) é recalculado e outra iteração é realizada de modo que as distâncias são recalculadas. Serão feitas tantas iterações quanto forem necessárias até que os centróides convirjam e não mais se movam, não sendo conhecida, previamente, a quantidade dessas iterações e ainda não sendo garantida a convergência. A Figura 4.10 retrata a iteração deste algoritmo [1].

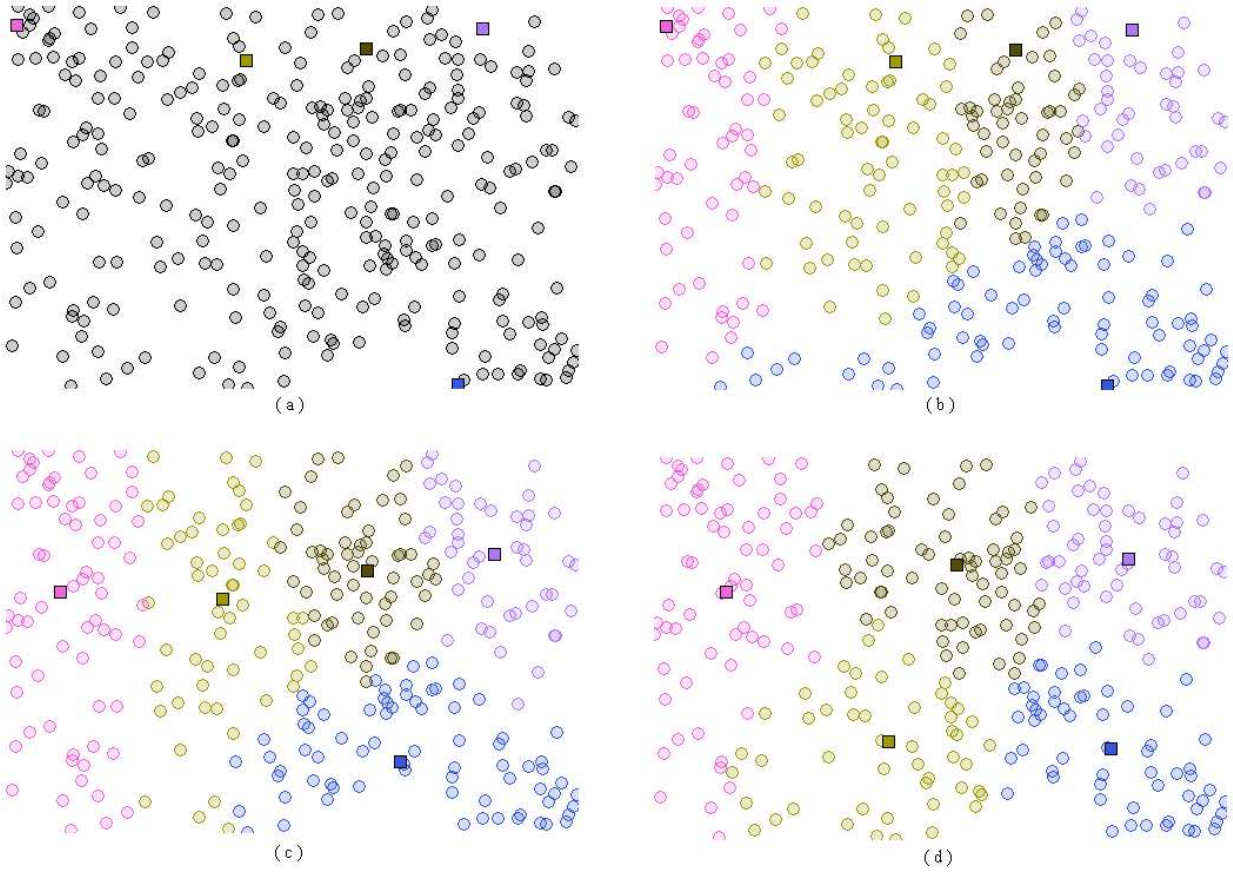


Figura 4.10: Funcionamento do algoritmo *K-Means*

A Figura 4.10(a) mostra uma distribuição de 200 dados (círculos cinzas) e cinco centróides (quadrados

[1]Demonstração disponível em *http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/AppletKM.html*

coloridos). Em 4.10(b), ocorre a primeira iteração agrupando os dados mais próximos de cada centróide. Estes centróides são recalculados e redistribuídos na imagem, assim como os dados que são reclassificados para os centróides mais próximos como visto na segunda iteração na Figura 4.10(c). A Figura 4.10(d) mostra a convergência dos centróides e o resultado final da classificação após 11 iterações.

Este algoritmo é de fato eficiente porém tem um grande problema que o torna inviável para este projeto. O *K-Means* demanda que se informe necessariamente quantas categorias devem ser utilizadas para classificar os dados e na imagem capturada pela *webcam* nunca se sabe ao certo quantos objetos aparecerão se movendo em cena a não ser que a imagem seja submetida a um prévio procedimento como *split-and-merge*, porém a cada nova varredura, mais demorada se tornaria toda a análise prejudicando a eficiência do aplicativo. Outra desvantagem é que o resultado final das iterações muda de acordo com o posicionamento inicial dos centróides, podendo até acontecer desses centróides não convergirem.

Outro ponto a se considerar também é que o *K-Means* é um algoritmo de *hard clustering*, ou seja, os dados podem pertencer a somente uma única categoria, ao contrário do *soft clustering* que considera cada dado como provável pertencente a uma ou outra categoria sejam elas quantas forem. Então a idéia de se dividir a imagem em *k* regiões não seria útil pois seria possível e muito provável de separar a imagem em regiões que retratariam erroneamente os objetos em movimentos.

Outros dois algoritmos cogitados para dividir os objetos em movimento na imagem foram o *C-Means* [10] e o Mapa de *Kohonen* [11]. O primeiro em muito se assemelha ao *K-Means* porém é um algoritmo de *soft clustering*. Como explicado anteriormente, ele, apesar de poder atribuir probabilidades de classificação de dados a vários objetos simultaneamente, ainda assim necessita que se aponte previamente o número *c* de objetos da imagem e o resultado depende também do posicionamento inicial dos centróides.

O Mapa de *Kohonen* é uma técnica de rede neural que organiza e aprende a reconhecer padrões e ainda fazer previsões. Como o propósito inicial da classificação de objetos nos quadros capturados pela câmera deveria ser feito de imediato, um algoritmo de aprendizado não seria de interesse nesse momento portanto não foi levado adiante seu uso.

Finalmente, a técnica final utilizada foi desenvolvida a partir da observação dos próprios vetores e busca por conectividade. Com a informação desses vetores de movimento, foram armazenados em uma variável ”mapa” todos os macroblocos que possuíam vetores com módulo acima de quatro *pixels*. Esse mapa sofreu uma erosão de modo que se numa vizinhança N_4 a soma dos vetores de movimento não fosse maior que 25 *pixels* (de deslocamento tanto vertical como horizontal) então o *pixel* central seria erodido (desmarcado) como visto na Figura 4.11.

O próximo passo consiste em varrer o mapa e realizar uma dilatação. A verificação condicional para esse processo é que o *pixel* central e pelo menos um macrobloco de uma vizinhança de N_8 fossem marcados. Se a condição fosse atendida então toda a vizinhança N_8 seria marcada. Uma observação importante a se fazer é que a cada operação morfológica realizada, novos mapas eram utilizados para escrever os resultados (novo mapa) para que, obviamente, as análises das iterações anteriores não influenciassem nas iterações seguintes.

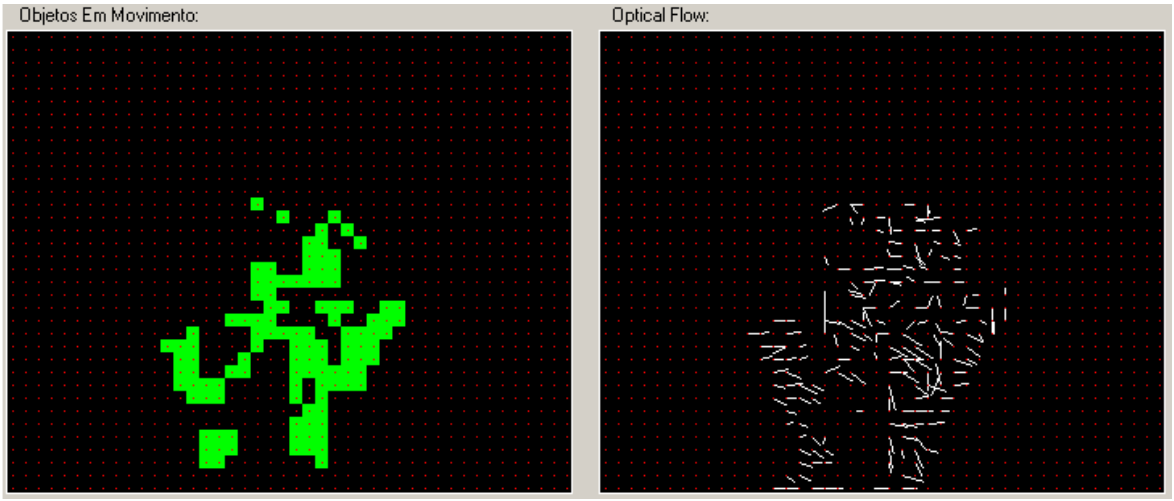


Figura 4.11: Janela mostrando o mapa de macroblocos depois de erodidos baseado nos vetores de movimento mostrado na janela de vetores (*optical flow*) ao lado

O mapa resultante é visto na Figura 4.12. Os macroblocos pintados são os resultantes da abertura (erosão seguida de dilatação).

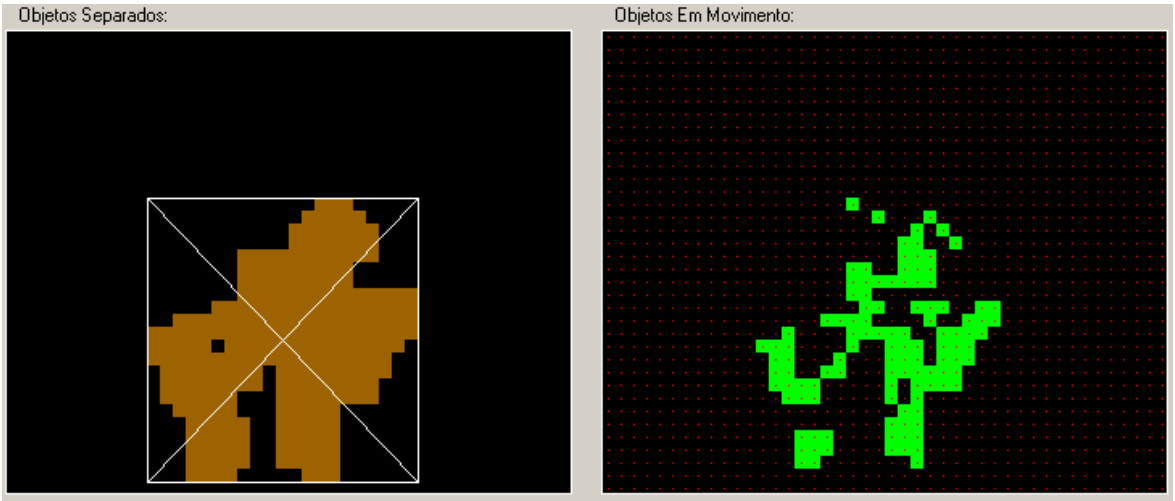


Figura 4.12: Janela mostrando os macroblocos dilatados e conexos baseados em movimento e a janela ao lado com os blocos antes da dilatação

Neste momento, utiliza-se um critério de conectividade entre esses macroblocos. São feitas varreduras para que os macroblocos sejam rotulados de acordo com sua proximidade. Se macroblocos estiverem separados até no máximo dois macroblocos de distância (vizinhança N_8 e seus vizinhos diretos na camada imediatamente externa a ela) então os mesmos são considerados como mesmo objeto, ou seja, a eles são atribuídos o mesmo rótulo.

Pode ser percebido na imagem que um retângulo é traçado ao redor do objeto conexo. As diagonais são traçadas somente para efeito de visualização do centróide dessa caixa.

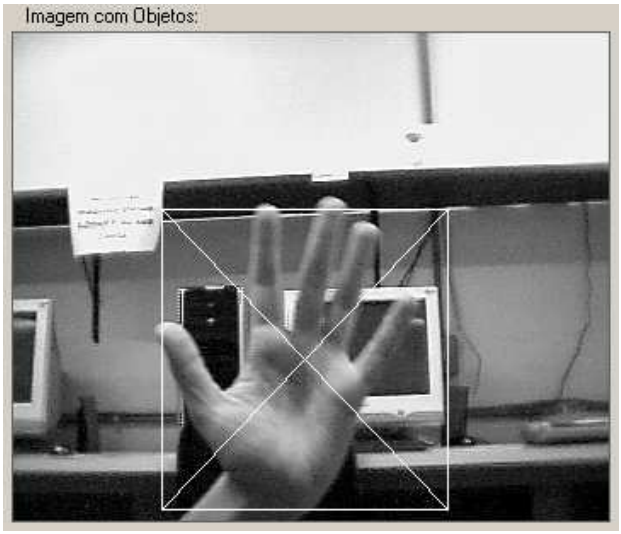


Figura 4.13: Janela mostrando a imagem original e a caixa que envolve o objeto em movimento

Uma outra janela ainda é mostrada na Figura 4.13. Para visualizar essa janela basta clicar no menu *Detecção de Movimentos* e posteriormente em *Mostrar objetos* como visto na Figura 4.14. Essa imagem nada mais é que uma versão em níveis de cinza da imagem original com as caixas que envolvem os objetos em movimento.

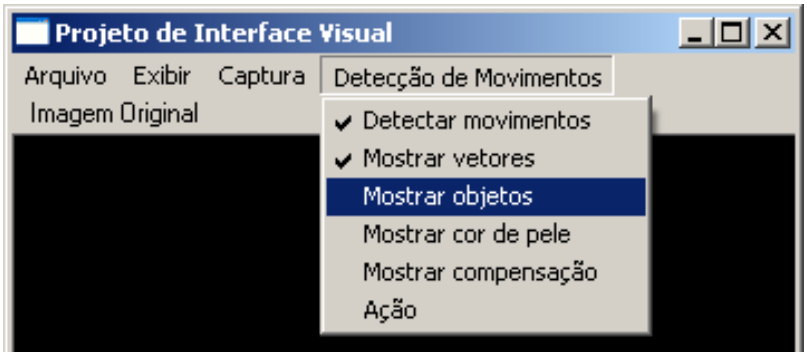


Figura 4.14: Menu responsável por mostrar os distintos objetos capturados pela câmera

Nenhuma outra informação de fato é acrescentada desde o último processamento (dilatação) mostrado

na Figura 4.12, porém a Figura 4.13 existe para que sejam observados de fato quais objetos reais estão a se mover e que foram capturados após a estimação e as operações morfológicas.

Mais exemplos com mais de um objeto se movendo na frente da *webcam* são explicitados na Figura 4.15 e 4.16.

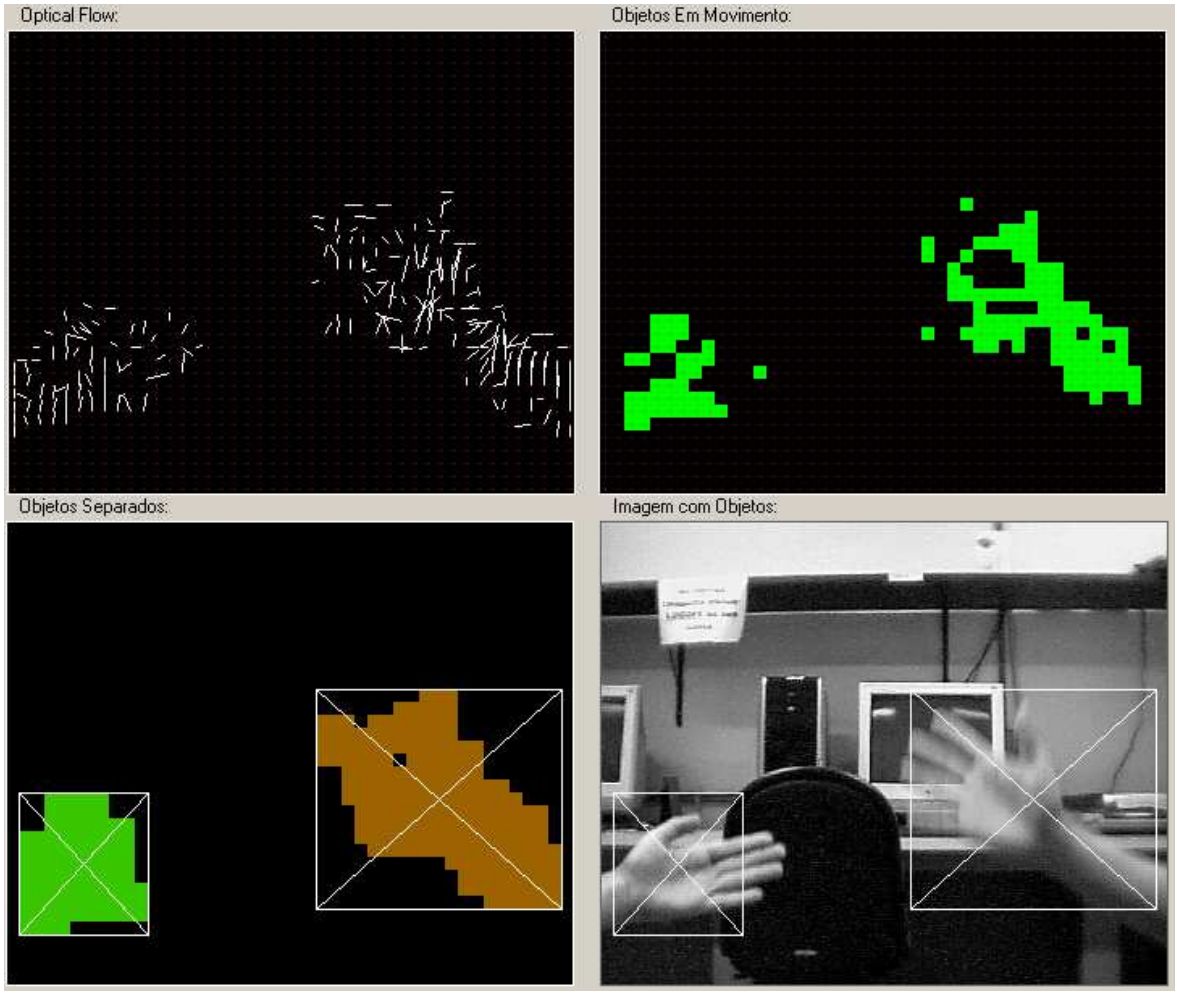


Figura 4.15: Janela mostrando dois objetos em movimento e, respectivamente, seus vetores de movimento, macroblocos erodidos, dilatados e conectados, e imagem real com as caixas de demarcação de dimensões

O programa reconhece tantos quantos forem os objetos desde que os mesmos estejam distantes a pelo menos três macroblocos de espaçamento entre eles. Nessa etapa, quaisquer agrupamentos de vetores podem ser considerados como objetos.

Um problema desse método é que pouco se pode fazer em relação a alguns tipos de ruídos, fundos dinâmicos e adaptações constantes da câmera.

A adaptação da câmera ocorre pois a *webcam* tende a sempre calibrar sua distribuição de cores a partir de uma referência que seria o ponto mais claro da imagem que está sendo capturada. Então de acordo

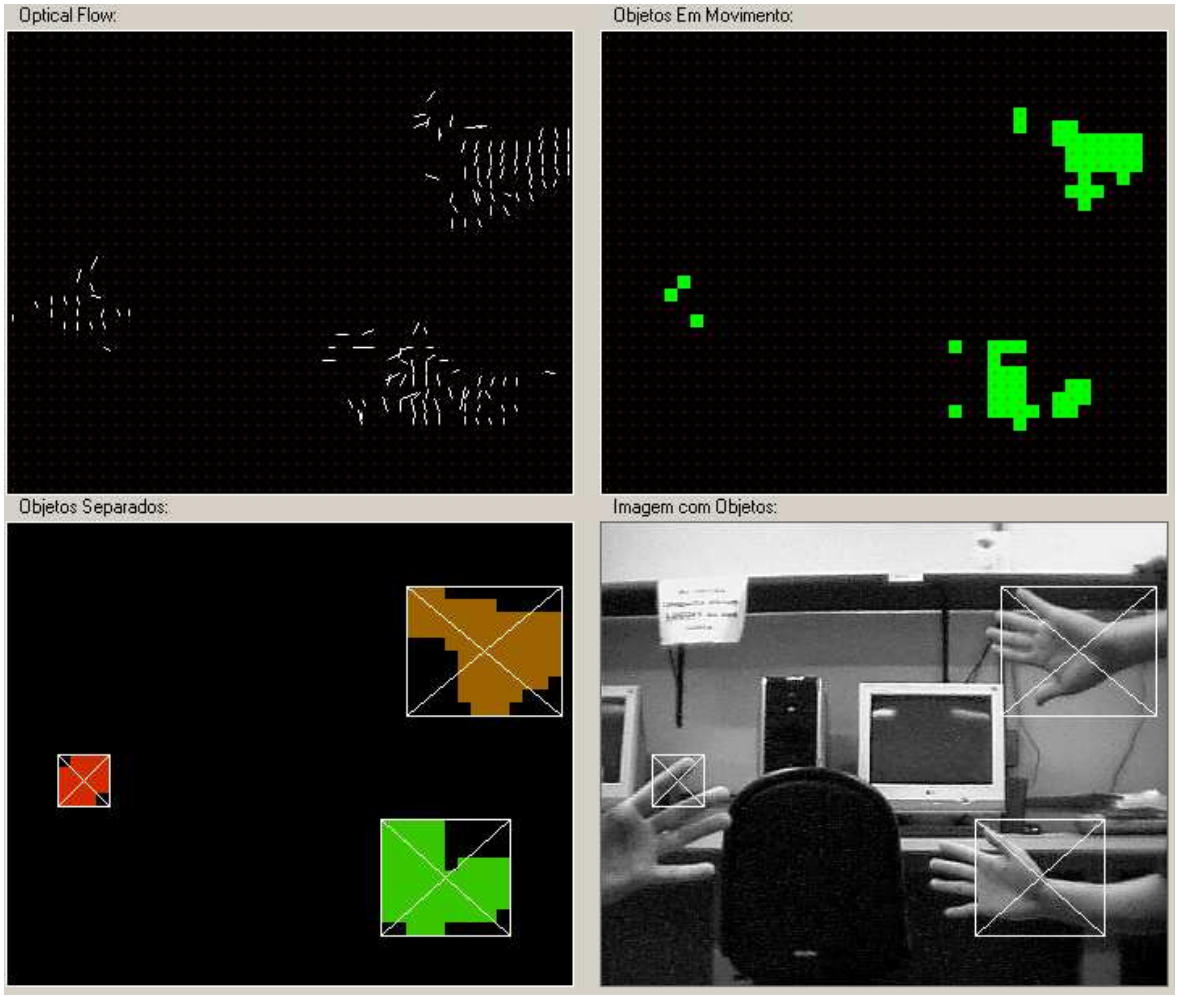


Figura 4.16: Janela mostrando três objetos em movimento e, respectivamente, seus vetores de movimento, macroblocos erodidos, dilatados e conectados, e imagem real com as caixas de demarcação de dimensões

com a luz incidente no ambiente ou pela reflexão de luz dos objetos em cena, a câmera adapta as cores da imagem. Dessa forma, de um quadro para outro quando as cores sofrem ligeira alteração, e ao realizar a estimação de movimento pode ser considerado que houve movimento quando de fato não houve, criando vetores e, consequentemente, um ”pseudo-objeto”.

Fundos dinâmicos criam um empecilho difícil de contornar. Ao se utilizar somente a abordagem de movimento é possível ter até um quadro inteiro como um só objeto, caso todo o fundo esteja se movendo.

Para tentar diminuir esses efeitos indesejáveis e para tornar mais viável à descoberta de objetos específicos na imagem então é preciso utilizar alguma outra técnica aliada ao método explicado nessa Seção. Uma outra técnica foi então colocada em prática e se apóia não em movimento, mas em cores. A Seção seguinte detalha essa escolha.

4.4 OBJETOS COM PRESENÇA DE COR DE PELE

A cor de pele, como explicada na Seção 3.2, pode ser encontrada numa imagem a partir de diversos algoritmos. Várias técnicas são conhecidas porém nenhuma delas possui pleno acerto na identificação correta de *pixels* com cor de pele e plena rejeição de *pixels* que não o são. Sua capacidade de reconhecimento de fato são boas porém até hoje ainda falíveis.

A técnica utilizada no programa para encontrar cor de pele nos quadros capturados pela *webcam*, apesar de não ser a mais eficaz, foi a mais simples, direta e rápida dentre as encontradas [6]. O método consiste de, no próprio espaço RGB, obedecer alguns limiares descritos nas relações que se seguem.

$$R > 95 \tag{4.13}$$

$$G > 40 \tag{4.14}$$

$$B > 20 \tag{4.15}$$

$$max\,(R, G, B) - min\,(R, G, B) > 15 \tag{4.16}$$

$$|R - G| > 15 \tag{4.17}$$

$$R > G \tag{4.18}$$

$$R > B \tag{4.19}$$

Se um *pixel* obedece todas essas relações então é dito que ele é considerado como cor de pele. Exemplo da utilização desse algoritmo são mostrados na Figura 4.17, 4.18 e 4.19.

Os *pixels* pintados de branco não são cor de pele. Os *pixels* pintados de preto e vermelho já são considerados como cor de pele. A diferença entre os *pixels* marcados com preto ou vermelho consiste numa filtragem feita onde o *pixel* central se torna vermelho somente se pelo menos 89% dos *pixels* de uma vizinhança N_8, contando também o *pixel* central, for considerado como cor de pele. Em outras palavras, se pelo menos oito dos nove *pixels* (vizinhança e central) forem pele. Do contrário, o *pixel* central permanece com a mesma designação tomada anteriormente a ele (branco, ”não-pele”, ou preto, ”pele”).

A proposta nesse momento é juntar a informação de objetos determinados por movimento com a identificação de cor de pele. A idéia é reavaliar todos os objetos encontrados previamente considerando agora a presença ou não de cor de pele dentro dessas caixas.

Todas as caixas são analisadas individualmente tentando ao máximo ”encaixotar” justamente o objeto cor de pele. No caso de não haver *pixels* cor de pele substanciais, considerado nesse caso como pelo menos

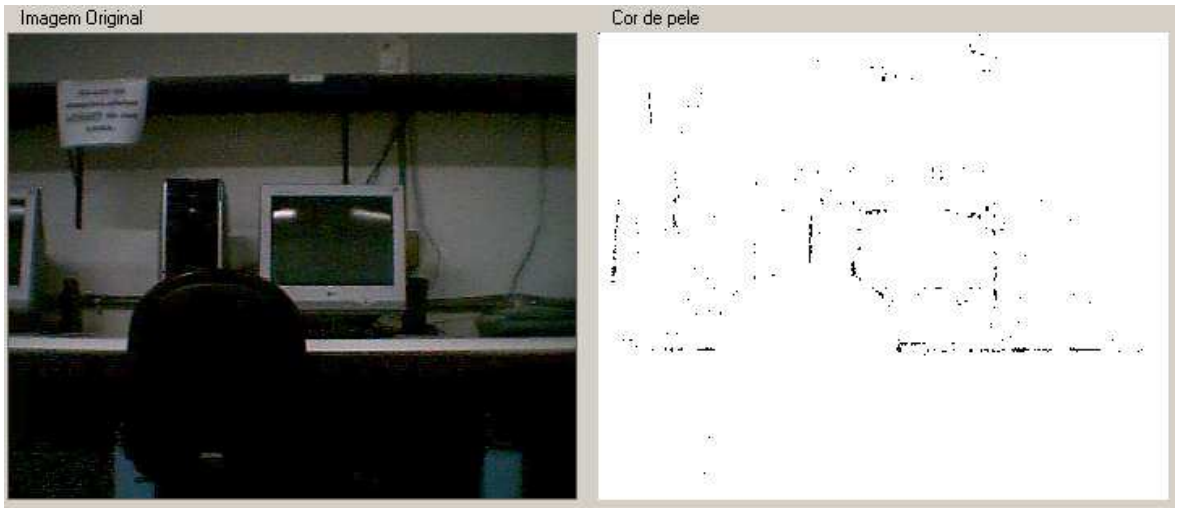


Figura 4.17: Janela mostrando o cenário utilizando identificação de cor de pele

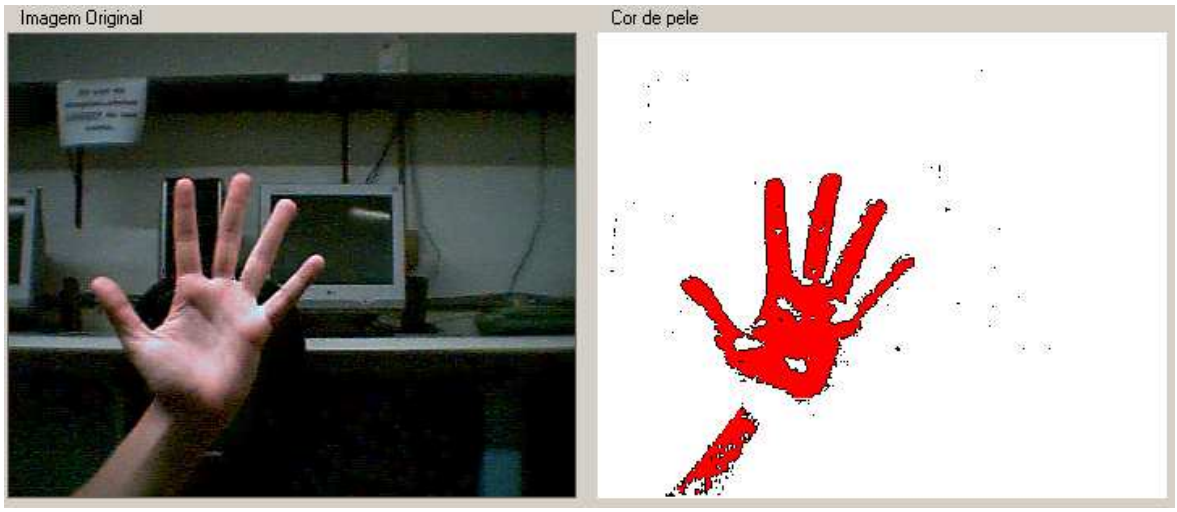


Figura 4.18: Janela mostrando o cenário e a mão utilizando identificação de cor de pele

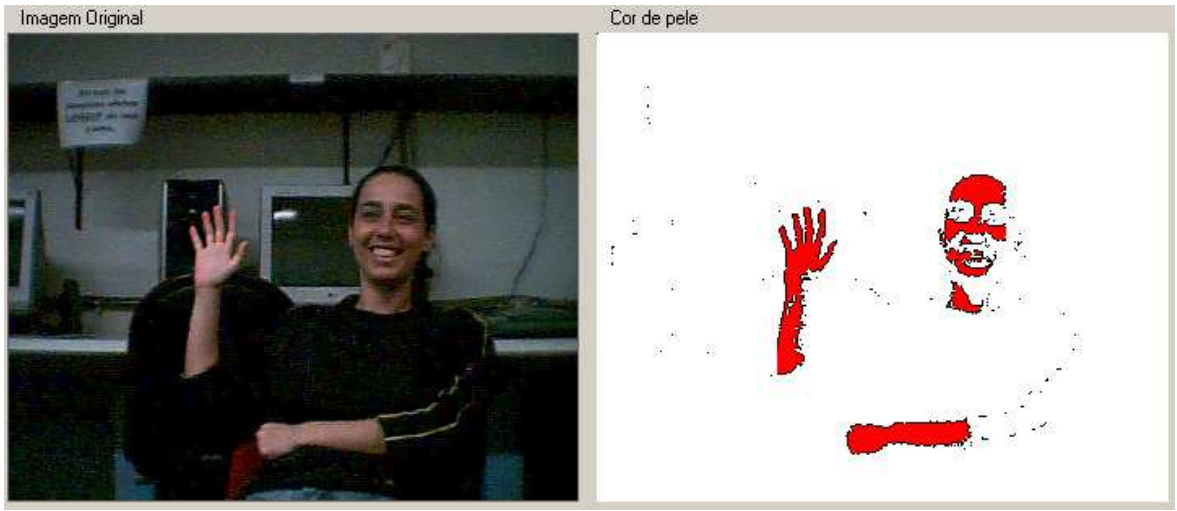


Figura 4.19: Janela mostrando o cenário, mão, braço e rosto utilizando identificação de cor de pele

10% da área total da caixa, dentro das bordas da caixa então o objeto é desconsiderado. Se houver pelo menos 10% de *pixels* cor de pele dentro da caixa então a mesma é reconhecida como uma caixa válida e seu tamanho é ajustado para as dimensões do objeto cor de pele.

Exemplos desse redimensionamento são mostrados nas Figuras 4.20, 4.21 e 4.22.

O retângulo amarelo na imagem retrata os limites do movimento em si. O retângulo verde mostra o redimensionamento da caixa do objeto para englobar toda a dimensão do que foi considerado como cor de pele.

Na Figura 4.22 pode ser percebido que foram encontrados dois objetos, a caneta e a mão. A mão está envolta num retângulo amarelo (redimensionado para outro retângulo verde) e a caneta está envolta por um retângulo branco. Retângulos brancos são desconsiderados por não terem quantidade de *pixels* cor de pele consideráveis dentro de suas bordas. Portanto os únicos retângulos resultantes válidos são os verdes que é a junção de presença de movimento e de cor de pele em seu interior.

Portanto, após utilizar a informação de cores aliada a informação de movimento se tem no final somente caixas candidatas a serem nossos objetos de real interesse que nesse caso são mãos.

Alguns problemas encontrados com esse método para encontrar cor de pele são as falsas identificações como cores que de alguma forma se assemelham à cor de pele porém não o são. Pesquisou-se em muitos artigos algoritmos mais eficientes porém correspondia a um julgamento totalmente seguro e preciso do que seria pele e do que não seria. Dessa forma, o algoritmo utilizado está a contento além de ter a vantagem supracitada de possuir grande simplicidade e rapidez de análise.

Além das falsas identificações, ainda houve uma outra dificuldade em relação a sombras. Pode-se perceber que mesmo a mão sendo toda cor de pele, o algoritmo identificador não classifica essas regiões menos iluminadas como pele. Novamente a justificativa para o uso deste algoritmo é a mesma explicada anteriormente. Foi percebido que os outros algoritmos também seriam falhos em alguns momentos e ainda demandariam mais processamento e tempo para as análises.

4.5 CONTAGEM DE DEDOS NO INTERIOR DE CLUSTERS

Após serem determinadas as dimensões dos objetos com presença de cor de pele, o próximo passo seria escolher aqueles que pudessem ser bons candidatos a serem considerados como mão. Algumas abordagens foram estudadas para essa tarefa como realizar detecção de bordas para encontrar contornos e transformada

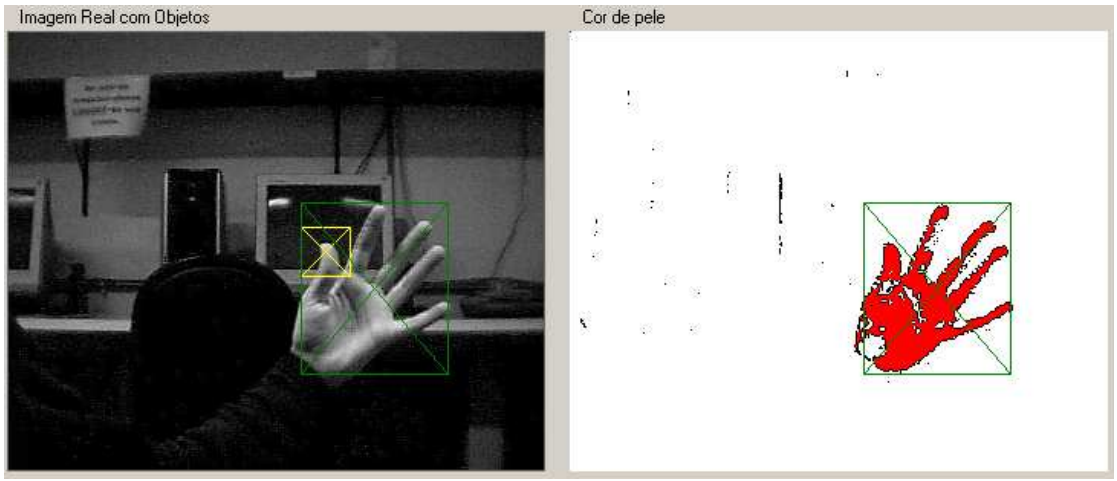


Figura 4.20: Janela mostrando o redimensionamento da caixa se adaptando ao objeto cor de pele

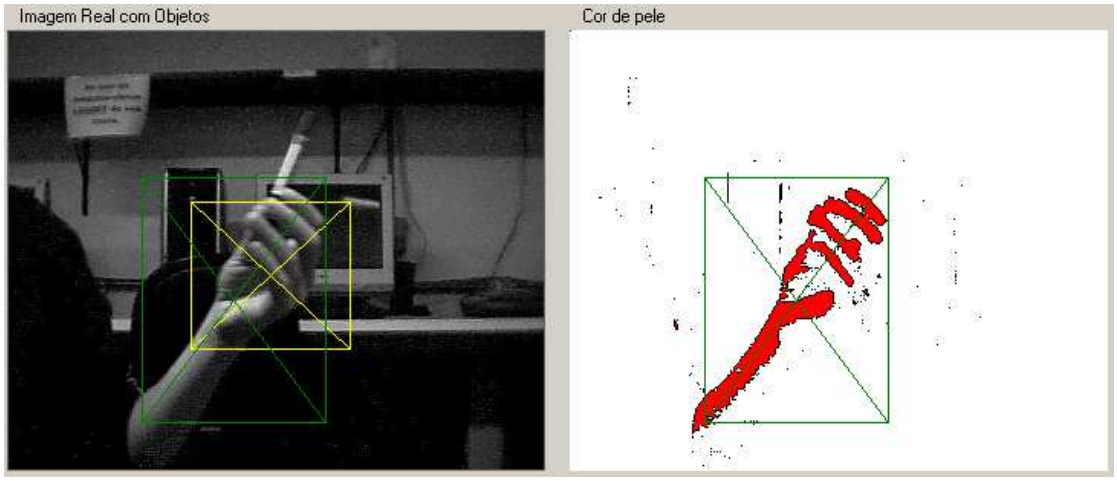


Figura 4.21: Janela mostrando outro redimensionamento da caixa se adaptando ao objeto cor de pele

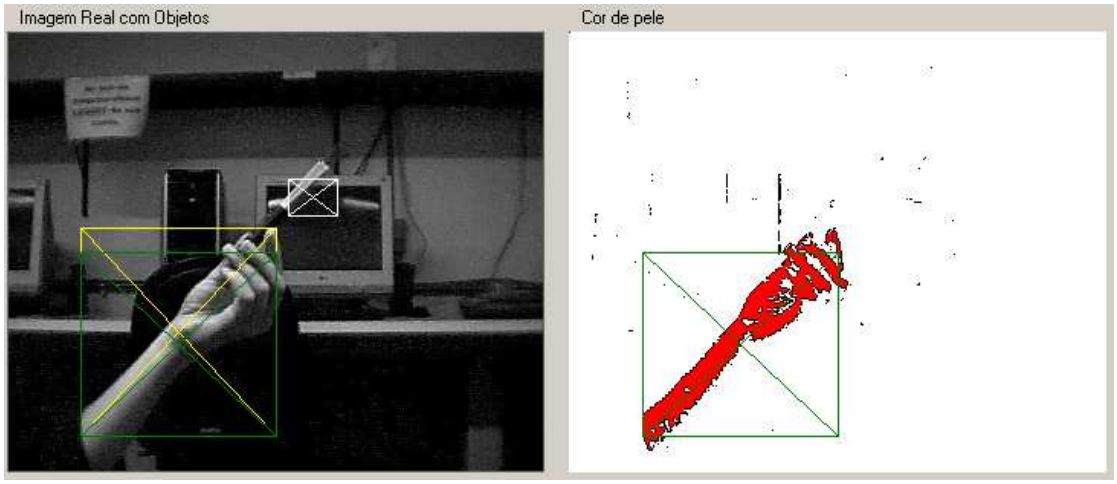


Figura 4.22: Janela mostrando o redimensionamento da caixa se adaptando ao objeto cor de pele e ignorando o objeto sem cor de pele

de Hough [12] para encontrar linhas paralelas que fossem interpretadas como dedos. Porém nenhuma dessas abordagens rendeu algum resultado próximo de satisfatório.

A transformada de Hough gera um diagrama que indica a presença de linhas predominantes na imagem. Alguns exemplos simples da produção desse diagrama se encontram a seguir nas Figuras 4.24 e 4.26.

Para realizar a transformada de Hough é preciso detectar as bordas da imagem. As Figuras 4.23 e 4.25 mostram as linhas originais em (a) e suas respectivas bordas em (b). É possível perceber que o diagrama de Hough indica a predominância de linhas a 90° no caso das linhas verticais (4.24) e de linhas a 45° nas linhas diagonais (4.26).

No caso de uma imagem com linhas com diversos ângulos a lógica é exatamente a mesma. O resultado de um caso desses pode ser visto nas Figuras 4.27 e 4.28. Pode-se perceber a predominância de linhas com ângulos 90°, 135° e 180° (0°) pois são os pontos no diagrama onde ser observa a cor clara mais evidente.

Esses últimos exemplos são consideravelmente mais simples que as imagens capturadas pela *webcam*. As próximas imagens explicitam uma imagem muito mais rica em detalhamento o que dificulta a tarefa de traçar bordas e, principalmente, de achar alguma relação entre o diagrama de Hough e o que seriam dedos (retas relativamente paralelas).

A idéia de utilizar a transformada de Hough partiu do pressuposto de que ao capturar a imagem da mão aberta então seria visível no diagrama uma concentração de linhas com certa angulação que seria interpretada como dedos. O problema é que para usar a transformada é preciso fazer a detecção de bordas resultando numa imagem binária com a menor quantidade de ruídos possível pois do contrário o diagrama final vai se tornando cada vez menos conclusivo.

As dificuldades começam a se mostrar incontornáveis à medida que se observa o resultado da detecção de bordas da imagem capturada pela câmera. Alguns exemplos de filtros utilizados e o resultado de suas filtragens realizadas no MatLab® [2] se encontram nas Figuras 4.29 e 4.30.

Percebe-se que a maioria (Sobel [13], Prewitt [14], Roberts [15] e Laplaciano [14]) das imagens resultantes tem linhas desinteressantes, com exceção do método de Canny [16] que produziu uma imagem tanto quanto satisfatória para prosseguir com a transformada de Hough. O diagrama de Hough resultante então da imagem 4.30(a) pode ser vista na Figura 4.31.

Analisando a Figura 4.31 não se pode concluir com precisão qual seria o ângulo predominante teoricamente imposto pela presença de dedos ou até mesmo se a mão está presente na imagem. Dessa

[2] Mais informações em http://www.mathworks.com/

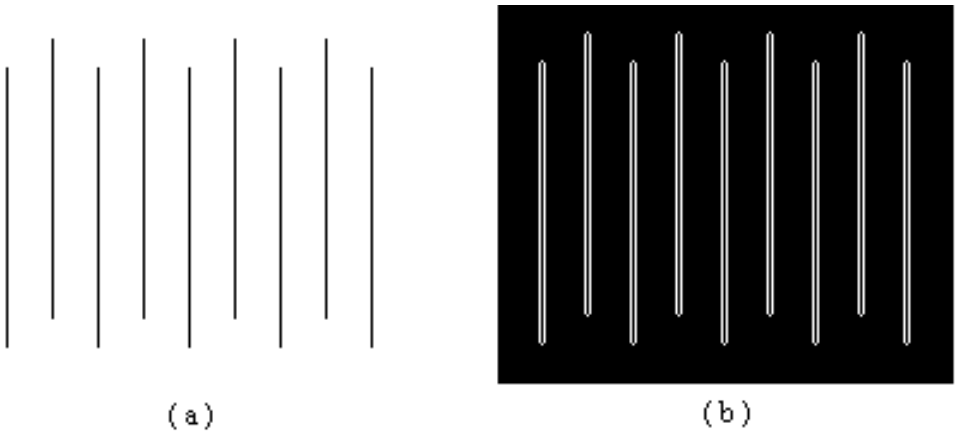


Figura 4.23: Exemplo de linhas verticais e suas bordas

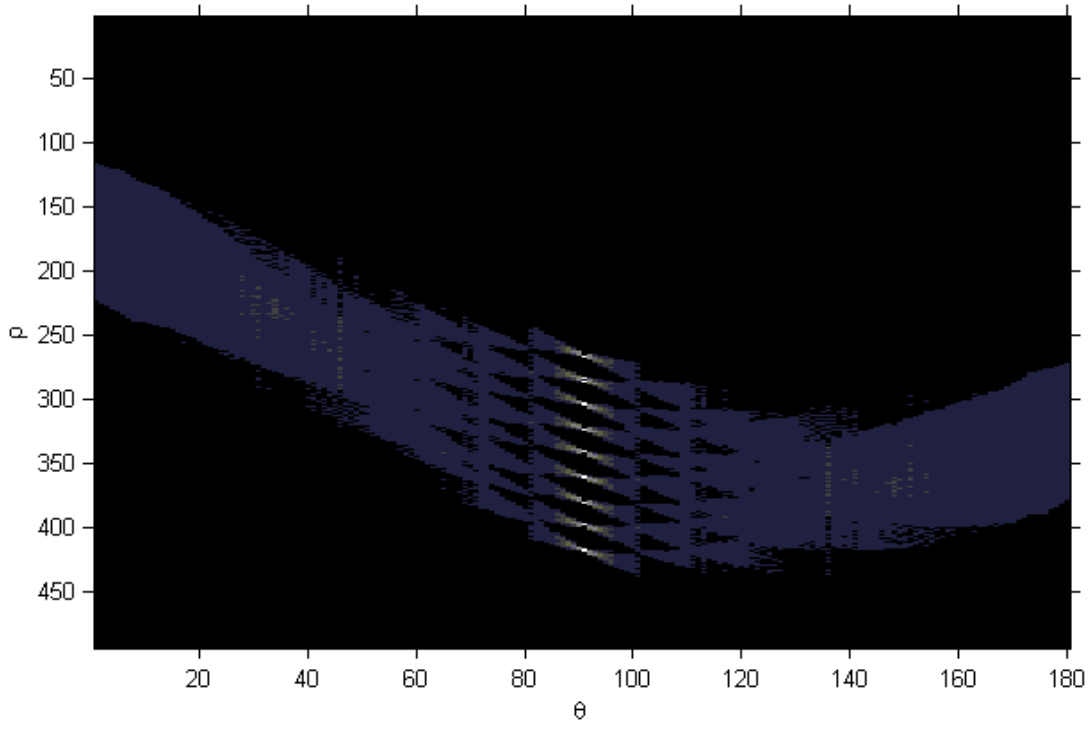


Figura 4.24: Transformada de Hough baseada nas linhas verticais

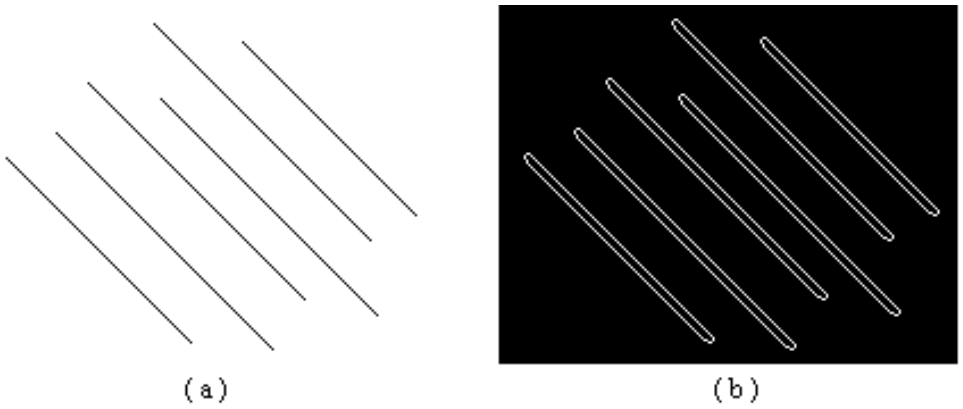


Figura 4.25: Exemplo de linhas diagonais e suas bordas

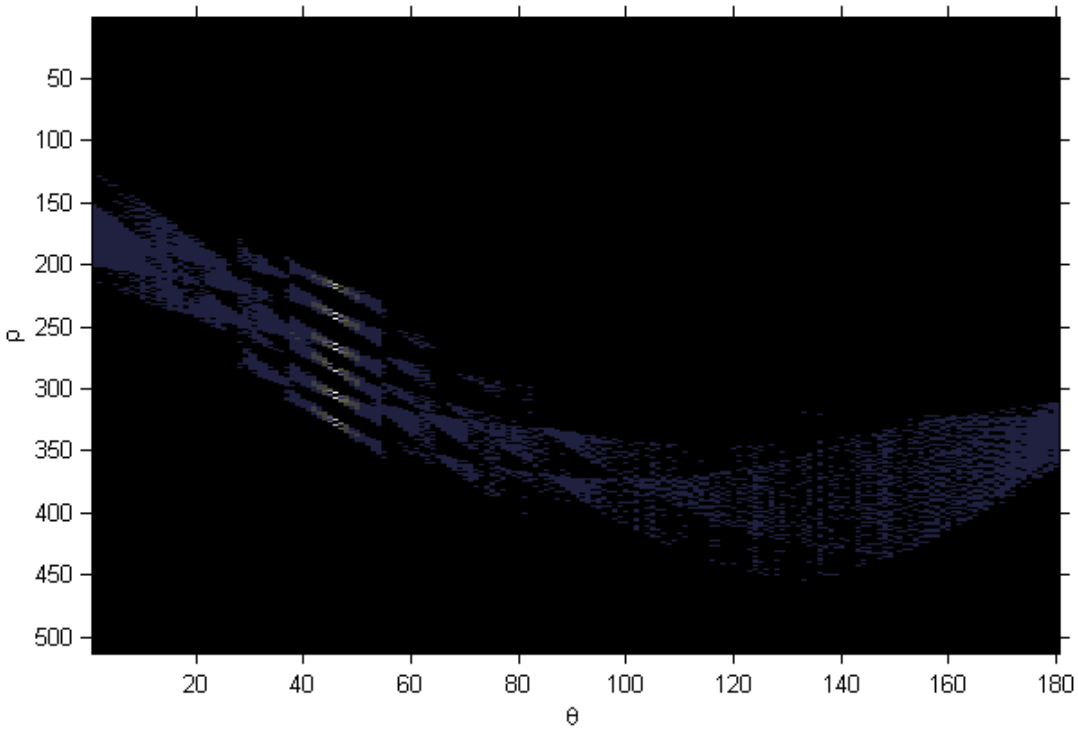


Figura 4.26: Transformada de Hough baseada nas linhas diagonais

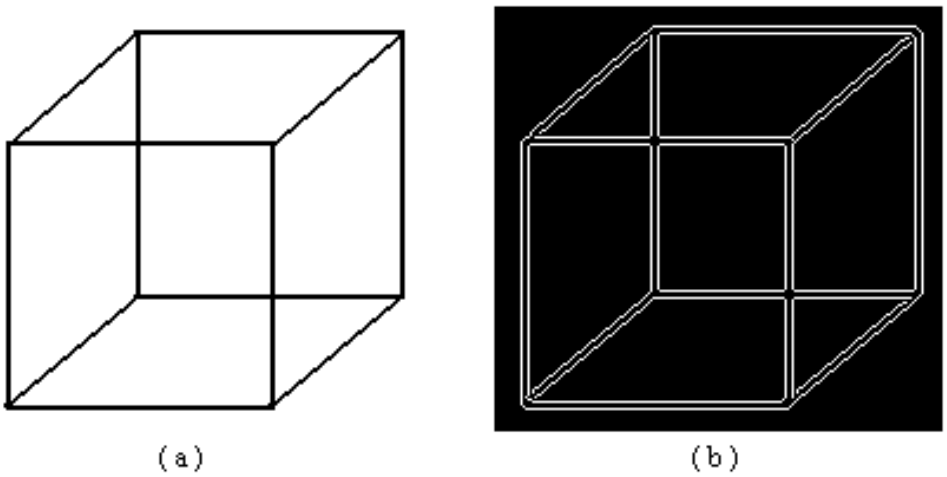


Figura 4.27: Exemplo de linhas verticais, horizontais e diagonais e suas bordas

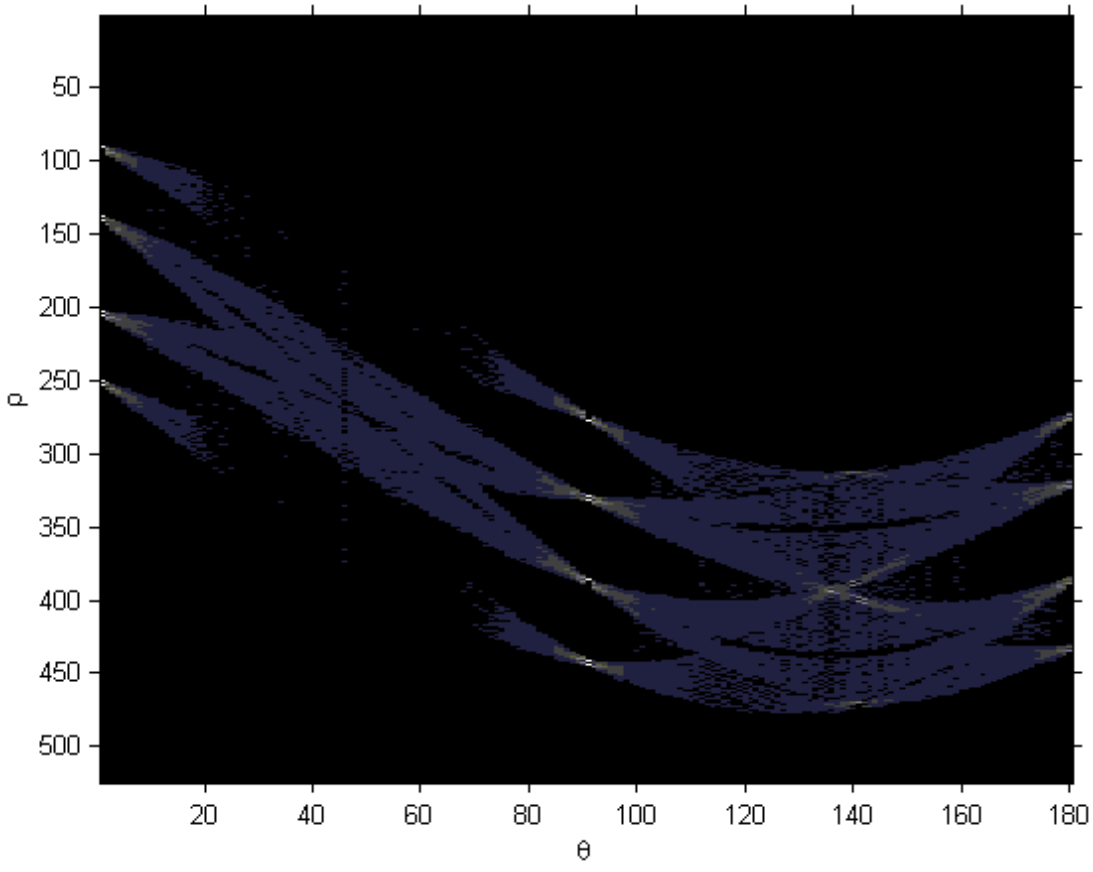


Figura 4.28: Transformada de Hough baseada nas linhas verticais, horizontais e diagonais

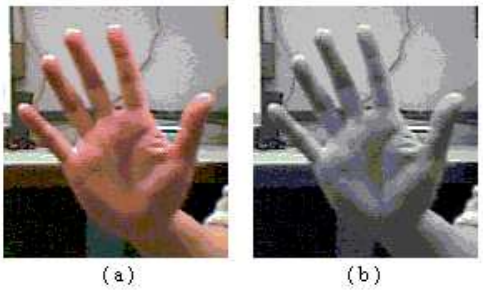


Figura 4.29: Imagem da mão e sua imagem em níveis de cinza

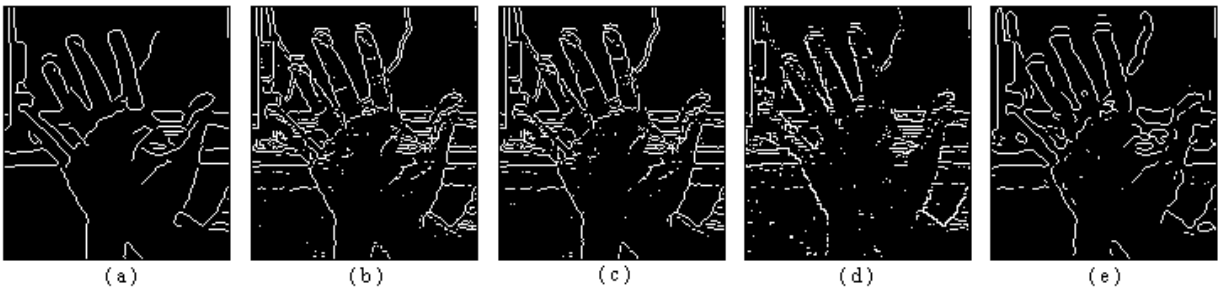


Figura 4.30: Imagens resultantes dos filtros utilizados na imagem em níveis de cinza da mão: (a) Canny, (b) Sobel, (c) Prewitt, (d) Roberts e (e) Laplaciano

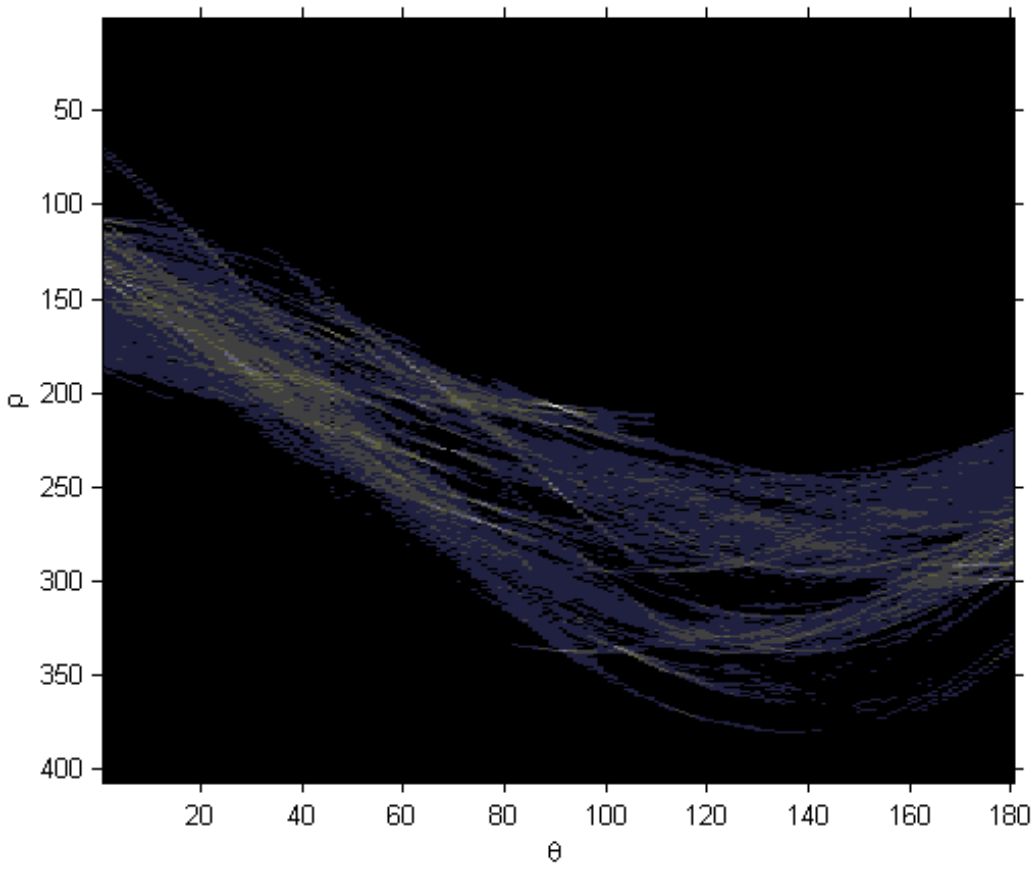


Figura 4.31: Diagrama de Hough para as bordas de uma imagem da mão

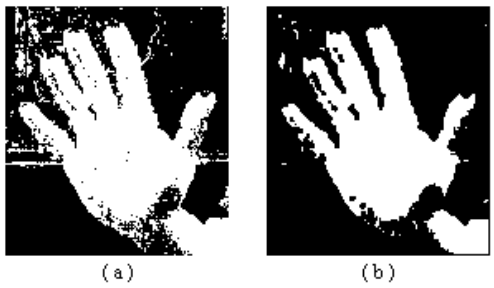


Figura 4.32: Imagem com cor de pele e a mesma imagem filtrada

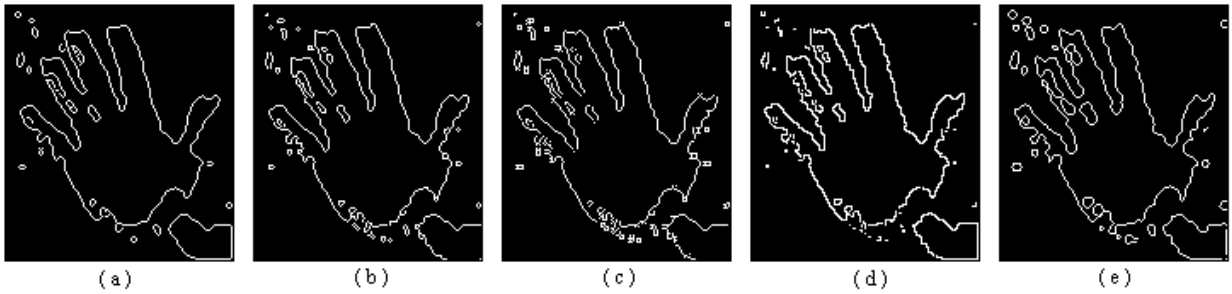


Figura 4.33: Imagens resultantes dos filtros utilizados na imagem binária de cor de pele: (a) Canny, (b) Sobel, (c) Prewitt, (d) Roberts e (e) Laplaciano

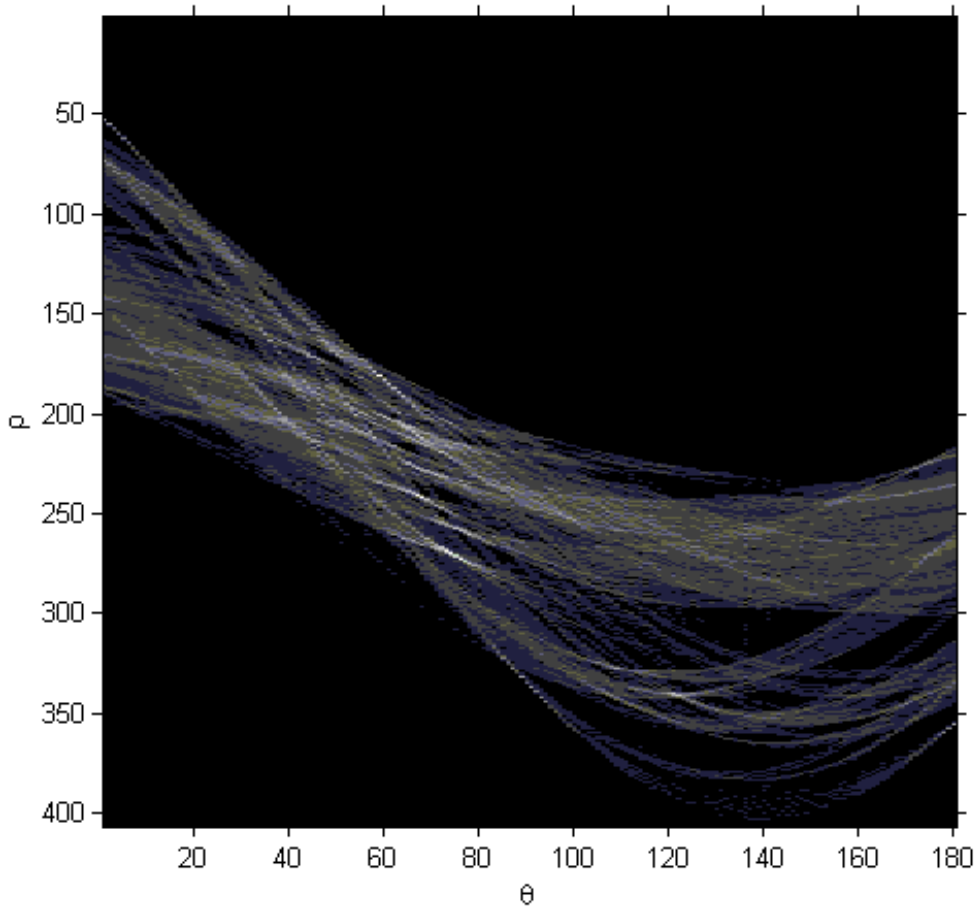


Figura 4.34: Diagrama de Hough para as bordas de uma imagem binária de cor de pele

forma não há muito com que trabalhar com esse diagrama.

Foi passada então para uma abordagem ligeiramente diferente, ao invés de utilizar a imagem real (em níveis de cinza) para determinar as bordas, foi utilizada a imagem binária que indica ou não cor de pele. A imagem retratando cor de pele (4.32(a)) foi filtrada (4.32(b)) e utilizada para encontrar bordas como visto na Figura 4.33.

Novamente o resultado da transformada de Hough (4.34) continua insatisfatório e inconclusivo para a análise de encontrar dedos em uma imagem.

Foi pensada então em uma abordagem diferente. Utilizando a informação das dimensões das caixas e da cor de pele presentes nelas é possível fazer uma contagem de dedos. Foi percebido que mantendo a mão numa posição mais vertical na imagem pode-se traçar uma linha em algum ponto perto do limite superior da caixa que passa por pelo menos três dedos se a mão estiver aberta.

O polegar é desconsiderado por ser bem menor e estar posicionado muito abaixo em relação aos outros. O dedo mínimo também não é levado como certeza de aparecer na contagem pois dependendo da inclinação da mão ele pode aparecer mais baixo que a linha que conta os dedos. Portanto para a mão ser considerada como mão aberta deve-se contar pelo menos três dedos (que provavelmente serão: indicador, médio e anular).

Observou-se a partir de imagens de caixas envolvendo a mão que uma linha posicionada a um sexto de distância do limite superior da caixa (e, obviamente a cinco sextos do limite inferior) possivelmente cruza três dedos da mão aberta (Figura 4.35(a)), podendo, dependendo do caso, também cruzar o dedo mínimo, somando quatro dedos. Ao fechar a mão e se levantar somente um dedo a linha cruza somente esse dedo (Figura 4.35(c)). O mesmo acontece para dois dedos, a linha cruza somente por dois dedos (Figura 4.35(b)).

A questão agora que surge é o método para contar os dedos. A solução foi guardar em um vetor a quantidade de *pixels* de fundo (”não pele”), de objeto (”pele”), de fundo, de objeto - e assim por diante - na ordem em que fossem aparecendo. Dessa forma seria possível determinar quantas contagens de pele existem e deduzir que cada contagem dessas seria um dedo.

O problema dessa técnica é ela não ser imune a ruído. Para obter um método mais robusto algumas condições foram estabelecidas. Primeiramente, toda contagem que marque menos de 4 *pixels* cor de pele será zerado, ou seja, ele é desconsiderado.

A outra melhoria no processo é um tanto mais elaborada. Sabe-se que as larguras dos dedos são muito

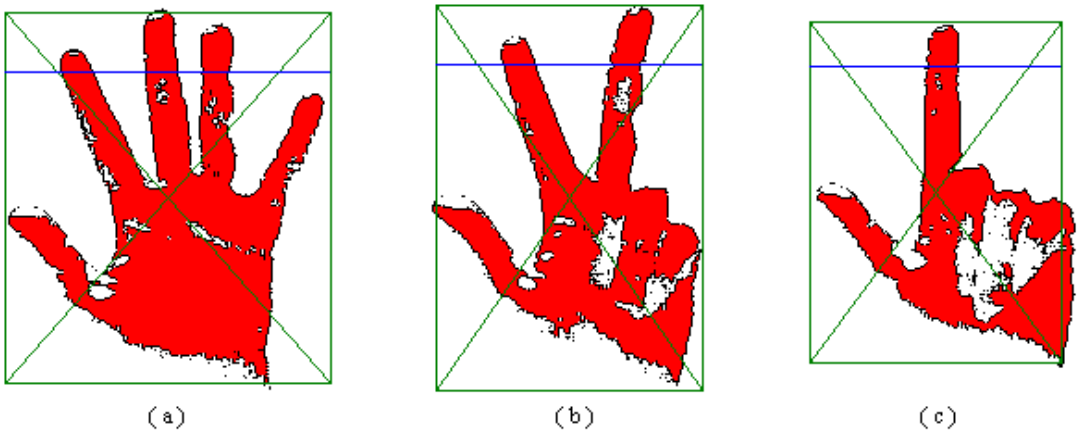


Figura 4.35: Exemplo de mão em diferentes posturas

semelhantes entre si, ou seja, o dedo indicador tem praticamente a mesma largura do dedo anular e assim por diante. Dessa forma é deduzido que ao se fazer a média e o desvio padrão da largura dos dedos tende-se a encontrar um desvio padrão pequeno perto da média aritmética.

$$media = \overline{x} = \frac{1}{N}\sum_{n=1}^{N} x_n \tag{4.20}$$

$$desvio\ padrao = \sigma = \sqrt{\frac{1}{N-1}\sum_{n=1}^{N}(x_n - \overline{x})^2} \tag{4.21}$$

A rotina do programa foi escrita de forma que se for encontrado um desvio padrão com um valor acima de 80% da média então toda essa amostra é desconsiderada. Se o desvio padrão estiver entre 30% e 80% da média então se tenta corrigir a amostra retirando a contagem que destoa das restantes. Os intervalos que indicam quais contagens estão dentro ou fora do padrão são descritos pelas igualdades a seguir.

$$min = \overline{x}(amostras) - 2 \cdot \sigma(amostras) \tag{4.22}$$

$$max = \overline{x}(amostras) + 2 \cdot \sigma(amostras) \tag{4.23}$$

$$min \leq contagem\ correta \leq max \tag{4.24}$$

$$contagem\ destoante < min \tag{4.25}$$

$$contagem\ destoante > max \tag{4.26}$$

A correção retirando as contagens destoantes prossegue até que o desvio padrão se torne abaixo de 30% da média que é o limite aceitável para uma boa amostra de contagem, do contrário a amostra é descartada. Quando a amostra for (ou alcançar depois da correção) o valor aceitável de desvio padrão abaixo de 30% da média então essa etapa de eliminação de ruídos é terminada.

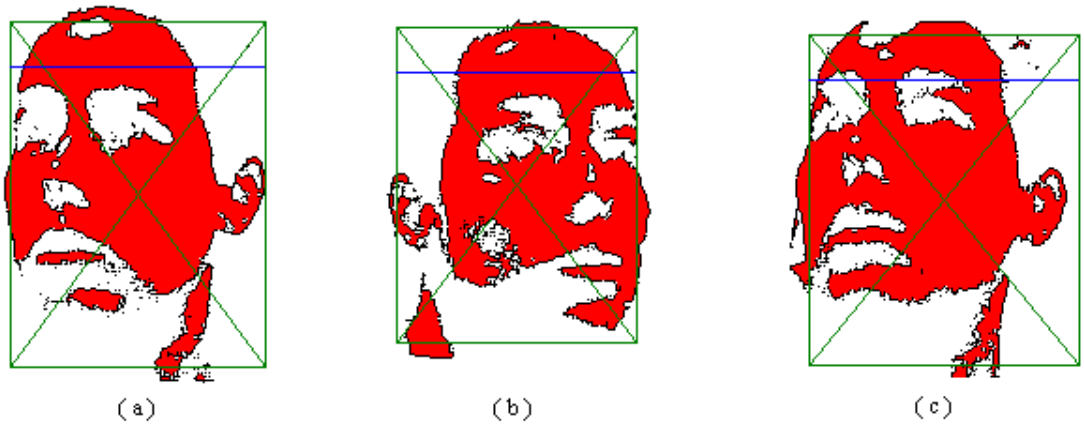


Figura 4.36: Exemplo de rosto em diferentes posições

Quando se termina a eliminação de ruídos é esperado que as contagens de conjuntos de *pixels* cor de pele restantes sejam exatamente os dedos que se procura. Após a limpeza fica mais fácil contar de fato quantos dedos estão em cada caixa, resta somente contar quantos conjuntos de *pixels* cor de pele existem no vetor. Finalmente, é sabido quantos dedos estendidos se possui na imagem.

Vale apontar que com essa técnica de média e desvio padrão, o problema de identificar dedos erroneamente ao se capturar o rosto (outro objeto com cor de pele frequentemente capturado pela câmera) é drasticamente diminuído pois foi também imposto um filtro que se a contagem de cor de pele for maior que 30% da largura da caixa (Figura 4.36(a) e (b))então essa contagem é descartada sem risco de ser considerada como um dedo.

Outro ponto importante é o fato de que as larguras contadas no rosto não são tão regulares quanto os dedos (Figura 4.36(c)), caindo também na filtragem de correção de desvios padrão acima de 30% da média.

Antes de considerar a contagem de dedos de um quadro capturado como sendo uma contagem decisiva para dizer se existe um, dois ou mais de dois dedos estendidos, é realizada uma verificação de que pelo menos dez quadros seguidos tenham essa mesma informação para se certificar que aquela postura da mão está realmente sendo tomada e que aquela contagem não foi fruto de ruído que porventura não tenha sido filtrado pela técnica.

As Figuras 4.37, 4.39 e 4.38 exemplificam mais imagens capturadas variando um pouco a angulação da posição da mão para demonstrar que mesmo com certa variação na postura o programa é capaz também de realizar uma contagem de dedos acertada.

A Tabela 4.1 retrata algumas amostras de contagens realizadas pelo programa. As contagens dessa Tabela são realizadas antes de qualquer filtragem e mostram a quantidade de *pixels* de ”pele” ou ”não-

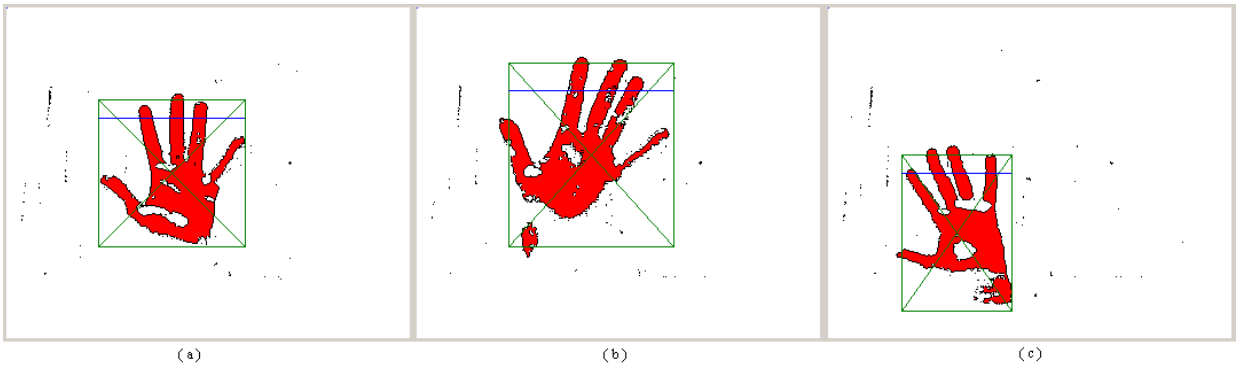


Figura 4.37: Exemplos de mão aberta (três ou quatro dedos estendidos)

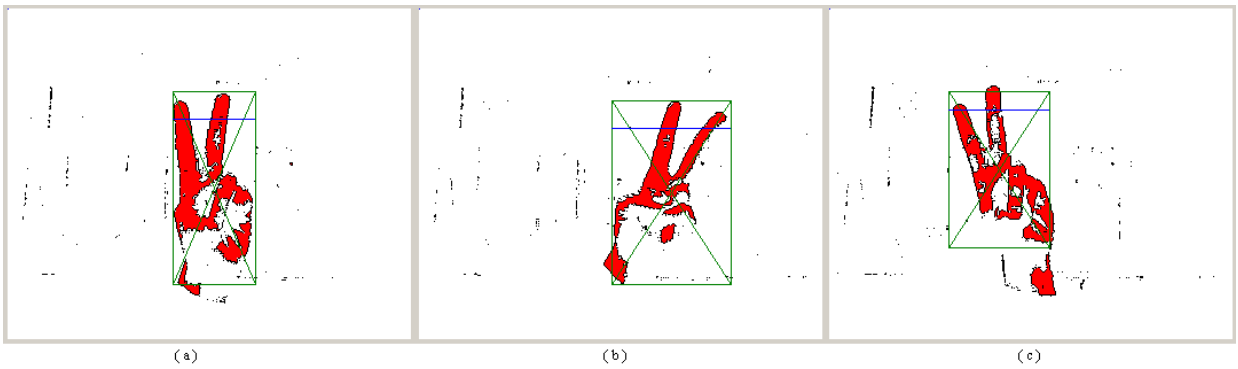


Figura 4.38: Exemplos de mão com dois dedos estendidos

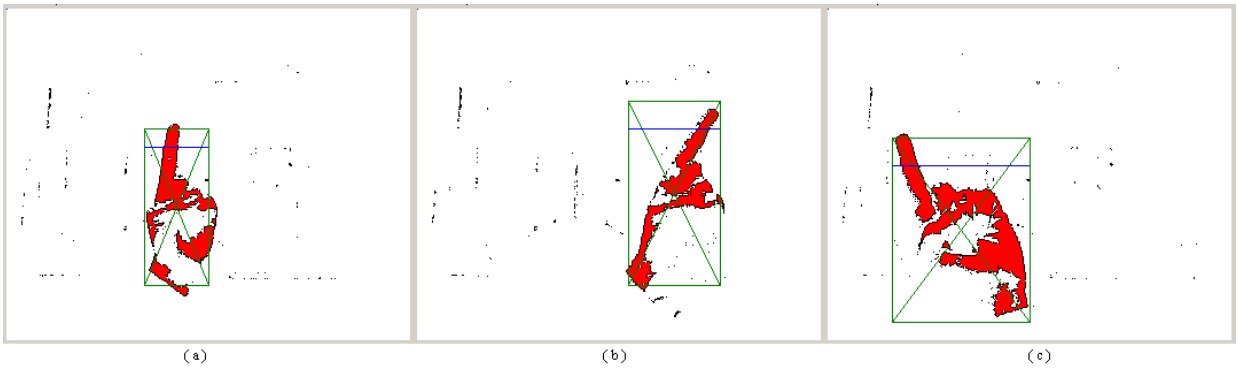


Figura 4.39: Exemplos de mão com um dedo estendido

#	Fundo	Pele	Fundo	Pele	Fundo	Pele	Fundo	Pele	Fundo
1	29	13	14	12	16	11	15	10	6
2	29	12	12	15	14	11	29	-	-
3	31	16	14	18	19	15	35	-	-
4	28	22	24	27	30	25	31	-	-
5	28	14	16	17	16	13	31	-	-
6	34	14	12	16	13	-	-	-	-
7	34	11	13	13	10	-	-	-	-
8	32	16	17	18	16	-	-	-	-
9	39	20	17	23	21	-	-	-	-
10	41	25	21	27	26	-	-	-	-
11	41	20	20	-	-	-	-	-	-
12	40	18	3	-	-	-	-	-	-
13	40	12	20	-	-	-	-	-	-
14	40	19	17	-	-	-	-	-	-
15	39	17	15	-	-	-	-	-	-
16	31	8	15	31	16	9	35	-	-
17	40	5	2	6	5	13	31	-	-
18	32	12	10	20	21	-	-	-	-
19	23	3	5	2	32	-	-	-	-
20	11	52	3	-	-	-	-	-	-

Obs.: ”Fundo” e ”Pele” são contagens em *pixels* de cada uma destas categorias

Tabela 4.1: Tabela com amostras contando pele e fundo

#	Média Arit.	Desvio Padrão	λ (%)	Menor Cont.	τ (%)	Problema Constatado	Decisão Tomada	Provável Captura
1	11,5	1,3	11,2	10	10,3	sem problema	ok	4 dedos
2	12,7	2,1	16,4	11	12,3	sem problema	ok	3 dedos
3	16,3	1,5	9,4	15	12,2	sem problema	ok	3 dedos
4	24,7	2,5	10,2	22	14,4	sem problema	ok	3 dedos
5	14,7	2,1	14,2	13	12,6	sem problema	ok	3 dedos
6	15,0	1,4	9,4	14	18,0	sem problema	ok	2 dedos
7	12,0	1,4	11,8	11	16,0	sem problema	ok	2 dedos
8	17,0	1,4	8,3	16	18,2	sem problema	ok	2 dedos
9	21,5	2,1	9,9	20	19,2	sem problema	ok	2 dedos
10	26,0	1,4	5,4	25	19,3	sem problema	ok	2 dedos
11	20,0	não há	0,0	20	24,7	sem problema	ok	1 dedo
12	18,0	não há	0,0	18	29,5	sem problema	ok	1 dedo
13	12,0	não há	0,0	12	16,7	sem problema	ok	1 dedo
14	19,0	não há	0,0	19	25,0	sem problema	ok	1 dedo
15	17,0	não há	0,0	17	23,9	sem problema	ok	1 dedo
16	16,0	13,0	81,3	8	21,4	DP > 80% da média	descarte	olhos
17	8,0	4,4	54,5	5	12,7	DP > 30% da média	correção e ok	dedos com ruído
18	16,0	5,7	35,4	12	21,1	DP > 30% da média	correção e descarte	dedos juntos ou cabeça
19	2,5	0,7	28,3	2	4,6	pele < 4 *pixels*	correção e anulação	ruídos
20	52,0	0,0	0,0	52	78,8	pele > 30% da caixa	correção e anulação	testa ou braço

λ: desvio padrão/média aritmética (%)

τ: maior contagem de pele/largura da caixa (%)

Tabela 4.2: Tabela com verificações matemáticas para validação de amostras

pele” (fundo) encontradas na varredura da linha de interesse (um sexto do topo da caixa).

A Tabela 4.2 mostra as análises matemáticas feitas pelo programa que definem se a amostra é válida ou não, e, no caso de ser inválida, se poderia haver correção ou se a amostra seria descartada. As 15 primeiras amostras são consideradas válidas e retratam uma interpretação de três ou quatro dedos estendidos (cinco primeiras amostras), dois dedos (sexta amostra até a décima) e ainda um dedo estendido (décima primeira a décima quinta amostra). As últimas cinco amostras são consideradas a princípio inválidas porém podem ainda sofrer correção dependendo do caso.

Na décima sexta amostra o que se observa é uma contagem de poucos *pixels* na esquerda, muitos *pixels* no meio e poucos *pixels* na direita como visto na tabela 4.1. É provável que se tenha capturado na imagem o rosto, mais especificamente a região dos olhos. O problema com essa amostra é o alto desvio padrão (maior que 80% da média) gerado pelas contagens irregulares de pele. A reação do programa a esse tipo de amostra é o descarte.

Na décima sétima amostra se percebe que as duas primeiras contagens são pequenas e estão separadas por uma contagem de fundo muito pequena. A terceira e última contagem de pele tem praticamente o valor das duas primeiras contagens juntas o que nos faz deduzir que esta amostra provavelmente foi a captura de dois dedos estendidos porém no primeiro dedo foi capturado também ruído. A análise matemática do programa encontra o problema como sendo o desvio padrão acima de 30% da média e tenta então corrigir a amostra. A correção eliminará a amostra destoante e o resultado será a interpretação acertada de dois dedos estendidos.

Na décima oitava amostra são contabilizadas duas contagem de pele, uma pequena e uma grande. As prováveis capturas nesse caso são três dedos estendidos porém dois deles juntos (maior contagem) ou ainda alguma outra região da cabeça vista lateralmente. De qualquer forma, o programa constata um problema ao analisar as contagens pois o desvio padrão, como o caso acima, é maior que 30% da média. Ao se tentar realizar uma correção não se consegue pois o critério de eliminação de contagens destoantes (fora do intervalo de média mais/menos duas vezes o desvio padrão) não se aplica nesse caso, não eliminando nenhuma das contagens. Neste caso o algoritmo tenta sem sucesso mais algumas vezes a correção. A reação final nesse caso é o descarte da amostra.

A décima nona amostra exibe contagens muito pequenas de *pixels* de pele o que pode significar a captura de ruídos. A resposta do programa é tentar corrigir esse problema eliminando contagens de pele de valor muito baixo. Como não há mais nenhuma contagem significativa a amostra acaba sendo anulada no final.

Por fim, a vigésima amostra tem uma contagem de pele que de fato é única porém exageradamente grande, ou, em condições matemáticas do programa, maior que 30% da caixa. A provável captura nessa caso poderia ser a testa ou alguma outra região contínua muito grande de pele (braço, por exemplo). A reação do programa é a tentativa de correção anulando essa contagem. Na falta de outra contagem de pele, a amostra inteira acaba sendo desprezada.

Os resultados alcançados por esse processo foram de fato satisfatórios tendo o programa reconhecido com grande quantidade de acerto as posturas da mão e ainda tendo uma baixa quantidade de interpretações errôneas das mesmas.

4.6 COMANDOS PARA O SISTEMA

O último passo do programa foi passar ao sistema operacional o resultado tanto do movimento resultante (para mover o *mouse*) quanto à postura final da mão (para clicar o *mouse*).

Nesta etapa, para obter o movimento resultante entre dois quadros deve-se somar os vetores de todos os macroblocos no interior das caixas que possuem cor de pele. Se não houver caixa de cor de pele o mouse não se movimenta mesmo que haja outros movimentos no quadro. Ao se encontrar os vetores resultantes dos eixos horizontal e vertical então se transmite esses dois parâmetros ao sistema, de forma que o mouse obedeça ao vetor final fruto dessas duas componentes.

Para clicar o *mouse*, o procedimento é semelhante. Na Seção anterior foi mostrado como se chegar a contagem de dedos, então agora se faz necessário somente associar cada uma das posturas a um clique. As associações escolhidas são mostradas na tabela que se segue.

Postura	Ação
Mão aberta (três ou mais dedos estendidos)	sem clique
Um dedo estendido	clique esquerdo
Dois dedos estendidos	clique direito
Mão aberta (após algum clique)	desfazer o clique

Tabela 4.3: Tabela com associações de posturas e cliques

É importante citar que em qualquer uma das posturas é possível também mover o *mouse* normalmente.

Dessa forma, para clicar com o botão esquerdo deve-se fechar a mão deixando somente um dedo estendido (e esperar que o programa faça o reconhecimento dessa postura esperando pelo menos dez

quadros nesse mesmo gesto, um tempo em torno de 330 ms) e depois abrir a mão para soltar o clique. Se desejar arrastar algo é só clicar realizando a postura com um dedo estendido e não desfazer o clique (manter a mão ainda somente com um dedo levantado).

O mesmo vale para o clique do botão direito do *mouse* sendo a única diferença a postura com dois dedos estendidos ao invés de um. Para soltar o clique se faz necessário abrir a mão como descrito acima.

5 CONCLUSÃO

Porque tão importante quanto o caminho percorrido é também onde se consegue chegar.

5.1 CONCLUSÃO

O presente trabalho proporcionou não só o aprendizado de diversas técnicas de processamento de imagens porém também um ótimo instrumento de aplicação das mesmas. As técnicas utilizadas e até mesmo as que somente foram estudadas para o presente projeto foram de demasiada importância ao conhecimento da área de imagens.

O projeto possibilitou a junção prática de técnicas já conhecidas como erosão, dilatação, rotulação, estimação de movimento, soma de diferenças absolutas... e também de outros métodos concebidos no decorrer do trabalho para resolver situações inusitadas encontradas tais como agrupamento de vetores de movimento para identificar objetos, expansão e contração de regiões de interesse por constatação de presença de cores específicas e ainda contagem de dedos por média aritmética e desvio padrão.

Foi constatado que inúmeras ferramentas estão disponíveis à utilização no âmbito da estimação de movimento, identificação de objetos, reconhecimento de cores etc., porém nem todas elas puderam ser aplicadas a esse projeto por motivos de falta de praticidade, elevado tempo de processamento ou mesmo inconclusão de resultados em casos gerais ou particulares.

Uma dificuldade encontrada foi justamente a falta de adequação no projeto de técnicas já apreciadas como transformada (*Hough*) ou algoritmos não supervisionados de identificação de objetos (*K-Means*, *C-Means*, mapa de *Kohonen*) que auxiliassem na resolução das questões envolvidas.

Muito se tentou inserir métodos matemáticos e científicos porém a maioria das melhores saídas foi embasada em lógicas mais simples ou rotinas computacionais de mais baixo processamento. Algumas das alternativas mais conhecidas foram preteridas por não serem a solução mais otimizada.

Outro ponto a se considerar nesse escopo é que a todo o momento foi ponderada a relação qualidade *versus* complexidade. Em muitos pontos deste projeto optou-se por soluções de qualidade considerada apenas suficiente e não a melhor possível uma vez que não se teria o benefício de todo tempo e processamento disponíveis pois o projeto foi concebido para ser executado o máximo possível em tempo

real como um aplicativo auxiliar paralelo a outros, inclusive em máquinas de tecnologia menos recente.

A interface visual, em sua versão final, pode ser executada em um computador pessoal comum o que faz a interface uma alternativa viável para uso geral. Outro ponto positivo é o fato da interface necessitar somente de um dispositivo de captura de imagens trivial como uma *webcam*.

Finalmente, a proposta de gerar uma interface usuário-computador para controle não só de movimento do *mouse* como também de certas ações como cliques direito, esquerdo e arrasto foram desempenhadas com sucesso. O sistema ainda pode ser aprimorado para aumentar a robustez porém os resultados encontrados certamente cumprem as expectativas.

5.2 PROPOSTAS PARA TRABALHOS FUTUROS

Como foi dito, o programa pode ainda ser melhorado de forma a ter aumentada sua robustez.

Algumas sugestões são o uso de técnicas de busca na estimação de movimento que possam melhorar ainda mais o resultado, tentando ainda outros tamanhos de macroblocos e sobreposição (*overlapping*) no deslocamento desses macroblocos sem que o processamento envolvido não seja comprometido.

Outro ponto a se considerar é a utilização de um algoritmo de identificação de cor de pele que forneça menos falsos positivos. A maior problemática constatada no projeto foi justamente essa identificação errônea de cor de pele o que pode atrapalhar o redimensionamento das caixas de objetos de interesse.

Uma última questão importante a citar é que se tente utilizar artifícios e técnicas matemáticas mais conhecidas que sejam eficientes e, de preferência, de baixo custo computacional que possam inserir mais ciência ao projeto para que pudessem ser publicados artigos sobre o assunto.

REFERÊNCIAS BIBLIOGRÁFICAS

[1] OLIVEIRA, R. G. de; ANDRADE, J. B. de. *Implementação de Interface Visual Baseada em Estimação de Movimentos e Posição para Controle de Mouse e Teclado*. 2005. Projeto Final de Graduação - Universidade de Brasília.

[2] HUNT, R. W. G. *The Reproduction of Colour*. [S.l.]: Wiley, 2004.

[3] GONZALEZ, R. C.; WOODS, R. E. *Processamento de Imagens Digitais*. [S.l.]: Edgard Blücher LTDA, 2000.

[4] COLANTONI, P.; AL. *Color Space Transformations*. 2004. Couleur.org, disponível em *http://colantoni.nerim.net/download/colorspacetransform-1.0.pdf*.

[5] RICHARDSON, I. E. G. *H.264 and MPEG-4 Video Compression*. [S.l.]: Wiley, 2003.

[6] VEZHNEVETS, V.; SAZONOV, V.; ANDREEVA, A. A survey on pixel-based skin color detection techniques. *Proceedings Graphicon*, pp. 85-92, 2003.

[7] SINGH, S. K. et al. A robust skin color based face detection algorithm. *Tamkang Journal of Science and Engineering*, Vol. 6, No. 4, pp. 227-234, 2003.

[8] CHANG, H.; ROBLES, U. *Face Detection*. 2000. University of Stanford, disponível em *http://www-cs-students.stanford.edu/∼robles/ee368/main.html*.

[9] KANUNGO, T. et al. An efficient k-means clustering algorithm: Analysis and implementation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 24, Issue 7, pp. 881-892, 2002.

[10] LIM, Y. W.; LEE, S. U. On the color image segmentation algorithm based on the thresholding and the fuzzy c-means techniques. *Pattern Recognition*, Volume 23, Issue 9, pp. 935-952, 1990.

[11] MAHONEN, P.; HAKALA, P. Automated source classification using a kohonen network. *Astrophysical Journal Letters*, Volume 452, pp. L77-L80, 1995.

[12] DUDA, R. O.; HART, P. E. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, Volume 15, Issue 1, pp. 11-15, 1972.

[13] ALEXOPOULOS, L. G.; ERICKSON, G. R.; GUILAK, F. A method for quantifying cell size from differential interference contrast images: validation and application to osmotically stressed chondrocytes. *Journal of Microscopy*, Volume 205, Issue 2, pp. 125-135, 2002.

[14] ATKOCIUNAS, E. et al. Image processing in road traffic analysis. *Nonlinear Analysis: Modelling and Control*, Volume 10, No. 4, pp. 315-332, 2005.

[15] ROBERTS, L. G. Machine perception of three-dimensional solids. *Computer Methods in Image Analysis*, 1977.

[16] CANNY, J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 8, Issue 6, pp. 679-698, 1986.

ANEXOS

I. LISTA DE MÉTODOS DO PROGRAMA INTERFACE VISUAL

O código fonte do programa Interface Visual assim como seu arquivo executável se encontram em uma mídia gravável (CD) que está em anexo junto a este trabalho. A seguir segue a lista de métodos contidos nesse programa.

I.1 UNIDADE PRINCIPAL

Lista de métodos do arquivo unit.cpp:

int ConverteParaInteiro(String s)
bool TForm1::LerArquivo()
bool TForm1::TestaArquivo()
void __fastcall TForm1::Timer1Timer(TObject *Sender)
void __fastcall TForm1::FormCreate(TObject *Sender)
void __fastcall TForm1::Abrir1Click(TObject *Sender)
void __fastcall TForm1::Sair1Click(TObject *Sender)
void __fastcall TForm1::SempreVisivel1Click(TObject *Sender)
void __fastcall TForm1::Iniciar1Click(TObject *Sender)
void __fastcall TForm1::Finalizar1Click(TObject *Sender)
void __fastcall TForm1::DetectarMovimentos1Click(TObject *Sender)
void __fastcall TForm1::Aes1Click(TObject *Sender)
void __fastcall TForm1::Mostrarresultados1Click(TObject *Sender)
void __fastcall TForm1::Mostrarvetores1Click(TObject *Sender)
void __fastcall TForm1::Mostrarobjetos1Click(TObject *Sender)
void __fastcall TForm1::Mostrarcordepele1Click(TObject *Sender)

I.2 REGIÕES

Lista de métodos do arquivo regiao.cpp:

REGIAO::REGIAO()
REGIAO::∼REGIAO()

void REGIAO::Zera()

void REGIAO::Adapta(int w, int h)

bool REGIAO::DefineRegiao(int reg,int tipo,int xi,int yi,int xf,int yf, int tolerancia)

bool REGIAO::DefineCodigos(int reg,int cod0,int cod1,int cod2,int cod3)

void REGIAO::SomaVetores()

void REGIAO::Acoes()

I.3 CAPTURA DE IMAGENS

Lista de métodos do arquivo capturaimagens.cpp:

CAPTURA::CAPTURA()

CAPTURA::∼CAPTURA()

LRESULT CALLBACK CAPTURA::FrameCallbackProc(HWND hWnd, LPVIDEOHDR lpVHdr)

int CAPTURA::IniciarCaptura(HWND JanelaPrincipal)

bool CAPTURA::FinalizarCaptura()

int CAPTURA::Adapta()

bool CAPTURA::Aloca()

void CAPTURA::CapturaFrame()

int CAPTURA::RetornaLargura()

int CAPTURA::RetornaAltura()

void CAPTURA::RetornaCoordenadasCluster()

I.4 DETECÇÃO DE MOVIMENTO

Lista de métodos do arquivo detectarmovimento.cpp:

DETECTARMOVIMENTO::DETECTARMOVIMENTO()

DETECTARMOVIMENTO::∼DETECTARMOVIMENTO()

int DETECTARMOVIMENTO::SAD(int xA, int yA, int dx, int dy)

void DETECTARMOVIMENTO::BuscaD(pixel **FrameAnterior, pixel **FrameAtual)

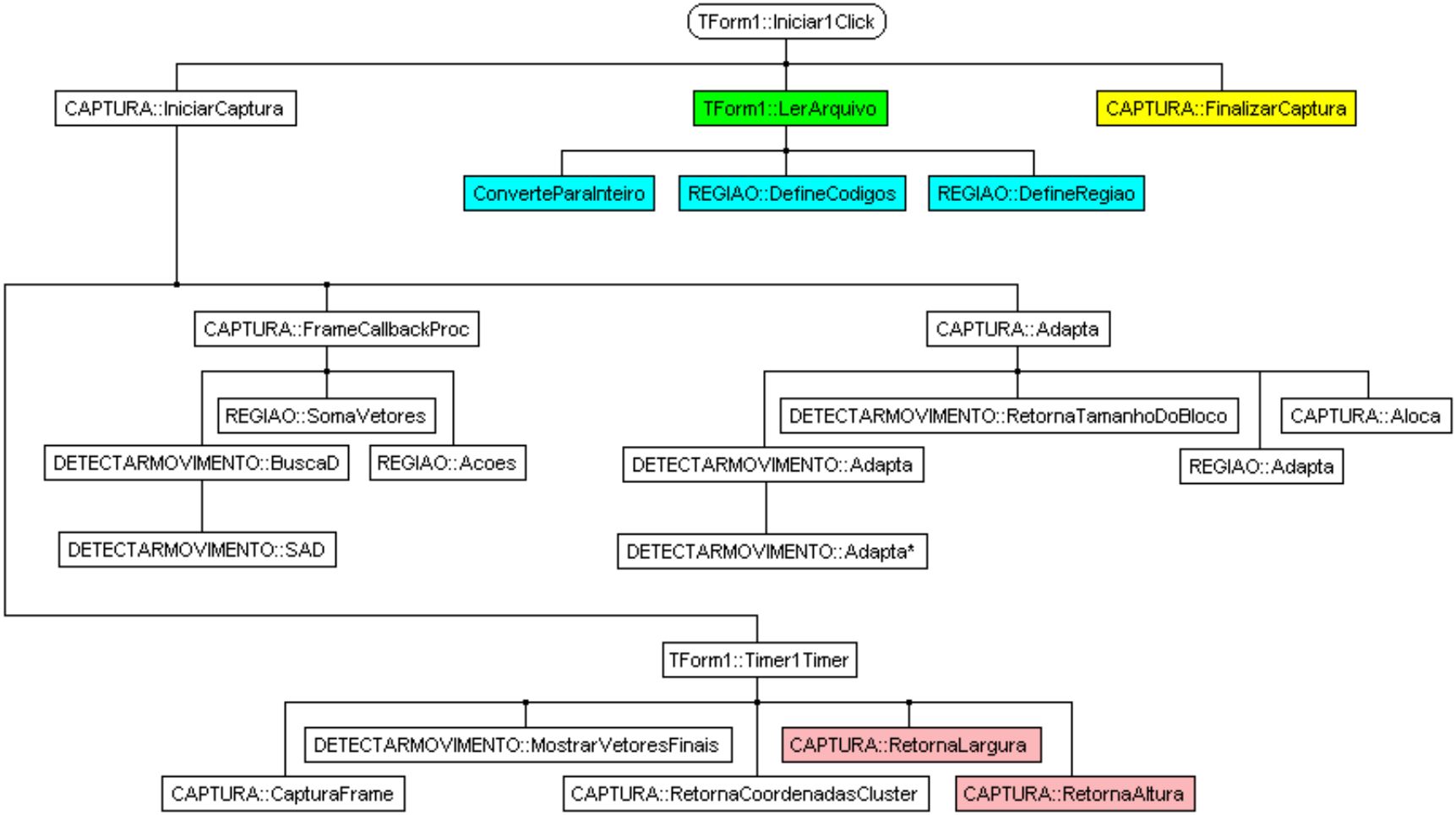
bool DETECTARMOVIMENTO::Adapta(int width, int height)

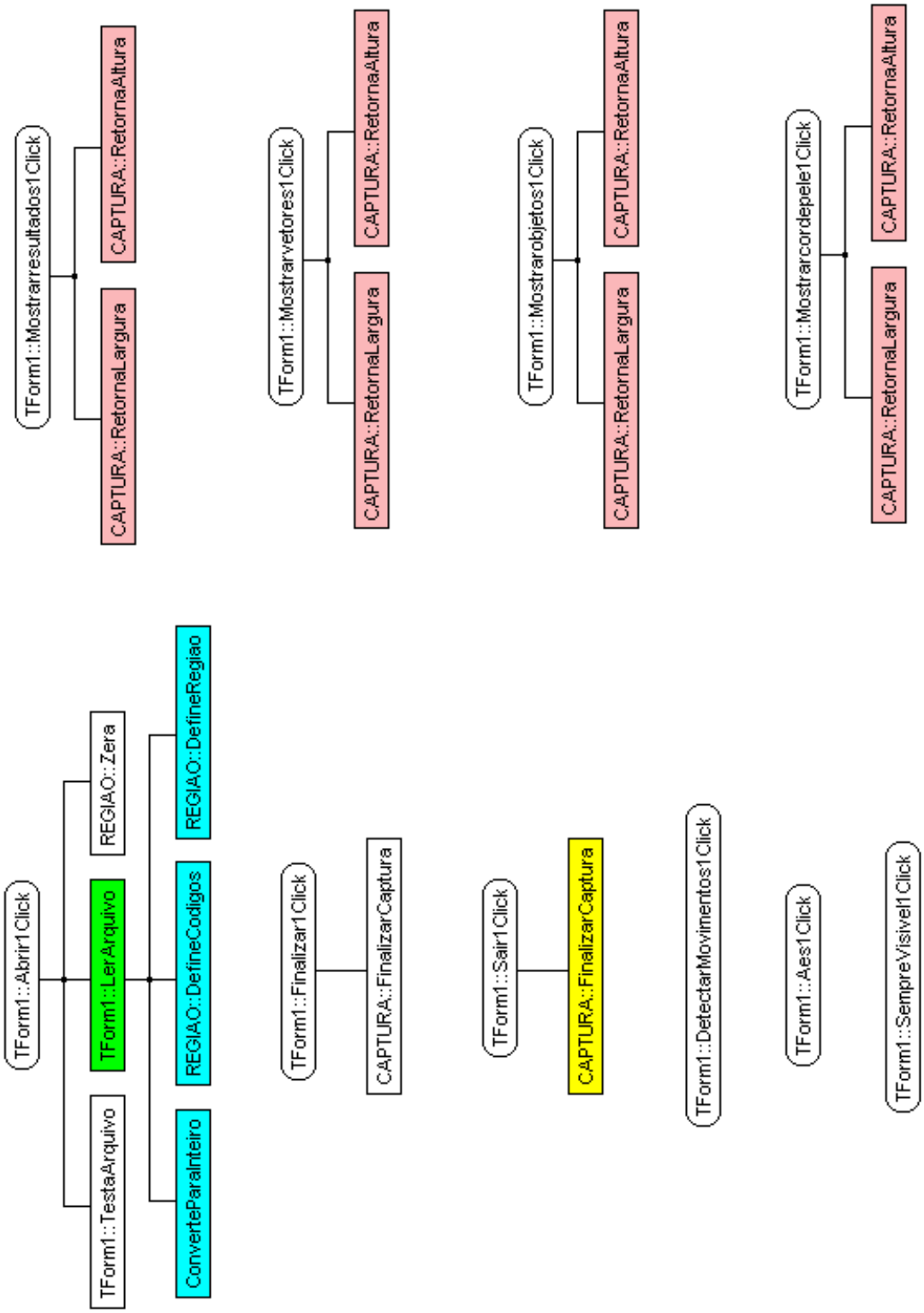
bool DETECTARMOVIMENTO::Adapta()

int DETECTARMOVIMENTO::RetornaTamanhoDoBloco()

void DETECTARMOVIMENTO::MostrarVetoresFinais()

II. GRAFOS DO PROGRAMA INTERFACE VISUAL


TForm1::Iniciar1Click
CAPTURA::IniciarCaptura
TForm1::LerArquivo
CAPTURA::FinalizarCaptura
ConverteParaInteiro
REGIAO::DefineCodigos
REGIAO::DefineRegiao
CAPTURA::FrameCallbackProc
CAPTURA::Adapta
REGIAO::SomaVetores
DETECTARMOVIMENTO::RetornaTamanhoDoBloco
CAPTURA::Aloca
DETECTARMOVIMENTO::BuscaD
REGIAO::Acoes
DETECTARMOVIMENTO::Adapta
REGIAO::Adapta
DETECTARMOVIMENTO::SAD
DETECTARMOVIMENTO::Adapta*
TForm1::Timer1Timer
DETECTARMOVIMENTO::MostrarVetoresFinais
CAPTURA::RetornaLargura
CAPTURA::CapturaFrame
CAPTURA::RetornaCoordenadasCluster
CAPTURA::RetornaAltura



Obs:

- Métodos envoltos em caixas arredondadas são chamados ao se clicar em alguma opção no programa Interface Visual.

- Métodos envoltos em caixas retangulares são chamados internamente por outros métodos.

- Caixas coloridas são chamadas por mais de um método.