TP2: Feature Detection

March 25, 2009

1 Introduction

Feature detection is a low-level image processing operation that aims to identify the presence of features in an image location. Usually, the extracted features are then used to perform more advanced operations like classification, image matching, texture analysis and texture recognition etc. As features we can define the edges, the corners, the blobs and the ridges. Depending on the application different features or a combination of features can be used. For example, in the case of aerial images and for the problem of road detection it is natural to consider the ridges as appropriate features.

A big difficulty in feature detection is that the way we perceive and find meaningful differences depends heavily on the scale of observation. This is a well known fact in physics, where a great range of levels of scale is used in order to describe different phenomena. In order to be able to take into account the fact that the importance of information in an image varies with the scale, either because it reflects multi-scale information of the real world or is due to noise introduced during the acquisition process, the notion of scale should be incorporated in the image representation. This leads to a multi-scale representation of the image data and thus, to methods that try to detect features across different scales.

2 Blob Detection

Blobs are defined as points or regions of an image that are either brighter or darker than their neighbor points. By introducing the notion of scale, blobs will be determined as significant by their stability in the scale-space. The idea is that by studying the evolution properties of normalized differential operators over the scales, we should be able to localize interesting image structures like blobs.

One of the simplest way to detect blobs is reported in [4] [3] and consists of the following steps:

1. The image is convolved by different Gaussian Kernels of increasing sizes in order to create a scale-space representation $L(x, y, \sigma) = g(x, y, \sigma) * I(x, y)$

2. The normalized Laplacian operator ∇^2_{norm} is then applied to the scale-space volume. The normalized Laplacian operator is defined as

$$\nabla^2_{norm} L = \sigma^2 (L_{xx} + L_{yy}) \tag{1}$$

3. The scale-space maxima/minima, that is the points that are at the same time maxima/minima of $\nabla^2_{norm} L$ with respect to both the space and scale, are then detected. In other words, a point is regarded as bright (dark) blob if its value is greater (smaller) than the value of its neighbors in its scale, as well as in one finer and one coarser scale. In that way, both the point and the scale is selected in an automatic way.

3 Work

3.1 Work to do

You should send us back a report containing the code that you wrote, some results from your experiments and explaining your observations. For each result you return, please give us the parameters you used. The report should be handed back in seven days. The eighth day the solutions will be posted so no further reports will be accepted.

3.2 Blob Detection

The previously described algorithm for Blob detection should be implemented.

In order to do so, you should implement each step of the algorithm by taking advantage of the code that you wrote for the first lab. However, you should be careful of the relation between the standard deviation and the radius of the kernel. With a small radius and a big standard deviation, only a small part of the Gaussian will be used, and thus the used filter will not be a Gaussian one. Defining a radius as 3σ is a common choice.

Your algorithm should be able to get as input the number of the different scales of the scale-space description of the image, the parameters for the initial Gaussian Kernel as well as a threshold value to select the most significant blobs. The scale-space representation is created by applying Gaussian Kernels of increasing standard deviations. The standard deviations should be a geometric series with a step of $\sqrt{2}$. Similarly to the previous lab, the normalized Laplacian operator should be applied to the image, and then a local scale-space search should be performed in order to identify the points of interest. In order to display the blobs a function will be given to you.

Two images will be given to you, you should use the smallest one in order to obtain results in a reasonable time.

3.3 Studying the effect of the scale selection

For diverse initial Gaussian Kernels, different numbers of scales used and different thresholds, you should study the output of your algorithm. For the smallest image, you should use an

initial standard deviation of 1, along with a threshold value of 25.

3.4 Bonus

CImg includes a fast method to apply the Gaussian Filter: *get_blur*(). By replacing your own method, compute the blobs for the full size image. Moreover, in order to understand how edge detectors wok, you should experiment with the magnitude of the gradient of the smoothed images at various scales and add to your report the results obtained, as well as your observations.

4 Usefull tools for this lab

4.1 Defining Tables

By tables we refer to arrays of a given size. In order to use such tables, one has to allocate some memory. It may be done either by static or dynamic allocation. Both methods do not apply in any situation. Static allocation may only be done when the size of the table to allocate is known before the compilation, whereas dynamic allocation may be done on the fly.

Static allocation may be done only when the size of the table is known before compilation.

```
double coord[3];    // allocate a table of 3 double
coord[0] = 1;       // set the value 1 to the first element of the table
```

Dynamic allocation may be done using a size that is known only when the application is being executed. The drawback is the requirement for explicit freeing of memory.

```
int size = 5;
double* coord = new double[size];    // allocate a table of 5 double
coord[0] = 1;       // set the value 1 to the first element of the table
delete [] coord;    // free the memory used by the table
```

4.2 Vectors

Unlike tables, vectors allow to work with objects which size is not known at initialization and may vary through time. Vectors are defined in the standard C Library. They are templated classes. First the vector should be included.

```
#include <vector>
```

We will now present some useful methods using vectors. First a vector should be defined, then we may add some elements using the push_back method. Access to any element is given as for tables, however when the element is not defined an error will be returned. More information may be found in [2] [1].

```
std::vector<double> vect; // define a vector of doubles
vect.push_back(2);        // add 2 at the end of the vector
vect.clear();             // remove all the elements inside the vector
vect[2];                  // access to the 3rd element
vect.size();              // number of elements inside the vector
```

4.3 Provided code

Two classes will be provided. The first class will manage Blobs. It is only a container, whose members give:

- x,y coordinates of the blob
- t the scale where the blob was found

The second class DisplayBlob will manage the display of blobs in a standard way. Each blob will be displayed as a circle, whose radius will be proportional to the scale of the blob. Display should be done in two steps. First a display should be created by creating a DisplayBlob object, with parameters as an image and a vector of blobs. All given blobs will be displayed. Then the method Launch will manage user interaction, along with saving an image with blob when a path is given. By pressing any key one may switch from the blob view to the initial image.

```
DisplayBlob window(img, vBlob);   // vBlob vector of blobs to display
window.Launch("./img.bmp");       // manages user interaction and save
```

References

[1] Vectors on C++. `http://www.cplusplus.com/reference/stl/vector/`.

[2] Vectors on SGI. `http://www.sgi.com/tech/stl/Vector.html`.

[3] Wikipedia on Blob Detection. `http://en.wikipedia.org/wiki/Blob_detection`.

[4] Tony Lindeberg. Feature detection with automatic scale selection. *International Journal of Computer Vision*, 30:79–116, 1998.