

# Rapport Projet IPT : Détection de contours dans une image

Charlotte Thomas  
Naïs Baubry

30 avril 2021

## Résumé

De la reconnaissance faciale à la médecine de pointe, de nombreuses applications modernes demandent de reconnaître les contours d'une image. C'est pour ces raisons que par ce projet, nous allons étudier une méthode matricielle de détection de contours dans une image : le filtre de Canny.

## Table des matières

<b>1</b>	<b>Détourer une image</b>	<b>2</b>
1.1	Solutions envisageables . . . . .	2
1.2	Algorithme de Canny [1] . . . . .	2
<b>2</b>	<b>Implémentation</b>	<b>4</b>
2.1	Raffinage . . . . .	4
2.2	Exemple de la fonction <i>masque</i> . . . . .	5
<b>3</b>	<b>Problèmes rencontrés et exemples</b>	<b>6</b>
3.1	Problèmes rencontrés . . . . .	6
3.2	Exemple : fonctions avec des jeux de valeurs . . . . .	6
<b>4</b>	<b>Résultats et dépassement</b>	<b>8</b>
4.1	Résultats . . . . .	8
4.2	Dépassement . . . . .	10
4.2.1	Calcul de complexité . . . . .	10
4.2.2	Améliorations . . . . .	10

# 1 Détourer une image

## 1.1 Solutions envisageables

En traitement d'images et en vision par ordinateur, on appelle détection de contours les procédés permettant de repérer les points d'une image matricielle qui correspondent à un changement brutal de l'intensité lumineuse. Pour traiter ces images, plusieurs méthodes sont possibles :

- Le filtre de Roberts met en évidence la différence d'intensité en suivant les diagonales de l'image.

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{pmatrix} \text{ et } \begin{pmatrix} 0 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

- Le filtre de Prewitt met en évidence la différence d'intensité en suivant les axes verticaux et horizontaux.

$$\frac{1}{3} \begin{pmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{pmatrix} \text{ et } \frac{1}{3} \begin{pmatrix} -1 & -1 & -1 \\ 0 & 0 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

- Le filtre de Sobel utilise un principe assez semblable aux deux premiers filtres présentés, seul l'opérateur est modifié. Le principal avantage des filtres de Sobel et Prewitt est leur facilité de mise en œuvre ainsi que la rapidité de leur traitement. De plus, ils utilisent un opérateur de lissage qui améliore le rendu final. Mais leur grande sensibilité au bruit les rend moins performant.

$$\frac{1}{4} \begin{pmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{pmatrix} \text{ et } \frac{1}{4} \begin{pmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{pmatrix}$$

- Le filtre de Canny est une amélioration du filtre de Sobel et utilise un opérateur gaussien. Il suit trois critères :

1. *bonne détection* : faible taux d'erreur dans la signalisation des contours,
2. *bonne localisation* : minimisation des distances entre les contours détectés et les contours réels,
3. *clarté de la réponse* : une seule réponse par contour et pas de faux positifs.

Lors de ce projet, nous avons étudié plus particulièrement le filtre de Canny.

## 1.2 Algorithme de Canny [1]

L'algorithme de Canny permet de détecter les bordures d'une image, il est découpé en plusieurs phases

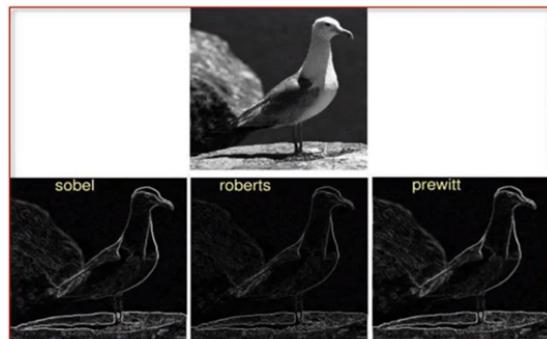


FIGURE 1 – Différents résultats en fonction du filtre utilisé

1. Un filtre gaussien de taille impaire et de paramètre sigma ( l'écart type ) est appliqué sur la matrice de l'image pour réduire le bruit de l'image et améliorer la détection des contours : c'est l'étape de ***bruit***.
2. On fait ensuite la convolution de la matrice de l'image et de deux matrices de gradient pour avoir le gradient en X et en Y de la matrice ( prise comme une fonction de deux variables ), ces deux matrices servent à calculer la matrice de gradient ( la norme du gradient ) pour avoir une première approximation, c'est l'étape de ***gradient***.
3. L'étape précédente fait des bordures plutôt grossières, alors ensuite on traite la matrice par bloc de 9 pixels ( carré de 3 par 3 ) et on cherche les maxima locaux pour affiner l'image, c'est l'étape ***d'affinage***.
4. Ensuite, on applique un seuillage ( avec deux paramètres haut et bas ). L'algorithme de seuillage permet de mettre tous les pixels à 0 ou 1 basé sur s'ils sont sur une bordure ou non, afin d'obtenir un meilleur contraste, c'est l'étape de ***seuillage***.



FIGURE 2 – Exemple d'une image passé par l'algorithme de Canny

## 2 Implémentation

### 2.1 Raffinage

Nous avons découpé au maximum le code afin qu'il soit lisible et agréable à maintenir. *N.B : les types renvoyés sont en **gras** et en *italique**

**Fonctions de support :** elles sont utilisées dans les 4 fonctions principales représentant les étapes de l'algorithme

- *image\_to\_array (finalename : str) → np.array* : transforme une image donnée en entrée (sous une forme de sa localisation ) en array numpy.
- *superposition(i : int; j : int; tableau : np.array; masque : np.array) → bool* : renvoie si le masque peut être superposé centré en (*i, j*).
- *calcul(i : int, j : int, tableau : np.array; masque : np.array) → float* : renvoie l'indice (*i, j*) de la convolution de *tableau* par *masque*.
- *masque(image : np.array, mask : np.array, name="Masque" : str, char="#" → np.array* : renvoie la convolution *d'image* par le tableau *mask*, avec une progressbar de titre masque.
- *gausse(x :int ; y : int ; sigma : float) : → float* : fonction de Gauss de paramètre sigma
- *voisin(i : int ; j : int ; image : np.array) → bool* : renvoie si le pixel (*i, j*) a un voisin blanc

**Fonctions principales :** ce sont les 4 fonctions qui mettent en place les différentes étapes de l'algorithme

- *bruit(taille : int ; sigma : float ; image : np.array) → np.array* : calcule le masque de Gauss et l'applique au tableau image
- *gradient(image : np.array) → (np.array,np.array,np.array)* : calcule le gradient en X, en Y et le tableau des normes des gradients et les renvoient
- *affinage(Gx : np.array ; Gy : np.array ; image : np.array) → np.array* : applique l'algorithme d'affinage par recherche de minimum locaux
- *seuillage (seuil\_haut : float, seuil\_bas : float, image : np.array) → np.array* : Applique l'algorithme de seuillage au tableau image

**Fonctions annexes :** Les deux fonctions qui servent à éviter de copier-coller le code pour demander les entrées clavier à l'utilisateur

- *get (string : str, f : function, except\_string : str, default=0, predicate=lambda x : True :function, additionnal\_string="" :str)* : Récupère une entrée clavier et vérifie sa validité et la renvoie ( le type de renvoi est donc variable )

- *etape (traitement : function,out : str, arguments : list, additional\_args\_functions = [] : function list) → np.array* : fonction générique pour effectuer une étape du processus de détourage

## 2.2 Exemple de la fonction *masque*

La fonction masque est une fonction effectuant la convolution d'un tableau d'entrée par un masque ( donné lui aussi en entrée ), il utilise pour ça deux sous-fonctions : la fonction *calcul* qui calcule la valeur d'une case du tableau de sortie en admettant que cette case peut être calculé et la fonction superposition qui effectue cette vérification.

La fonction *calcul* effectue le calcul d'une case à l'aide d'une formule que l'on a trouvée. Cette formule permet de “lier” une case du tableau du masque à l'endroit où elle est présente sur le tableau (dont on effectue la convolution). Cette liaison est le produit de la superposition. Pour chaque pixel  $(i', j')$  dans masque, on associe le pixel  $(i + i' - p//2, j + j' - p//2)$  dans le tableau. Cette formule a été trouvée à l'aide des deux conditions de bords sur les deux coins du masque.

La fonction masque effectue la convolution en elle-même, pour chaque case  $(i, j)$ , elle vérifie si le calcul est possible, et si oui effectue le calcul, et si non on associe  $c_{i,j}$  à  $t_{i,j}$  ( le multiplicateur sert à le mettre à 0 si on est dans le gradient )

```
def calcul(i,j,tableau,masque):
    #masque recurrence (0,0)->(p-1,p-1) ==> (i',j') dans tableau
    #liaison une case de masque et une case de tableau
    #(p//2,p//2) masque <--> (i,j) tableau (1)
    #@(0,0) masque <--> (i-p//2,j-p//2) tableau (2)
    #@(p-1,p-1) masque <--> (i-p//2+p-1, j-p//2+p-1) = (i+p//2,j+p//2)
    #(i',j') masque <--> (i+i'-p//2,j+j'-p//2) tableau
    n,m = tableau.shape
    p,_ = masque.shape
    p2 = p//2
    somme = 0
    for ip in range(p):
        for jp in range(p):
            somme += masque[ip][jp] * tableau[i+ip-p2][j+jp-p2]

    return somme
```

FIGURE 3 – Code de la fonction *calcul* [2]

```

def masque(image, mask, multiplicateur=1, name="Masque", char="#"):
    #traitement sur chaque (i,j) de image
    C = np.zeros(image.shape)
    n,m = image.shape
    bar = Progressbar(name, n, char=char)
    for i in range(n):
        for j in range(m):
            if(superposition(i,j,image,mask)):
                C[i][j] = calcul(i,j,image,mask) #on calcule
            else:
                C[i][j] = image[i][j]*multiplicateur #c'est en dehors
        bar.update(i)
    print("")
    return C

```

FIGURE 4 – Code de la fonction masque [3]

### 3 Problèmes rencontrés et exemples

#### 3.1 Problèmes rencontrés

Une ligne était mal placée dans le while (cf figure 5) à la ligne 281, cette ligne mal placée dans la boucle while a causé une décrementation de la variable *temp* à chaque tour de la boucle *for* intérieure et non à chaque fois qu'un pixel blanc était détecté dans le voisinage. Donc les bords étaient un peu plus épais que ce qu'il sont avec cette ligne bien placée dans le *if*.

Nous avons eu des soucis pour visualiser le vecteur gradient et particulièrement vers quel pixel pointait le vecteur. Pour trouver un solution, nous avons d'abord calculé l'angle du vecteur gradient avec l'horizontal ( i.e la ligne qui coupe le rectangle immédiatement à droite en deux parties égales ) et séparé en 8 zones espacés de  $\frac{\pi}{4}$  pour symboliser les 8 carrés. Les résultats sont cohérents avec ce qui est présenté dans le document du sujet.

```

274     while temp != 0:
275         for i,j in s21:
276             blanc = voisin(i,j,image) #on vérifie si le pixel a un voisin blanc
277             if blanc == True:
278                 sortie[i][j] = 1
279                 s21.remove((i,j))
280                 temp-=1
281

```

FIGURE 5 – Ligne mal placée ligne 281 [4]

#### 3.2 Exemple : fonctions avec des jeux de valeurs

Nous avons testé la fonction masque avec un tableau généré au hasard et le masque unitaire ( de taille 1 et de valeur 1 ) qui doit redonner le même tableau, cet exemple est bien concluant.

Ensuite, nous avons testé les fonctions suivantes avec l'image donnée dans le sujet ce qui nous a permis de vérifier la qualité et l'efficacité de nos fonctions à l'aide des témoins fournis dans le sujet. Après la fonction *bruit* :



(a) Image originale

(b) taille 3, sigma 0.8

(c) taille 21, sigma 2

FIGURE 6 – Exemple du bruit avec l'image témoin.



(a) après le *Gradient*

(b) après l'*affinage*

(c) après le *seuillage*

FIGURE 7 – Exemple des différentes fonctions.

## 4 Résultats et dépassement

### 4.1 Résultats

Nous avons choisi plusieurs images pour tester notre programme. Le résultat semble très convenable pour la plupart des images.



(a) Image originale



(b) Contours

FIGURE 8 – Résultats avec la maison carrée à Nîmes, les contours du bâtiments sont très précis, les rayons des lampes en arrière plan causent des fausses détections.



(a) Image originale



(b) Contours

FIGURE 9 – Résultats avec une image en dessin "manga" - les contours noir rendent la détection précise et simple.



(a) Image originale



(b) Contours

FIGURE 10 – Résultats avec une photo de M. Robert, la détection est très satisfaisante ici.

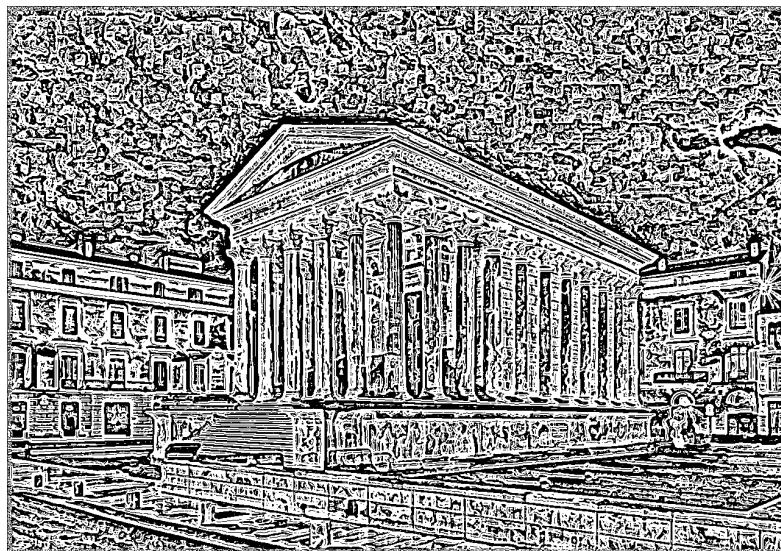


FIGURE 11 – Maison carrée importée en jpg

On peut voir que pour cette image, le programme n'a pas fonctionné correctement. En effet, l'image de base a été chargée en jpeg donc en int et non en float. Ici, les nuances de gris sont représentées par des entiers et non pas par des réels entre 0 et 1 donc toutes les valeurs à 0 restent à 0 et les autres sont mises à 1 d'où l'excès de blanc sur l'image. Cette erreur amène à apprécier la régularité de l'image. Un enchaînement de 0 représente une faible variation de luminosité donc une zone très régulière. Afin de contourner cette erreur, il faut donc soit changer l'image de base en png, soit bouger le seuil pour pouvoir prendre en compte l'intervalle représentant les nuances de gris ( 0 à 255 ).

## 4.2 Dépassemment

### 4.2.1 Calcul de complexité

*Méthodologie de calcul :*

- On compte toutes les opérations usuelles comme  $\mathcal{O}(1)$
- Le randint est compté en  $\mathcal{O}(1)$
- Les chaînes étant de tailles constantes on compte le slicing comme  $\mathcal{O}(1)$
- On calcule nos complexités avec  $m$  comme variable pour la hauteur en pixel de l'image,  $m$  la largeur de l'image (en pixel) et  $p$  comme taille du masque de bruit

*Calcul :*

- **image\_to\_array** Nous utilisons du slicing sur le tableau image à la fois en largeur et en hauteur donc  $\mathcal{O}(nm)$
- **superposition** Seul un test est fait d'où  $\mathcal{O}(1)$
- **calcul** La formule de calcul est utilisée  $p^2$  fois donc  $\mathcal{O}(p^2)$
- **masque** Dans le pire des cas, le calcul est utilisé sur tous les pixels, donc c'est en  $nm \cdot C(calcul)$  d'où  $\mathcal{O}(nmp^2)$
- **gausse** Seul un calcul est fait donc  $\mathcal{O}(1)$
- **bruit** On a une boucle en  $p$  et un appel en  $\mathcal{O}(nmp^2)$  donc comme  $nm >> 1$  on a donc du  $\mathcal{O}(nmp^2)$
- **gradient** La taille des masques est fixée à 3 d'où  $\mathcal{O}(nm)$
- **affinage** On applique un traitement en coût constant pour chaque pixel donc  $\mathcal{O}(nm)$
- **voisin** On ne teste qu'un nombre constant de valeur ( ici : 9 ) donc  $\mathcal{O}(1)$
- **seuillage** On applique un traitement sur tous les pixels et ensuite au pire, il n'y a que  $n*m$  pixels blanc donc la complexité de la boucle est toujours plus grande que celle du while d'où :  $\mathcal{O}(n * m)$
- **get** non applicable
- **etape** non applicable

### 4.2.2 Améliorations

Nous avons eu des difficultés pour prévoir le temps nécessaire à l'exécution des différentes fonctions et particulièrement pour bruit à cause de sa complexité ( dû à sa boucle while). Nous avons donc créé d'incroyables barres de progression afin de savoir si il se passe quelque chose..

Nous avons longuement réfléchi à une solution pour éviter les copier-coller lors de l'enchaînement des fonctions. Nous avons donc créé les fonctions génériques get et étape qui résolvent le problème.



FIGURE 12 – Exemple d'une barre de progression avec la durée écoulée ainsi qu'une estimation de la fin de la tâche.

```

309 def etape(traitement,out, arguments, additional_args_functions = []):
310     count = 0
311     while True:
312         args = []
313         for f in additional_args_functions:
314             try:
315                 args.append(f())
316             except TypeError:
317                 args.append(f(args[-1])) #si elles en ont un => seuil_bas => on met le dernier argument en date (seuil_haut)
318
319         argument = args + arguments #on ajoute les arguments dans l'ordre (image à la fin)
320         sortie = traitement(argument)
321
322         image = sortie
323         if isinstance(sortie, tuple): #si la sortie est un tuple => gradient => l'image est le 3eme element
324             |   image = sortie[2]
325
326
327         plt.imsave("result/*out*.png",image,cmap='gray') #on sauvegarde l'image
328         a = input(get_phrase(count)) #on demande si l'utilisateur.ice est satisfait.e
329         if a == "0":
330             |   print(" ")
331             |   return sortie #si oui on renvoi la sortie du traitement
332         count += 1
333

```

FIGURE 13 – Code de la fonction étape [5] .

## Références

- [1] Algorithme de canny, wikipédia. [https://fr.wikipedia.org/wiki/Filtre\\_de\\_Canny](https://fr.wikipedia.org/wiki/Filtre_de_Canny).
- [2] Charlotte Thomas et Naïs Baubry. Code de la fonction calcul. [https://github.com/coco33920/projet\\_ipt\\_2/blob/master/projet.py#L136](https://github.com/coco33920/projet_ipt_2/blob/master/projet.py#L136).
- [3] Charlotte Thomas et Naïs Baubry. Code de la fonction masque. [https://github.com/coco33920/projet\\_ipt\\_2/blob/master/projet.py#L154](https://github.com/coco33920/projet_ipt_2/blob/master/projet.py#L154).
- [4] Charlotte Thomas et Naïs Baubry. Ligne originellement mal placée. [https://github.com/coco33920/projet\\_ipt\\_2/blob/master/projet.py#L281](https://github.com/coco33920/projet_ipt_2/blob/master/projet.py#L281).
- [5] Charlotte Thomas et Naïs Baubry. Code de la fonction étape. [https://github.com/coco33920/projet\\_ipt\\_2/blob/master/projet.py#L310](https://github.com/coco33920/projet_ipt_2/blob/master/projet.py#L310).
- [6] Charlotte Thomas et Naïs Baubry. Repo github. [https://github.com/coco33920/projet\\_ipt\\_2](https://github.com/coco33920/projet_ipt_2).

*Les sources du programme ainsi que du présent document sont accessible sur un repository github public [6]*