Artificial Vision Report - Lab 2

Olivier Jais-Nielsen

April 2, 2009

0.1 C++ functions

0.1.1 Gaussian distribution

The `gaussianDist` fucntion is the one that was written for the previous lab. Cf. Algorithm 0.1.

Algorithm 0.1 gaussianDist

```
float gaussianDist(float r, float deviation)
{
        return (exp(-(r*r)/(2*deviation*deviation)))/(
            deviation*sqrt(2*M_PI));
}
```

The arguments are the considered point `r` and the standard deviation `deviation`.

0.1.2 Gaussian mask

The function `gaussianKernel` is almost the same as the one in the previous lab, except that in order to be more memory efficient, it takes as a first argument a reference to an image to fill with the mask. Cf. Algorithm 0.2.

Algorithm 0.2 putGaussianKernel

```
void putGaussianKernel(CImg<float> &mask, int radius,
    float deviation)
{
        float r;
        float I;
        for (int i = -radius; i <= radius; i++)
        {
                for (int j = -radius; j <= radius; j++)
                {
                        r = i*i + j*j;
                        r = sqrt(r);
                        I = gaussianDist(r, deviation);
                        mask(i + radius, j + radius) = I;
                }
        }
        mask /= mask.norm(1);
}
```

0.1.3 Main program

The main program gets the image file name as the first console argument (after the name of the program itself). Then it asks successively for the initial standard deviation `firstDeviation`, the number of scale levels to compute `scalesNb`, the threshold value for selecting the blobs `threshold` and wether it should save the picture showing the blobs. All the radius values are chosen as three times the corresponding standard deviation.

Then, it creates a regular laplacian kernel `laplacianMask` according to this formula : $\begin{pmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{pmatrix}$.

0.1.4 Computation of the scale-space volume

After that, a loop fills an image list `gaussianMasks` with gaussian kernels corresponding to the different scale levels (the n-th kernel has a deviation equals to $\sqrt{2}^n$ multiplied by the first standard deviation, and a kernel equals to three times the previous deviation), scale-normalizes the laplacian kernel, stores in another image list `scaleSpace` the convolution of the input image with the previous gaussian and laplacian kernels. Cf. Algorithm 0.3 on the following page.

0.1.5 Blob search

For each scale level from the second one to the second last one, for each pixel which is not on the edges of the frame, we check if its value is bigger (resp. smaller) than his neighbouring pixels which are the one that are at most one pixel away and one scale away from it. If that is the case and if the pixel value is higher (resp. lower) than the threshold value, we add a blob to the previously created vector of blobs `blobs`, with the corresponding deviation and coordinates. The deviation value is computed similarly as before. Cf. Algorithm 0.4 on page 5.

Eventually, the main program displays the blobs over the input image and optionaly save these results, indicating in the file name the different parameters.

0.2 Efficiency of the blob detection

Using the `get_blur` method dramatically speeds up the algorithm. Without that, with a initial deviation of 1 and 10 scale levels, the computing time seemed endless.

For the small image, with an initial deviation of 1, 7 scale levels and a threshold of 25, the result is quite good, all the principal blobs being detected with very few wrong ones. The same quality is obtained for the big image and a number of scale levels of 10. Cf. figure 1 on page 4.

However, when the scale levels become too high, some extra big blobs are detected. Cf. figure 2 on page 4.

Algorithm 0.3 First loop

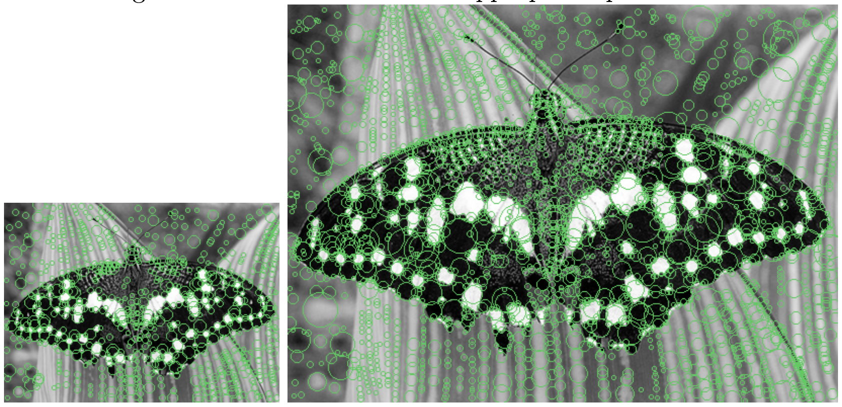
```
CImgList<float> gaussianMasks(scalesNb);
CImgList<float> scaleSpace(scalesNb);
float deviation = firstDeviation;
int radius;

laplacianMask *= deviation * deviation;
for (int i = 0; i < scalesNb; i++)
{
        radius = 3 * deviation;
        gaussianMasks[i].assign(2 * radius + 1, 2 *
            radius + 1);
        putGaussianKernel(gaussianMasks[i], radius,
            deviation);
        scaleSpace[i] = img.get_convolve(gaussianMasks[i
            ]);
        // scaleSpace[i] = img.get_blur(deviation,
            deviation, 0);
        scaleSpace[i].convolve(laplacianMask);
        laplacianMask *= 2;
        deviation *= sqrt(2.);
}
```

After normalizing the laplacian mask with the initial deviation value, the loop performs each operation with the previously defined deviation and updates it value. In the same way, it updates the normalization of the laplacian mask at each iteration.
The commented line can be used in replacement of the previous one, for a much faster convolution of the gaussian kernels.

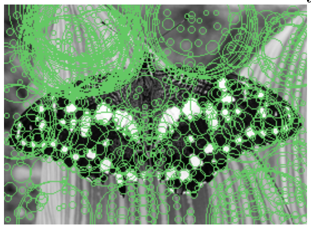
A too high threshold makes it impossible to detect many of the dark blobs while a too low threshold increase dramatically the number of blobs detected, some of the little ones being actually part of bigger ones also detected. Cf figure 3 on page 6.

Figure 1: Blob detection with appropriate parameters



First standard deviation : 1
Scale levels : 7 and 10
Threshold : 25

Figure 2: Blob detection with too many scale levels



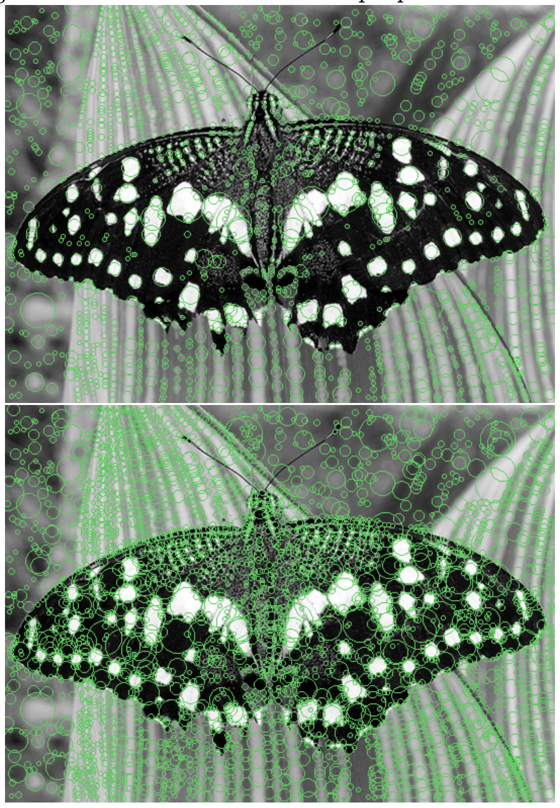
First standard deviation : 1
Scale levels : 13
Threshold : 25

Algorithm 0.4 Second loop

```
float point;
bool isBigger, isSmaller;
vector<Blob> blobs;

deviation = firstDeviation;
for (int i = 1; i < scalesNb - 1; i++)
{
        deviation *= sqrt(2.);
        for (int x = 1; x < w - 1; x++)
        {
                for (int y = 1; y < h - 1; y++)
                {
                        point = scaleSpace[i](x,y);
                        isBigger = true;
                        isSmaller = true;
                        for (int k = i - 1; k <= i + 1; k
                            ++)
                        {
                                for (int u = x - 1; u <=
                                    x + 1; u++)
                                {
                                        for (int v = y -
                                            1; v <= y + 1;
                                             v++)
                                        {
                                                if (k !=
                                                    i || u
                                                     != x
                                                    || v
                                                    != y)
                                                {
                                                        isBigger
                                                            =
                                                            isBigger
                                                            &&
                                                            (
                                                            point
                                                            >
                                                            scaleSpace
                                                            [
                                                            k
                                                            ](
                                                            u
                                                            ,
                                                            v
                                                            )
                                                            )
                                                            ;
```

Figure 3: Blob detection with inapropriate thresholds



First standard deviation : 1
Scale levels : 10
Threshold : 100 and 5