Instituto de Matemática e Estatística
Universidade de São Paulo

Casamento de formas com o descritor Shape Context

Trabalho de Formatura Supervisionado

Aluno: Diego Mira David

Supervisor: Ronaldo Fumio Hashimoto

1 de Dezembro de 2008

Agradecimentos

Ao meu orientador, Prof. Dr. Ronaldo Fumio Hashimoto, sou muito grato pela orientação e por todo interesse demonstrado durante a realização do trabalho.

A Wonder A. L. Alves e Alexandre Noma, alunos de pós-graduação do Instituto, pelas sugestões e contribuições.

Conteúdo

1 Introdução

1.1 Visão Geral

O reconhecimento de formas é um dos problemas fundamentais na área de Visão Computacional e diversas abordagens têm sido propostas para tratá-lo. Este trabalho consiste no estudo e implementação de algoritmos para casamento de formas baseados no descritor *shape context*, proposto por S. Belongie e J. Malik em [1] no ano 2000.

Um **descritor** de uma forma é uma representação computacional que contém informação sobre as características da forma. Descritores são usados sobretudo para distinguir formas diferentes.

Embora compartilhe limitações características da área de Visão Computacional, a utilização do descritor *shape context* tem apresentado bons resultados em trabalhos recentes na área.

1.2 Motivações e objetivos do trabalho

O objetivo inicial deste trabalho era estudar e implementar o algoritmo descrito no artigo "Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA" [3], cujos conceitos também são utilizados em outros trabalhos desenvolvidos no Instituto. Entretanto, após a leitura de alguns artigos relacionados, houve interesse em realizar um trabalho com foco maior no descritor e no desenvolvimento de algumas aplicações.

Muito foi publicado a respeito do tema, sobretudo pelos autores do descritor. Este trabalho não visa cobrir todos os detalhes presentes na literatura, mas sim estudar os conceitos principais e desenvolver ferramentas para analisar o método utilizado e testar algumas aplicações. Para maiores informações, consulte artigos citados nas referências bibliográficas.

Em geral, as publicações na área são bastante compactas. Durante a etapa de implementação, surgiram uma série de dúvidas referentes a detalhes que necessitam de maior atenção e de outras fontes para auxiliar no desenvolvimento. Um das motivações

principais na realização do trabalho foi contribuir com implementações que possam ser utilizadas em outros projetos que utilizem os conceitos estudados.

O tema proporcionou ao aluno o contato com trabalhos recentes desenvolvidos na área de reconhecimento de objetos e casamento de formas. Apesar do trabalho aparentemente ter um foco num tema específico, permitiu lidar com diversos aspectos da área de Visão Computacional. Tiveram de ser implementados tanto algoritmos na área de processamento de imagens como algoritmos para resolver subproblemas.

1.3 Organização do texto

A monografia foi organizada na seguinte estrutura:

- Parte técnica:
  - Apresentação de definições e dos conceitos principais que envolvem a utilização do descritor *shape context*.
  - Descrição das ferramentas desenvolvidas, onde serão mencionados alguns detalhes de implementação, resultados obtidos e dificuldades encontradas.
- Parte subjetiva
  - Relato pessoal sobre a experiência de desenvolver o projeto, desafios, frustrações encontradas e relacionamento do projeto com o curso.

Parte I

Parte Técnica

2 Conceitos e tecnologias estudadas

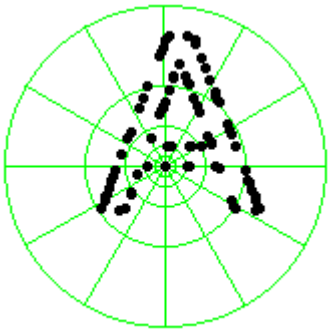
2.1 Shape Context

Shape Context é o nome do descritor proposto pela primeira vez em 2000 por S. Belongie e J. Malik em "Matching with Shape Contexts" [1]. Abordagens que fazem uso do descritor tem sido usadas com sucesso em diversos trabalhos como [1], [2], [3], [**?**], [4], [5], entre outros.

Neste capítulo será dada a definição e apresentados os conceitos principais relacionados ao uso do descritor.

2.1.1 Histograma

Uma forma pode ser representada por uma amostra de pontos do contorno. Seja $P = \{p_1, \ldots, p_n\}$, $p_i \in R^2$, uma amostra de pontos dos contornos interno e externo da imagem de um objeto. Para cada ponto p_i da amostra, é calculado um **histograma** das coordenadas relativas dos demais pontos do contorno em relação à p_i.



Para a construção do histograma, utiliza-se um diagrama log-polar (ilustrado na imagem ao lado), particionado em 60 *bins*. A utilização do diagrama log-polar torna o descritor mais sensível às posições mais próximas de p_i, do que à pontos situados em regiões mais distantes.

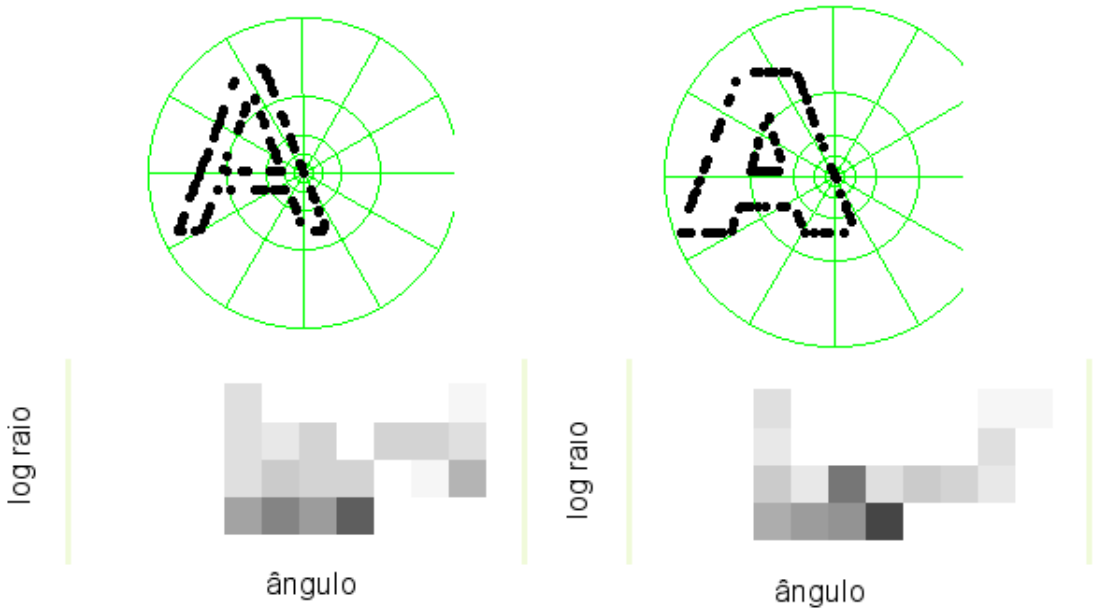
O cálculo do histograma é feito da seguinte forma:

$$h_i(k) = \#\{q \neq p_i : (q - p_i) \in bin(k)\}$$

O histograma h_i é definido como o *shape context* de p_i. Ele incorpora informação global da forma em relação à p_i ao armazenar a distribuição dos pontos nos *bins*.

2.1.2 Comparação

A imagem abaixo ilustra a semelhança entre os histogramas calculados para pontos localizados em uma mesma região do contorno em duas formas semelhantes.



Para comparar dois *shape contexts* é utilizada a distância χ^2. O custo para casar um ponto p_i de uma forma com um ponto q_j de outra, que é uma medida da diferença entre os dois *shape contexts*, é dado por:

$$C(p_i, q_j) = \frac{1}{2} \sum_{k=1}^{K} \frac{[h_i(k) - h_j(k)]^2}{h_i(k) + h_j(k)}$$

Uma vez que temos uma medida para comparar dois *shape contexts* associados a dois pontos, podemos usar tais custos para casar duas formas, ou seja, encontrar uma correspondência entre os pontos de uma forma e de outra. Isto pode ser feito por meio do algoritmo *Kuhn-Munkres*, que será descrito adiante.

2.1.3 Invariâncias

Translação

Invariância à translação decorre da própria definição do descritor, uma vez que ele é baseado nos pontos do contorno.

Escala

Invariância à escala pode ser obtida normalizando as distâncias entre os pontos pela distância média entre todos os pares de pontos do contorno.

Rotação

Caso seja desejável, também pode ser obtida invariância à rotação, como descrito em [4]. Para isto, em vez de usar um eixo absoluto como referência para o cálculo do ângulo que determina o *bin*, pode-se considerar como referência o vetor tangente ao ponto.

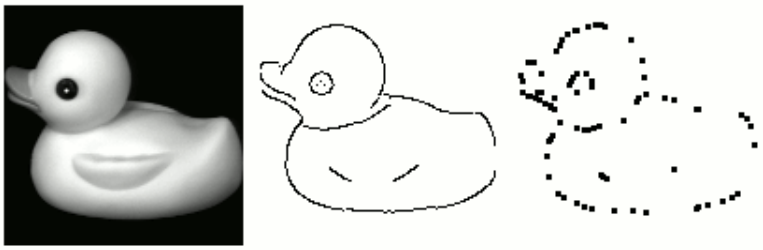
2.2 Algoritmo de Canny para detecção de bordas

A localização de bordas é um problema de fundamental importância na área de processamento de imagens. Neste trabalho, foi implementado o método de Canny, que é um algoritmo clássico da área.

O operador de detecção de bordas Canny [6] foi desenvolvido por John Canny em 1986 a partir das idéias de Marr e Hildreth. A elaboração do algoritmo parte das seguintes premissas:

- **Premissa da boa detecção:** deve localizar as bordas realmente existentes na imagem, ou seja, deve haver uma baixa taxa de erro.
- **Premissa da boa localização:** deve minimizar a distância entre a borda localizada e a borda verdadeira.
- **Premissa da boa resposta:** deve haver uma única resposta para cada borda existente.

Neste trabalho, é necessário obter o contorno da imagem de um objeto.



A imagem acima exibe o contorno obtido com a aplicação do algoritmo em uma imagem em níveis de cinza, assim como uma amostra aleatória de pontos do contorno.

2.3 Algoritmo Kuhn-Munkres

O algoritmo *Kuhn-Munkres*, também conhecido como **Método Húngaro**, resolve o problema de atribuição de tarefas (*assignment problem*). Trata-se de uma etapa fundamental no método utilizado para calcular a semelhança entre duas formas. O algoritmo é usado para determinar uma correspondência de custo total ótimo entre os pontos do contorno de duas imagens.

2.3.1 O problema de atribuição de tarefas

Seja C uma matriz $n \times n$ contendo os custos para que cada um de n trabalhadores realize quaisquer n tarefas. O problema consiste em atribuir as tarefas para os trabalhadores de forma que o custo total seja mínimo. Cada trabalhador pode realizar uma única tarefa e cada tarefa pode ser feita por um único trabalhador. Assim, o objetivo é encontrar uma permutação de $\{1, \ldots, n\}$ que minimize o custo total. Ou ainda, numa outra formulação para este mesmo problema, podemos dizer que o objetivo é encontrar um emparelhamento de valor mínimo em um grafo bipartido com custo nas arestas.

Uma solução por força bruta é inviável, uma vez que existem $n!$ possibilidades. O algoritmo *Kuhn-Munkres* resolve o problema em tempo polinomial ($O(n^3)$).

O implementação do algoritmo é repleta de detalhes e consistiu num desafio a mais na realização do projeto. O código foi desenvolvido na linguagem C++ e a versão implementada é baseada em [9].

2.3.2 Matriz de custos não-quadrada

O algoritmo assume como entrada uma matriz quadrada. Entretanto, podemos facilmente lidar com matrizes retangulares, bastando para isso preencher com zeros linhas ou colunas de forma a tornar a matriz quadrada.

2.4 Thin plate splines

Dada a correspondência entre os pontos de duas formas, pode ser estimada uma transformação que melhor alinhe as duas formas.

Um modelo bastante usado com esta finalidade é o de *thin plate splines* (TPS) [8]. Trata-se de uma generalização de splines cúbicas para o caso 2D. O modelo de *thin plate splines* é usado para representar transformações de coordenadas. Há trabalhos interessantes que utilizam TPS para realizar transformações mórficas em imagens.

$$(x, y) \rightarrow (f_x(x, y),\ f_y(x, y))$$

Uma *thin-plate spline* f(x,y) minimiza a energia de dobramento (*bending energy*) dada por:

$$\int\int_{\mathbf{R}^2} (\frac{\partial^2 f}{\partial x^2})^2 + 2(\frac{\partial^2 f}{\partial x \partial y})^2 + (\frac{\partial^2 f}{\partial y^2})^2 dxdy$$

e tem a forma

$$f(x, y) = a_0 + a_x x + a_y y + \sum_{i=1}^{n} w_i U(|(x, y) - P_i|)$$

onde $U(r) = r^2 \log(r^2)$

Informações detalhadas podem ser encontradas em Bookstein [8].

Neste trabalho foi implementado um algoritmo para estimar transfomações utilizando o modelo TPS com base no livro "Shape analysis and classification: Theory and Practice" [7, p.317-330]. Entretanto, o algoritmo acabou não sendo utilizado nas ferramentas implementadas. As implementações feitas utilizam técnicas de menor custo computacional, descritas nas próximas seções.

2.5 Shape Context generalizado

Em "Efficient Shape Matching Using Shape Contexts" [2], os autores apresentam uma extensão mais rica do descritor *shape context*, chamada *generalized shape context*, que considera vetores tangentes t_i de comprimento unitário para indicar a direção do contorno em cada ponto p_i.

$$\hat{h}(k) = \sum_{q_i \in Q} t_i \;\; onde \;\; Q = \{q \neq p_i : (q - p_i) \in bin(k)\}$$

Desta forma, cada *bin* passa a carregar um vetor com a direção dominante da contorno dentro do *bin*. O custo para casar um ponto p_i de uma forma com um ponto q_i de outra forma passa a ser dado por:

$$C(p_i, q_j) = \sum_{k=1}^{K} \frac{||\hat{h}_i(k) - \hat{h}_j(k)||^2}{||\hat{h}_i(k) + \hat{h}_j(k)||}$$

O descritor original pode ser visto como um caso particular desta extensão em que todos os vetores tangentes têm ângulos iguais a zero.

Esta abordagem se mostrou mais adequada e foi utilizada nas implementações feitas em conjunto com a técnica de filtragem *representative shape contexts*, descrita na próxima seção.

2.6 Técnica de filtragem

Em [2] é apresentada uma técnica de filtragem (*Representative Shape Context*) para tornar mais eficiente o processo de determinar as formas mais semelhantes a uma dada imagem de consulta.

2.6.1 Objetivo

O objetivo é obter, com baixo custo computacional, uma pequena lista de candidatos que inclua a forma mais semelhante à consulta. Para verificar que duas formas são diferentes não é necessário comparar todos os pares de pontos. O método, descrito a seguir, usa esta intuição para tornar o processo mais eficiente.

2.6.2 Método

Para cada forma já conhecida, é calculado uma quantidade razoável (por volta de 100) de *shape contexts*. Já para a imagem de consulta, são selecionados apenas alguns pontos do contorno e calculados apenas os *shape contexts* associados à estes pontos para representar a forma. Para preencher o histograma, todos os demais pontos do contorno são considerados.

Para calcular a distância entre uma dada forma e uma forma conhecida, devem ser localizados os melhores casamentos para cada um dos *shape contexts* que representam a forma dada. Seja G_i o conjunto de índices dos k *shape contexts* que possuem as menores distâncias [1] ao realizar a comparação com uma forma S_i. A distância entre a consulta Q e uma forma conhecida S_i é dada por:

$$d(Q, S_i) = \frac{1}{k} \sum_{u \in G_i} \frac{d_{GSC}(SC_Q^u, SC_i^{m(u)})}{N_u}$$

$$\text{onde } m(u) = arg\ min_j d_{GSC}(SC_Q^u, SC_i^j)$$

Para medir o quão discriminante é o *shape context* SC_Q^u, é usado um fator de normalização, dado por:

[1] Em imagens com ruídos, se faz necessário descartar alguns *shape contexts* de maior distância.

$$N_u = \frac{1}{|S|} \sum_{s_i \in S} d_{GSC}(SC_Q^u, SC_i^{m(u)})$$

onde S é o conjunto com todas as formas conhecidas.

Ordenando as distâncias obtidas, pode-se descartar boa parte das imagens do conjunto e manter apenas uma pequena lista de candidatos. Com isso, obtém-se um ganho considerável de tempo, sem prejuízo à qualidade dos resultados. Na seção 3.2.1, é apresentada uma comparação do tempo gasto.

3 Atividades realizadas

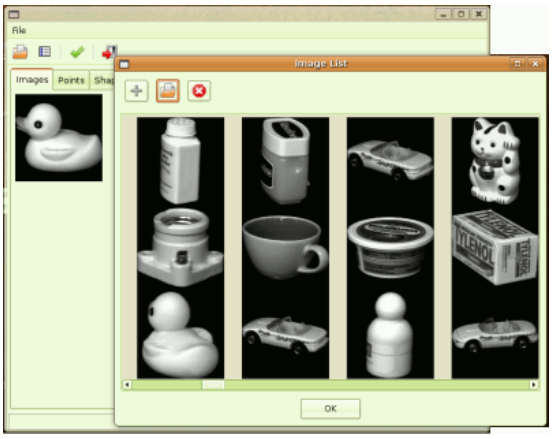
Após estudar os artigos e implementar alguns algoritmos para testes, começaram a ser desenvolvidas algumas aplicações. A parte prática deste trabalho pode ser divida em três partes:

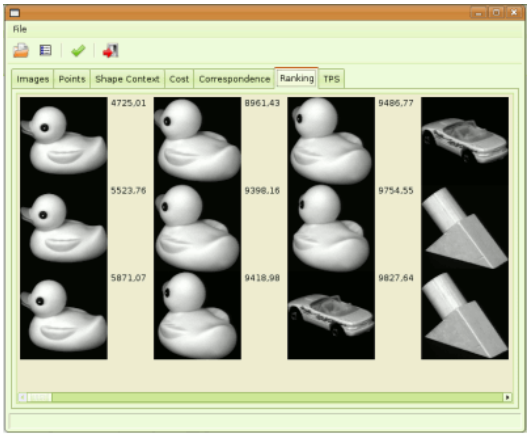
- **Recuperação de formas semelhantes:** Foi desenvolvido um programa para recuperar, dentro de um conjunto de imagens, formas semelhantes à de uma imagem de consulta.
- **Reconhecimento de gestos do mouse:** Tirando proveito dos resultados alcançados na aplicação acima, foi desenvolvido um programa para reconhecer gestos feitos com o movimento do mouse.
- **Quebra do captcha EZ-GIMPY:** Implementação de um dos algoritmos descritos em *"Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA"* [3]

A seguir serão descritos cada um destes problemas, com alguns detalhes de implementação, dificuldades encontradas durante o desenvolvimento e resultados obtidos.

3.1 Recuperação de formas semelhantes

Com base no descritor *shape context*, foi desenvolvido um programa para recuperar formas semelhantes. Dados um conjunto de imagens e uma imagem de consulta, o programa exibe um *ranking* com as imagens do conjunto mais semelhantes à imagem de consulta.





As imagens ilustram o funcionamento do programa para algumas imagens do dataset *Columbia Object Image Library (COIL-20)*. Na primeira tela estão as imagens de entrada e na segunda tela está um *ranking* com as imagens mais semelhantes.

3.1.1 Ferramenta de estudo

O programa foi construído de forma a também fornecer informações para acompanhar o funcionamento do método utilizado. Permite visualizar:

- Contorno da imagem
- Pontos da amostra
- *Shape context* associado a cada ponto
- Tabela de custos
- Correspondência entre os pontos de custo ótimo.

3.1.2 Implementação

Inicialmente é aplicado o operador Canny, descrito na seção [6], em cada imagem fornecida e selecionada uma amostra aleatória de pontos do contorno. Calcula-se o *shape context* associado a cada ponto da amostra, considerando no cálculo do histograma todos os demais pontos do contorno.

Foram utilizados 60 *bins*. São 12 divisões de ângulo (cada *bin* tem um ângulo de abertura de 30 graus). São 5 divisões de raio, que variam de 0.125λ a 2λ, onde λ é a distância média entre todos os pontos do contorno.

Inicialmente, foi adotada a versão tradicional do descritor, que considera apenas a contagem de pontos. Mais tarde foi utilizada a extensão que incorpora informação da aparência local utilizando tangentes e os resultados obtidos foram melhores.

Cálculo da tangente

Um método simples foi utilizado para obter a tangente em cada ponto do contorno. É considerada uma janela 3 pixels $\times$ 3 pixels centrada no ponto. A tangente é fornecida de acordo com a configuração dos pixels desta janela.

Exemplo de uma configuração

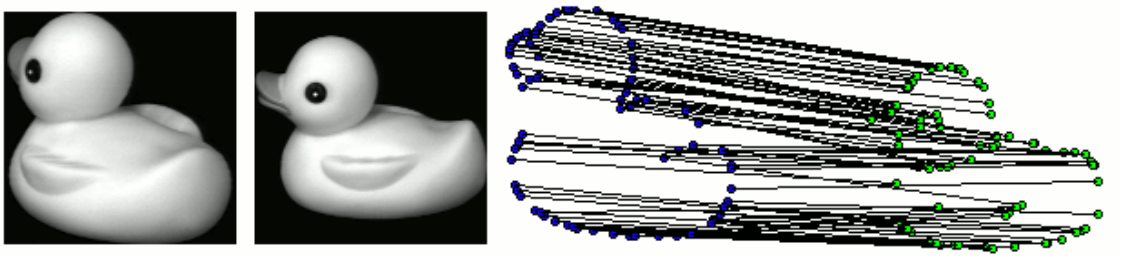
O método implementado não é o ideal, visto que pressupõe um contorno contínuo e depende de muitas configurações. Outros métodos para resolver este problema poderiam ser implementados de forma a obter resultados mais robustos.

Filtragem

Após calcular os *shape contexts* generalizados para cada forma, é utilizada a técnica de filtragem descrita na seção 2.6 para reduzir o tamanho do conjunto a uma pequena lista de candidatos.

Correspondência de pontos

Para cada imagem na lista restante, é aplicado o algoritmo Kuhn-Munkres, descrito na seção 2.3. O algoritmo nos fornece uma permutação que resolve o problema da correspondência de pontos com custo ótimo. A imagem abaixo ilustra o resultado.



Uma vez que temos o custo para casar a imagem de consulta com cada imagem do conjunto, ele é usados como critério de classificação para obter o *ranking* com as imagens mais semelhantes. Note que neste trabalho não foi estimada uma transformação com *thin plate splines*.

3.1.3 Resultados

O programa funciona muito bem para formas simples. Características como cor ou textura não foram consideradas. Para imagens mais complexas, cujo contorno obtido com o operador Canny não contenha apenas pontos do objeto desejado, ou para localização de objetos dentro de outras imagens, devem ser consideradas outras técnicas além das implementadas neste trabalho.

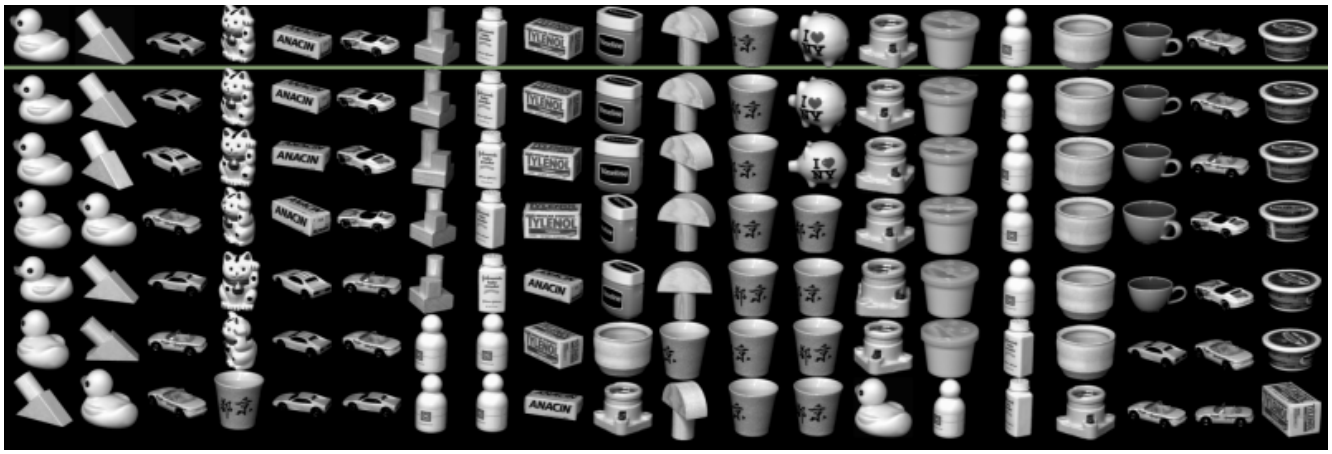
Experimento

Foi utilizado o *dataset Columbia Object Image Library (COIL-20)* [11], que contém imagens processadas de 20 objetos. As imagens de cada objeto em níveis de cinza estão separadas e possuem fundo preto. Para cada objeto, há 72 imagens com orientações diferentes que variam de 0 a 360 °. Há um total de 1440 imagens.

Em nosso experimento, com auxílio de um script em *python*, foram selecionadas 5 imagens de cada objeto, totalizando 100 imagens.

As 5 imagens de cada objeto foram tomadas com as seguintes orientações: $-30\,^{\circ}$, $-15\,^{\circ}$, $0\,^{\circ}$, $15\,^{\circ}$, $30\,^{\circ}$. A imagem com orientação correspondente a $0\,^{\circ}$ de cada objeto foi comparada com todas as 100 imagens.

O resultado do experimento é exibido na imagem abaixo. Para cada objeto são apresentados aqueles que foram considerados os mais semelhantes.



Comparação de tempo

Foi feita uma comparação entre o tempo, em segundos, levado para encontrar as 10 imagens mais semelhantes dentre as 100 imagens do conjunto, utilizando a técnica de filtragem e sem utilizá-la. O tempo relacionado ao cálculo dos *shape contexts* foi desconsiderado.

Neste experimento foram utilizados 20 *shape contexts* para representar a imagem de consulta e 80 *shape contexts* para representar as imagens conhecidas.

A tabela abaixo mostra o tempo de execução, em segundos, para executar o algoritmo para três imagens de consulta retiradas ao acaso do conjunto.

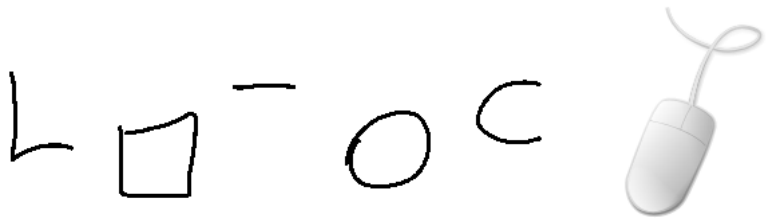
Tempo total sem filtragem	Tempo total com filtragem
41.32s	21.92s
35.98s	21.65s
41.88s	21.93s

O resultado da etapa de filtragem é uma lista com 10 candidatos. A tabela abaixo separa o tempo envolvido na etapa de filtragem e na ordenação da lista de candidatos.

Filtragem	Ordenação
19.00s	2.92s
19.27s	2.38s
18.90s	3.03s

3.2 Gestos do mouse

Gestos do *mouse* (*mouse gestures*) são movimentos feitos com o *mouse* e que podem ser associados à comandos pré-estabelecidos. São usados em *browsers* e também para lançar aplicativos como uma forma interessante de interação.



Neste trabalho, testamos o uso do descritor *shape context* para reconhecer gestos do *mouse*. Considerando que os movimentos feitos com o *mouse* descrevem uma forma, podemos aplicar a mesma abordagem usada no programa acima. A vantagem é a flexibilidade adquirida com o uso do descritor, não se limitando a apenas algumas combinações de direções. Neste caso, invariância à rotação não é desejada, pois é preciso diferenciar entre linhas horizontais e verticais por exemplo. Invariância à escala e translação são fundamentais neste tipo de aplicação.

3.2.1 Implementação

A implementação feita utiliza a abordagem já descrita para comparar a semelhança de formas. A biblioteca *wxWidgets* foi utilizada para construir uma imagem a partir do movimento feito com o *mouse*.

O estado atual do *mouse* é obtido por meio da função *wxGetMouseState()*, que devolve uma instância da classe *wxMouseState*, por meio da qual é possível obter informações sobre botões pressionados e posição do ponteiro do *mouse* em coordenadas da tela.

Demarcação do movimento

O início e o fim de um movimento são demarcados pelo pressionamento de um botão do *mouse* ou uma tecla.

Construção da imagem

A partir da demarcação de início é construída uma imagem associada ao movimento realizado. Para isto, a cada instante é marcada a posição do ponteiro do *mouse*, de forma que ao final do movimento teremos uma imagem binária com a forma do movimento realizado. Após a demarcação do fim do movimento, a imagem resultante tem seu tamanho reduzido e é então comparada com as formas armazenadas.

Comparação

A princípio, é realizada apenas a etapa de filtragem, com um baixo número de *shape contexts*. Caso a distância entre os primeiros colocados esteja abaixo de um certo limiar, é executado o processo mais custoso de encontrar a correspondência entre os pontos.

Caso a distância da imagem associada ao movimento feito em relação à imagem melhor classificada tenha um valor acima de um limiar, nenhuma associação é aceita.

Caso contrário, o gesto feito com o *mouse* é reconhecido como sendo a forma que possui a menor distância.

Associação à comandos

Cada forma está associada a um comando, como por exemplo a execução de um aplicativo ou simulação do pressionamento de teclas.

Algumas melhorias que podem ser consideradas:

- Utilização de uma técnica de espalhamento de pontos, em substituição à amostra aleatória.
- Inclusão da direção como característica a ser considerada.

Os resultados preliminares são animadores dada à flexibilidade do descritor e pretendo seguir com o desenvolvimento desta ferramenta.

3.3 Captchas

3.3.1 Definição

CAPTCHA é acrônimo de "*Completely Automated Public Turing test to tell Computers and Humans Apart*". O termo foi cunhado em 2000 por Luis Von Ahn, Manuel Blum, Nicholas Hopper e John Langford da universidade *Carnegie Mellon*. Na época, foi desenvolvida a primeira versão para utilização pela *Yahoo*.

Trata-se de um teste que visa distinguir humanos de computadores para impedir acessos automatizados a um sistema. Atualmente são muito utilizados, entre outros motivos, para prevenir contra:

- Ataques por força bruta em sistemas de senha
- SPAM em comentários de blogs
- Criação de contas de e-mail de forma abusiva
- Alto número de acessos automatizados a um serviço de disponibilidade limitada

O uso deste dispositivo muitas vezes incorre em problemas de usabilidade e acessibilidade. Para maiores informações, consulte [13].

3.3.2 Segurança

Atualmente, diversos projetos têm demonstrado que captchas visuais baseados em caracteres podem se quebrados por computadores. Alguns exemplos são os projetos **PWNtcha** [14] e **aiCaptcha** [15].

Publicações também têm mostrado que muitos desses sistemas podem ser quebrados. Neste trabalho foi estudado o algoritmo para quebrar o captcha *EZ-Gimpy* descrito em *"Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA"* [3].

No estágio atual de desenvolvimento, a implementação feita ainda apresenta problemas que impossibilitam o reconhecimento satisfatório de todas as classes de imagem. Detalhes sobre o método utilizado e dificuldades encontradas estão nas próximas seções.

3.3.3 Implementação

A implementação feita tem como base o Algoritmo A descrito em *"Breaking a Visual CAPTCHA"* [3] .

Em alto nível, o algoritmo pode ser resumido da seguinte forma:

1. Localizar possíveis letras em diversas localizações da imagem.
2. Utilizando as hipóteses obtidas, construir um grafo dirigido acíclico com todas as letras consecutivas que podem ser usadas para formar uma palavra.
3. Dentre todos os caminhos possíveis no grafo, selecionar apenas palavras existentes num dicionário.
4. Avaliar cada palavra em um processo mais preciso e caro computacionalmente.

Lista de hipóteses

Usamos a técnica de *representative shape contexts* para obter um conjunto de tuplas com letras e possíveis posições.

Construção do grafo

As letras (com posições) melhores classificadas na lista de hipóteses são consideradas na criação de um grafo dirigido acíclico. O grafo é construído de forma que as letras são os vértices e toda letra tem arestas ligadas a cada uma das letras à direita. É aplicada a restrição de localização, não são inseridas arestas para letras muito distantes. Assim, todas as palavras possíveis na imagem são dadas pelos caminhos possíveis no grafo.

O captcha *EZ-GIMPY* é baseado em um dicionário com 561 palavras. Este fato é usado para reduzir o conjunto de palavras possíveis, já que muitos caminhos no grafo não correspondem à nenhuma palavra do dicionário. Para isto, são calculados todos os trigramas (sequência de três letras) das palavras do dicionário. Das palavras do dicionário, são mantidas todas àquelas que têm todos os seus trigramas contidos no grafo, na ordem correta.

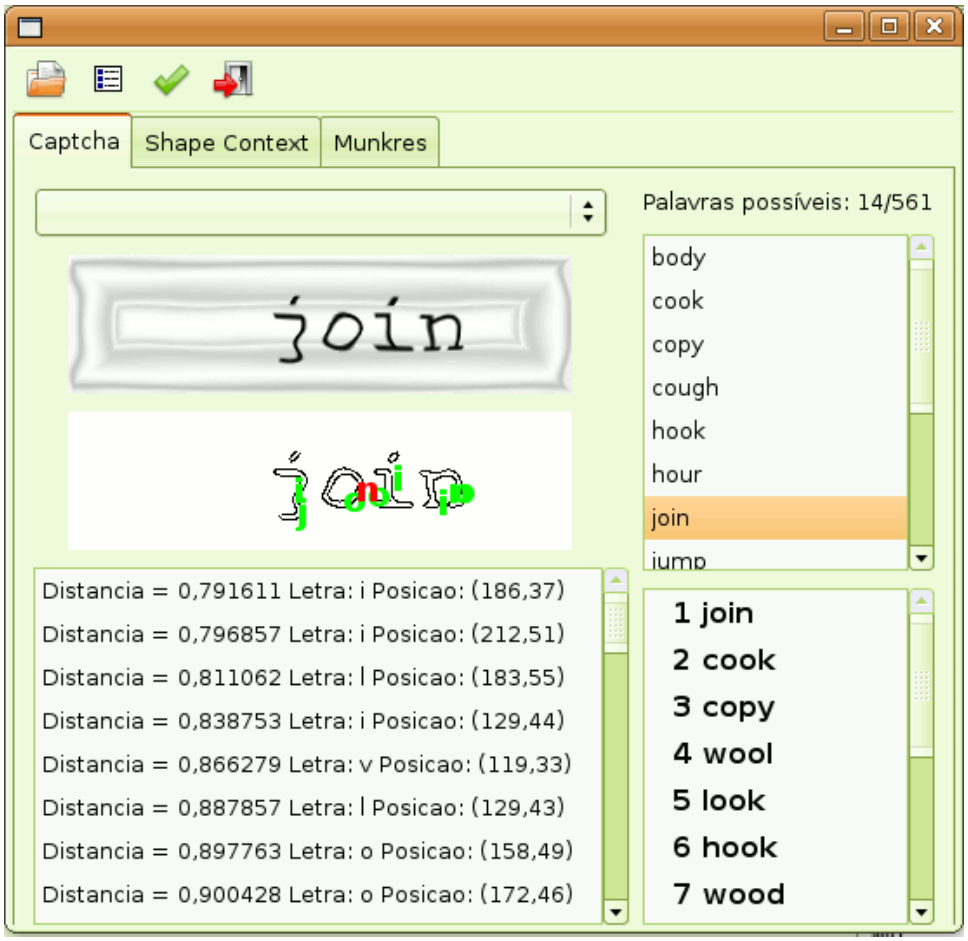
Ranking

É atribuída uma distância a cada palavra do conjunto de palavras possíveis. Na implementação feita, tenta-se casar os pontos de cada letra da palavra com os melhores pontos da imagem. Isso é feito aplicando o algoritmo *Kuhn-Munkres* com uma matriz de custos retangular, uma vez que neste caso estamos interessados em localizar uma forma dentre outras que estão numa mesma imagem. Então é feita a média dos custos de cada letra da palavra e esta média é usada como critério de classificação.

Para informações mais detalhadas sobre o algoritmo implementado, consulte [3].

3.3.4 Resultados

Foi desenvolvida uma ferramenta que implementa o algoritmo descrito acima.



Na figura acima estão exemplos de palavras que o programa é capaz de reconhecer. O resultado ainda não é satisfatório, uma vez que é possível obter a palavra correta apenas para um pequeno subconjunto de imagens.

No estágio atual de desenvolvimento, não há um pré-processamento adequado das imagens, nem uso de um operador de textura para distinguir entre os pontos pertencentes ao fundo da imagem e os pontos pertences à palavra. Além disso, a forma como são calculadas as tangentes, mencionada anteriormente, pressupõe um contorno contínuo após a aplicação do operador Canny, o que não ocorre em grande parte dessas imagens.

Abaixo, podemos ver algumas palavras que não podem ser reconhecidas pela ferramenta implementada devido aos problemas citados.

4 Tecnologias utilizadas

Os programas foram desenvolvidos em linguagem C++. A escolha se deu pela questão de desempenho e também pelo meu interesse em adquirir mais experiência com a linguagem. Para o desenvolvimento da interface gráfica, foi utilizada a biblioteca *wxWidgets*.

wxWidgets

O wxWidgets é uma biblitoteca livre e multiplataforma para construção de interfaces gráficas. A biblioteca é implementada em C++, mas possui *bindings* que permitem utilizar a API em outras linguagens como *Python*, *Perl* e muitas outras. Uma característica interessante é que o *look and feel* da aplicação é adaptado à plataforma usada. A biblioteca possui uma documentação bem completa e mostrou-se uma excelente escolha.

CodeBlocks

Inicialmente foi utilizado o ambiente de desenvolvimento *Eclipse*, mas logo foi trocado pelo ambiente Code::Blocks, por este possuir melhor integração com linguagem e bibliotecas utilizadas no projeto.

Boost

Trata-se de uma excelente coleção de bibliotecas que faz uso extensivo de programação genérica. Neste trabalho, foi usada para lidar com matrizes, grafos e resolução de sistemas lineares.

Todo o trabalho foi realizado no sistema operacional *Linux*, utilizando *software livre*. A monografia, a apresentação e o pôster do projeto foram feitos utilizando *LaTeX*.

5 Conclusão

Neste trabalho foi possível estudar o descritor *shape context* e ter contato com trabalhos recentes na área de Visão Computacional.

Como resultado, foram desenvolvidas algumas aplicações utilizando os conceitos estudados. Acredito que o trabalho desenvolvido seja de grande valia no estudo de métodos que envolvam o uso do descritor *shape context* para o reconhecimento formas.

No estágio atual, as aplicações desenvolvidas estão mais relacionadas ao estudo das técnicas do que aplicações de caráter mais prático. Dado que o código será disponibilizado, espero que este trabalho possa contribuir de alguma forma para o desenvolvimento de trabalhos futuros relacionados ao tema.

Referências Bibliográficas

[1] S. Belongie and J. Malik. Matching with Shape Contexts. CBAIVL 2000

[2] G. Mori, S. Belongie, J. Malik. Efficient Shape Matching Using Shape Contexts. IEEE PAMI Nov. 2005

[3] G. Mori, J. Malik. Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA. CVPR 2003

[4] S. Belongie, J. Malik e J. Puzicha. Shape Matching and Object Recognition Using Shape Contexts. PAMI 2002

[5] G. Mori, J. Malik. Recovering 3D Human Body Configurations Using Shape Contexts. IEEE PAMI Jul. 2006

[6] J. F. Canny. A computational approach to edge detection. IEEE Trans. Pattern Analysis and Machine Intelligence, p. 679-698, 1986.

[7] L. F. Costa and R. M. Cesar. Shape analysis and classification: Theory and Practice. CRC PRESS, 2001, 4.9.6, p317-330.

[8] F.L. Bookstein. Principal Warps: Thin-plate splines and the decomposition of deformations. IEEE Transactions on Pattern Analysis and Machine Intelligence, 11(6):567-585, June 1989.

[9] Munkres' Assignment Algorithm
http://csclab.murraystate.edu/bob.pilgrim/445/munkres.html

[10] The Official CAPTCHA Site
http://www.captcha.net/

[11] Columbia Object Image Library (COIL-20)
http://www1.cs.columbia.edu/CAVE/software/softlib/coil-20.php

[12] reCAPTCHA
http://recaptcha.net/captcha.html

[13] Inaccessibility of CAPTCHA
http://www.w3.org/TR/turingtest/

[14] Pretend We're Not a Turing Computer but a Human Antagonist
http://caca.zoy.org/wiki/PWNtcha

[15] aiCaptcha
http://www.brains-n-brawn.com/default.aspx?vDir=aicaptcha

Parte II

Parte Subjetiva

6 Parte Subjetiva

Para o trabalho de conclusão, gostaria de realizar algo que envolvesse tanto um estudo teórico, quanto desenvolvimento de alguma aplicação. Inicialmente defini que faria um trabalho na área de Visão Computacional, pois queria conhecer um pouco mais essa área de pesquisa.

A escolha do tema foi um processo demorado. Conversei com praticamente todos os professores do grupo de Visão Computacional do Departamento de Computação em busca de um tema para o projeto. Gostei do tema sugerido pelo professor Ronaldo Fumio Hashimoto, relacionado ao artigo *"Recognizing Objects in Adversarial Clutter: Breaking a Visual CAPTCHA"*. Após uma pesquisa mais aprofundada sobre o tema e o interesse despertado, resolvi conversar com meu orientador para aumentar um pouco o escopo do trabalho. O foco passaria a ser o estudo do descritor *shape context* e desenvolvimento de pequenas aplicações, se possível também incluindo a implementação do algoritmo do artigo da proposta inicial.

O apoio do orientador durante o desenvolvimento do trabalho foi muito importante, sempre disposto a ajudar e se mostrando interessado no trabalho que vinha sendo feito.

6.1 Desafios

O primeiro desafio foi realizar o trabalho numa área nova para mim e já lidar com diversos artigos com textos densos e que exigem certa maturidade. No início, houve grande dificuldade no entendimento dos artigos, que são extremamente concisos.

Durante a etapa de implementação, percebi que muitos detalhes que considerava importantes não eram abordados da forma que eu desejava, ou em alguns casos extremos eram mencionados apenas no rodapé das páginas.

Embora no início o assunto parecesse bem específico, o trabalho se mostrou bem mais abrangente e trabalhoso do que imaginava.

6.2 Frustrações

O projeto teve início tarde demais. A frustração maior é por não ter tido tempo de realizar maiores contribuições e trabalhado também em outras aplicações que tinha em mente.

Com a proximidade do prazo final, percebi que não haveria tempo de resolver todas as questões que ficaram em aberto na implementação do algoritmo para quebra do *captcha EZ-Gimpy*.

No entanto, o fato de ter que lidar com vários desafios e algumas frustrações é visto por mim como uma parte interessante no desenvolvimento do trabalho. Além da questão técnica, tive um grande aprendizado com os erros cometidos durante a realização do trabalho e a experiência proporcionou grande crescimento pessoal.

6.3 Disciplinas relevantes

- **MAC0122 - Princípios de Desenvolvimento de Algoritmos**
  Considero esta uma das disciplinas mais importantes do curso. Lembro de um exerício-programa feito no início da disciplina que consistia na implementação e comparação de desempenho de diversos métodos de ordenação. A análise dos resultados, com diferenças tão acentuadas de tempo, é tão impressionante e revela a importância do estudo de algoritmos, que é o foco dessa disciplina.

- **MAC0211 - Laboratório de Programação I**
  Outra disciplina que considero de fundamental importância. Lidamos pela primeira vez com o desenvolvimento, em grupo, de um projeto grande. Entre outras coisas, aprendemos a modularizar o código, usar controle de versões, Makefile e documentar o código com ferramentas como *doxygen*. Também aprendemos *LaTeX*, que considero uma ferramenta essencial. Também tive a oportunidade de ser monitor desta disciplina e, curiosamente, no ano em que o projeto envolvia Visão Computacional, com a construção de um jogo em que a interação se dava pela *webcam*.

- **MAC0323 - Estruturas de Dados**
  Extensa atividade de programação com diversas estruturas de dados. Esta disci-

plina nos mostra como a escolha de uma boa estrutura para organizar os dados de um problema tem impacto direto na qualidade das implementações.

- **MAC0417 - Visão e Processamento de Imagens**
  Disciplina em que fui apresentado à essa área de pesquisa e aprendi todos os conceitos básicos. Também tivemos a oportunidade de realizar alguns trabalhos em *python* envolvendo manipulação de imagens. Essa disciplina me motivou na escolha do tema do trabalho.

- **MAT0111 - Cálculo Diferencial e Integral I**
  Ao meu ver, a importância dessa disciplina também é indiscutível. Integrais e derivadas estão por toda a parte. Sem o conhecimento básico de cálculo, torna-se inviável o estudo de trabalhos de grande parte das áreas de pesquisa em computação.

- **MAT0139 - Álgebra Linear para Computação**
  Alguns conceitos estudados nesta disciplina também apareceram em alguns momentos no desenvolvimento deste trabalho. No meu caso, não cursei exatamente esta disciplina, pois ainda era aluno do curso de Bacharelado em Matemática Aplicada e Computacional. Cursei duas disciplinas equivalentes e também cursei a displina Aplicações de Álgebra Linear.

- **MAC0441 - Programação Orientada a Objetos**
  Não entendo porque esta disciplina ainda não é obrigatória. Embora seja feito uso de SmallTalk na disciplina e o projeto tenha sido desenvolvido em C++, todos os conceitos do paradigma de orientação a objetos foram muito importantes para o desenvolvimento do projeto.

A experiência como monitor de algumas disciplinas do curso (Introdução à Computação, Laboratório de Programação I, Laboratório de Programação II) foi bastante enriquecedora e também contribuíram na minha formação.

6.4 Passos a seguir se continuar atuando na área

Espero ter a oportunidade de continuar meus estudos no programa de pós-graduação do Instituto, embora não tenha ainda definido uma área de pesquisa específica.

Se seguir na mesma área em que se insere este trabalho, pretendo me envolver em projetos que são desenvolvidos no grupo de Visão Computacional do Instituto.

Por fim, agradeço a oportunidade de ter desenvolvido este trabalho.