Optimisation de la reconnaissance automatique de cartes à jouer

Marie Picard

2023-2024

1. Introduction

On s'intéresse dans ce TIPE au développement d'une méthode de reconnaissance automatique de cartes à jouer robuste à des perspectives et rotations importantes, et sans réseaux de neurones. Plus particulièrement, on développera l'optimisation de la reconnaissance de la détection des bords de la carte, grâce à une approche proposée par Fernandes et Oliveira [2].

2. Algorithme naïf et premières complexités

2.1 Détail de l'algorithme naïf

Afin d'identifier correctement la carte, on procède en 3 étapes majeures. Tout d'abord, on détecte les bords et coins de la carte, puis on la redresse, et enfin on identifie sa valeur et son enseigne.

2.1.1 Détection des bords et des coins de la carte

Un filtre de flou gaussien ($\sigma = 1.3$) suivi d'un filtre de Canny appliqué sur le gradient de l'image [1] permettent d'obtenir une image binaire correspondant aux pixels de contour de la carte. De là, on produit une liste des pixels de contour puis on applique la transformation de Hough sur le gradient (GHT) [1] afin de déterminer les droites auxquelles correspondent ces points de contour. Les droites sont repérées en coordonnées polaires (ρ, θ) (cf figure 1).

1. Filtre à hystérésis (dont le seuil dépend de l'état) détaillé si besoin en annexe C.

Algorithme 1 transformation de Hough

Entrée : (liste de pixels de bord B, pas de discrétisation δ)

$Votes \leftarrow 0$ ▷ initialisation de la matrice de vote
pour $(x, y) \in B$ **faire**
  pour $\theta \in [0°, 180°[$ avec un pas de δ **faire**
    $\rho \leftarrow x\cos(\theta) + y\sin(\theta)$
    $Votes(\rho, \theta) \leftarrow Votes(\rho, \theta) + 1$
  fin pour
fin pour
renvoyer maxima locaux($Votes$)

Propriété 1

La transformation de Hough ainsi décrite a une complexité temporelle $\mathcal{O}\left(\frac{|B| + R}{\delta}\right)$, où R est le rayon de l'image.

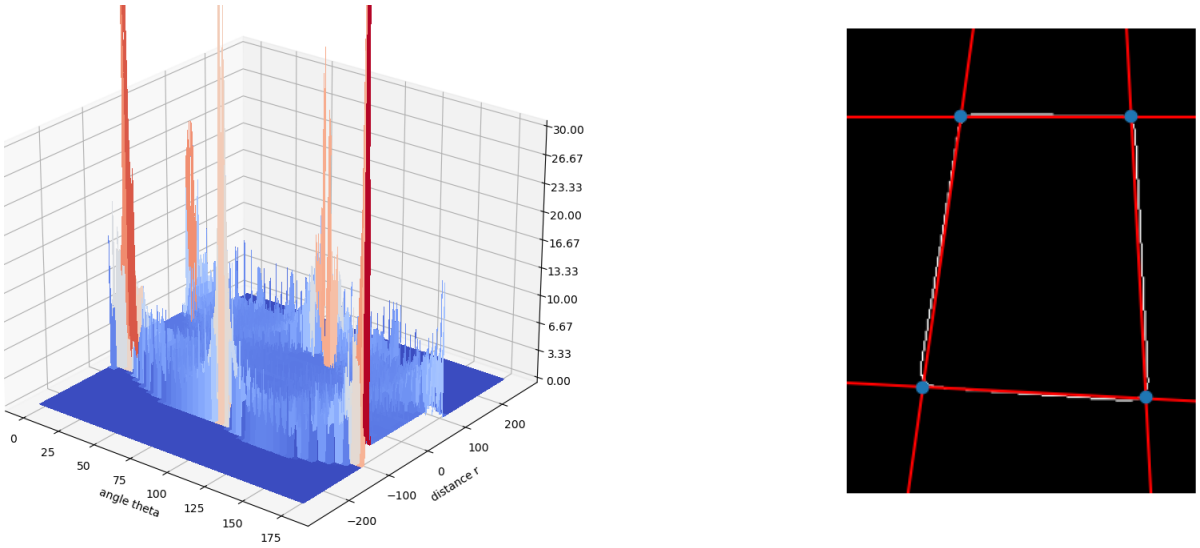

FIGURE 1 – Espace de Hough de l'as de carreau, et les droites correspondantes

2.1.2 Redressement de la carte

On redresse la carte en calculant ses coordonnées en 3 dimensions : on définit un système de coordonnées fictif (cf figure 2) constitué d'un centre optique, d'un plan d'image $z = 100$ et on détermine (à homothétie près) les coordonnées de la carte en 3 dimensions dans le système ainsi défini [2]. Il est cependant impossible de savoir quel bord correspond au bord court : il faut donc éventuellement tester les deux orientations possibles lors de la reconstruction de la carte. Pour cela, on teste simplement la présence dans le bon sens du motif dans un des coins de l'image.

2. On préfère cette méthode à celle d'un calcul de réciproque d'homographie, car elle ne nécessite pas de savoir à l'avance quel coin sur l'image est associé à quel coin réel. Les performances en terme de temps sont parfaitement équivalentes. Le maillage a également été évité, car il déforme les images dans des cas de grandes perspectives (cf annexe A).

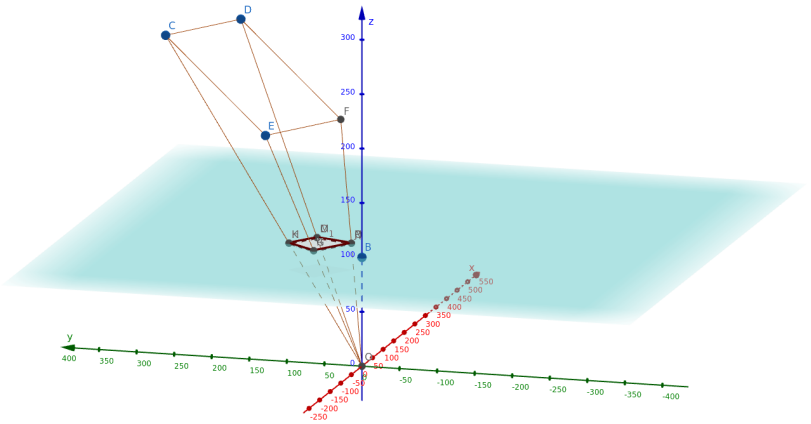

FIGURE 2 – Système de coordonnées défini.

Un point A de l'espace est repéré par (x_A, y_A, z_A) et son projeté sur le plan $z = 100$ est donné par $(X_A, Y_A, 100)$. On obtient alors :

$$\forall A \in \{C, D, E, F\}, X_A = 100\frac{x_A}{z_A} \tag{1}$$

$$\forall A \in \{C, D, E, F\}, Y_A = 100\frac{y_A}{z_A} \tag{2}$$

$$\overrightarrow{CD} = \overrightarrow{EF} \tag{3}$$

i.e. un système de 11 équations à 12 inconnues (les 3 coordonnées des coins B, C, D, E).

Propriété 2

Le redressement est correct dès que les 4 points ne sont pas alignés et s'effectue en $\mathcal{O}(L \cdot l)$, où L et l sont les dimensions de l'image à reconstituer.

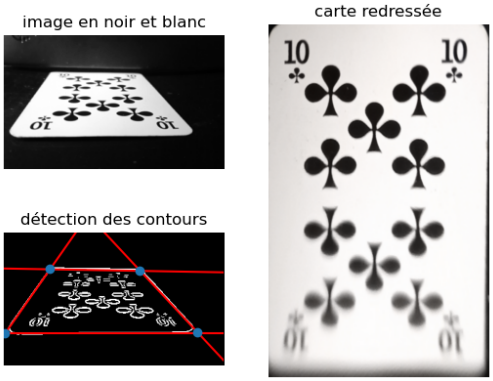

FIGURE 3 – Le système de redressement reste insensible aux grandes perspectives.

2.1.3 Identification de la carte grâce à des arbres de décision

Une fois reconstitué un des coins de la carte, on peut retrouver la position de la valeur et l'enseigne grâce à un simple algorithme $A*$. Des arbres de décisions s'appuyant sur des

critères simples à calculer (nombre de composantes connexes de l'arrière-plan, défaut de convexité, symétries, etc) permettent d'identifier le motif.

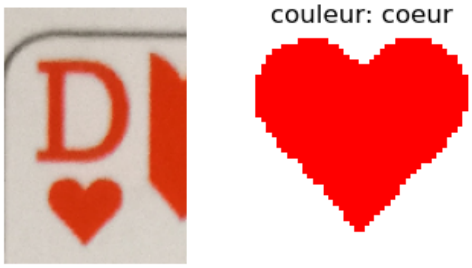

FIGURE 4 – Exemple d'une enseigne correctement reconnue.

2.2 Premiers résultats pratiques

Sur le jeu de 52 cartes test, l'algorithme identifie correctement 51 d'entre elles. En revanche, le temps d'exécution est assez important. Le filtre de Canny et la transformation de Hough sont les plus chronophages (cf figure 5), suivis du redressement.

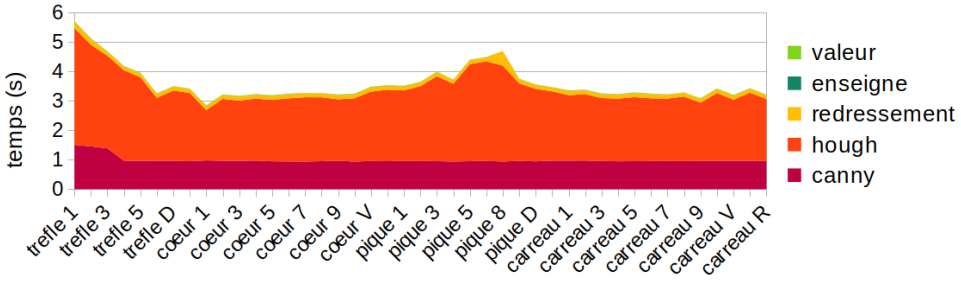

FIGURE 5 – Temps de calcul avec l'algorithme naïf[3].

Il est difficile d'optimiser efficacement le filtre de Canny, aussi on cherche plus particulièrement à améliorer les performances de la transformation de Hough.

3. Optimisation

3.1 Amincissement des bords

L'épaisseur des bords obtenus par le filtre de Canny a deux inconvénients : elle rend imprécise la position de la droite correspondante et augmente le temps de calcul de la transformation de Hough. L'amincissement par suppression des non-maxima (cf figure 6) proposé dans un filtre classique de Canny est insatisfaisant, car il bruite les données. Les coins et zones recourbées sont en outre plus épais que les zones droites ; ainsi, bien qu'inutiles, ils gaspillent beaucoup de temps de calcul pour la transformation de Hough.

3. Les images ont une résolution 4160×3120 pixels, le code a été exécuté en Python.

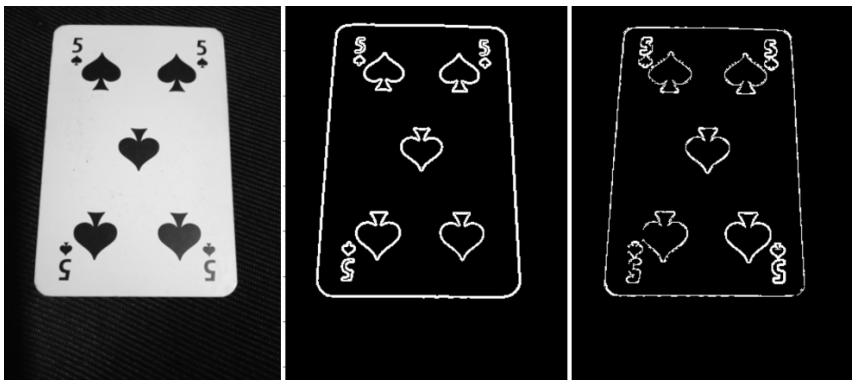

FIGURE 6 – Résultats du filtre de Canny sans puis avec suppression des non-maxima.

On préfère donc l'algorithme d'amincissement de Zhang et Suen [5] (cf figure 7) : supprimer tant que possible les pixels de bords du Nord-Ouest, puis du Sud-Est, non nécessaires au squelette du bord, afin de conserver la connexité du motif, sa forme globale, et réduire son épaisseur à 1 pixel [4].

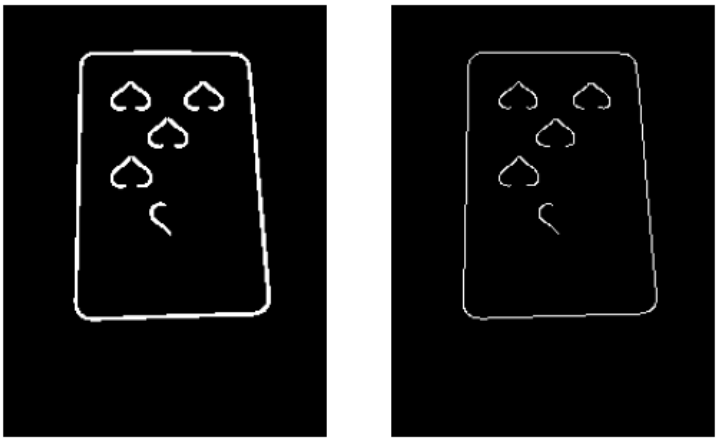

FIGURE 7 – Résultats de l'amincissement.

Algorithme 2 Passe Nord-Ouest

Entrée : image binaire I des points de contour

1: $M \leftarrow 0$ ▷ `matrice des points supprimés`
2: $Cpt \leftarrow 0$ ▷ `nombre de modifications effectuées`
3: **pour** $(x, y) \in I$ **faire**
4: **si** $(x, y) \in$ {contour Nord-Ouest, ou coin Sud-Est} **alors**
5: $M(x, y) \leftarrow 1$
6: $Cpt \leftarrow Cpt + 1$
7: **fin si**
8: **fin pour**
9: $I \leftarrow I - M$
10: **si** $Cpt = 0$ **alors renvoyer** I
11: **sinon renvoyer** $PasseSE(I)$
12: **fin si**

4. Les conditions de suppression d'un pixel et des éléments de preuve de la correction de l'algorithme sont présents si besoin en annexe B.

De même pour la passe Sud-Est.

Propriété 3

L'algorithme d'amincissement a une complexité $\mathcal{O}(l \cdot h \cdot e)$, où e est l'épaisseur d'un trait de contour, l et h les dimensions de l'image binaire.

3.2 Détection de chaînes de pixels approximativement colinéaires

Afin d'optimiser encore le calcul de la transformation de Hough, on cherche à détecter des chaînes de pixels approximativement colinéaires (cf figure 8) afin de pouvoir voter uniquement aux environs des droites qui approximent au mieux ces chaînes. À cet effet, on lie puis subdivise des chaînes de pixels. Le lien est un simple parcours en profondeur depuis un pixel de bord. Sa complexité est $\mathcal{O}(k)$, avec k la longueur de la chaîne ainsi liée.

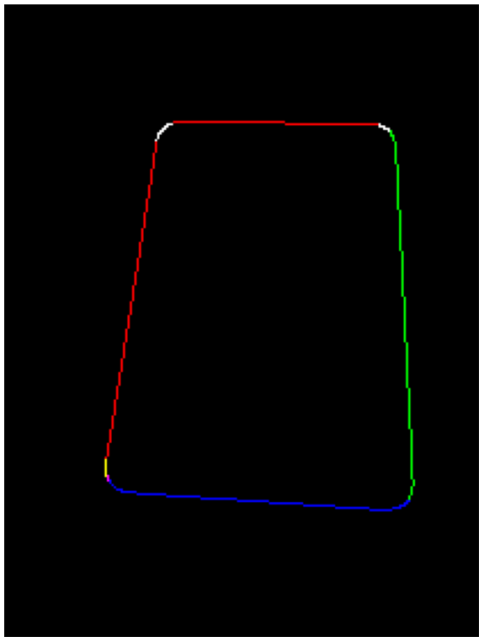

FIGURE 8 – Subdivision en chaîne de pixels approximativement colinéaires. [5]

Une fois une chaîne liée, on la subdivise récursivement au pixel le plus éloigné [4], en utilisant comme critère de colinéarité le rapport $r = \frac{d}{l}$ où l est la distance entre les deux points extrémaux de la chaîne et d la distance maximale d'un point de la chaîne à la droite passant par les deux points extrémaux. Si une chaîne est plus colinéaire que chacun de ses sous-segments, alors elle est conservée, sinon, ce sont ses sous-segments qui le sont.

Propriété 4

La subdivision s'effectue dans le pire des cas en $\mathcal{O}(k^2)$, où k est la longueur de la chaîne et en $\mathcal{O}(k)$ pour un quadrilatère.

5. Les zones blanches (coins) ont été éliminées par l'algorithme, car pas assez droites.

3.3 Ellipses gaussiennes et transformation de Hough optimisée

On cherche à approcher au mieux chaque cluster de pixels colinéaires par une ellipse [2]. Pour cela, on calcule $\overline{p}$ le barycentre du cluster et les valeurs et vecteurs propres de la matrice symétrique $A = p \cdot p^\top$, où chaque colonne de $p \in \mathcal{M}_{2,n}(\mathbb{R})$ correspond aux coordonnées d'un pixel du cluster centré. On se place alors dans la base orthonormée $(\vec{u}, \vec{v})$ de vecteurs propres de A, $\vec{u}$ étant associé à la plus grande valeur propre. La droite qui approxime au mieux le cluster est celle dirigée par $\vec{u}$ et passant par $\overline{p}$.
On cherche alors à voter selon une gaussienne bi-dimensionnelle (cf figure 9) définie par :

$$G_k(\rho, \theta) = \frac{1}{2\pi\sigma_\rho\sigma_\theta\sqrt{1-r^2}} \exp\left(\frac{-z}{1-r^2}\right)$$

où

$$z = \frac{(\rho-\rho_k)^2}{\sigma_\rho^2} + \frac{(\theta-\theta_k)^2}{\sigma_\theta^2} - 2r\frac{(\rho-\rho_k)(\theta-\theta_k)}{\sigma_\rho\sigma_\theta} \qquad r = \frac{\sigma_{\rho\theta}}{\sigma_\rho\sigma_\theta}$$

σ_ρ, σ_θ et $\sigma_{\rho\theta}$ étant respectivement la variance associée à ρ_k, θ_k, et la covariance de ρ_k et θ_k associées à la droite approchant au mieux l'ellipse de points du cluster k, qu'on peut approcher par régression linéaire, et propagation d'incertitude [6]. En pratique, on ne vote qu'aux environs de ρ_k et θ_k, la valeur de $G_k(\rho, \theta)$ devenant rapidement négligeable.

Pour limiter le temps de calcul, on ne vote que pour les noyaux gaussiens ayant une hauteur suffisamment importante [7]. On repère ensuite les 4 premiers maxima locaux sur la carte lissée.

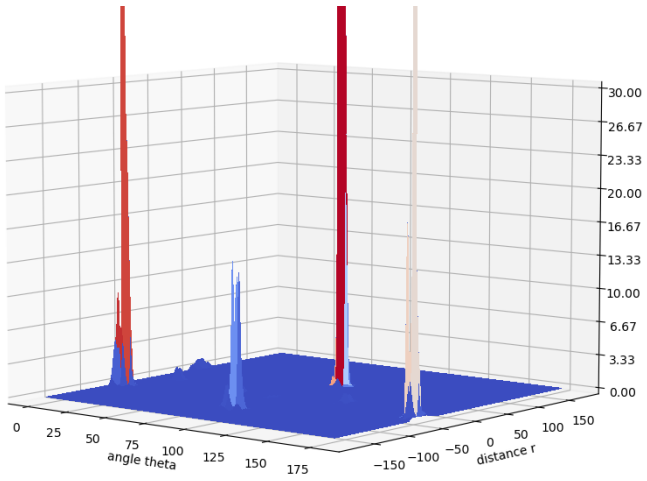

FIGURE 9 – Espace de Hough après les votes pour l'as de carreau, avec la nouvelle méthode

Propriété 5

Le processus de vote pour un cluster s'effectue en $T_{vote}(k) = \mathcal{O}\left(\frac{a_k b_k}{\delta}\right)$, avec a_k, b_k les axes de l'ellipse du cluster $\mathcal{C}_k$ – et au total en $\mathcal{O}\left(\sum_{k=1}^{c} T_{vote}(k) + \frac{R}{\delta}\right)$, avec c le nombre de clusters.

6. Des détails sur l'origine des formules et l'implémentation sont si besoin donnés en annexe D.
7. Supérieure à 0.0005 fois la hauteur maximale d'un des cluster.

3.4 Résultats pratiques

L'algorithme proposé a un taux de réussite satisfaisant dans le cas d'une prise de vue droite, et acceptable dans une prise de vue en perspective (cf figures 11 et 12).

FIGURE 10 – Exemple de carte en prise droite et perspective.

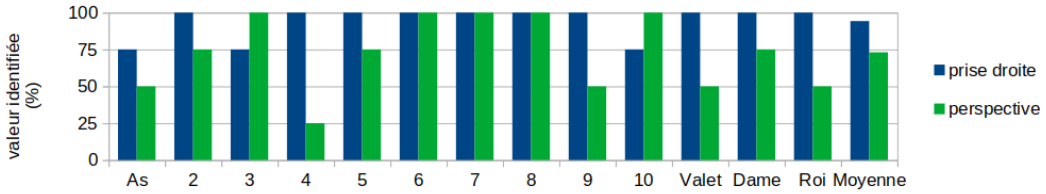

FIGURE 11 – Taux de réussite sur la valeur, relevé sur un jeu de 52 cartes

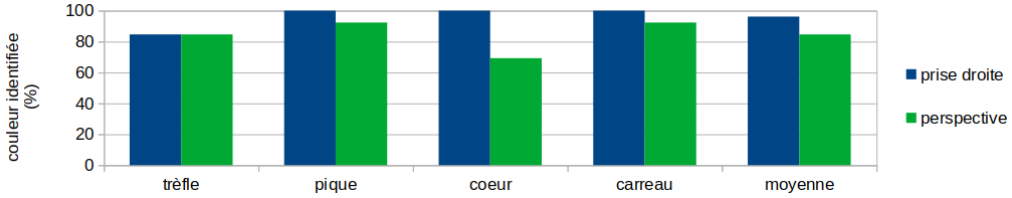

FIGURE 12 – Taux de réussite sur l'enseigne, de même

L'optimisation a permis de diviser par 1000 environ le temps de calcul pour la transformation de Hough, et produit des espaces de Hough moins bruités (ce qui permet de réduire la taille de l'image sur laquelle on travaille, et donc d'accélérer également le filtre de Canny). Le temps de calcul total a environ été divisé par 8.

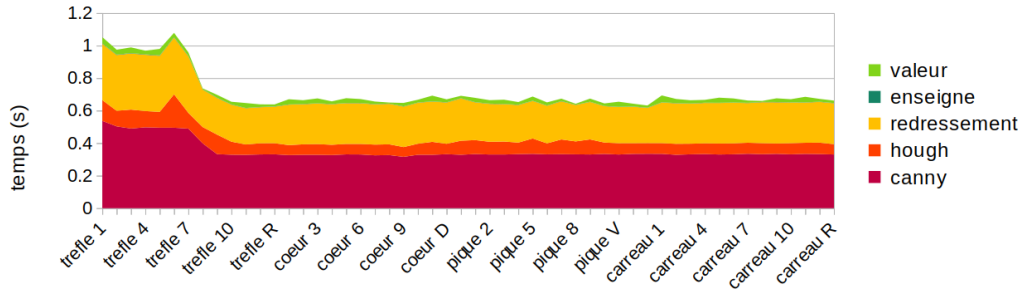

FIGURE 13 – Temps de calcul pour chacune des 52 cartes testées

Références

[1] R. O. Duda & P. E. Hart. « Use of the Hough transformation to detect lines and curves in pictures ». In : *Communications of the ACM* 15 (1972), p. 11-15.

[2] L. A. F. Fernandes & M. M. Oliveira. « Real-time line detection through an improved Hough transform voting scheme ». In : *Pattern Recognition* 41 (2008), p. 299-314.

[3] P. Heckbert. « Projective mappings for image warping ». In : *Fundamentals of Texture mapping and image warping (master's thesis)* (1989), p. 17-21.

[4] D. G. Lowe. « Three-dimensional object recognition from single two-dimensional images ». In : *Artif. Intell.* 31 (1987), p. 355-396.

[5] T. Y. Zhang & C. Y. Suen. « A fast parallel algorithm for thinning digital patterns ». In : *Communications of the ACM* 27 (1984), p. 236-239.

A. Maillage

Le maillage a tendance à déformer une image en perspective, car il préserve les proportions de l'image projetée. On lui préfère donc la méthode avec les axes fictifs (cf 2.1.2).

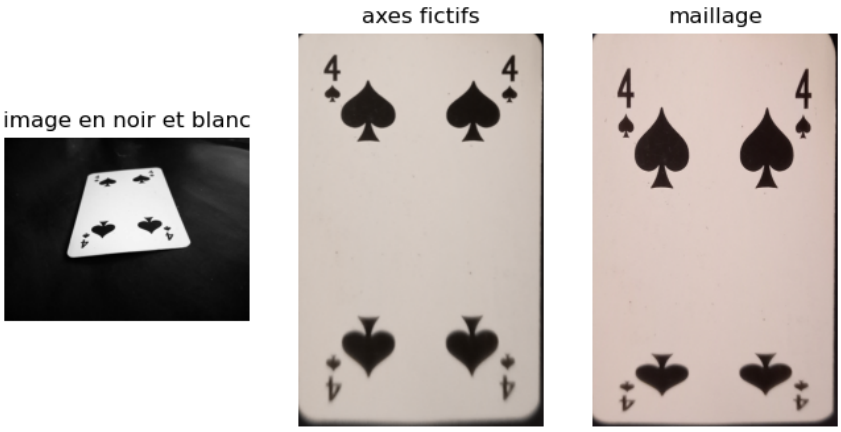

Figure 14 – 4 de piques redressé grâce à une méthode de maillage et la méthode décrite

B. Algorithme d'amincissement

Un pixel de contour est considéré comme appartenant au contour Nord-Ouest si :

1. Parmi ses 8 voisins, entre 2 et 6 sont des points de contour
2. En parcourant ses 8 voisins dans un sens, on rencontre exactement une séquence 01
3. Parmi ses 3 voisins Nord, Est et Ouest, au moins un n'appartient pas au contour
4. Parmi ses 3 voisins Nord, Ouest et Sud, au moins un n'appartient pas au contour

De même pour le Sud-Est, en modifiant dans la condition 3 par Nord, Sud et Est, et la condition 4 par Est, Sud et Ouest.

La condition 1 assure que les pixels de fin de chaîne, ou au centre du motif, ne sont pas supprimés. La condition 2 garantit la connexité (cf figure 15). Les conditions 3 et 4 assurent que les points supprimés lors de la passe Nord-Ouest (resp. Sud-Est) font partie du bord Nord-Ouest du motif ou qu'il s'agit du coin Sud-Est (resp. du bord Sud-Est ou du coin Nord-Ouest).

FIGURE 15 – Conservation des points extrêmaux et de la connexité [8]

C. Filtre de Canny

Un pixel peut avoir deux états. Il est noté $Vrai$ si un de ses voisins est un point de contour. Il est noté $Faux$ sinon.

Algorithme 3 Filtre de Canny

Entrée : carte de gradient $Grad$

$Grad \leftarrow$ Filtre gaussien($Grad$)
$img \leftarrow$ tableau vide de dimensions de $Grad$
$seuil_{max} \leftarrow 0.9 \cdot \max(Grad)$
$seuil_{min} \leftarrow 0.7 \cdot seuil_{max}$
$pile \leftarrow \{((0,0), Faux)\}$ ▷ la pile contient le pixel et son état
tant que $pile \neq \varnothing$ **faire**
    $(p, etat) \leftarrow$ Dépiler($pile$)
    si $img[p] \neq 1$ **alors**
        si $Grad[p] \geq seuil_{max}$ **ou** ($Grad[p] \geq seuil_{min}$ **et** $etat$) **alors**
            $img[p] \leftarrow 1$ ▷ le pixel fait partie du contour
            pour v voisin de p **faire**
                Empiler($(v, Vrai)$, $pile$)
            fin pour
        sinon
            si $img[p] \neq 0$ **alors** ▷ pixel pas encore traité
                $img[p] \leftarrow 0$
                pour v voisin de p **faire**
                    Empiler($(v, Faux)$, $pile$)
                fin pour
            fin si
        fin si
    fin si
fin tant que
renvoyer img

8. Source de l'image : [5]

D. Code de la transformation de Hough améliorée

On travaillera par la suite dans le système de coordonnées défini partie 3.3.

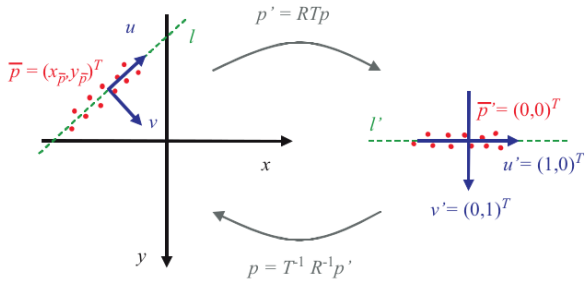

FIGURE 16 – Système de coordonnées adapté [9]

L'ellipse est séparée en 4 cadrans lors du processus de vote afin de simplifier le vote. On accorde une attention particulière aux moments où θ est proche de 0 ou 180, car alors, la distance (algébrique) peut changer de signe. On change alors de cadran où voter.

```
1  import numpy as np
2  import numpy.linalg as npl
3
4  def g_kernels(img, clusters, h, l):
5      """calcul des noyaux elliptiques gaussiens selon l'algorithme de
           ↪ Fernandes et Oliveira"""
6      origine = np.array([l//2, h//2])
7      l_param = []
8      for cl in clusters:
9          n = len(cl)
10         s = np.array(cl) - origine
11         pm = np.mean(s, axis=0) #barycentre
12         s = s - pm #on centre le cluster
13         m = np.dot(np.transpose(s), s)
14         vp, vect = npl.eig(m)
15         if vp[0] > vp[1]:
16             u = vect[:,0] #base orthonormee adaptee a l'ellipse
17             v = vect[:,1]
18         else:
19             u = vect[:,1]
20             v = vect[:,0]
21         if v[1] < 0 :
22             v = -v #pour la compatibilite avec l'equation, y_v >= 0
23         #calcul des parametres des noyaux gaussiens
24         rho = int(np.dot(v, pm))
25         theta = int(180*np.arccos(v[0])/np.pi)
26         a = np.dot(s, u)
```

9. Source de l'image [2]

```
27          sigma_m2 = 1/np.dot(a, a)
28          sigma_b2 = 1/n
29          #definition de la matrice jacobienne
30          #on verifie son inversibilite
31          if v[0] == 1:
32              j = np.array([(-np.dot(u, pm), 1), (1, 0)])
33          else:
34              j = np.array([(-np.dot(u, pm), 1), (u[0]/abs(u[0]), 0)])
35          M = np.dot(np.dot(j, np.array([(sigma_m2, 0),
36              (0, sigma_b2)])), np.transpose(j))
37          sigma_r2 = 4 * M[0][0]
38          if M[1][1] == 0 :
39              sigma_t2 = 0.4 #pour eviter la division par 0
40          else :
41              sigma_t2 = 4*M[1][1]
42          cov_rt = M[0][1]
43          l_param.append((rho, theta, sigma_r2, sigma_t2, cov_rt))
44      return np.array(l_param)
45
46  def G(p_k, rhoj, thetaj):
47      """gaussienne qui donne le coefficient de vote pour chaque cluster
            ↪ """
48      rho, theta, sigma_r2, sigma_t2, cov_rt = p_k
49      r2 = cov_rt**2 / (sigma_r2*sigma_t2)
50      z = (rhoj -rho)**2/sigma_r2 + (thetaj - theta)**2/sigma_t2 -
51          2*cov_rt*(rhoj - rho)*(thetaj-theta)/(sigma_r2*sigma_t2)
52      return np.exp(-z/(2*(1-r2)))/
53          (2*np.pi*np.sqrt(sigma_r2*sigma_t2*(1-r2)))
54
55  def height_c(p_k):
56      """hauteur d'une ellipse gaussienne"""
57      rho, theta, sigma_r2, sigma_t2, cov_rt = p_k
58      r2 = cov_rt**2 / (sigma_r2*sigma_t2)
59      return 1/(2*np.pi*np.sqrt(sigma_r2*sigma_t2*(1-r2)))
60
61  def vote(votes, rho0, theta0, delta_r, delta_t, gs, p_k, R):
62      """procedure de votes pour chaque cluster en utilisant une
            ↪ repartition gaussienne"""
63      rho, theta, sigma_r2, sigma_t2, cov_rt = p_k
64      theta_j = theta0
65      loop = True
66      #la boucle continue tant que des votes sont ajoutes
67      while loop:
68          # on teste si l'orientation de la droite est modifiee i.e la
                ↪ distance algebrique change de signe
69          if theta_j < 0 or theta_j > 180:
70              delta_r *= -1
71              rho *= -1
72              cov_rt *= -1
```

```
            rho0 *= -1
            if theta_j < 0:
                theta += 180 - delta_t
                theta_j = 180 - delta_t
            else:
                theta += delta_t - 180
                theta_j = 0
        rho_j = rho0
        g = round(gs * G(p_k, rho_j, theta_j))
        c = 0
        while g > 0 and -R <= rho_j <= R:
            c+=1
            votes[int(rho_j)][int(theta_j)] += g
            rho_j += delta_r
            g = round(gs * G(p_k, rho_j, theta_j))
        if c==0:
            loop = False
        theta_j += delta_t

def hough_amelioree(img, clusters, h, l):
    """transformation de Hough decrite par Fernandes et Oliveira."""
    R = int(np.sqrt(h**2 + l**2)//2) + 1
    delta = 1
    parametres = g_kernels(img, clusters, h, l)
    #suppression des clusters trop bas
    hmax = 0
    for param in parametres:
        if height_c(param) >= hmax:
            hmax = height_c(param)
    ncl = []
    cl = []
    for i in range(len(parametres)):
        if height_c(parametres[i]) >= hmax * 0.0005:
            ncl.append(parametres[i])
            cl.append(clusters[i])
    parametres = np.array(ncl)
    #seuil gm pour voter avec des entiers
    gm = 1
    for (rho, theta, sigma_r2, sigma_t2, cov_rt) in parametres:
        M = np.array([(sigma_r2, cov_rt), (cov_rt, sigma_t2)])
        vp, vect = npl.eig(M)
        if vp[0] > vp[1]:
            vp_min = vp[1]
            w = vect[:,1]
        else:
            vp_min = vp[0]
            w = vect[:,0]
        gm = min(gm, G((rho, theta, sigma_r2, sigma_t2, cov_rt),
        rho+w[0]*np.sqrt(vp_min), theta + w[1]*np.sqrt(vp_min)))
```

```
122
123     #matrice de votes (i.e espace de Hough)
124     #les distances negatives sont dans la partie 2 du tableau pour
            ↪ utiliser la convention python tab[-k] = tab[n - k]
125     vot = np.zeros((2*R + 2, 180))
126     gs = max(1 / gm, 1)
127     for p_k in parametres:
128         (rho, theta, sigma_r2, sigma_t2, cov_rt) = p_k
129         #on vote dans chaque cadran
130         vote(vot, rho, theta, delta, delta, gs, p_k, R)
131         vote(vot, rho, theta - delta, delta, -delta, gs, p_k, R)
132         vote(vot, rho - delta, theta -delta, -delta, -delta, gs, p_k, R)
133         vote(vot, rho - delta, theta, -delta, delta, gs, p_k, R)
134
135     #detection des pics
136     vot = ndimage.gaussian_filter(vot, 1.5) #espace lisse
137     #on trie les cases non vides de vot par ordre decroissant
138     lst = ((vot >= 1).nonzero())
139     l_triee = ((np.transpose(lst))[np.argsort(vot[lst])])[::-1]
140     vus = np.zeros((2*R +1, 180))
141     pics = []
142     i = 0
143     while i<len(l_triee) and len(pics)<4:
144         idx = l_triee[i]
145         r, theta = idx[0], idx[1]
146         vus[r][theta] = 1
147         #on verifie si le pic n'est pas voisin d'un pic deja vu avec une
                ↪  attention particuliere pour theta = 0 ou 179
148         if theta==0:
149             aij = vus[r][1] + vus[r+1][0] + vus[r+1][1] + vus[-r][179] +
                    ↪  vus[-r+1][179] + vus[r-1][0] + vus[-r-1][179] + vus[r
                    ↪ -1][1]
150         elif theta==179:
151             aij = vus[r][178] + vus[r+1][179] + vus[r-1][179] + vus[r
                    ↪ +1][178] + vus[r-1][178] + vus[-r][0] + vus[-r-1][0] +
                    ↪  vus[-r+1][0]
152         else:
153             aij = vus[r][theta+1] + vus[r][theta-1] + vus[r-1][theta] +
                    ↪ vus[r-1][theta-1] + vus[r-1][theta+1] + vus[r+1][theta
                    ↪ ] + vus[r+1][theta-1] + vus[r+1][theta+1]
154         if aij == 0:
155             if r > R+1:
156                 r -= 2*R +1
157             #conversion en coordonnees cartesiennes
158             a = np.cos(theta*np.pi/180)
159             b = np.sin(theta*np.pi/180)
160             pics.append((a, b, -(r + a*(l//2) + b*(h//2))))
161         i += 1
162
```

```
163     return pics
```

Dans le nouveau système, on définit les coefficients m' et b' tels que la droite approchant l'ellipse est donnée par $y = m'x + b' = 0$. On peut approcher leurs variances et covariances par :

$$\sigma_{m'}^2 = \frac{1}{\sum_{i=0}^{n-1} (x_{p'_i})^2} \qquad \sigma_{b'}^2 = \frac{1}{n} \qquad \sigma_{m'b'} = 0$$

où p'_i sont les pixels translatés et pivotés.

On obtient alors

$$\rho = x_v x_{\overline{p}} + y_v y_{\overline{p}} + b' - (x_u x_{\overline{p}} + y_u y_{\overline{p}})m' \qquad \theta = \arccos(x_v - x_u m')$$

et leurs variances et covariances associées :

$$\begin{pmatrix} \sigma_\rho^2 & \sigma_{\rho\theta} \\ \sigma_{\rho\theta} & \sigma_\theta^2 \end{pmatrix} = \nabla f \begin{pmatrix} \sigma_{m'}^2 & \sigma_{m'b'} \\ \sigma_{m'b'} & \sigma_{b'}^2 \end{pmatrix} \nabla f^\top$$

où ∇f est la matrice jacobienne définie par

$$\nabla f = \begin{pmatrix} \frac{\partial \rho}{\partial m'} & \frac{\partial \rho}{\partial b'} \\ \frac{\partial \theta}{\partial m'} & \frac{\partial \theta}{\partial b'} \end{pmatrix} = \begin{pmatrix} -(x_u x_{\overline{p}} + y_u y_{\overline{p}}) & 1 \\ \operatorname{sgn}(x_u) & 0 \end{pmatrix}$$

qu'on obtient en évaluant la dérivée en 0.

Notons que la valeur de ∇f est mal définie quand $x_u = 0$: on devrait obtenir $\frac{\partial \theta}{\partial m'} = 0$ mais alors ∇f n'est plus inversible, et le changement de base n'a plus de sens. Remarquons plutôt que si on prolonge par continuité à droite, ou à gauche, la seule modification est le signe de $\sigma_{\rho\theta}$. On prolonge donc par continuité des valeurs de σ_ρ et σ_θ, et $\sigma_{\rho\theta} = 0$.