Revista TECCEN – Edição Especial – volume 2 – número 1 - março de 2009 – ISSN 1984-0993

Interação Humano - Computador usando Visão Computacional

Bernardo Bucher B. Barbosa[1], Júlio César Silva[2]

[1]UNIFESO, Centro de Ciências e Tecnologia, Curso de Ciência da Computação, brgenius@gmail.com

[2]UNIFESO, Centro de Ciências e Tecnologia, Curso de Ciência da Computação; PUC-Rio, Departamento de Engenharia Civil, jcesarop@gmail.com

Resumo. *Este trabalho visa estudar maneiras de se explorar a Interação Humano Computador, usando Visão Computacional. A idéia tem como objetivo um esforço para tornar o computador mais interativo com o usuário, sem a necessidade da compra de um hardware ou acessório específico para tal. O produto final deste trabalho em desenvolvimento é um software que contempla esta funcionalidade, tornando o computador mais interativo.*

1 Introdução

A proposta deste trabalho é estudar formas de interação do humano com o computador, usando recursos de visão computacional, excluindo dispositivos gráficos como o *mouse* ou teclado, especificamente o uso de *webcam* para servir de interface. A idéia inicial é usar as imagens capturadas através de uma *webcam* para que se possa mover o mouse, usando comandos dados pelas mãos do usuário. Depois de concluído o projeto, este *software* poderá ser usado em várias áreas da computação e até mesmos para meios de entretenimento, ou ainda, para uso rotineiro.

Porém, será necessário que alguns passos básicos sejam feitos antes que se possa encontrar uma aplicação viável para o *software*. Para poder fazer com que o computador de fato entenda o que as imagens capturadas "significam", é necessário uso de técnicas de processamento e reconhecimento de imagens. Tendo as imagens tratadas, pode-se então executar os comandos necessários para que se alcance o objetivo.

Como foi dito anteriormente, será necessário adotar técnicas de processamento e reconhecimento de imagens. Porém, como reconhecer uma imagem? Esta é a chave da questão. Como se pode reconhecer a mão sem um dispositivo de *hardware* acoplado? E mais, como fazê-lo em um computador, que trabalha essencialmente com dados que são discretos? Todas as repostas destas perguntas levam a apenas uma única resposta, o reconhecimento de padrões. Neste caso, alguns padrões se mostram melhores para a finalidade, como o formato da mão e suas cores características (os tons avermelhados, devido ao sangue). Estes fatores terão de ser usados para que o reconhecimento possa ser efetuado. Para alcançar-se o objetivo é necessário ter em mente que neste *software*, o custo computacional das técnicas adotadas tem de ser relativamente baixo, caso contrário sua aplicabilidade será diminuída, para comprovar este fato, basta imaginar que o *software* será usado como ferramenta para a manipulação do mouse e, também, será necessária a captura de imagens de uma *webcam*.

2 Processamento de Imagens

2.1 Formato

Para a pesquisa e o desenvolvimento desta aplicação é necessário que se conceitualize o significado de formato. O formato de um objeto pode ser entendido pelo contorno de sua forma em um espaço 2D ou 3D. Assim, o formato de uma régua é retangular, enquanto, o de uma garrafa é cilíndrico ou cônico. A fim de reconhecer o formato de um objeto é necessário definir um formato que funcione de base de comparação, identificar o formato do objeto em teste, e por fim fazer a comparação entre os dois respeitando certas taxas de variação. Para que se possa a fazer a extração de um contorno de uma imagem adquirida de uma *webcam*, é necessário usar um algoritmo para detecção de bordas de uma imagem, que são os contornos de suas formas. Existem alguns algoritmos para tal, com o de *Roberts*, o Operador de *Sobel*, o método de *Canny*, ou mesmo técnicas de *Relief Mapping* (Mapeamento de Relevo), porém apenas os dois primeiros se mostraram mais eficientes (Figura 1).

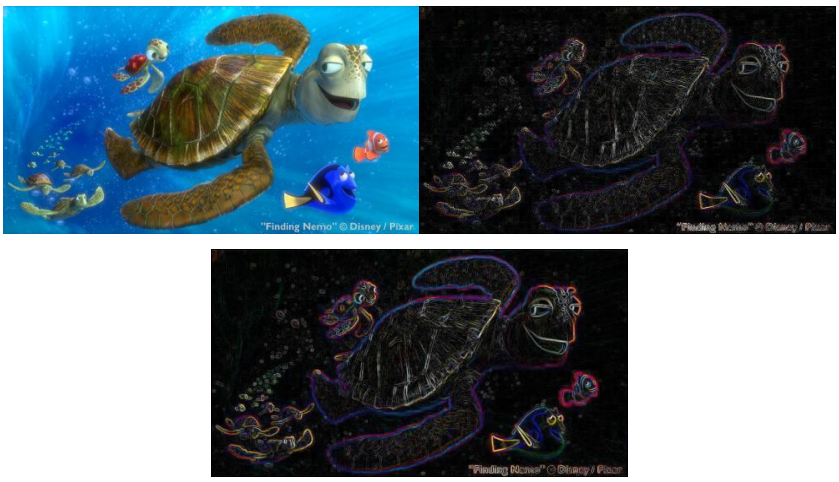

Figura 1. Acima, à esquerda: a imagem original, à direita a mesma imagem quando aplicado o operador de *Roberts*, e a imagem abaixo com o operador de *Sobel* [PDI 2008].

O que caracteriza um algoritmo eficiente é o seu uso de processamento, considerando o resultado da detecção. Quanto menos ruídos e menor uso do processador, mais precisão o algoritmo tem [von Wangenheim 2008]. Pesquisou-se qual melhor se encaixava no contexto de uso deste *software*, onde ambos os fatores tem de serem pesados, tendo o fator ruído um maior peso. Dos algoritmos de detecção de bordas, os mais utilizados são o de *Sobel*, o de *Roberts* e o *Canny*. A Tabela 1 apresenta um quadro comparativo entre os algoritmos citados anteriormente.

Tabela 1. Comparação ente os tipos mais conhecidos de reconhecimento de bordas de uma imagem, constatados experimentalmente.

Algoritmo	Ruídos	Custo Computacional	Avaliação final
Roberts	Apresenta	Baixo	Rápido, mas impreciso
Canny	Apresenta Menos	Baixo	Rápido, mas impreciso
Sobel	Quase nenhum	Médio	Eficaz

O método de *Sobel* foi escolhido como o método para extração de bordas por apresentar uma avaliação final que mais se compara com o tipo de algoritmo esperado.

2.2 Cores

Mesmo o algoritmo de *Sobel* sendo um ótimo extrator de bordas, ainda capta elementos que são desnecessários para a fase de reconhecimento de imagens, que acabam por se traduzir, neste contexto, em ruído. Imaginem, por exemplo, uma imagem que possua objetos que não somente a mão do indivíduo. Todos os objetos - exceto a mão - são ruídos. Assim, mesmo o contorno sendo uma chave para se encontrar a mão em uma imagem, deve-se filtrar parte desta imagem, de modo a excluir os ruídos, que podem ser deixados pelo algoritmo de *Sobel*. Logo, há necessidade de uma de filtragem da imagem, para se obter um reconhecimento melhor da mão.

Toda mão tem certo padrão de tons avermelhados, devido ao sangue da pessoa, por isso se procura por este padrão. Com esta técnica de procura de cor, delimita-se a área de busca na imagem, facilitando e aumentando a precisão do reconhecimento. Porém, nem sempre esses tons serão evidentes numa imagem, por exemplo, dependendo das condições de iluminação do ambiente, os tons podem aparentar ser diferentes. Assim, mesmo fazendo comparações com um padrão de cor, pode-se não estar sendo fiel ao padrão que se quer. Por isso, é preciso considerar que mesmo comparando com o padrão que é exatamente igual ao previamente estabelecido, é necessário considerar certa taxa de erro. Um algoritmo bastante interessante para realizar esta tarefa é o *CamShift*.

CamShift (*Continuously Adaptive Mean-SHIFT*), em português, Transformação Contínua de Adaptação, é um algoritmo que para cada frame do vídeo capturado, pega a imagem bruta e a converte para uma distribuição probabilística de cores da imagem, usando um modelo de histograma da cor a ser rastreada [OpenCV 2002], ou seja, neste caso, a cor característica da mão. Ele opera basicamente, se valendo de uma busca inicial, para ajustar uma janela para procura. Nesta janela, ele busca pela distribuição provável de cor, que mais se aproxima do padrão, previamente estabelecido. Após ter reconhecido um provável padrão, ele focaliza as buscas apenas no rastreio deste. Por isso, fazendo ajustes de tamanho máximo da janela de pesquisa, e do desvio máximo permitido, ele é capaz de rastrear com perfeição o objetivo. Além disso, internamente o *CamShift* faz a conversão do sistema de cores captado (RGB) para HSV (Figura 2), pois neste é separado a matiz da saturação, reduzindo problemas de intensidade luminosa.

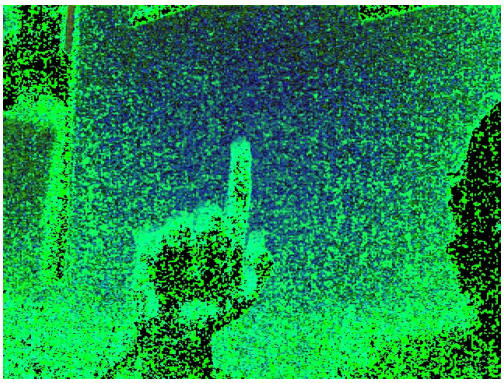

Figura 2. Imagem no formato HSV, em baixa iluminação (lâmpada de 60W).

Aumentando o contraste da imagem em HSV e mudando alguns parâmetros da janela de procura, obtêm-se uma detecção probabilística mais pura, ou seja, apenas os pixels relevantes são destacados, possibilitando ajustar uma área de interesse com foco no alvo, que é a mão. Assim fazendo os ajustes necessários ele pode definir esta área com muito mais precisão. Com esta área configurada, o reconhecimento de imagens é facilitado. Seu custo computacional é relativamente baixo e é bastante preciso, sendo

por estas razões escolhido como método para o rastreio do padrão de cores. Após esta fase é necessário que se faça uma conversão da imagem para uma escala de cinza, facilitando os processamentos posteriores. O uso de *CamShift* acentua a capacidade do uso *software* em ambientes de baixa iluminação.

Um método alternativo seria através da binarização da imagem, tendo em vista que usuário estivesse usando uma luva, de cor bem característica, diferente das cores que normalmente são encontradas nos ambientes, como o verde florescente ou o rosa *pink*. Usando esta alternativa, bastaria aplicar um filtro de cor que apenas deixasse na imagem a cor escolhida, tornando-a binária. Assim, não seria necessário o método de *Sobel,* pois somente a imagem do objetivo seria resultante, ou seja, onde houvesse algo desenhado, a mão do usuário. Esta alternativa tem baixo custo computacional e também é bastante precisa, porém obrigaria que o usuário usasse algo a mais em sua mão, dificultando o uso do *software*.

3 Reconhecimento de Padrões

Após os processamentos da imagem necessários, vem a fase de reconhecimento de padrões. Nesta etapa, é necessário usar de um padrão previamente estabelecido, neste caso, uma mão que esteja na posição indicada para o comando que o *mouse* terá de obedecer. Existem várias alternativas para que o reconhecimento de padrões possa ser feito, passando pelo uso de redes neurais, sistemas especialistas, dentre outros. Porém, os que mais se encaixam no contexto do *software*, são aqueles que oferecem um baixo custo computacional, além de apresentarem uma solução que seja portável, como por exemplo, o uso de *template matching*, ou uso de *shape matching* [OpenCV 2002]. As técnicas de *matching* são ideais para este propósito já que os comandos não vão ser variáveis e por isso se a mão do usuário não estiver na posição do mesmo, ele não será reconhecido (Figura 3).

Figura 3. Template de um comando da mão, tendo sua borda extraída pelo algoritmo de *Sobel*, em ambiente bem iluminado.

Ambos oferecem suporte que atendem as necessidades do projeto, porém o *Shape Matching* além de mais simples, tem baixo custo é ideal para a resolução deste projeto. A Tabela 2 apresenta um quadro comparativo entre as duas técnicas citadas anteriormente.

Tabela 2. Comparativo entre *Shape matching* e *Template Matching*, constatados experimentalmente, a partir do protótipo produzido.

Algoritmo	Imagem Teste	Template	Tempo de Execução	Taxa de Acertos
Shape Matching	640 x 480 pixels	196 x 275 pixels	**3,73 ms**	**± 65%**
Template Matching			**53,90 ms**	**± 85%**

Shape matching é uma técnica que utiliza comparação das formas de dois objetos com o intuito de encontrar alguma semelhança entre ambos [OpenCV 2008b].

Então, para finalmente encontrar o padrão da mão, enquanto fazendo o gesto que simboliza o comando do *mouse,* deve-se usar o algoritmo *CamShift* para reconhecer o padrão da mão, ajustando assim a região de interesse da imagem (ROI - *Region Of Interest*). Tendo o ROI ajustado, é necessário usar o operador de *Sobel* para extrair as bordas da imagem e, finalmente, usar estas bordas extraídas para fazer o *Shape Matching* com uma imagem que foi previamente ajustada.

4 Captura de imagens

Por fim, para a captura de imagens, existem várias alternativas para a linguagem C/C++, que foi escolhida por sua conhecida alta performance. Dentre as diversas alternativas para a captura, tem-se o *DirectShow*, que é uma API (*Application Programming Interface*) de manipulação de *webcam*, presente na API do *DirectX*, além da possibilidade de ser usada a própria API do *Windows*, ou ainda através de bibliotecas de terceiros como a *VideoInput* [VideoInput 2008]. A alternativa mais interessante entre todas as pesquisadas foi a *OpenCV* (*Open Computer Vision*), que é uma biblioteca *free*, criada pela Intel, que oferece diversas funções para a manipulação de imagens, como por exemplo, algoritmos embutidos para a detecção de bordas de imagens e reconhecimento das mesmas, captura de uma *webcam*, e até exibição das mesmas, através de simples funções [OpenCV 2008a].

5 *OpenCV* e algoritmo final

O uso da *OpenCV* facilitará muito o desenvolvimento do aplicativo final, pois reúne todos os elementos necessários. A seguir apresenta-se de forma resumida a forma de execução do *software*:

1. Obtêm-se uma imagem que servirá como *template*, ou referência para os comandos para o mouse.
2. Captura-se a imagem de uma *webcam*.
3. Aplica-se o *CamShift* para que se possa determinar qual a região da imagem possuí a mão, definindo assim a *ROI* e iniciando o rastreio desta área específica.
4. Usa-se o algoritmo de *Sobel*, para a obtenção das bordas da imagem.
5. Aplica-se o *Shape Matching* a imagem obtida da *webcam*, comparando-a com o template.
    a. Se o *Shape Matching* retornar alguma semelhança, respeitando certa taxa de erro, então se encontra a imagem
        i. Aplica-se o movimento no *mouse* de acordo com a posição do dedo indicador, que servirá como referência para a movimentação do *mouse*.
    b. Se o *Shape Matching* não retornar semelhança alguma, então o processo recomeça.

6 Descrição do Protótipo

Com base nos dados adquiridos com os estudos sobre processamento de imagens e reconhecimento de padrões em imagens foi feito um aplicativo para teste destas características.

Implementaram-se diversos métodos para a aquisição da imagem, na maneira desejada. O aplicativo desenvolvido converte as imagens para cinza, aplica o operador de *Sobel*, realiza *shape matching, template matching*, verifica os tempos de execução para os algoritmos e suas respostas (Figura 4). Depois, captura imagens da *webcam* e aplica processamentos em tempo real, além de realizar o algoritmo de *CamShift*. Por fim, aplica o *shape matching* na imagem resultante do *CamShift* e retorna uma resposta para o algoritmo.

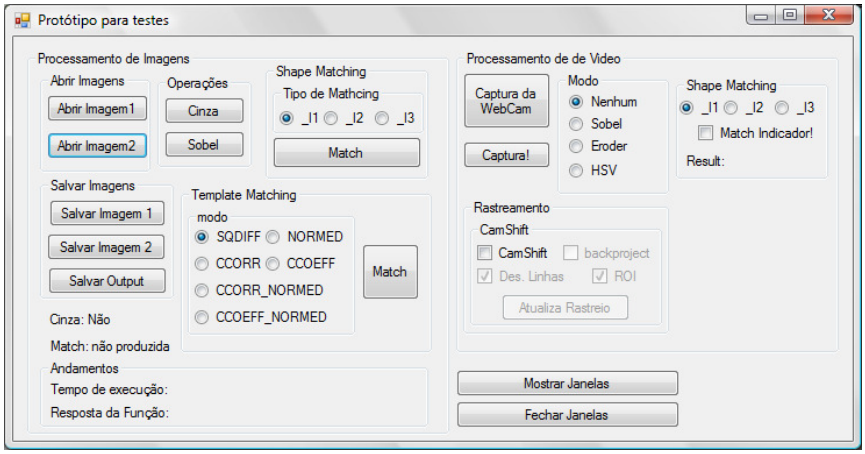

Figura 4. Imagem do protótipo.

No protótipo implementaram-se as rotinas citadas anteriormente, através da seguinte seqüência de comandos (Figura 5):

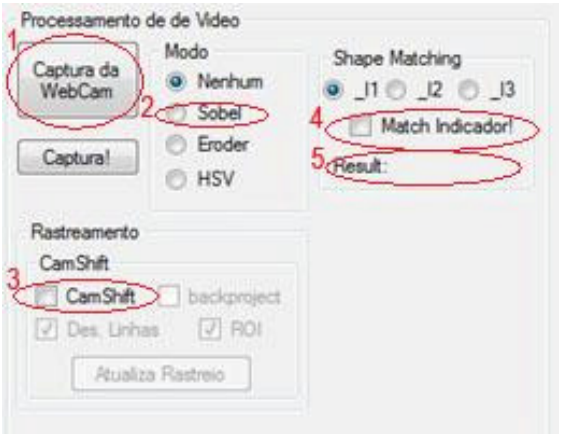

Figura 5. Ordem para o reconhecimento do indicador, e obter o movimento do *mouse*.

Como mostrado acima, inicia-se a captura (1); logo após isso, é necessário aplicar o operador de *Sobel* para a extração de bordas, da imagem (2); no próximo passo, implementa-se o algoritmo *CamShift* na imagem, para definição da região de interesse (ROI), onde os processamentos posteriores serão realizados (3); e por fim, ao habilitar o *Shape Matching*, é possível escolher entre três tipos de comparações diferentes (4). Após isso, os resultados colhidos do *Shape Matching* são apresentados no campo *Result* (5). Por *default*, o *mouse* só se movimenta caso o valor no campo *Result* seja menor ou igual a 0,026, valor avaliado experimentalmente. Então, é realizada uma comparação entre o tamanho total da imagem, o tamanho da tela e a posição do dedo no *template*, fazendo-se com que o *mouse* possa se mover de acordo com a ponta do dedo.

7 Resultados

O *software*, como já foi dito anteriormente implementa diversos métodos de processamento de imagens e alguns de reconhecimento. Para a sua utilização é necessária, no quesito processamentos de imagens, escolher as imagens nos botões *"Abrir Imagem..."*, convertê-las para cinza, e então os processamentos podem ser efetuados. Os resultados, quando numéricos, são apresentados na área *"Andamentos"*, bem como os seus tempos de processamento. Para o processamento de vídeos da *webcam,* basta iniciar a captura e escolher os parâmetros para os testes. Os resultados são mostrados logo abaixo das opções de processamentos. É possível salvar as imagens processadas. A imagens obtidas, sejam de arquivo ou da *webcam,* são mostradas na janela "*Image...*" e os seus processamentos são mostrados na janela "Imagem Processada". A Tabela 3 apresenta os tempos de execução dos algoritmos implementados no *software*.

Tabela 3. Tempos de execução dos algoritmos.

Algoritmo	Resolução Entrada	Resolução Saída	Profundidade	Tempo Médio
Camshift	640 x 480 pixels	418 x 413	24 bits	20,59 ms
Sobel	418 x 413 pixels	418 x 413	8 bits	24,64 ms
Shape Matching	418 x 413 pixels da captura e 185 x 259 pixels do *template*	418 x 413	8 bits	1,11 ms
Total:				46,35 ms

O protótipo ainda não funciona perfeitamente, pois o algoritmo *CamShift*, para poder colher exatamente um área de interesse que seja no mínimo relevante exige uma configuração rica em detalhes. Por isso, o sistema como um todo ainda não funciona tão bem quanto deveria, mas após ajustes constantes ele tende a melhorar e tornar a precisão do reconhecimento melhor. Por causa da conversão para HSV implementada internamente no *Camshift*, o fator iluminação não é tão crítico nesta parte do algoritmo. Porém, para o *Sobel* é necessária uma iluminação do ambiente relativamente boa, para que o operador possa ser usado em toda sua plenitude e então capturar imagens mais ricas em bordas, diminuindo os ruídos da região de interesse, o que resulta num aumento da exatidão do movimento do *mouse*.

8 Conclusão

Com os testes realizados a partir do protótipo, ficou claro que o método adotado é bastante viável, desde que se faça um apuramento nas configurações do algoritmo *Camshift* e, se possível, no método de *Sobel*, já que o primeiro é responsável pela detecção da área de interesse (*R.O.I.*). Esta área é utilizada pelos processamentos posteriores, implicando em maior precisão na detecção da mão e dos possíveis comandos por ela realizados. O tempo de execução de todo o processo está diretamente ligado ao apuramento das configurações corretas desses algoritmos, além de ser fator crítico, pois não serão admitidos pelo usuário grandes atrasos entre um comando realizado pela mão e sua efetivação no computador.

Um aplicativo que implemente está solução para a finalidade aqui discutida será ótimo, numa relação custo benefício final, tendo em vista que não há necessidade de aquisição de equipamentos que não sejam comumente encontrados, além de ser uma

solução computacionalmente barata, fatores esses que indicam um alto fator de aceitação entre os mais diversos tipos de usuários.

Referências Bibliográficas

OpenCV (2002). Documentação Oficial da OpenCV. Encontrada no diretório de instalação da mesma, *OpenCV Reference Guide*.

OpenCV (2008a). *OpenCV Official* - Página oficial da Intel, informações sobre utilização e licença. Disponível em: http://www.intel.com/technology/computing/opencv/index.htm. Último acesso em 13 de setembro de 2008.

OpenCV (2008b). *Wiki da OpenCV*. Tutoriais e exemplos de sua utilização. Disponível em: http://opencvlibrary.sourceforge.net/Welcome/#head-59654a1ce94786b994e1e a7b59ac0d98bb7318c7. Último acesso em 13 de setembro de 2008.

PDI (2008). Processamento Digital de Imagens Filtragens Espaciais. Disponível em: http://www.dpi.inpe.br/~carlos/Academicos/Cursos/Pdi/pdi_filtros.htm. Último acesso em 06 de setembro de 2008.

von Wangenheim, Aldo (2008). Introdução à Visão Computacional - Encontra a Linha Divisória: Detecção de Bordas. Disponível em: http://www.inf.pucrs.br/~eduardob/disciplinas/laproiv/HomeAssignments/DeteccaoContorno/bordas.pdf. Último acesso em 06 de fevereiro de 2009.

VideoInput (2008). Biblioteca *VideoInput* para captura de imagens. Disponível em: http://muonics.net/school/spring05/videoInput/. Último acesso em 09 de fevereiro de 2008.