TUTORIAL 4: IMAGE FEATURES

Lecturer: Christopher Brunner

This tutorial is MARKED.
Due date: 9am Tuesday, the week after the tutorial.

Instructions

- This tutorial will be evaluated. This is GROUP work. You are to do this in your major project groups (3 students).
- A printed copy of this report is to be handed in and a .pdf copy and code is to be emailed by the start of the following lecture.
- A title page must include: AMME4710 Image Processing Tutorial, student name and SID, email address. The electronic version MUST be as a .PDF and be titled GROUPNUMBER_AMME4710_Tutorial4.pdf.
- The email subject MUST BE: **AMME4710 Tutorial 4 – Image Features**.
- Answer questions briefly and clearly. Ensure that you include the correct title for each question. I strongly suggest you answer each question under the headings Aim, Method, Results, Discussion for clarity.
- "Aim" should quickly say what you are doing and why it is an important part of computer vision.
- "Method" needs to be explained briefly using flow charts and/or the theory involved (i.e. not using code or saying "followed instructions" – show you understand what you are doing and how the theory/code works in general.)
- "Results" shows your images and other results with captions explaining what it is (relate it to the method). You can include brief descriptions of what the results are showing.
- "Discussion" discusses, explains and expands on the results.
- If you are displaying images, please make sure the image appears as you expect it to in the hard copy and the pdf. Please use appropriately sized images and remember that if you are comparing images, make them the same size and put them next to each other.
- Work together. Make sure each member understands each question.

Marking Scheme:

Format (clarity, grammar, image size, captions, white space etc): 10
General:
Intro/Aims: 10
Methods: 10
Results: 10
Discussion: 10
Q1: 10
Q2: 10
Q3: 15
Q4: 15

Introduction

This tutorial works through the various concepts in extracting features in images. There are four main parts to this tutorial.

- Harris corner detector
- Canny edge detector
- Hough transform
- SIFT

We use functions from the Matlab Image Processing Toolbox. Use the Matlab "help" to get more details, and see the lecture slides for more information on the principles behind the processes. A zip file including images and other relevant code and functions is included in the website.

You should experiment with this tutorial yourself, by changing the parameters while testing different images available, or on your own pictures. **Test each method with at least three different images**. Starting with default values, the parameters need to be tuned with the objective of bringing out principal features (edges, lines, etc) while reducing irrelevant details (or wrongfully detected features) as much as possible. **Remember that your choice of images and parameters will reflect your understanding of the subject matter.**

1. Harris corner detector

Use the Matlab command `cornermetric` to compute corner response function R.

```
filtCoeff = fspecial('gaussian',[9 1],3);
sensFactor = 0.05;
R = cornermetric(I,'Harris','FilterCoefficients', filtCoeff
,'SensitivityFactor',sensFactor);
figure,imshow(imadjust(R)), colormap(jet)
```

Implement step 4 of the algorithm by yourself, i.e. apply a mask over the response image R based on a threshold.

> Harris corner detector algorithm
> 1. Compute Gaussian derivatives at each pixel
> 2. Compute second moment matrix *M* in a Gaussian window around each pixel
> 3. Compute corner response function *R*
> 4. Find points with large corner response (R > threshold)
> 5. Take the points of local maxima of R

Use the following commands to select the points of local maxima of R and plot the results.

```
corner_peaks = imregionalmax(R);
[cornerY,cornerX]  = find(corner_peaks == true);
figure, imshow(I), hold on
plot(cornerX, cornerY, 'r*');
```

Comment on the results. Are they what you expect? Where does this corner detector work? Where does it not work? Why?

2. Canny Edge Detector

You will test the Canny edge detector using Matlab command `edge`. The thresholds and the smoothing constant have to be tweaked by trial and error to work well for each image (the values indicated were tuned for the 'chess' image). Note that the smoothing has to remove the high spatial frequency texture in the background. After finding the edges, display the binary edge map in a new figure.

```
canny_thresh = [0.1 0.2];
canny_sigma = 2;
BW = edge(I,'canny', canny_thresh, canny_sigma);
figure, imshow(BW)
```

Compare the results of the Canny edge detector with the other algorithms available: Sobel and Laplacian of Gaussian (LoG) detectors. Is there any algorithm that is superior? Why?

3. Hough Transform

Use the Matlab toolbox `hough` function to perform the Hough tranform. In the display of the accumulator array, the rho axis runs vertically, and the theta axis runs horizontally. The two roughly vertical rows of bright dots correspond to the two sets of lines at different angles on the chessboard.

The default settings for the bin sizes mean that, for this input image, there are more bins in the rho direction than in the theta direction. This can easily be changed, but the defaults work well in this case.

```
[H,theta,rho] = hough(BW);

figure, imshow(imadjust(mat2gray(H)),[],'XData',theta,'YData',rho,...
        'InitialMagnification','fit');
xlabel('\theta (degrees)'), ylabel('\rho');
axis on, axis normal, hold on;
colormap(hot)
```

Find the peaks in the transform. We use the toolbox function to locate the peak positions. Again, the parameters have to be tweaked to suit the specific image. For the 'chess.png' image, the maximum number of peaks to find is 21, and peaks must not be closer together than 27 bins in rho and 11 bins in theta.

```
maxPeaks = 30;
peakSep = [27 11];
peakThresh = 0;
P = houghpeaks(H, maxPeaks, 'Threshold', peakThresh, 'NhoodSize', peakSep);

x = theta(P(:,2));
y = rho(P(:,1));
plot(x,y,'s','color','blue');
```

Find the lines using the `houghlines` Matlab function and then use a loop to plot each line superimposed over the original image. The results are good but not perfect. But that's to be expected, given that there are many objects and textures in the image.

```
lineGap = 5;
lineMinL = 7;
lines = houghlines(BW,theta,rho,P,'FillGap',lineGap,'MinLength',lineMinL);

figure, imshow(I), hold on
for k = 1:length(lines)
   xy = [lines(k).point1; lines(k).point2];
   plot(xy(:,1),xy(:,2),'LineWidth',2,'Color','red');
end
```

The standard Hough transform does not localise the beginning and end of each line in the image. Can you explain how Matlab implements this functionality?

4. SIFT

Run the `sift` command to call the appropriate binary to extract SIFT features. The keypoint features are returned in matrix form. Use `showkeys` to display the keypoints superimposed on the image.

```
[image, descrips, locs] = sift('chess2.jpg');
showkeys(image, locs);
```

The `match` command is takes two images and shows the two input images next to each other, with lines connecting the matching locations as a result. It extracts SIFT features from each image and matches the features between the two images.

```
match('scene.pgm','book.pgm');
```

This function uses an approximation of the distance as the dot product for speed. Implement the matching using the sum of square differences and ratio distance, as described in the lecture.

Given a feature in I_1, how to find the best match in I_2?

- Define distance function that compares two descriptors
- Test all the features in I_2, find the one with min distance

$SSD(f_1, f_2)$ is the sum of square differences between entries of the two descriptors.
Better approach: ratio distance = $SSD(f_1, f_2) / SSD(f_1, f_2')$
f_2 is best SSD match to f_1 in I_2
f_2' is 2nd best SSD match to f_1 in I_2