Lista de Exercícios 2

Vinicius Graciano Santos - vgs@dcc.ufmg.br

10 de maio de 2010

1 Exercícios Teóricos

1. Ruído pode ser considerado qualquer entidade, dado ou resultado intermediário que não é interessante para os propósitos da computação principal [1]. Por exemplo, em processamento de imagens, o ruído pode ser a flutuação dos valores reais dos pixels, introduzida pelo sistema de aquisição de imagens. Em algoritmos numéricos, o ruído pode vir de erros de arredondamento e também devido a limitada precisão dos computadores.

2. - *Photon Noise:* Devido à natureza quântica da luz e a natureza estatística de produção de fótons, não é possível garantir que o número de fótons sobre um *grid* do CCD seja igual em duas observações consecutivas e independentes.
   - *Thermal Noise:* Elétrons podem ser liberados do CCD através da vibração térmica, assim como podem também ficar presos, não sendo distinguidos de fotoelétrons "verdadeiros".
   - *On-chip Electronic Noise:* Originado no processo de leitura e transmissão do sinal do CCD, gerado inerentemente pelo sistema eletrônico.
   - *Quantization Noise:* São erros introduzidos pelo processo de discretização do sinal analógico.
   - *Film Grain:* Na fotografia analógica, é possível que pequenos grãos tornem-se grãos escuros após absorver uma certa quantidade de fótons, gerando um ruído dependente do sinal na imagem final.

3. Um sistema é linear quando podemos representá-lo por uma transformação T aplicada a uma função $f(x)$ que respeita a regra da superposição:

$$T\{\alpha f_1(x) + \beta f_2(x)\} = \alpha T\{f_1(x)\} + \beta T\{f_2(x)\}$$

   A aplicação de um filtro linear g em um sinal f pode ser modelada através de uma convolução:

$$(f \star g)(t) = \int_{-\infty}^{\infty} f(s)g(t-s)ds = \int_{-\infty}^{\infty} f(t-s)g(t)ds$$

   O filtro da mediana é um exemplo de filtro não-linear, pois esse método não pode ser modelado por uma convolução [1]. Logo, dadas duas imagens I_1 e I_2, no caso geral, $M\{I_1 + I_2\} \neq M\{I_1\} + M\{I_2\}$ - onde M é a mediana da imagem.

4. Queremos encontrar um filtro $h(t)$ que aproxima $\mathcal{H}(s)$, a transformada de Fourier $\mathcal{F}$ de $h(t)$. Logo:

$$\begin{aligned} \mathcal{F}\{h(t)\} &= \mathcal{H}(s) \\ h(t) &= \mathcal{F}^{-1}\{\mathcal{H}(s)\} \end{aligned}$$

A transformada de Fourier pode ser representada no caso discreto por uma matrix M, $n \times n$ [2], assim, dadas as n amostras $\vec{H}$ do filtro no domínio da frequência, temos:

$$\begin{aligned} M\vec{h} &= \vec{H} \\ \vec{h} &= M^{-1}\vec{H} \end{aligned}$$

Como M é quadrada, esse método de design de um filtro é equivalente a resolver um sistema de n incógnitas a partir de n equações linearmente independentes. Esse fato mostra que o método produz um filtro com uma resposta à frequências que é exatamente as frequências amostradas, não colocando restrição alguma entre os pontos da amostra, o que pode levar a resultados de baixa qualidade.

Dessa maneira, nosso objetivo é encontrar um $\vec{h}$ tal que $M\vec{h} \approx \vec{H}$, com M sendo a matriz de transformação $N \times n$, $\vec{H}$ as N amostras de $\mathcal{H}$ e $\vec{h}$ o filtro resultante de tamanho n, com $N > n$. Podemos resolver esse sistema definindo uma fórmula de erro:

$$E(\vec{h}) = |M\vec{h} - \vec{H}|^2$$

Para minimizar esse erro devemos fazer $\frac{dE(\vec{h})}{dt} = 0$, desenvolvendo essa equação encontramos:

$$\vec{h} = (M^T M)^{-1} M^T \vec{H}$$

Note que quando o número de amostras é o mesmo tamanho do filtro, $n = N$, o método acima é equivalente ao primeiro apresentado.

5. Descrição do detector de bordas de Canny:

   - Dada uma imagem I, filtre o máximo possível o seu ruído. Caso o modelo de ruído não seja conhecido, assuma uma distribuição gaussiana. A saída é uma imagem J.
   - Para cada pixel (i, j) de J, calcule os componentes de $\nabla J(i, j)$, J_x e J_y. Estime a intensidade da borda $b_i(i, j) = ||\nabla J(i, j)||$ e a orientação da normal a mesma, $b_o(i, j) = \arctan(\frac{F_y}{F_x})$
   - As bordas contidas em b_i são espessas e logo não estão localizadas com uma precisão aceitável. Para contornar esse problema, aplicamos o algoritmo *Nonmaximum Suppression*.
     Considerando-se as direções $d \in \{0^o, 45^o, 90^o, 135^o\}$, com relação ao eixo horizontal da imagem, para cada pixel (i, j) encontre um d que mais aproxima $b_o(i, j)$ e avalie b_i nos vizinhos de (i, j) na direção d. Se $b_i(i, j)$ for menor do que um dos vizinhos, suprima-o, i.e. $I_n(i, j) = 0$; caso contrário faça $I_n(i, j) = b_i(i, j)$.
   - $I_n(i, j)$ ainda contém bordas geradas por ruído. Assim, devemos aplicar o algoritmo *Hysteresis Threshold*.
     Dados dois limiares $\tau_b < \tau_a$, para todos os pixels de I_n, percorrendo-os em ordem fixa, localize o próximo pixel não visitado tal que $I_n(i, j) > \tau_a$. Percorra ambas as direções perpendiculares à normal da borda enquanto $I_n(i, j) > \tau_b$, marcando todos os pixels visitados e salvando uma lista dos mesmos. A saída é um conjunto de listas descrevendo a posição de um contorno conectado na imagem, além da imagem de intensidade e de orientação.

   Em imagens com muito ruído uma grande quantidade de bordas "falsas"podem ser detectadas, além de que bordas "verdadeiras"podem acabar não sendo totalmente encontradas. Dessa maneira, um dos passos cruciais para o funcionamento do algoritmo é a filtragem do ruído.

2 Exercícios Práticos

1. A filtragem no domínio da frequência obteve um resultado muito mais suave do que no domínio do espaço, mas o design escolhido para esse último filtro foi derivado do caso mais simples, quando o filtro espacial possui a mesma dimensão das amostras no domínio da frequência. Abaixo segue o resultado da filtragem mais suave. O algoritmo foi desenvolvido no Scilab com o auxílio da biblioteca SIVP.

Figura 1: Imagem original e com filtragem no domínio da frequência

```
//Gera um filtro de butterworth.
function FILTER = butterworth(order, cutoff, sze)
  //Cria uma matriz com distancias relativas ao centro.
  RADIUS = zeros(sze, sze);
  for i = 1:sze, for j = 1:sze
      RADIUS(i, j) = sqrt((i - ceil(sze/2))^2 + (j - ceil(sze/2))^2);
  end, end

  //Gera o filtro
  FILTER = 1 ./ (1.0 + (RADIUS ./ cutoff).^(2*order));
endfunction

//Nao estou testando, mas apenas aceita imagens quadradas.
I = rgb2gray(imread("lena.bmp"));

//Filtrando no dominio do espaco.
//O design foi simplificado para o caso mais simples. :-)
SPACIAL_FILTER = abs(fftshift(fft(butterworth(1, 0.4, 5), 1)));
OUTPUT = mat2gray(filter2(SPACIAL_FILTER, I));
imwrite(OUTPUT, "lena_space.bmp");

//Filtrando no dominio da frequencia.
FREQ_FILTER = butterworth(1, 35, size(I, 1));
OUTPUT =  mat2gray(abs(fft(fftshift(fft(im2double(I), -1)).*FREQ_FILTER, 1)));
imwrite(OUTPUT, "lena_freq.bmp");
```

2. O extrator de características *cvGoodFeaturesToTrack* encontra as quinas mais proeminentes em uma imagem. Primeiramente, a função calcula as bordas utilizando-se um algoritmo escolhido pelo usuário, por exemplo, o método de Harris, então uma supressão é aplicada (*non-maximum supression*) e uma cadeia de *thresholds* é feita para eliminar quinas não interessantes. O extrator SURF gera descritores que são invariantes a escala e rotação, sendo parcialmente invariante a iluminação. O método é explicado com um pouco mais de detalhes na próxima questão.

   O algoritmo desenvolvido compara os descritores SURF de uma imagem parametrizada com um vídeo proveniente de uma webcam. O *matching* é feito utilizando-se uma abordagem ingênua para encontrar o vizinho mais próximo de um descritor de um *keypoint*. Dado um keypoint u no vídeo, queremos verificar a sua relação com um keypoint qualquer da imagem. Para isso, encontramos o

descritor da imagem que possui a menor distância euclidiana com relação ao descritor de u. Após esse processo, aplicamos um threshold com relação à diferença entre as duas menores distâncias para definir se algum dos keypoints no vídeo casa com algum na imagem. O tracking foi feito apenas apresentando os keypoints encontrados pelo algoritmo. Essa maneira é interessante para verificarmos os erros no processo de casamento, que geralmente ocorrem em alguns poucos pontos.

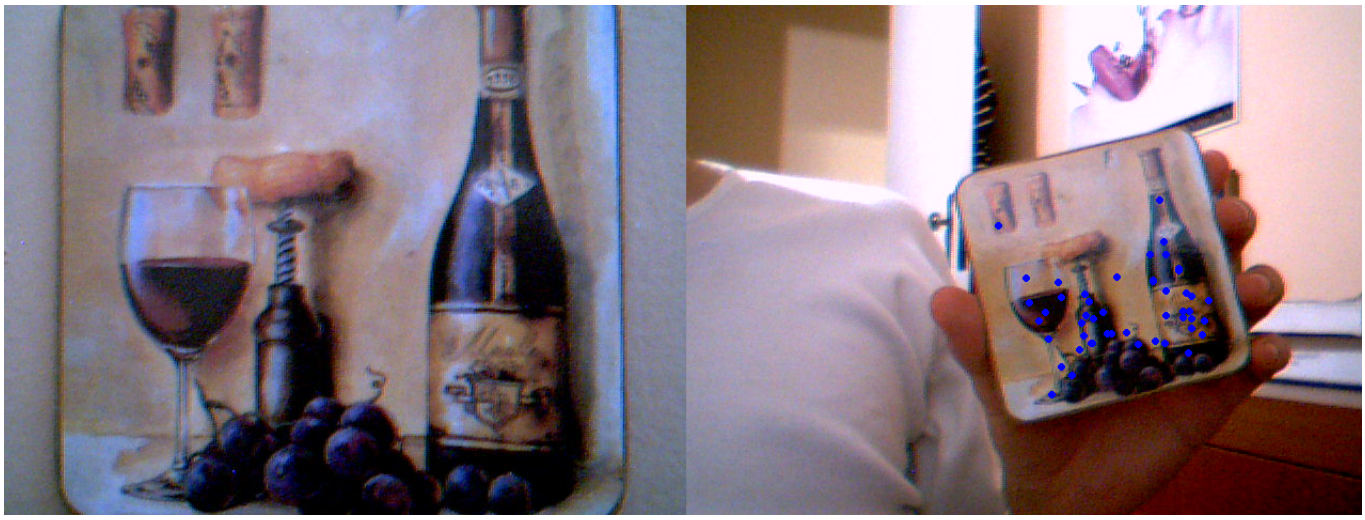

Figura 2: Imagem fonte e tracking (SURF keypoints)

```
1  #include "cv.h"
2  #include "highgui.h"
3
4  #include <iostream>
5
6  using namespace cv;
7  using namespace std;
8
9  bool hasPair(KeyPoint a, float *descriptor, int descriptorSize, int objectKeyPointsNumber, vector <float> &
       objectDescriptors)
10 {
11   double dist, dist1 = 20E+10, dist2 = 20E+10;
12
13   //Verifica se existe uma correspondencia de um keypoint com algum outro na imagem fonte.
14   assert(descriptorSize % 4 == 0);
15   for (int i = 0; i < objectKeyPointsNumber; i++){
16     dist = 0;
17     for(int j = 0; j < descriptorSize; j += 4){
18       double d1 = descriptor[j] - objectDescriptors[descriptorSize*i + j];
19       double d2 = descriptor[j + 1] - objectDescriptors[descriptorSize*i + j + 1];
20       double d3 = descriptor[j + 2] - objectDescriptors[descriptorSize*i + j + 2];
21       double d4 = descriptor[j + 3] - objectDescriptors[descriptorSize*i + j + 3];
22       dist += d1*d1 + d2*d2 + d3*d3 + d4*d4;
23     }
24
25     if (dist < dist1){
26       dist2 = dist1;
27       dist1 = dist;
28      } else if (dist < dist2)
29        dist2 = dist;
30   }
31
32   //Threshold de casamento.
33   if (dist1 < 0.4*dist2)
34     return true;
35   return false;
36 }
37
38 int main(int argc, char **argv)
39 {
40   SURF surf(500);
41   VideoCapture capture(0);
42   Mat frame_bw, frame_gauss;
43   vector <float> source_descriptors;
44   vector <KeyPoint> source_keypoints;
45
46   if (argc < 2){
47     cout << "Usage: simpleTracker [image_source]\n";
48     return -1;
49   }
50
51   if(!capture.isOpened()){
52     cout << "Could not open the video feed!\n";
```

```
53      return -1;
54    }
55
56    frame_bw = imread(argv[1], 0);
57    if (frame_bw.data == NULL){
58      cout << "Could not load the file" << argv[1] << endl;
59      return -1;
60    }
61
62    GaussianBlur(frame_bw, frame_gauss, Size(7,7), 1.5, 1.5);
63    surf(frame_gauss, 255*Mat::ones(frame_bw.rows, frame_bw.cols, CV_8U), source_keypoints, source_descriptors);
64
65    while(1){
66      Mat frame;
67      Point object(0, 0);
68      vector <float> descriptors;
69      vector <KeyPoint> keypoints;
70
71      capture >> frame;
72      cvtColor(frame, frame_bw, CV_BGR2GRAY);
73      GaussianBlur(frame_bw, frame_gauss, Size(7,7), 1.5, 1.5);
74      surf(frame_bw, 255*Mat::ones(frame.rows, frame.cols, CV_8U), keypoints, descriptors);
75
76      for (size_t i = 0; i < keypoints.size(); i++){
77        if (hasPair(keypoints[i], &descriptors[i*surf.descriptorSize()], surf.descriptorSize(), source_keypoints.size
              (), source_descriptors))
78          circle(frame, keypoints[i].pt, 2, Scalar(255,0,0,0), 2);
79      }
80      imshow("Camera feed", frame);
81
82      if(waitKey(30) >= 0)
83        break;
84    }
85    return 0;
86  }
```

3 Exercícios de Pesquisa

O algoritmo *SIFT* (Scale-Invariant Feature Transform) [3] é muito utilizado para encontrar a correspondência de imagens, uma aspecto fundamental de vários problemas em visão computacional. O método provê características (*features*) que são invariantes a escala e rotação, sendo parcialmente invariantes a mudanças na iluminação. O algoritmo é bem robusto para imagens com oclusão, *cluttering* e ruído. Os passos principais do algoritmo são:

- **Scale-space extrema detection:** Faz uma busca considerando todas as escalas e posições da imagem. É implementada eficientemente utilizando-se uma função de diferença de Gaussianas (*difference-of-Gaussian - DoG*) para identificar os potenciais pontos de interesse que são invariantes a escala e rotação.

- **Keypoint localization:** Em cada localização candidata, um modelo detalhado é encabido de determinar a localização e a escala. *Keypoints* são selecionados baseados em medições da sua estabilidade.

- **Orientation assignment:** Uma ou mais orientações são assimiladas para cada *keypoint* baseado nas direções locais do gradiente da imagem. Todas as próximas operações são feitas em dados da imagem que foram transformados de acordo com a orientação, escala e localzação definidas para cada *feature*, provendo a invariância a essas transformações.

- **Keypoint descriptor:** Os gradientes locais da imagem são medidos na escala selecionada dentro da região em volto de cada *keypoint*. Eles são transformados em uma representação que permitem uma boa variação de distorção da sua forma e de alterações na iluminação.

Com relação à correspondência de imagens e ao reconhecimento, os descritores SIFT são primeiramente extraídos de um conjunto de imagens de referência e salvos em um banco de dados. A correspondência com uma nova imagem é feita comparando-se os novos descritores com os de cada imagem previamente armazenados.

O método *SURF* (*Speeded-Up Robust Features*) [4] possui o mesmo objetivo que o *SIFT*, aproximando ou até mesmo superando esse algoritmo com respeito a repetibilidade, distinção e robustez - ainda assim

possuindo uma complexidade menor de tempo. Isso é conseguido utilizando-se imagens integrais - um algoritmo eficiente para encontrar a soma dos valores em um subconjunto retangular de um grid - para acelerar as convoluções bidimensionais discretas, além do aprimoraramento dos pontos fortes dos melhores detectores e descritores existentes, usando medições baseadas em matrizes Hessianas - uma matriz quadrada que contém as derivadas parciais de segunda ordem de uma função. Essa nova abordagem gerou um algoritmo muito mais rápido do que os outros descritores já inventados.

O extrator SIFT normalmente gera mais keypoints do que o SURF, mas em compensação é ineficiente com relação ao tempo de execução [5]. Em imagens onde o SURF não encontra keypoints suficientes para o propósito da computação, o SIFT pode ser mais adequado.

Referências

[1] E. Trucco, A. Verri, "Introductory Techniques for 3-D Computer Vision", Prentice Hall, 1998.

[2] H. Farid. "Fundamentals of Image Processing". Disponível em: http://www.cs.dartmouth.edu/~farid/tutorials/fip.pdf

[3] D. G. Lowe, "Distinctive image features from scale-invariant keypoints,"International Journal of Computer Vision, 60, 2 (2004), pp. 91-110.

[4] H. Bay, A. Ess, T. Tuytelaars, L. V. Gool, "SURF: Speeded Up Robust Features", Computer Vision and Image Understanding (CVIU), Vol. 110, No. 3, pp. 346–359, 2008.

[5] J. Bauer, N. Sunderhauf, P. Protzel. "Comparing Several Implementations of Two Recently Published Feature Detectors". Em Proc. of the International Conference on Intelligent and Autonomous Systems, IAV, Toulouse, France, 2007.

[6] Quantitative Imaging Group, Delft University. Image Processing Fundamentals, disponível em http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip-_Noise.html