TD 1 : Recalage et programmation procédurale

Question 1 : Comprendre ce que fait le code ci-dessous.

```
double optimize(double (*f)(double),
                double min          ,
                double max          )
{
double res = min;
double minFunction = f(min);
double c;
for (int i=1;i<11;i++)
  {
  double b = min + i * (max-min)/10;
  c = f(b);
  if (c<minFunction)
    {
    minFunction = c;
    res = b;
    }
  }
return res;
}

double myFunction(double value)
{
return (value-2)*(value-2);
}

int main()
{
double res3 = optimize(myFunction,0.,10.);
}
```

Question 2 : Concevoir une application en C (écrire les prototypes des fonctions (dire ce qu'elles font...)) qui permet de recaler de manière rigide deux images. Pour chaque fonction, dire les fonctions qu'elle utilise et prévoir la manière dont elle va être testée.

Quid du recalage

Recaler deux images peut être vu comme un problème d'optimisation : il s'agit d'estimer les paramètres Θ d'une transformation géométrique T_Θ de manière à maximiser la ressemblance entre une image de référence (I_{ref}) et l'image à recaler (I) déformée par T_Θ ($I \circ T_\Theta$) :

$$\Theta_{opt} = \arg\min_{\Theta} E(I_{ref}, I \circ T_\Theta), \tag{1}$$

où E est une mesure de similarité entre les deux images.

Il s'agit donc d'optimiser une fonction de coût qui dépend des paramètres de la transformation géométrique. La fonction de coût associée au paramètre Θ est calculée comme suit :

— on déforme l'image I suivant la transformation T_Θ pour obtenir l'image déformée I_{def} ;

— la valeur de la fonction de coût est $E(I_{def}, I_{ref})$.

Par la suite, on va définir la transformation géométrique utilisée, la manière dont on déforme une image ainsi que le critère de similarité.

La transformation

Nous utiliserons une transformation rigide. On a ainsi $\Theta = \{t_x, t_y, \theta\}$ avec :

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = T_{t_x,t_y,\theta} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} cos(\theta) & -sin(\theta) \\ sin(\theta) & cos(\theta) \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} t_x \\ t_y \end{pmatrix} \tag{2}$$

Déformation de l'image I suivant la transformation géométrique $T_{t_x,t_y,\theta}$

L'image I déformée par la transformation $T_{t_x,t_y,\theta}$ est notée I_{def}. L'image I_{def} sera calculée sur le même support que l'image I : les deux images auront ainsi même taille. La valeur de l'image I_{def} au point $\mathbf{s}$ est calculée de la manière suivante (il suffit ensuite d'itérer sur tous les pixels de l'image I_{def}) :

— calcul de $\mathbf{s}' = T_{t_x,t_y,\theta}(\mathbf{s})$;

— $I_{def}(\mathbf{s})$ est fixée égale à $I(\mathbf{s}')$

Remarque 1 : $\mathbf{s}'$ n'est pas obligatoirement une coordonnée discrète si bien que l'image I n'y est donc pas forcément définie. Il faut donc utiliser une méthode d'interpolation (cf paragraphe suivant) : elle permet de calculer la valeur de l'image I au point $\mathbf{s}'$

Remarque 2 : il est possible que le point $\mathbf{s}'$ soit en dehors du support de l'image I. Dans ce cas, on peut fixer $I_{def}(\mathbf{s})$ à une valeur arbitraire et fournir une seconde image binaire qui informe des pixels qui ont un sens ou non.

Méthode d'interpolation

Une méthode d'interpolation permet de calculer la valeur d'une image I pour des coordonnées non entières. On propose d'utiliser une méthode d'interpolation bilinéaire. Le code-ci dessous calcule la valeur d'une image I (tableau à 2 dimensions) de taille $H \times L$ au point de coordonnées (x, y).

```
1  if ( (x<0) || (x>H-1) || (y<0) || (y>L-1)
       )
2    ;
3    // On ne peut pas interpoler car le
         point n'est pas dans le support de l
         'image
4    // Que faire?
5
6  else
7    {
8    int i1,i2,j1,j2;
9    double dx,dy,dfy,dfx,dfxy,res;
10
11   i1 = floor(x);
12   j1 = floor(y);
13   if (i1==H-1)
14     i1--;
15
16   if (j1==L-1)
17     j1--;
18   i2 = i1+1 ;
19   j2 = j1+1 ;
20   //Le point (x,y) est dans le carré défini
         par les points discrets (i1,j1),(i1,
       j2),(i2,j2),(i2,j1).
21
22   dx = x - i1;
23   dy = y - j1;
24   dfx  = I[i2][j1]-I[i1][j1];
25   dfy  = I[i1][j2]-I[i1][j1];
26   dfxy = I[i1][j1]+I[i2][j2]-I[i2][j1]-I[i1
         ][j2];
27   res = I[i1][j1] + dfx * dx + dfy * dy +
       dx*dy*dfxy;
28   //la valeur de I au point (x,y) est res
29   }
```

Critère de similarité

Pour le critère de similarité E de l'Eq. 1, on pourra utiliser un critère quadratique :

$$E(I_1, I_2) = \frac{1}{N} \sum_{\mathbf{s}} (I_2(\mathbf{s}) - I_1(\mathbf{s}))^2, \tag{3}$$

où N est le nombre de termes dans la somme. La somme sera réalisée uniquement sur le support commun des deux images (cf image binaire qui informe sur les pixels qui ont un sens ou non).

TD 2 (début) : Introduction d'extensions

Modifier votre conception de manière à prendre en compte les possibles variations futures suivantes :

(a) mesure de similarité entre les deux images : la moyenne de la norme L2 sera codée mais d'autres critères pourront être ajoutés par la suite.

(b) méthode d'interpolation : la méthode d'interpolation bilinéaire sera codée mais d'autres méthodes d'interpolation pourront être ajoutées par la suite.

(c) Est-ce que la fonction « optimize » sera ré-utilisable ou inversement, est-ce que l'on pourra ré-utiliser une fonction « optimze » d'une librairie de mathématique ? Que faire ?

TD 2 (fin) : Algorithme d'optimisation : programmation orientée objet

On considère l'algorithme d'optimisation d'une fonction f de $\mathbb{R}$ dans $\mathbb{R}$ (cf code du TD1) définie en C au TD précédent. Traduire l'algorithme avec une solution « orientée objet » de deux manières différentes. Comment prendre en compte les paramètres muets ?

TD 3 : recalage et programmation orientée objet

Préparer la conception de votre application (recalage d'images) en C++. Pour chaque classe, donner sa responsabilité et son interface. Prévoir ensuite les éventuelles données et méthodes privées. Prévoir également comment vous allez tester chaque classe (certaines classes peuvent être testées indépendamment des autres...).

Une classe responsable de l'optimisation vous sera donnée en TP. Cette dernière a peu d'influence sur votre conception. On vous donnera également une classe « Image » dont la responsabilité sera de manipuler une image (il y a différents constructeurs, des méthodes pour modifier la taille de l'image, pour accéder et modifier la valeur d'un pixel...).

Les images seront fournies au format `pgm ASCII`. Le fichier au format texte commence par « P2 largeur hauteur 255 » et les valeurs des pixels sont ensuite données ligne par ligne. Vous pouvez déclarer une méthode pour lire (traduire le fichier texte en un objet « Image ») et sauver ces images (traduire un objet « Image » en un fichier texte). L'affichage se fera avec un éditeur d'images quelconque.

TPs : recalage en programmation orientée objet

Question 1 : Codes et tests.
Question 2 : Ajouter la possibilité d'utiliser une méthode d'interpolation au plus proche voisin : la valeur de I au point $\mathbf{s}'$ (coordonnée non entière) est égale à la valeur de I au point $\mathbf{s}$ où $\mathbf{s}$ est le point de coordonnée entière le plus proche de $\mathbf{s}'$.
Question 3 : Avant de lancer le recalage, modifier les niveaux de gris d'une image (pas les deux) de la manière suivante :

$$I(\mathbf{s}) = min(2|I(\mathbf{s}) - 128|, 255) \tag{4}$$

Lancer l'algorithme de recalage. Que se passe-t-il ? Pourquoi ? Pour résoudre ce problème, on peut utiliser le critère de l'information mutuelle.

$$E(I_1, I_2) = \sum_{x_0 y_0} p(I_1(.) = x_0, I_2(.) = y_0) log \left(\frac{p(I_1(.) = x_0, I_2(.) = y_0)}{p(I_1(.) = x_0) p(I_2(.) = y_0)} \right) \tag{5}$$

où :

— $p(I_1(.) = x_0, I_2(.) = y_0)$ représente la probabilité que l'on ait $I_1(.) = x_0$ et $I_2(.) = y_1$ pour un même pixel noté « . »,

— $p(I_1(.) = x_0)$ est la probabilité que $I_1(.) = x_0$,

— $p(I_2(.) = y_0)$ représente la probabilité que $I_2(.) = y_0$.

En pratique, pour estimer fidèlement l'information mutuelle, il convient de quantifier le signal. Aussi, on pourra utiliser la démarche suivante.

— Diviser chacune des images par 16 (division entière si bien que les valeurs de chacune des images seront dans l'intervalle entier $[0, 15]$). On note I_1^q et I_2^q les deux images obtenus. Les tableaux permettant de définir $p(I1(.))$ et $p(I12(.))$ seront de taille 16 alors que celui pour définir $p(I1(.), I2(.))$ sera de taille 16 x 16.

— Le calcul de l'information mutuelle entre I_1^q et I_2^q se fait alors de la manière suivante :

$$\sum_{x_0=0}^{15} \sum_{y_0=0}^{15} p(I_1^q(.) = x_0, I_2^q(.) = y_0) log \left(\frac{p(I_1^q(.) = x_0, I_2^q(.) = y_0)}{p(I_1^q(.) = x_0) p(I_2^q(.) = y_0)} \right) \tag{6}$$

Ces probabilités pourront être estimées en se basant sur des simples comptages. De la même manière que pour le critère quadratique, on ne considérera que le support commun des deux images. Par exemple, $p(I_1^q(.) = x_0, I_2^q(.) = y_0)$ est égal au nombre de pixels $\mathbf{s}$ (du support commun) pour lesquels $I_1^q(\mathbf{s}) = x_0$ et $I_2^q(\mathbf{s}) = y_0$ divisé par le nombre de pixels dans le support commun. De même, $p(I_1^q(.) = x_0)$ est égal au nombre de pixels $\mathbf{s}$ du support commun pour lesquels $I_1^q(\mathbf{s}) = x_0$ divisé par le nombre de pixels dans le support commun.

Comme le critère de l'information mutuelle est maximal quand les deux images sont bien recalées, il est nécessaire de minimiser l'opposé du critère.

Question 4 : Passer en argument du main le choix de la méthode d'interpolation et du critère de similarité.