Université de Technologie de Compiègne

SY31

Rapport de TP

5 - Traitements d'images

Automne 2014

Romain PELLERIN - Kyâne PICHOU - Nianfei SHI - Ibraim SOEROPAIMAN
Groupe de TP 2
27 décembre 2014

Table des matières

1 Introduction

L'objectif de ce TP est d'étudier les opérateurs de dérivation spatiale en traitement d'images. Pour cela, nous allons développer un module RTMaps prenant comme entrée le flux d'images de la caméra pan/tilt et donnant en sortie une image de la norme du gradient.

Nous commencerons par convertir l'image couleur fournie par la webcam en **niveaux de gris**. Puis nous appliquerons le **masque de Sobel**, qui est très utilisé dans les algorithmes de détection de contours. Enfin, nous appliquerons un autre masque très utilisé, le **masque de Prewitt**.

2 Principe de filtre dérivateur

Il existe de nombreux types de traitement d'images. Celui que l'on utilisera ici est un filtre dérivateur. Dans ce cas, on peut approcher la dérivée par une différence finie tel que :

$$\frac{\partial f}{\partial x} \simeq f(x+1) - f(x) \tag{2.1}$$

On écrit aussi :

$$\frac{\partial f}{\partial x} \simeq g_{-1} f(x+1) + g_0 f(x) \tag{2.2}$$

Soit :

$$\frac{\partial f}{\partial x} \simeq \sum_{k=-1}^{k=0} g_k f(x-k) \tag{2.3}$$

Ce qui correspond à une opération de convolution et donc de filtrage numérique.

Cependant ceci est vrai pour un cas en une dimension. Il faut donc généraliser au cas en 2D pour le traitement de l'image. En général, on nous donne la formule de calcul suivante pour le filtrage linéaire d'une image f de taille M x N avec un masque de taille m x n.

$$g(x,y) = \sum_{s=-a}^{a} \sum_{t=-b}^{b} w(s,t) f(x+s, y+t) \tag{2.4}$$

où a=(m-1)/2 et b=(n-1)/2.

Cette équation doit-être appliquée pour tout les points de l'image afin d'obtenir une image filtrée. Étant donné que notre masque de filtrage se base sur les pixels autour de celui « selectionné », il conviendra d'éviter les effets de bords. On choisira d'ajouter des zéros autour de l'image.

3 Développement

3.1 Conversion de l'image en niveaux de gris

Le filtrage s'effectue sur une image simple canal (une seule composante entre 0 et 255, le gris). En revanche le RGB (Red Green Blue) fourni par la caméra utilise 3 canaux. Le conversion s'effectue comme suit :

$$Y = 0.299R + 0.587G + 0.114B \tag{3.1}$$

3.2 Implémentation du filtrage dérivateur

On utilisera 2 masques de dérivation différents.

— Masque de Sobel

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.2}$$

— Masque de Prewitt

$$G_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix} G_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{3.3}$$

Pour chaque masque l'image dérivée de sortie sera la norme du gradient en chaque pixel c'est à dire :

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.4}$$

3.3 Questions et Code C++

Nous allons donc développer nos filtres en C++ en complétant les fichiers fournis. Il sera utilisé une structure `MAPS::IplImage` contenant la plupart des informations d'une image. Le plus important est un tableau contenant les informations de chaque pixel.
En niveau de gris, le tableau associé à un pixel est un `unsigned char` codant le niveau. En RGB, le tableau se visualise par groupe de 3 cases. Ces trois cases contiennent chacune un `unsigned char` correspondant aux niveaux de rouge, vert et bleu de chaque pixel.

3.3.1 Question 1

On donne le pseudo-code capable d'appliquer une fonction F à chaque pixel d'une image en niveau de gris. On utilise `img` qui correspond à l'adresse du premier pixel dans le tableau.

$i \leftarrow 0$
for (i=0 ; $i <$ taille(img) ; i++) **do** F($img[i]$)
end for

3.3.2 Question 2

De même que pour la question 1, sauf qu'ici on veut appliquer la fonction à une image couleur RGB.

$i \leftarrow 0$
for (i=0 ; $i <$ taille(img) ; i++) **do** F(img[3*i],img[3*i+1],img[3*i+2])
end for

3.3.3 Question 3

Dans la structure `MAPS::IplImage` les pixels sont stockés dans un tableau ligne après ligne. C'est à dire que la première ligne de pixels de l'image occupe les premières cases du tableau, puis la seconde ligne occupe les cases suivantes, etc... Dans une image couleur RGB, on se souvient aussi que chaque pixel prend 3 cases dans le tableau. Donc pour accèder au pixel d'indice (i,j), on fait :

$pixel \leftarrow (j * largeur * 3) + (3 * i)$

où `largeur` correspond à la largeur de l'image (en nombre de pixels).

3.3.4 Question 4

Lors de la conversion RGB vers niveau de gris, il faut s'assurer que le niveau de gris ne soit pas supérieur à 255 puisque sur 8 bits on peut stocker des valeurs décimales allant de 0 à 255. En réalité ce n'est pas un vrai problème. En effet on remarque que dans notre formule de calcul du niveau de gris, la somme des coefficients multipliant chaque canal est égale à 1 (0.299+0.587+0.114=1). Cela signifie que quelque soit la valeur de notre pixel couleur (de (0,0,0) à (255,255,255)), le résulat sera toujours compris entre 0 et 255.

3.3.5 Code C++

RGB to Gray

On se propose maintenant de coder réellement notre filtre. On commence par notre fonction de conversion :

C++

```
void SobelComponent::rgb2gray(IplImage const& src, IplImage& dst)
{
  int i;
  for(i=0;i<src.width*src.height;i++)
  {
    dst.imageData[i]=src.imageData[3*i]*0.299+src.imageData[3*i+1]*0.587+src.
        imageData[3*i+2]*0.114;
  }

}
```

Cette fonction est relativement simple. Elle se contente d'implémenter la formule de conversion donnée, en prenant garde de parcourir correctement (sans erreur d'indice) le tableau image en RGB. Pour cela il suffit de bien se souvenir qu'un pixel occupe 3 cases. Il faut également faire attention à l'ordre des composantes (rouge, puis vert et enfin bleu).

Convolution

On passe ensuite à la fonction réalisant la convolution entre notre masque et notre image. Celle-ci prend en entrée les coordonnées x et y correspondant au point à traiter, ainsi qu'un

masque. Ceci nous permet d'utiliser la même fonction convolution quelque soit le masque utilisé (ici Sobel ou Prewitt) puisqu'il est passé en paramètre.

C++
```
int SobelComponent::convolution(IplImage const& src, int x, int y, int const*
    mask)
{
  int R=0;
  // On parcourt les pixels autour du pixel (x,y)
  for(int i=-1;i<=1;i++) // Parcours en hauteur
  {
    for(int j=-1;j<=1;j++) // Parcours en largeur
      R+=mask[3*(i+1)+(j+1)]*src.imageData[(src.width*(i+x))+(j+y)]; //
          Application du masque
  }
  return R; // On retourne la nouvelle valeur du pixel (x,y)
}
```

C'est encore une fois une application de la formule donnée. Nos 2 boucles `for` correspondent à l'imbrication des 2 sommes.

Sobel

On programme maintenant la fonction qui effectue le calcul complet de l'image en utilisant le masque de Sobel. On rappelle que l'image de sortie correspond à :

$$|G| = \sqrt{G_x^2 + G_y^2} \tag{3.5}$$

Notre fonction se contentera donc d'appliquer le produit de convolution entre les points et chacun des 2 matrices. On applique ensuite la formule ci-dessus et on n'oublie pas de normaliser notre pixel.

C++
```
void SobelComponent::compute_sobel(IplImage const& src, IplImage& dst)
{
  //Masque de SOBEL
  int Gx[9] ={-1,0,1,-2,0,2,-1,0,1};
  int Gy[9] ={1,2,1,0,0,0,-1,-2,-1};
  int x,y;

  for(int i=1;i<src.height-1;i++) // De 1 à la hauteur-1 pour éviter l'effet de
      bord
  {
    for(int j=1;j<src.width-1;j++) // De 1 à la largeur-1 pour éviter l'effet de
        bord
    {
    // On applique le masque de Sobel au pixel (i,j)
    x=convolution(src,i,j,Gx);
    y=convolution(src,i,j,Gy);
    int g=sqrt((double)(x*x)+(y*y));
    g=max(min(g,255),0); // On vérifie que la valeur de sortie est entre 0 et
        255
    dst.imageData[i*dst.width+j]=g; // On écrit dans l'image de sortie
```

```
    }
  }
}
```

Prewitt

De même on programme une fonction pour le masque de Prewitt. Son fonctionnement est strictement identique à la fonction précédente. Seul le masque change.

C++

```
void SobelComponent::compute_per(IplImage const& src, IplImage& dst){
  //Masque de Prewitt
  int Gx[9] ={-1,0,1,-1,0,1,-1,0,1};
  int Gy[9] ={1,1,1,0,0,0,-1,-1,-1};
  int x,y;
  for(int i=1;i<src.height-1;i++){
    for(int j=1;j<src.width-1;j++){
      x=convolution(src,i,j,Gx);
      y=convolution(src,i,j,Gy);
      int g=sqrt((double)(x*x)+(y*y));
      g=max(min(g,255),0);
      dst.imageData[i*dst.width+j]=g;
    }
  }
}
```

3.3.6 Question 5

Après avoir créé notre module RTMaps on effectue nos tests. On obtient les images ci-dessous. On peut voir une main venir de la gauche de l'image, tenant un stylo.

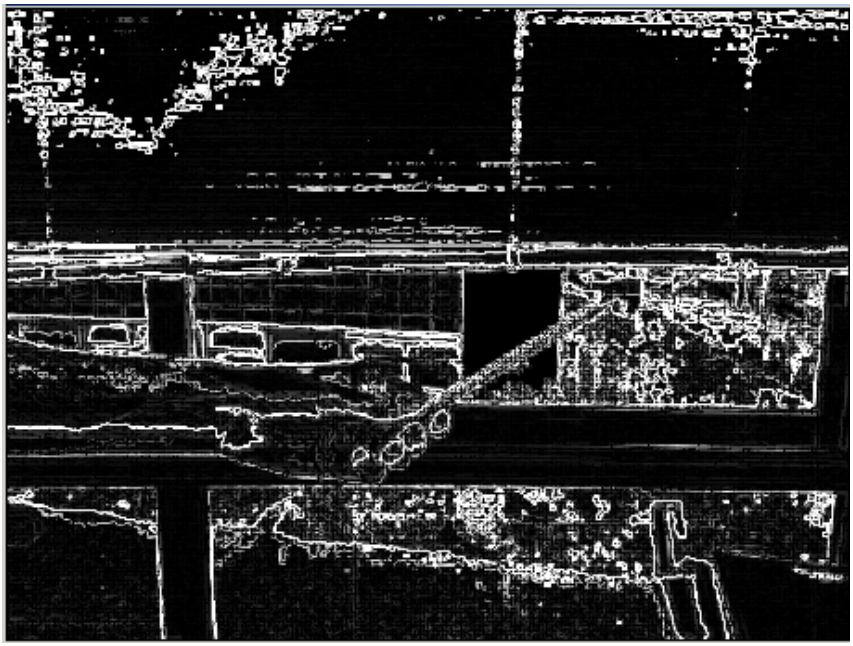

FIGURE 3.1 – Masque de Sobel

FIGURE 3.2 – Masque de Prewitt

L'image de sortie avec le masque de Prewitt semble contenir plus de points clairs. On a donc une certaine impression de « bruit » dans les zones où il y a de nombreuses lignes (par exemple au niveau des visages). L'image avec le masque de Sobel semble être moins « bruitée » et les contours semblent plus nets.

3.3.7 Question 6

Les autres types de masques dérivateurs utilisés sont, entre autres, les masques de Roberts, de Kirsch ou de Robinson.