Rapport Analyse d'Image
TP2

Novembre 2016

Benneton Anna - p1007382
Marot Fabien - p1006013

Sommaire

1/ Introduction

En continuité du dernier TP qui consistait à détecter les contours d'une image, nous avons lors de ce TP implémenter une application qui permet de détecter des formes simples (segments, cercles). Cette détection est faite avec la méthode de Hough, qui sera expliquée en détails dans ce rapport.

Notre interface graphique utilise *GTK* et la librairie *OpenCV* est utilisée pour les fonctions simples de traitement d'image.

Dans ce rapport, nous allons présenter nos choix de structures, nos choix algorithmiques pour implémenter les principales fonctions de traitement, nos résultats les plus pertinents, ainsi que les éléments qui nous paraissent améliorables. Vous trouverez également en annexe un manuel d'utilisation de notre application.

2/ Choix d'implémentation

a. Explication des classes

Nous sommes repartis du TP1 pour pouvoir utiliser facilement notre détection de contour qui nous paraissait plus qu'acceptable pour poursuivre. La plupart des classes sont donc identiques à celles détaillées dans le précédent rapport, celles utilisées sont décrites ci-dessous :

1. Hough

C'est le seul nouveau fichier de ce TP et c'est celui qui s'occupe de toutes les fonctions liées aux transformées de Hough. Ce fichier *hough.cpp* n'est pas une classe à proprement parlé mais seulement un fichier qui regroupe des fonctions et que nous avons voulu séparer du reste des traitements d'image (lissage, filtre différentiels…) du TP1 pour plus de clarté.

Ces principales fonctions sont :

- *hough_lines()*
- *hough_circles()*

qui sont utilisées respectivement pour la détection de droites et de cercles. Les appels à ces fonctions sont faites via la classe Image, détaillée ci-après :

2. Classe Image

Elle hérite de la classe *cv::Mat* d'*OpenCV.* Cela permet d'utiliser facilement les fonctions de chargement de fichiers, de sauvegarde ou des fonctionnalités de traitement d'image basique (GrayScale, parcours d'image…).
C'est dans cette classe que sont la plupart des méthodes utiles à la détection de contours, ainsi que 3 membres qui permettent de sauvegarder des informations supplémentaires entre chaque traitement : Un tableau des gradients, un tableau d'amplitudes de ces gradients et un tableau de directions de ces gradients.

Ces principales fonctions sont :

- *hough_transform* (met en évidence les droites ou les cercles sur une image)
- *lissage* (moyen, médian ou gaussien)
- *filtre_differentiel* (filtres de Prewitt, Sobel, Kirsch : selon différentes directions)
- *seuillage* (simple, local, global, hystérésis, hystérésis auto … par moyenne, médiane ou écart-type)
- *affinage* (suppression des épaisseurs de contour)
- *fermeture* (comble les trous dans les contours)
- *detection_contour* (appelle toutes les autres fonctions les unes à la suite des autres)
- *color_direction* (Color les contours en fonction de la direction des gradients)

S'ajoute à celles-ci des fonctions supplémentaires pour passer l'image en nuances de gris, la faire pivoter...

3. Classe Option

Cette classe stocke toutes les variables nous permettant de paramétrer au maximum notre application. Nous les avons regroupées dans une classe afin de pouvoir les passer facilement aux méthodes de la classe Image. En effet, l'objet *option* est le seul paramètre passé aux fonctions, ce qui rend le code plus facilement modifiable et modulable.

4. Classe MainWindow

Cette classe est présente uniquement dans la version graphique (GUI) de notre application et nécessite la librairie GTK pour fonctionner. Cette classe gère toute la partie interface et la gestion des signaux lorsque l'utilisateur paramètre son traitement d'image.
De plus dans cette version de l'application il est possible de revenir en arrière/avant à chaque traitement appliqué à l'image ou bien revenir à notre image de base. Ceci est possible grâce à un tableau d'Image : chaque nouveau traitement est considéré comme une nouvelle Image et ajouté à cette liste.

b. Options paramétrables

L'interface permet à l'utilisateur de modifier rapidement la valeur des paramètres utilisés lors de la détection de contour ou des transformées de Hough. La plupart des valeurs numériques utilisées dans nos traitements sont paramétrables.
Dans notre précédent rapport, nous avons présenté en détails les options de détection de contours (lissage, filtrage directionnel, seuillage, affinage et fermeture). Nous allons détailler ici les options utilisées par la transformée de Hough :

- Option générales :

- *le type de détection voulu* : détection seulement des droites, seulement des cercles ou bien détection des cercles et des droites pour une même image.
- *affichage de l'accumulateur :* au lieu de montrer le résultat.
- *effectuer d'abord une détection de contour* : dans le cas où l'image n'est pas binaire.
- *Résultat sur l'image d'origine* : dessiner les droites ou les cercles sur l'image de gauche au lieu de celle de droite.

- Options des droites :

- *seuil* : valeur minimales pour qu'une case de l'accumulateur soit considérées
- *tracer de droite au point par point* : vérification de l'appartenance de chaque point de chaque droite détectées à un contour.

- Options des cercles :

- *seuil :* pourcentage à partir duquel un cercle est considéré comme tel. Ce pourcentage est fonction du nombre de point détecté pour un rayon donné et du périmètre du cercle.
- *rayon min et max des cercles :* Les cercles détectés auront forcément un rayon compris entre ces 2 valeurs. Ces valeurs sont des pourcentages de la largeur de l'image.
- *distance minimale :* Utilisé lors du calcul des maximas locaux, distance minimale entre 2 cercles voisins.

3/ Transformée de Hough

La transformée de Hough s'applique sur des images binaires, représentant les contours d'une image.
Elle s'effectue en deux phases : d'abord le remplissage d'un accumulateur, ensuite l'extraction des droites ou des cercles… et éventuellement leur affichage sur l'image traitée.

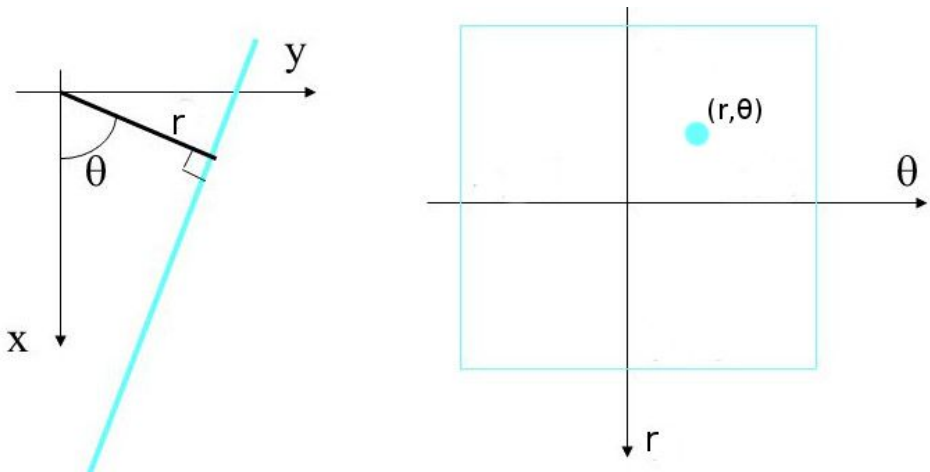
Explication des algorithmes :

a. Détection des droites

➢ Remplissage de l'accumulateur

Notre accumulateur est un simple tableau d'entier, qui correspond à la transformée d'une droite de notre image *(x,y)* en point dans l'espace de Hough *(r,θ).*
Pour le remplir, nous parcourons notre image à la recherche des contours. Lorsque nous tombons sur l'un d'entre eux, nous générons toutes les droites qui peuvent possiblement passer par ce point de contour. Nous faisons varier *θ* entre 0 et 180, et pour chaque valeur nous calculons *r* grâce à la formule d'une droite dans le système polaire :

$$r = x.cos(\theta) + y.sin(\theta)$$



Nous passons ensuite les coordonnées *r,* θ d'un tableau 2D en indice d'un tableau à 1 dimension (notre accumulateur) avec la formule :

$$indice = r * width_{acc} + \theta$$

Et nous incrémentons la valeur de la case *accumulateur[indice]*.
Cette première partie est suffisante si nous ne voulons que des droites. Mais pour pouvoir extraire des segments, nous stockons dans un deuxième accumulateur les points qui ont servi à incrémenter chacune des cases et donc nous ajoutons le *Point(x,y)* à *accumulateurPoint[indice].*

➢ Extraction des droites

Pour trouver les droites, nous parcourons notre accumulateurs et cherchons les valeurs supérieures à un certain seuil (choisi par l'utilisateur). Ce seuil vérifie qu'un nombre suffisant de points sont sur la droite.

Une fois un point validé, nous cherchons à éliminer les droites trop épaisses et les doublons en vérifiant que ce point est un maxima local, c'est à dire nous vérifions que les valeurs de l'accumulateur dans une fenêtre autour du point actuel sont inférieures à celle de ce point. Si c'est bien le cas, on garde la droite correspondante à notre case de l'accumulateur.
Cette droite est stockée sous la forme de 2 points *p1* et *p2* qui sont récupérés dans le deuxième accumulateur *accumulateurPoints,* parmis tous les points qui ont voté pour cette case. On recherche parmis ces derniers le minimum et le maximum, respectivement *p1* et *p2*.

➢ Affichage

Nous parcourons toutes les droites récupérées à l'étape précédente et pour chaque droite (paire de point) nous avons le choix entre 2 méthodes d'affichage :

- points par points :

On itère sur tous les points d'une droite, en vérifiant que ce point est bien un contour. Si c'est le cas, on le marque (on le passe en rouge).
Cette méthode permet un tracé plus précis des droites sans fausses détection mais ne trace pas de segments à proprement parlé.

- segment :

On itère également sur tous les points d'une droite, mais cette fois à la recherche des points minimum et maximum, de manière à ce que ces points soient également des contours sur l'image d'origine. Puis on trace en rouge le segment entre le minimum et le maximum.
Cette méthode génère des segments corrects, mais il arrive que les points de départ et d'arrivés soient hors de la véritable droite, trop à l'extérieur. (*voir les résultats plus bas)*

b. Détection des cercles

Au vue des temps de calculs très long pour les cercles, nous avons ajouté une parallélisation lors des doubles parcours d'image (#pragma omp parallel for). Nous avons ainsi divisé par 3 environ le temps d'exécution.

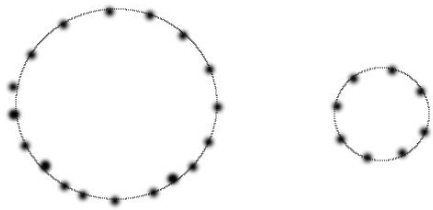
➢ Remplissage de l'accumulateur

La méthode de vote pour remplir l'accumulateur est semblable à celle des droites, seulement ici nous avons choisi de travailler en coordonnées cartésiennes et non polaires, car celles-ci ne nous donnait pas de résultats significatifs.
Pour chaque point de contour de notre image, nous générons tous les centres de cercles possibles dans un interval [rayon minimal - rayon maximal] dont le cercle passerait par notre point actuel.

Ceci se fait en parcourant l'image une 2e fois et en calculant la distance entre notre point p et le point p2 du 2e parcours. Si cette distance est comprise entre *rayon_min* et *rayon_max* (valeurs définies par l'utilisateur), alors nous incrémentons la valeur de notre accumulateur à la case (distance, p2).

➢ Affichage des cercles

Nous parcourons notre accumulateur et pour chaque paire (rayon, centre), nous vérifions que la valeur de l'accumulateur pour cette paire est bien supérieure au seuil choisi par l'utilisateur.
Ce seuil est un pourcentage, calculé en fonction du périmètre du cercle testé (et donc de son rayon) : Pour un même seuil, un rayon plus grand demandera un plus grand nombre de vote
dans l'accumulateur qu'un rayon plus petit.



Si un cercle est valable, nous vérifions qu'il est un maximum local, c'est à dire qu'il n'y ai pas un plus grand nombre de vote dans l'accumulateur pour des cercles de centre voisins ou de rayon proche. Cette fenêtre de contrôle est également passée en paramètre. (*distance min*)

Si ce cercle est à la fois supérieur au seuil et à la fois un maximum local, alors nous le dessinons en vert.

4/ Résultats

Nos résultats sont satisfaisant autant pour les droites que pour les cercles pour ce qui est de la détection (à condition de régler les paramètres correctement pour chaque image).

Par contre, le temps de détection des cercles est vite très long du fait du double parcours de l'image lors du remplissage de l'accumulateur (pour chaque contour, on vérifie chaque point de l'image qui peut être un centre) : la complexité est exponentielle ! (on limite quand même le parcours au rayon max, mais parfois ce n'est pas loin de l'image complète)

Cercles :

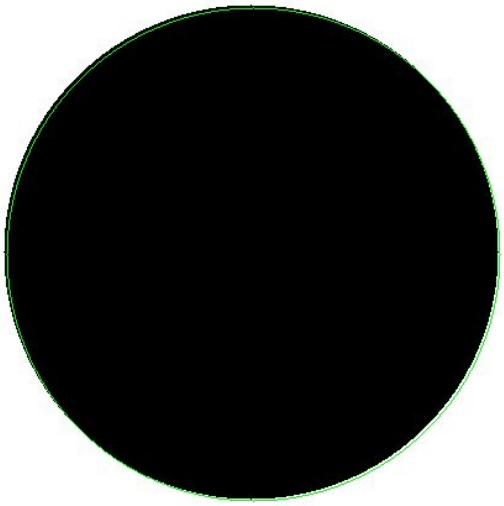


fig.1 - détection d'un seul cercle avec les réglages par défaut (trait vert)

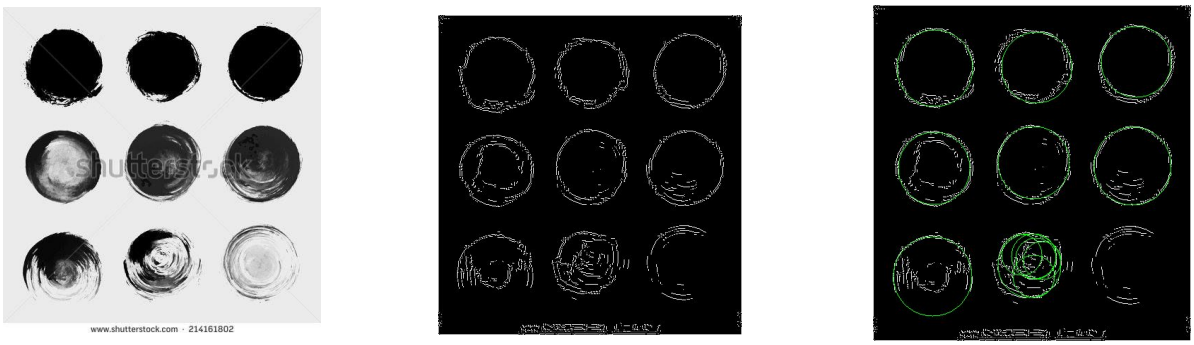
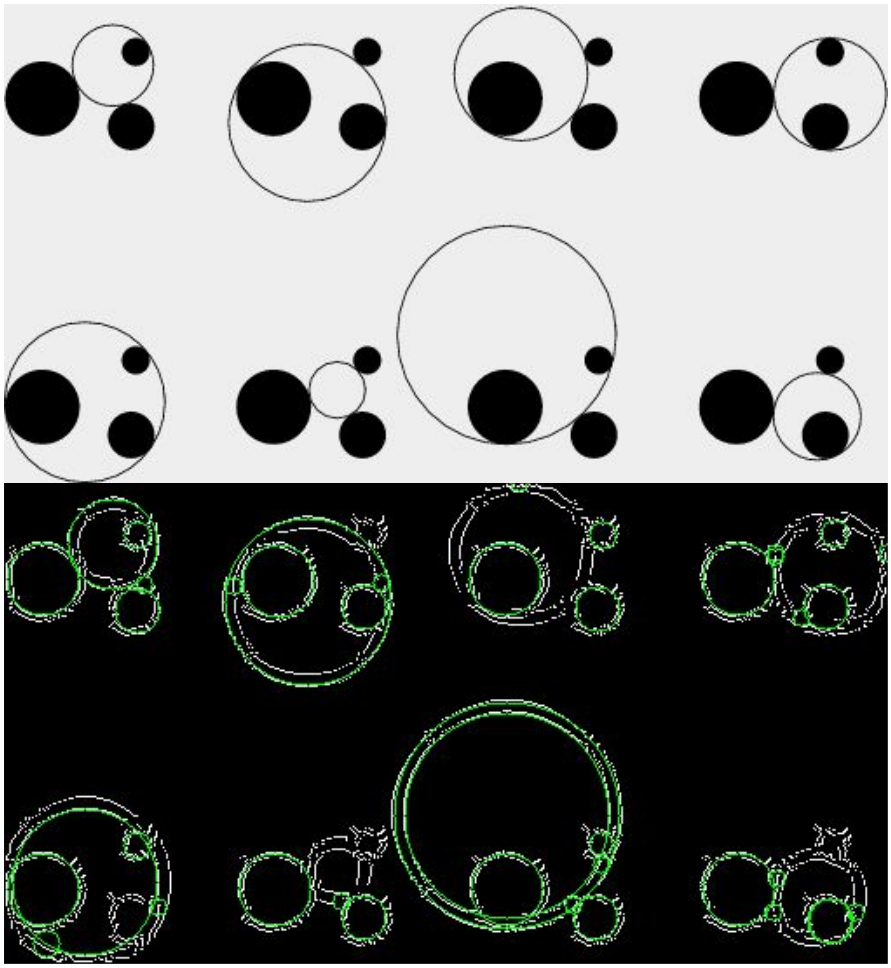


*fig.2 - détection de plusieurs cercles avec un affinage supplémentaire et un seuil de détection à 50. (durée 7 secondes pour une image 470*450)*

fig.3 - Détection de plusieurs cercles : détection de contour avec un affinage supplémentaire et un seuillage global
Hough : rayon minimum à 1% et maximum à 20%. (durée 2 secondes)

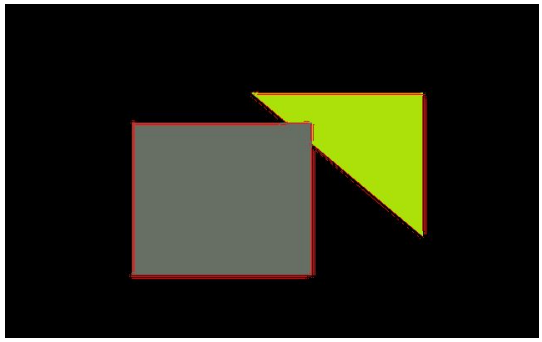


seuil à 80

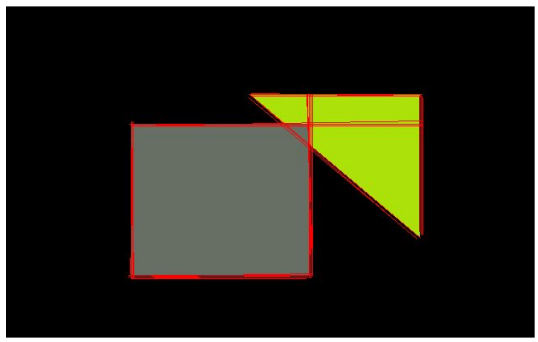
seuil à 100

Droites :

fig.4 - détection de droite : seuil droite à 100

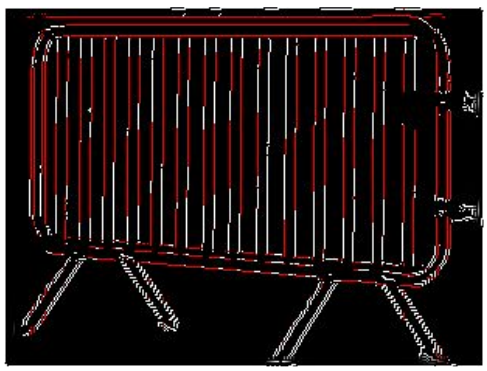


tracé point par point

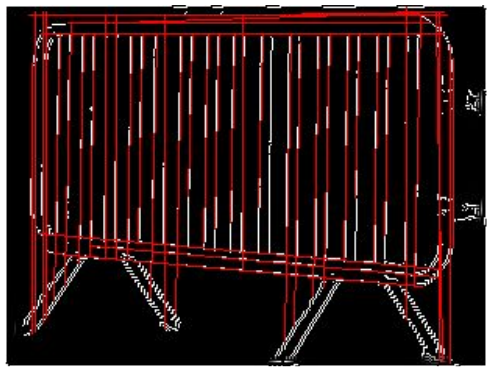


tracé par min/max (1)

fig.5 - détection de droite : seuil droite à 80



tracé point par point (2)



tracé par min/max (3)

(1)La ligne du haut du carré croise le triangle à droite, ce point est faussement considéré comme le maximum de la droite et on a donc le segment du carré qui traverse le triangle jaune.

(2) En point par point, certains points ne sont pas exactement sur la droite détectée et sont donc ignoré, d'où les trous.

(3) Certaines lignes verticales sont prolongées en bas car le maximum de la droite est détecté au niveau des pieds de la barrière.

5/ Conclusion

Pour ce TP nous avons donc implémenté les fonctions demandées, de manière totalement paramétrable par l'utilisateur. Les résultats d'un point de vue qualitatifs sont corrects, par contre, les temps d'exécutions laissent à désirer, surtout pour la détection des cercles.

De plus, un chose à améliorer peut être la visualisation de l'accumulateur pour les droites, ceci n'est pour l'instant possible que pour les cercles.

Il aurait également pu être intéressant lors de la détection de contours de remplir une liste de points correspondant aux contours, de manière à ne pas avoir à re-parcourir toute l'image à chaque fois quand ce que l'on cherche est justement tous ces contours.

Annexes

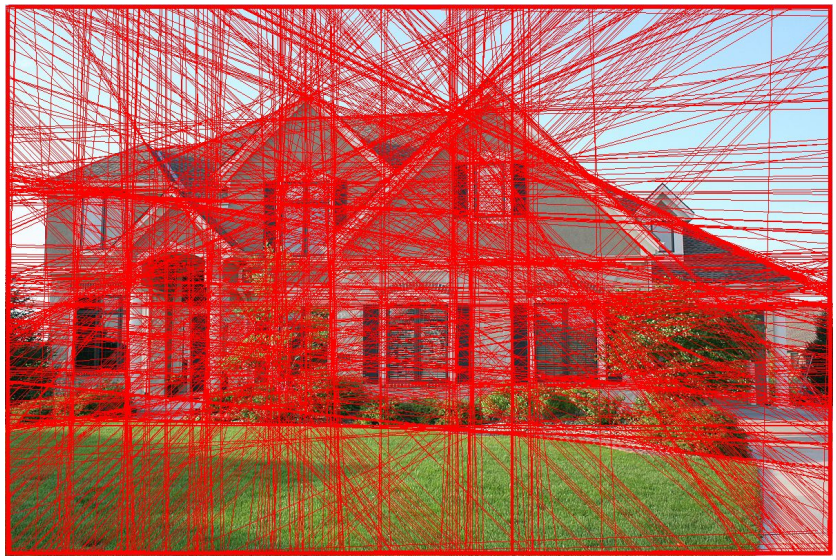
Interface Graphique

1. Ouvrir une image (jpg / png)
2. Enregistrer l'image de droite (-14-)
3. Undo : Annule la dernière action
4. Redo : Refait une action annulée
5. Reset : Supprime tous les traitements appliqués à l'image de base.
6. Apply : Remplace l'image de base à gauche -13- par l'image modifiée de droite -14-.
7. Zoom In/Out/Fit
8. GrayScale (3 types possibles) - Inverser les couleurs - Rotation de l'image.
9. Étape par étape de la détection de contour : les options pour chaque étapes se paramètre dans l'onglet du même nom dans le panel -12-.
10. Détection de contour qui applique automatiquement toutes les étapes précédentes avec les options définies dans le panel -12-.

10 - bis. Transformée de Hough avec les paramètres définis.

11. Faire varier l'intensité des couleurs / Nuance de Gris des résultats.
12. Panel des Options. Voir la partie 2/ b. de ce rapport pour les détails.
13. Image en entrée (voir outil -1-)
14. Image de sortie : on peut l'enregistrer ou la définir comme image d'entrée.
15. Informations sur l'image

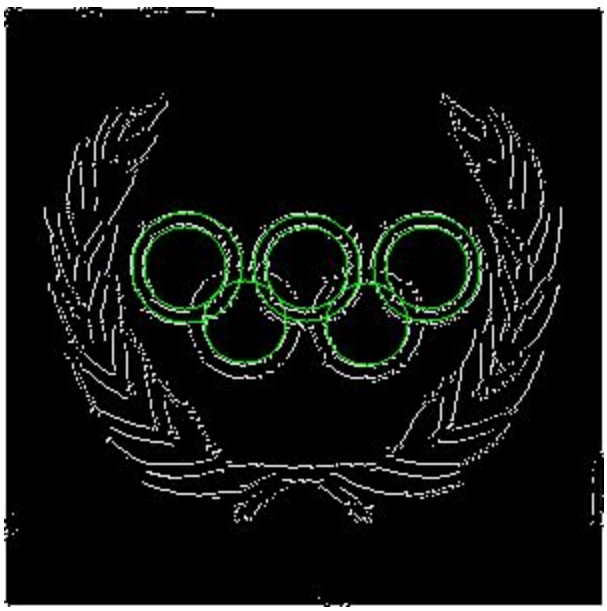
16. Console de Log : Historique de chaque traitement et de ses options appliqué à l'image.



tracé min/max

tracé point par point

tracé min/max ; seuil droite à 150



rayon_min = 5 ; seuil cercle = 65