Filtrage par convolution et détection de bords

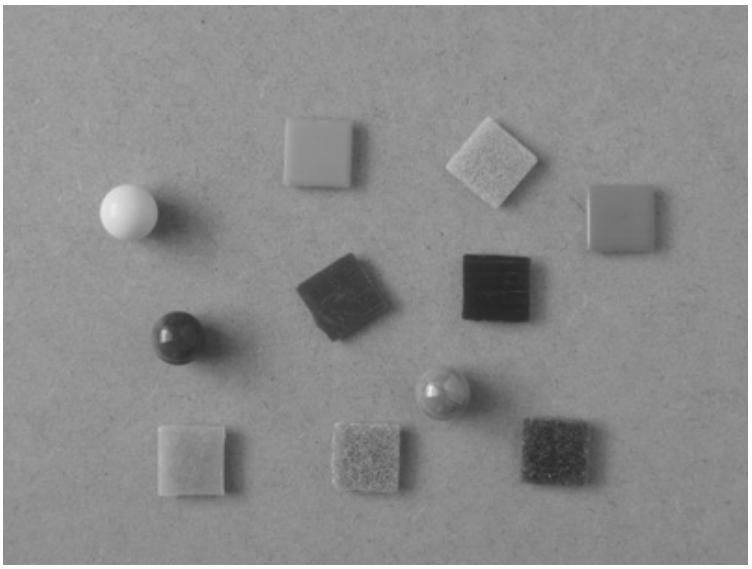
1. Filtrage par convolution

```
Needs['CV`'];
CVOpen[];
```

Lecture d'une image en niveaux de gris :

```
img=CVLoadImage['CIMG3181.jpg',CVLoadImageGrayScale];
s=CVGetSize[img];
```

```
Image[CVGetImageArray[img,1,1]]
```



Le noyau de convolution est une matrice carrée de taille impaire. On choisit ici une matrice 3x3, composée de flottants 32 bits et d'une seule couche :

```
noyau=CVCreateMat[3,3,CV32F,1];
```

La matrice est remplie (l'indice 1 indique la première couche) :

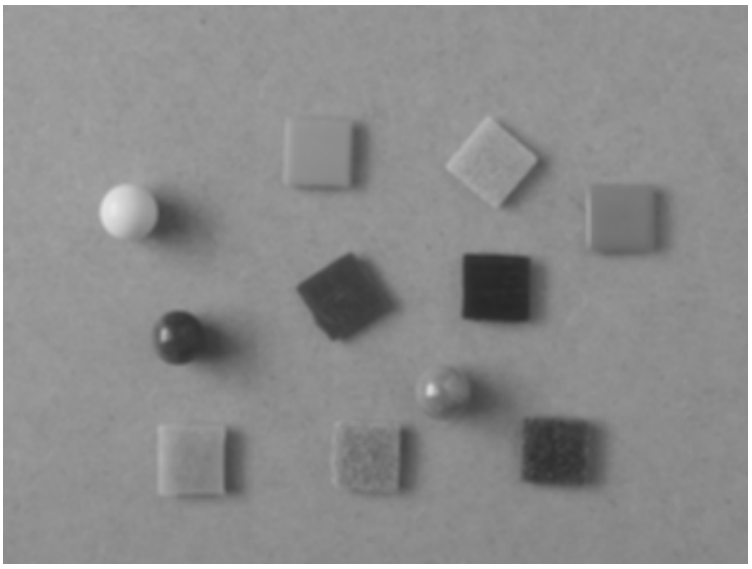
```
CVSetArray[noyau,1,N[{{1/9,1/9,1/9},{1/9,1/9,1/9},{1/9,1/9,1/9}}]]
```

Remarquer la présence de l'opérateur `N`, nécessaire pour convertir les fractions en nombres à virgule flottante.

Le filtrage consiste à effectuer le produit de convolution de cette matrice par l'image. Dans le cas présent, cela revient à remplacer chaque point de l'image par la moyenne arithmétique de ce point et des 8 points voisins. Le filtrage est réalisé par :

```
img2=CVCreateImage[s,IplDepth8U,1];
CVFilter2D[img,img2,noyau];
```

```
Image[CVGetImageArray[img2,1,1]]
```



2. Application : opérateurs de dérivation

2.a. Dérivation simple

Le noyau de convolution suivant permet d'obtenir une approximation de la dérivée première suivant x (avec un moyennage dans la direction y) :

```
noyau=CVCreateMat[3,3,CV32F,1];
CVSetArray[noyau,1,N[{{1,0,-1},{1,0,-1},{1,0,-1}}]]
```

La convolution conduit dans ce cas à des valeurs comprises entre -255 et 255. Il faut donc créeer une image de nombres 16 bits signés :

```
img3=CVCreateImage[s,IplDepth16S,1];
CVFilter2D[img,img3,noyau]
```

Voyons les valeurs minimales et maximales de l'image obtenue :

```
minmax=CVMinMaxLoc[img3]
```

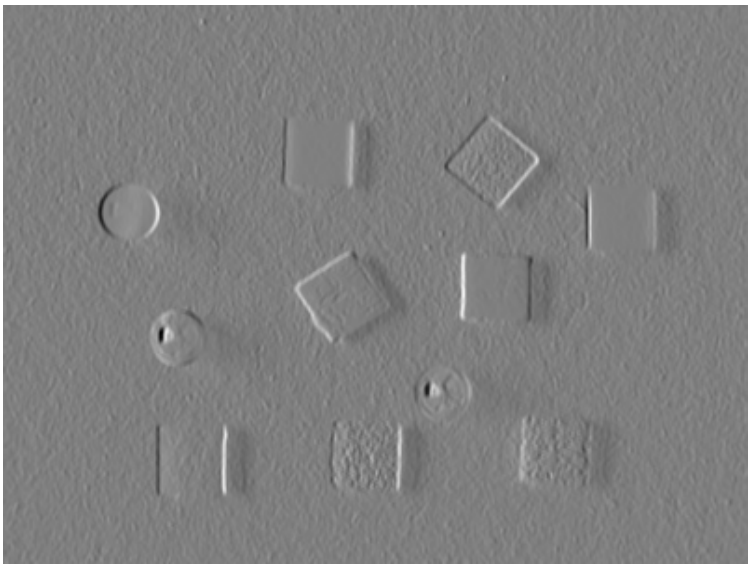
```
{{-248., 246.}, {84., 176.}, {247., 155.}}
```

Le premier doublet donne les valeurs minimale et maximale, les deux doublets suivants sont les positions du minimum et du maximum sur l'image.

Pour obtenir une image représentable, il faut transformer l'intervalle [-255,255] en [0,255] :

```
img4=CVCreateImage[s,IplDepth8U,1];
CVConvertScale[img3,img4,0.5,127.0];
```

```
Image[CVGetImageArray[img4,1,1]]
```



Les dérivées positives apparaissent comme des traits blancs, les dérivées négatives comme des traits noirs. Les zones où la dérivée est nulle sont gris moyen (valeur 127 environ).

2.b. Laplacien

Le laplacien de l'image est obtenu de la manière suivante :

```
img5=CVCreateImage[s,IplDepth16S,1];
CVLaplace[img,img5,3];
```

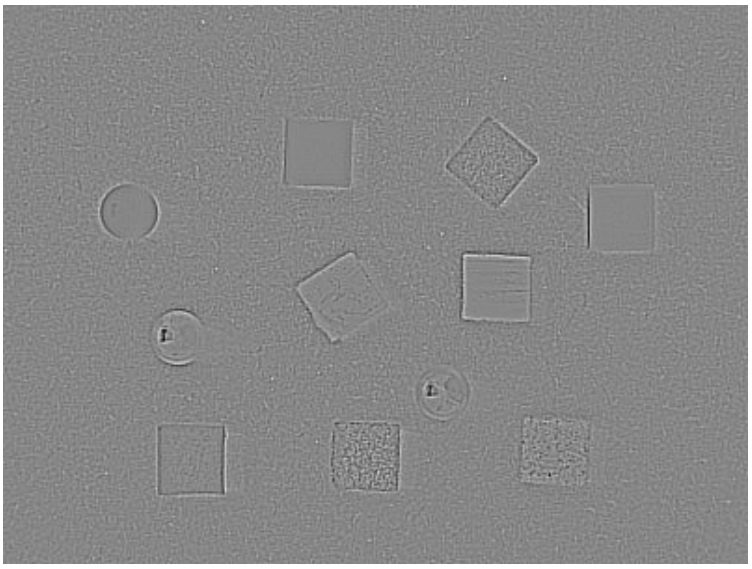
Le dernier argument indique la taille du noyau utilisé. Voyons le résultat :

```
minmax=CVMinMaxLoc[img5]
```

```
{{-382., 340.}, {86., 179.}, {269., 39.}}
```

```
img6=CVCreateImage[s,IplDepth8U,1];
range=minmax[[1,2]]-minmax[[1,1]];
CVConvertScale[img5,img6,255.0/range,-255.0*minmax[[1,1]]/range];
```

```
Image[CVGetImageArray[img6,1,1]]
```

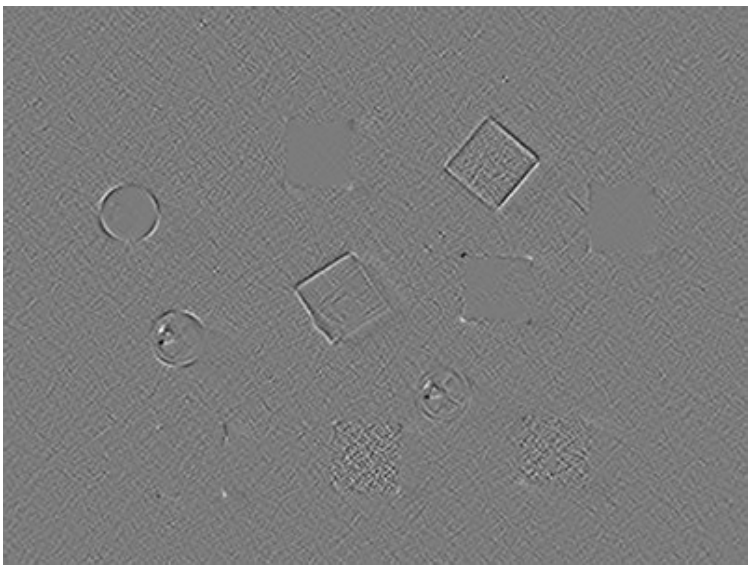


2.c. Opérateur de Sobel

Cet opérateur est principalement utilisé pour la détection de bords. Pour l'utiliser, on doit préciser l'ordre des dérivations par rapport à x et y :

```
img7=CVCreateImage[s,IplDepth16S,1];
xorder=1; yorder=1;
CVSobel[img,img7,xorder,yorder,3];
minmax=CVMinMaxLoc[img7];
img8=CVCreateImage[s,IplDepth8U,1];
range=minmax[[1,2]]-minmax[[1,1]];
CVConvertScale[img7,img8,255.0/range,-255.0*minmax[[1,1]]/range];


Image[CVGetImageArray[img8,1,1]]
```

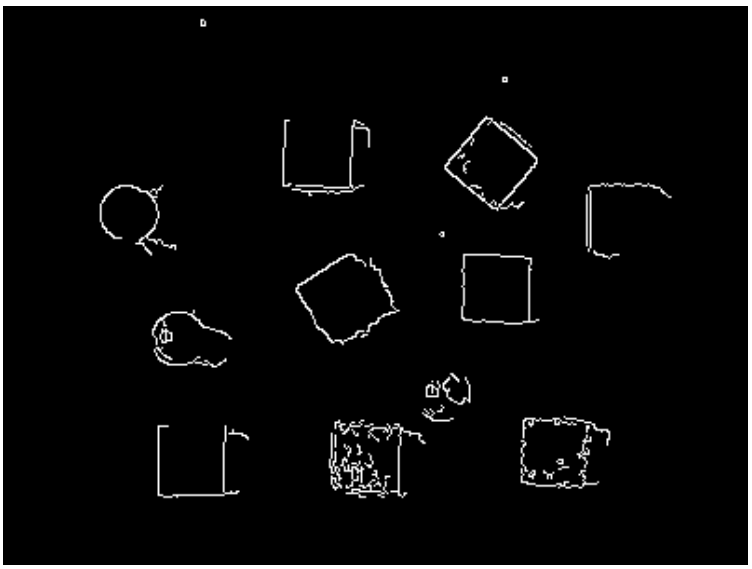


3. Détection de bords de Canny

La détection de bords de Canny repose sur le calcul de gradients. Il faut préciser un seuil bas et un seuil haut. Si un pixel a un gradient supérieur au seuil haut, il est considéré comme un bord et transformé en 255 (blanc). Si le gradient est inférieur au seuil bas, il est transformé en 0 (noir). Si le gradient est entre les deux seuils, le pixel est considéré comme faisant partie d'un bord seulement s'il est connecté à un autre pixel de bord.

```
img9=CVCreateImage[s,IplDepth8U,1];
seuilHaut=120.0;
seuilBas=60.0;
CVCanny[img,img9,seuilBas,seuilHaut,3];


Image[CVGetImageArray[img9,1,1]]
```



On remarque que certains bords d'objets ne sont pas détectés. Il s'agit toutefois d'une image difficile car le contraste entre les objets et le fond est très faible. D'autre part, des éléments de la texture de certains objets sont interprétés comme des bords. Une solution à ce problème consiste à appliquer un flou à l'image avant de détecter les bords.

4. Lissage

Le lissage est un filtrage par convolution qui consiste à estomper les détails d'une image, par exemple pour réduire le bruit ou faire disparaitre une texture gênante.

On commence par sélectionner la zone à filtrer avec l'éditeur d'image :

```
edit=CVJImageEditor['formes',True];
CVJShowImage[edit,img];
rectList = CVJGetPixelRectangleList[edit];
rect = Floor[rectList[[1]]]
```

```
{33, 34, 183, 171}
```

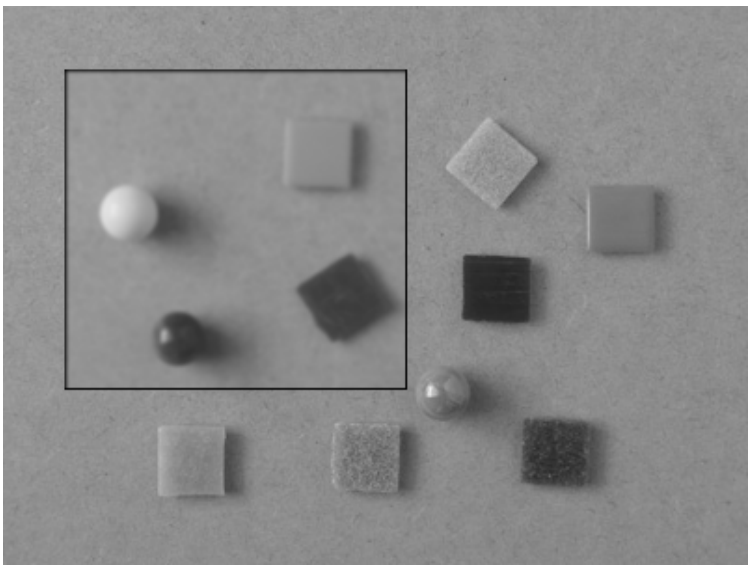
Cette zone rectangulaire sert à définir la région d'intérêt (Region Of Interest) :

```
img10=CVCreateImage[s,IplDepth8U,1];
CVConvertScale[img,img10,1.0,0.0];
CVRectangle[img10,rect,{0,0,0},1];
CVSetImageROI[img10,rect];
```

Comme exemple de lissage, appliquons un flou gaussien sur un noyau de 5x5 :

```
CVSmooth[img10,img10,CVGaussian,5,5,0.0];
CVResetImageROI[img10];
```

```
Image[CVGetImageArray[img10,1,1]]
```



```
CVClose[];
```