L'illusion de Kanizsa

par
Stéphane Drouin

17 décembre 2000

Problématique

But

Ce projet consiste à construire un circuit de neurones qui peut générer l'illusion de Kanizsa. Cette illusion est illustrée à la figure suivante. Pour les besoins du problème, on suppose que le carré de l'illusion est de taille 10×10 dans une image 100×100.

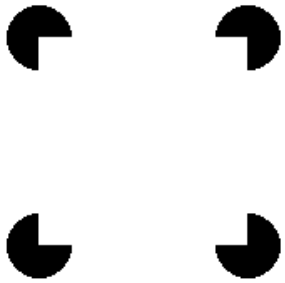

Figure 1: Illusion de Kanizsa

Définition du problème

On va maintenant définir ce qu'on entend par l'objectif de "générer l'illusion de Kanizsa". La première étape consiste à détecter les contours, réels et illusoires. Le résultat de cette étape devrait ressembler à l'image suivante.

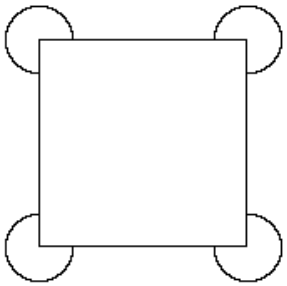

Figure 2: Bordures

La deuxième étape consiste à faire la segmentation de l'image, c'est-à-dire séparer les régions de l'image. On voudra identifier six régions distinctes, soit: l'arrière-plan, le carré, et chacune des quatre formes circulaires situées aux coins du carré.

Solution

Génération et manipulation de l'image

Une image de test est nécessaire pour vérifier la solution développée. Le code de la fonction **gen_img** est utilisé pour générer une illusion de Kanizsa positionnée aléatoirement dans une image. Tout au long du projet, on utilise la convention suivante: un pixel blanc est représenté par 1 et un pixel noir est représenté par 0. On évite également de générer une image trop près des bords de l'image.

Le résultat de cette fonction est une matrice. Deux autres fonctions sont utilisées pour passer de la représentation matrice à la représentation vecteur pour les images, soit **img2vec** et **vec2img**. Finalement, la fonction utilisée pour visualiser les images dans Matlab est **show_img**.

Détection des arêtes

La détection des arêtes se fait en utilisant le modèle de Grossberg, plus précisément avec la couche 1 du réseau de Grossberg. Les différents types d'arêtes sont détectés en choisissant judicieusement le patron de connexion défini par les matrices ${}^{+}\mathbf{W}$ et ${}^{-}\mathbf{W}$.

L'équation de la couche utilisée est la suivante:

$$\epsilon \frac{d\mathbf{n}}{dt} = -\mathbf{n} + ({}^{+}\mathbf{b} - \mathbf{n})[{}^{+}\mathbf{W}]\mathbf{p} - \mathbf{n}[{}^{-}\mathbf{W}]\mathbf{p}$$

où $\mathbf{p}$ est l'entrée du réseau, $\mathbf{n}$ est sa sortie. En régime permanent, l'équation devient:

$$\begin{aligned} 0 &= -\mathbf{n} + ({}^{+}\mathbf{b} - \mathbf{n})[{}^{+}\mathbf{W}]\mathbf{p} - \mathbf{n}[{}^{-}\mathbf{W}]\mathbf{p} \\ \mathbf{n}(\mathbf{1} + [{}^{+}\mathbf{W}]\mathbf{p} + [{}^{-}\mathbf{W}]\mathbf{p}) &= {}^{+}\mathbf{b}[{}^{+}\mathbf{W}]\mathbf{p} \\ \mathbf{n} &= \frac{{}^{+}\mathbf{b}[{}^{+}\mathbf{W}]\mathbf{p}}{\mathbf{1} + [{}^{+}\mathbf{W}]\mathbf{p} + [{}^{-}\mathbf{W}]\mathbf{p}} \end{aligned}$$

La sortie en régime permanent de cette couche est calculée par la fonction **gb** où ${}^{+}\mathbf{b} = \mathbf{1}$.

Dans les fonctions développées, l'opération modulo est réalisée par la fonction **mod**.

Détection des arêtes réelles

Les arêtes réelles sont détectées en utilisant un champ récepteur en forme de chapeau mexicain. La forme retenue est la suivante:

0	-1	0
-1	4	-1
0	-1	0

Figure 3: Champ récepteur pour les arêtes réelles

Soit une image n par n et le pixel i, i=1,...,n^2. Si $i > n$, $i \leq n^2 - n$, $mod(i-1,n) > 0$ et $mod(i,n) > 0$, alors le coefficient non nul de ${}^{+}_{i}\mathbf{W}$ est

$$ {}^{+}_{i}W_i = 4 $$

et les coefficients non nuls de ${}^{-}_{i}\mathbf{W}$ sont

$$ {}^{-}_{i}W_{i-1} = {}^{-}_{i} W_{i+1} = {}^{-}_{i} W_{i-n} = {}^{-}_{i} W_{i+n} = 1 $$

Sinon, ${}^{+}_{i}\mathbf{W} = {}^{-}_{i} \mathbf{W} = \mathbf{0}$ car le pixel i est sur le bord de l'image.
Ce champ récepteur permet de détecter un changement local de l'intensité de l'image, ce qui caractérise une bordure. La sortie de cette couche de Grossberg est rendue binaire à l'aide d'une fonction **hardlim** centrée à 0.5. Le choix de ce seuil, au centre de l'intervalle $[-{}^{-}b, {}^{+}b]$ assure que la sortie du système soit active quand le gradient au pixel d'intérêt est positif.
Ces opérations sont exécutées dans la fonction **ct_vrai**.

Détection des arêtes de l'illusion

Les arêtes de l'illusion sont détectées en utilisant un champ récepteur en forme de rectangle. La forme retenue est la suivante:

-1	-1	-1	-1	-1	-1	-1	-1	-1
0.5	0.5	1	1	3	1	1	0.5	0.5

Figure 4: Champ récepteur pour les arêtes de l'illusion

Soit une image n par n et le pixel i, i=1,...,n^2. Si $i > n$, $mod(i-1,n) \geq 4$ et $n - mod(i-n,n) - 1 \geq 4$, alors les coefficients non nuls de ${}^{+}_{i}\mathbf{W}$ sont

$$ \begin{aligned} {}^{+}_{i}W_i &= 3 \\ {}^{-}_{i}W_{i-1} &= {}^{-}_{i}W_{i+1} = {}^{-}_{i} W_{i-2} = {}^{-}_{i} W_{i+2} = 1 \\ {}^{-}_{i}W_{i-3} &= {}^{-}_{i}W_{i+3} = {}^{-}_{i} W_{i-4} = {}^{-}_{i} W_{i+4} = 0.5 \end{aligned} $$

et les coefficients non nuls de ${}^{-}_{i}\mathbf{W}$ sont

$$ \begin{aligned} {}^{-}_{i}W_{i-n} &= 1 \\ {}^{-}_{i}W_{i-n-1} &= {}^{-}_{i}W_{i-n-2} = {}^{-}_{i} W_{i-n-3} = {}^{-}_{i} W_{i-n-4} = 1 \\ {}^{-}_{i}W_{i+n+1} &= {}^{-}_{i}W_{i+n+2} = {}^{-}_{i} W_{i+n+3} = {}^{-}_{i} W_{i+n+4} = 1 \end{aligned} $$

Sinon, ${}^{+}_{i}\mathbf{W} = {}^{-}_{i} \mathbf{W} = \mathbf{0}$ car le pixel i est sur le bord de l'image.
Ce champ récepteur permet de détecter un pixel actif dont les voisins horizontaux immédiats sont aussi actifs, mais dont les voisins sur la ligne supérieure forment une bande inactive. La sortie de cette couche de Grossberg est rendue binaire à l'aide d'une fonction **hardlim**

centrée à 0.61. Le choix de ce seuil a été fait après l'observation des sorties pour le champ récepteur proposé. Il correspond au choix du gradient maximal dans ce cas.

Puisque qu'il y a quatre arêtes dans l'illusion, trois autres champs récepteurs correspondant à des rotations de 90, 180 et 270 degrés du récepteur présenté sont aussi utilisés.

Ces opérations sont exécutées dans la fonction **ct_faux**.

Toutes les bordures sont par la suite combinées dans une image binaire.

```
> C = max(ct_vrai(M),ct_faux(M));
```

L'étape suivante consiste à faire la segmentation de l'image, c'est-à-dire séparer les régions de l'image.

Segmentation de l'image

On utilise un algorithme séquentiel pour faire la segmentation de l'image. On prend en entrée l'image binaire contenant les bordures et la sortie est une image dont chaque pixel porte une étiquette correspondant à la région de l'image à laquelle il appartient.

Algorithme 1 Segmentation de l'image.

1. Pour chaque pixel (i, j) de l'image n'étant pas sur une bordure, identifier la région à laquelle il appartient par rapport à ses voisins immédiats.

   1.1. Si le pixel à la position $(i-1, j)$ possède une étiquette, la copier pour le pixel courant. De plus, si le pixel à la position $(i, j-1)$ possède également une étiquette, et que ces deux étiquettes ne sont pas identiques, noter la correspondance entre les deux régions.

   Sinon, si le pixel à la position $(i, j-1)$ possède une étiquette, la copier pour le pixel courant.

   Sinon, créer une nouvelle étiquette et l'assigner au pixel courant.

2. Faire un deuxième passage sur les pixels et assigner une seule étiquette dans le cas de correspondances.

Cet algorithme est implanté dans la fonction **segment**.

Résultats

Les fonctions Matlab développées ont été vérifiées avec une image de 50×50 pixels, pour des considérations de mémoire disponible. Les résultats avec une image de 100×100 pixels seraient identiques.

Figure 5: Image contenant l'illusion de Kanizsa

La première étape consiste à détecter les bordures de l'illusion. Cette opération est réalisée parfaitement comme le montre l'image suivante où les bordures sont en blanc et le reste de l'image est en noir.

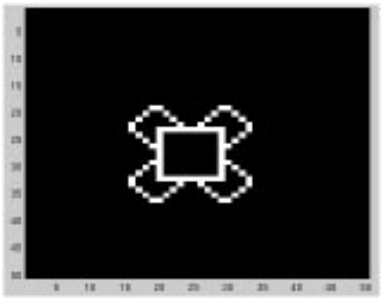

Figure 6: Bordures de l'illusion de Kanizsa

La deuxième étape consiste à faire la segmentation de l'image, c'est-à-dire séparer les régions de l'image. Sur l'image suivante, six niveaux de gris différents identifient six régions distinctes, soit: l'arrière-plan, le carré, et chacune des quatre formes circulaires situées aux coins du carré.

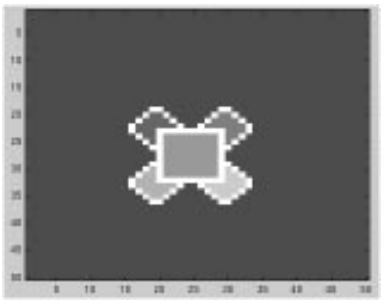

Figure 7: Segmentation de l'image en 6 régions

Conclusion

Comme les résultats précédents le montrent, on est en mesure de générer l'illusion de Kanizsa. Plus précisément, on en détecte les bordures avec un réseau de neurones contenant des cellules de Grossberg, puis on en fait la segmentation en utilisant un algorithme séquentiel.

Annexe

Fonctions pour la manipulation des images

```
function M = gen_img(size)

% function M = gen_img(size)
% Genere une image size*size contenant une illusion de Kanizsa 10*10
% SD

  % Generation de l'illusion de Kanizsa
  cercle = [1 1 0 0 1 1;
            1 0 0 0 0 1;
            0 0 0 0 0 0;
            0 0 0 0 0 0;
            1 0 0 0 0 1;
            1 1 0 0 1 1 ];

  kanizsa = ones(16);

  kanizsa(1:6,1:6) = cercle;
  kanizsa(11:16,1:6) = cercle;
  kanizsa(1:6,11:16) = cercle;
  kanizsa(11:16,11:16) = cercle;
  kanizsa(4:13,4:13) = ones(10);

  % Insere l'illusion de Kanizsa dans une image
  if (nargin < 1), size = 100; end
  M = ones(size);

  % Laisse 5 pixels de chaque cote de l'illusion dans l'image
  pos = floor((size-16-10) * rand(2,1) + 1) + 5;
  M(pos(1):pos(1)+15,pos(2):pos(2)+15) = kanizsa;
```

```
function v = img2vec(M)

% function v = img2vec(M)
% Transforme une image n * n en vecteur colonne de taille n^2 * 1
% SD

  [n,m] = size(M);
```

```
  v = [];
  for i=1:n, v = [v;M(i,:)']; end
```

```
function M = vec2img(v)

% function M = vec2img(v)
% Transforme un vecteur colonne de taille n^2 * 1 en image n * n
% SD

  [m,n] = size(v);
  if (n ~= 1), error('Mauvais vecteur d'entree'); end
  n = floor(sqrt(m));
  if (n*n ~= m), error('Dimension du vecteur d'entree mauvaise'); end

  M = [];
  for i=0:n-1, M = [M;v(i*n+1:(i+1)*n)']; end
```

```
function show_img(M)

% function show_img(M)
% Affiche l'image monochrome M
% SD

  % Pour l'affichage en niveaux de gris: 0=noir, 1=blanc, 2...8 = gris
  niveaux = linspace(0.3,0.9,7);
  colormap([0 0 0;1 1 1;niveaux' niveaux' niveaux']);
  image(M+1);
```

Fonctions pour la détection des bordures

```
function n = gb(p,Wp,Wn)

% function n = gb(p,Wp,Wn)
% Sortie de la couche 1 du reseau de Grossberg en regime permanent
% SD

  n = (Wp*p)  ./ (1 + (Wp*p) + (Wn*p));
```

```
function M = mod(X,Y)

% function M = mod(X,Y)
% Modulo de X divise par Y
% SD

  M = X;
  if (Y ~= 0), M = X - Y.*floor(X./Y); end
```

```
function CV = ct_vrai(M)

% function CV = ct_vrai(M)
% Detecte les vraies bordures dans une image
% SD

  p = img2vec(M);
  [m,n] = size(p);
  n = floor(sqrt(m));

  % Matrices pour les vraies bordures!
  % - 0 -
  % 0(1)0
  % - 0 -
    Wp = zeros(m);
    Wn = zeros(m);
    for i=1:m,
      % Verifie que le kernel sera symetrique sur les bords
      if (i-n > 0 & i+n <= m & mod(i-1,n) > 0 & mod(i,n) > 0),
        Wp(i,i) = 4;
        Wn(i,i-1) = 1;
        Wn(i,i+1) = 1;
        Wn(i,i-n) = 1;
        Wn(i,i+n) = 1;
      end
    end

  CV = gb(p,Wp,Wn);
  CV = hardlim(CV-0.5);
  CV = vec2img(CV);
```

```
function CF = ct_faux(M)

% function CF = ct_faux(M)
% Detecte les fausses bordures dans une image
% SD

  p = img2vec(M);
  [m,n] = size(p);
  n = floor(sqrt(m));

  % Matrices pour les fausses bordures!
  LARGEUR = (9-1)/2;

  % Matrices pour les fausses bordures!
  % 0 0 0 0 0 0 0
  % 1 1 1(1)1 1 1
    Wp = zeros(m);
    Wn = zeros(m);
    for i=1:m,
      % Verifie que le kernel sera symetrique sur les bords
      if (i-n> 0 & mod(i-1,n)>= LARGEUR & (n-mod(i-1,n)-1) >= LARGEUR)
        k = i-n;
        Wp(i,i) = 1;
        Wn(i,k) = 1;
        for j=1:LARGEUR,
          Wp(i,i-j) = 1;
          Wp(i,i+j) = 1;
          if (j> LARGEUR/2),
            Wp(i,i-j) = 1/2;
            Wp(i,i+j) = 1/2;
            Wp(i,i) = Wp(i,i) + 1;
          end
          Wn(i,k-j) = 1;
          Wn(i,k+j) = 1;
        end
      end
    end
  CF1 = gb(p,Wp,Wn);
  CF1 = hardlim(CF1-0.61);
  CF1 = vec2img(CF1);

  % Matrices pour les fausses bordures!
  % 1 1 1(1)1 1 1
```

```
% 0 0 0 0 0 0 0
  Wp = zeros(m);
  Wn = zeros(m);
  for i=1:m,
    % Verifie que le kernel sera symetrique sur les bords
    if (i+n<=m & mod(i-1,n)>=LARGEUR & (n-mod(i-1,n)-1)>=LARGEUR)
      k = i+n;
      Wp(i,i) = 1;
      Wn(i,k) = 1;
      for j=1:LARGEUR,
        if (j> LARGEUR/2),
          Wp(i,i-j) = 1/2;
          Wp(i,i+j) = 1/2;
          Wp(i,i) = Wp(i,i) + 1;
        else
          Wp(i,i-j) = 1;
          Wp(i,i+j) = 1;
        end
        Wn(i,k-j) = 1;
        Wn(i,k+j) = 1;
      end
    end
  end
CF2 = gb(p,Wp,Wn);
CF2 = hardlim(CF2-0.61);
CF2 = vec2img(CF2);

% Matrices pour les fausses bordures!
% 0 1
% 0 1
% 0 1
% 0(1)
% 0 1
% 0 1
% 0 1
  Wp = zeros(m);
  Wn = zeros(m);
  for i=1:m,
    % Verifie que le kernel sera symetrique sur les bords
    if (mod(i-1,n)>0 & floor((i-1)/n)>=LARGEUR & floor((m-i)/n)>=LARGEUR)
      k = i-1;
      Wp(i,i) = 1;
      Wn(i,k) = 1;
```

```
      for j=n:n:n*LARGEUR,
        if (j/n> LARGEUR/2),
          Wp(i,i-j) = 1/2;
          Wp(i,i+j) = 1/2;
          Wp(i,i) = Wp(i,i) + 1;
        else
          Wp(i,i-j) = 1;
          Wp(i,i+j) = 1;
        end
        Wn(i,k-j) = 1;
        Wn(i,k+j) = 1;
      end
    end
  end
CF3 = gb(p,Wp,Wn);
CF3 = hardlim(CF3-0.61);
CF3 = vec2img(CF3);

% Matrices pour les fausses bordures!
% 1 0
% 1 0
% 1 0
%(1)0
% 1 0
% 1 0
% 1 0
  Wp = zeros(m);
  Wn = zeros(m);
  for i=1:m,
    % Verifie que le kernel sera symetrique sur les bords
    if (mod(i,n)>0 & floor((i-1)/n)>=LARGEUR & floor((m-i)/n)>=LARGEUR)
      k = i+1;
      Wp(i,i) = 1;
      Wn(i,k) = 1;
      for j=n:n:n*LARGEUR,
        if (j/n> LARGEUR/2),
          Wp(i,i-j) = 1/2;
          Wp(i,i+j) = 1/2;
          Wp(i,i) = Wp(i,i) + 1;
        else
          Wp(i,i-j) = 1;
          Wp(i,i+j) = 1;
        end
```

```
          Wn(i,k-j) = 1;
          Wn(i,k+j) = 1;
        end
      end
    end
  CF4 = gb(p,Wp,Wn);
  CF4 = hardlim(CF4-0.61);
  CF4 = vec2img(CF4);

  CF = max(max(max(CF1,CF2),CF3),CF4);
```

Fonction pour la segmentation

```
function M = segment(M)

% function M = segment(M)
% Segmente l'image binaire M en regions
% SD

  % 0: region, 1: bordure
  BORDURE = 1;
  newlabel = 2;

  % Equivalence entre les etiquettes
  EQUI = [];

  [m,n] = size(M);

% ** Premier passage **

  % Traite le premier pixel
  if (M(1,1)~=BORDURE), M(1,1) = newlabel; newlabel = newlabel+1; end

  % Traite la premiere ligne
  for j=2:n,
    if (M(1,j)~=BORDURE),
      if (M(1,j-1)~=BORDURE), M(1,j) = M(1,j-1);
      else M(1,j) = newlabel; newlabel = newlabel+1; end
    end
  end

  % Traite la premiere colonne
```

```
for i=2:m,
  if (M(i,1)~=BORDURE),
    if (M(i-1,1)~=BORDURE), M(i,1) = M(i-1,1);
    else M(i,1) = newlabel; newlabel = newlabel+1; end
  end
end

% Traite les autres pixels
for i=2:m, for j=2:n,
  if (M(i,j)~=BORDURE),
    if (M(i-1,j)~=BORDURE),
      M(i,j) = M(i-1,j);
      if (M(i,j-1)~=BORDURE & M(i,j-1) ~= M(i-1,j)),
        EQUI = [EQUI;M(i,j-1) M(i-1,j)];
      end
    elseif (M(i,j-1)~=BORDURE), M(i,j) = M(i,j-1);
    else
      M(i,j) = newlabel;
      newlabel = newlabel+1;
    end
  end
end, end

% Correction des equivalences
Correction = linspace(2,newlabel-1,newlabel-2)';
[a,dum] = size(EQUI);

for i=1:a,
  Correction(EQUI(i,1)-1) = min(Correction(EQUI(i,1)-1),min(EQUI(i,:)));
  Correction(EQUI(i,2)-1) = Correction(EQUI(i,1)-1);
end

% Correction des indirections dans les equivalences
for i=2:newlabel-1, Correction(i-1) = Correction(Correction(i-1)-1); end

% Renomme les etiquettes a partir de 2 avec un pas de 1
R = Correction;
for i=2:newlabel-1,if (R(i-1)~=i) R(i-1) = 0; end, end
D = find(R);
[a,dum] = size(D);
renlabel = 2;
for i=1:a,
  for j=2:newlabel-1,
```

```
      if (Correction(j-1)==R(D(i))), Correction(j-1) = renlabel; end
    end
    renlabel = renlabel+1;
  end

% ** Deuxieme passage **

  for i=1:m, for j=1:n,
    if (M(i,j)~=BORDURE), M(i,j) = Correction(M(i,j)-1); end
  end, end
```
