



Prof. B. Schiele  
<schiele@informatik.tu-darmstadt.de>

Micha Andriluka  
<andriluka@mis.informatik.tu-darmstadt.de>

## Exercise 5: Interest Point Detection and Matching

(due June 5th 2008, 23:59)

### Question 1: Hessian Detector

In this exercise, we will implement a simple Hessian detector. This detector operates on the second-derivative matrix  $\mathbf{H}$  (called the “Hessian” matrix)

$$\mathbf{H} = \begin{bmatrix} D_{xx}(x, y; \sigma) & D_{xy}(x, y; \sigma) \\ D_{xy}(x, y; \sigma) & D_{yy}(x, y; \sigma) \end{bmatrix}. \quad (1)$$

It defines keypoints as those points for which the Hessian determinant is greater than a certain threshold  $t$ . (For reasons that will be explained in the next lecture, we include an additional scale normalization factor  $\sigma^4$ ):

$$\det(\mathbf{H}) = \sigma^4 (D_{xx}D_{yy} - D_{xy}^2) \stackrel{!}{>} t \quad (2)$$

- a) (6 points) Write a function `hessian` which computes the Hessian determinant for each pixel of a given image and displays the result image.

```
function imgDet = hessian(img,sigma)
...
...
end
```

- b) In order to obtain single points, we need to again extract maxima from the result image. Write a function `nonmaxsup2d` which performs non-maximum suppression in 2D on a given image. It should compare each pixel to its 8 direct neighbors and set the pixel to zero if it is not greater than all neighbors.

```
function imgMax = nonmaxsup2d(img)
...
...
end
```

- c) (2 points) Extend the function `hessian` to perform non-maximum suppression on the determinant image and return the coordinates of all points that pass the threshold. (Note: for obtaining the coordinates, you can use the following command: `[py px] = find(imgPts > thresh)`).

```
function [px py] = hessian(img,sigma,thresh)
...
...
end
```

- d) (2 points) Use the following function `drawpoints` to display the detected points overlaid on the original image. Load the images `graf.png` and `gantrycrane.png` and compute Hessian interest points for them. Experiment with the parameter settings. What do you observe?

```
function h=drawpoints(img, px, py, color)
h=figure; colormap('gray'); imagesc(img); hold on;
for i=1:length(px),
    plot( px(i),py(i),strcat(color,'+') );
end;
hold off;
end
```

## Question 2: Harris Detector

Next, we will implement a Harris detector. This detector searches for corner-like structures by searching for points  $p = (x, y)$  where the autocorrelation matrix  $\mathbf{C}$  around  $p$  has two large eigenvalues. The matrix  $\mathbf{C}$  can be computed from the first derivatives in a window around  $p$ , weighted by a Gaussian  $G(x, y; \tilde{\sigma})$ :

$$\mathbf{C}(x, y; \sigma, \tilde{\sigma}) = G(x, y; \tilde{\sigma}) \star \sigma \begin{bmatrix} D_x^2(x, y; \sigma) & D_x D_y(x, y; \sigma) \\ D_x D_y(x, y; \sigma) & D_y^2(x, y; \sigma) \end{bmatrix}, \quad (3)$$

where “ $\star$ ” denotes the convolution operator. Instead of explicitly computing the eigenvalues  $\lambda_1$  and  $\lambda_2$  of  $\mathbf{C}$ , the following equivalences are used

$$\det(\mathbf{C}) = \lambda_1 \lambda_2 \quad (4)$$

$$\text{trace}(\mathbf{C}) = \lambda_1 + \lambda_2 \quad (5)$$

to check if their ratio  $r = \frac{\lambda_1}{\lambda_2}$  is below a certain threshold. With

$$\frac{\text{trace}^2(\mathbf{C})}{\det(\mathbf{C})} = \frac{(\lambda_1 + \lambda_2)^2}{\lambda_1 \lambda_2} = \frac{(r\lambda_2 + \lambda_2)^2}{r\lambda_2^2} = \frac{(r + 1)^2}{r} \quad (6)$$

we can express this by the following condition

$$\det(\mathbf{C}) - \alpha \text{trace}^2(\mathbf{C}) > t. \quad (7)$$

In practice, the parameters are usually set to the following values:  $\tilde{\sigma} = 1.6$ ,  $\alpha = 0.06$ .

- a) (8 points) Write a function `harris` which computes the matrix  $\mathbf{C}$  for each pixel of a given image, calculates its trace and determinant, and combines them according to equation (7). After applying non-maximum suppression on the result image, it should compare the remaining points to the given threshold and return the coordinates of all points that pass the threshold.

```
function [px py] = harris( img, sigma, thresh )
...
...
end
```

- b) (2 points) Load the images `graf.png` and `gantrycrane.png` and compute Harris interest points for them. Compare the results to those of the Hessian detector from Question 1. Experiment with different parameter settings. What do you observe?

## Question 3: Region Descriptors

In order to find correspondences between interest points, we need to design region descriptors. In this question, we will implement several simple descriptors based on the histogram representations from the previous exercise sheet.

- a) Modify the functions `myhist3` and `myhist4` from the previous exercise sheet such that do not load the input image from disk, but take it as their first parameter. Rename the modified functions to `histrg` and `histrdxdy`. (Alternatively, you can download the functions `histrg` and `histrdxdy` from the class web page.)
- b) Write a function `descriptors_rg` which takes an input image and a list of interest points and computes an  $r, g$  color histogram over the  $m \times m$  sub-windows around each interest point (using the function `histrg`).

```
function D = descriptors_rg( img, px, py, m, nbins )
    rad = round((m-1)/2);
    [h w c] = size(img);
    D = zeros(length(px), bins^2);

    for i=1:length(px)
        minx = max(px(i)-rad, 1);
        maxx = min(px(i)+rad, w);
        miny = max(py(i)-rad, 1);
        maxy = min(py(i)+rad, h);
```

```

    imgWin = img(miny:maxy, minx:maxx, :);
    hist = histrgr(imgWin,bins);
    D(i,:) = hist';
end;

```

- c) (3 points) Write a similar function `descriptors_dxdy` which computes a  $dx$ ,  $dy$  histogram around each interest point (using the function `histrgr`).

```

function D = descriptors_dxdy(img, px, py, size, sigma, bins)
...
...
end

```

- d) (4 points) Write a another function `descriptors_maglap` which computes a  $mag$ ,  $lap$  histogram around each interest point (Note: experiment a little bit with an example image to select the histogram range appropriately).

```

function D = descriptors_maglap(img, px, py, size, sigma, bins)
...
...
end

```

- e) (3 points) Write a function `findnn_chi2` which takes two sets of region descriptors  $D_1$  and  $D_2$  and tries to find for each descriptor in  $D_1$  the nearest neighbor in  $D_2$  using the  $\chi^2$  distance.

```

function [Idx Dist] = findnn_chi2(D1,D2)
...
...
end

```

## Question 4: Matching

Now we have all the components for a small matching application. Download the packages `graff5.tgz` and `NewYork.tgz` from the class web page and extract them to your working directory. These packages contain test scenes with controlled image-plane rotations and viewpoint changes, for which we will try to find point correspondences.

- a) (3 points) First we want to try out the color descriptors. Load the two example images `graff5/img1.ppm` and `graff5/img2.ppm` and perform the following steps.

1. Compute Harris interest points for both images.
2. Compute  $r/g$  color histogram descriptors for all interest points.
3. Find the best matches using the function `findnn_chi2`.
4. Use the following function `displaymatches` to visualize the  $N$  best matches. What do you observe?

```

function displaymatches(img1, px1, py1, img2, px2, py2, Idx, Dist, N)
    % sort the matches according to their scores
    [SDist SIdx] = sort(Dist,'ascend');

    % visualize the N best matches
    h1 = drawpoints(img1, px1, py1, 'y');
    h2 = drawpoints(img2, px2, py2, 'y');
    for i=1:N
        disp(['Match',num2str(i),': dist=',num2str(SDist(i))]);
        figure(h1); hold on; plot( px1(SIdx(i)), py1(SIdx(i)), 'ro' );
        ht = text( px1(SIdx(i))+5, py1(SIdx(i))+14, num2str(i) );
        set(ht,'Color',[1 0 0]); set(ht,'FontWeight','bold');

        figure(h2); hold on; plot( px2(Idx(SIdx(i))), py2(Idx(SIdx(i))), 'ro' );
        ht = text( px2(Idx(SIdx(i)))+5, py2(Idx(SIdx(i)))+14, num2str(i) );
        set(ht,'Color',[1 0 0]); set(ht,'FontWeight','bold');
        pause
    end;
    close all;
end

```

b) (2 points) Now, try the same with Hessian interest points and the  $dx/dy$  histogram descriptors. What do you observe? Which combination gives the better results? Can you think of an explanation?

c) Next, we will try to find matches under image plane rotations. The package `NewYork.tgz` contains a series of test images for which the true homographies are known. On the class web page, you can find a small Matlab program `hom_gui_H` which lets you visualize the corresponding point locations. Load the two images `NewYork/im1.pgm` and `NewYork/im5.pgm` and start the program as follows. This will open a window showing the two images side by side. Click on one image and see what happens.

```
img1 = double(imread('NewYork/im1.pgm'));
img2 = double(imread('NewYork/im5.pgm'));
H = load('NewYork/H1to5');
hom_gui_H(uint8(img1),uint8(img2),H);
```

d) (2 points) Try to find matches between the two images `NewYork/im1.pgm` and `NewYork/im5.pgm` using Hessian interest points and the  $dx/dy$  histogram descriptors. What do you observe?

e) (3 points) Now try the *mag/lap* histogram descriptors on the same image pair. What performance do you get? Which descriptor performs better? Why?

---

*Please turn in your solution by sending an email to Micha Andriluka ([andriluka@mis.informatik.tu-darmstadt.de](mailto:andriluka@mis.informatik.tu-darmstadt.de)) including all relevant m-files before Thursday, June 5th, 23:59*