

Image, Acquisition, Analyse et Traitement de l'Image

Master 2 Recherche IGI (Informatique Graphique et Image)

Université de Lyon 1

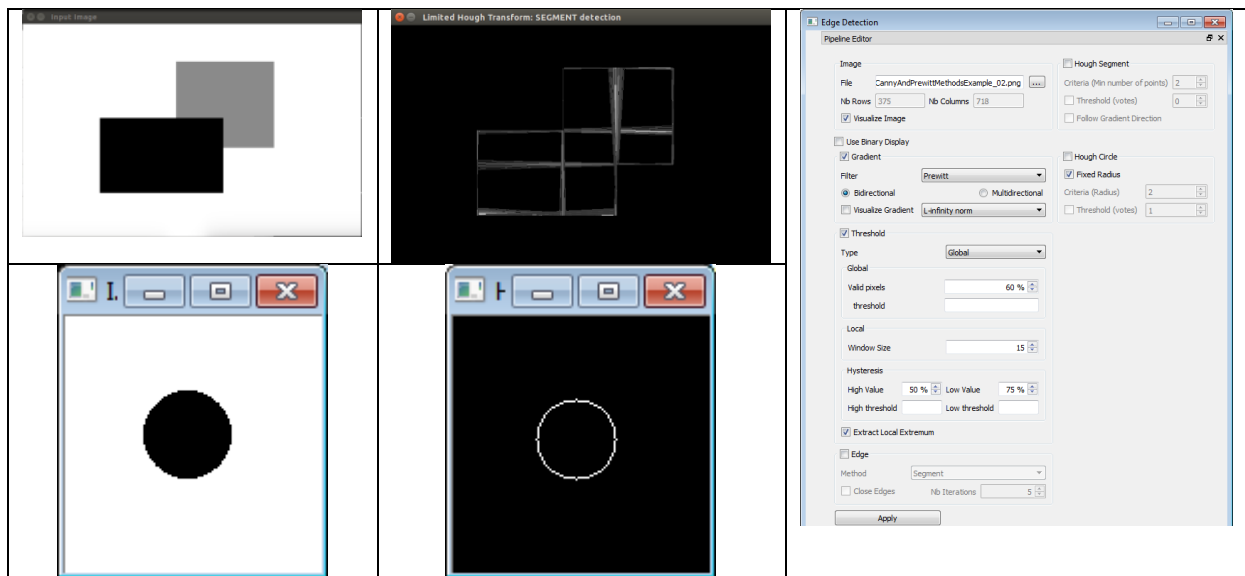
Elèves :

Promo : 2015 – 2016

Pascal GUEHL – p1511749

Clément PICQ –

TP 2 : Transformée de Hough



Contenu

Introduction.....	3
Contexte	3
I. Détection de segments par transformée de Hough.....	4
1. Algorithme	4
a. Principe général.....	4
2. Structure de données utilisée	6
3. Notion de vote.....	6
4. Comment tracer une droite.....	6
5. Limitations des droites	7
6. Résultats obtenus	7
II. Détection de cercles par transformée de Hough	9
1. Algorithme	9
a. Principe général.....	9
2. Structure de données	10
3. Résultats	12
III. Performances	14
1. Affichage des résultats	14
IV. Programme informatique.....	15
1. Dépendances	15
1. Environnement de développement	15
2. Compilation / Execution	15
3. Utilisation	16
Conclusion	17

Introduction

Contexte

Ce TP d'Analyse d'Images comporte 2 parties. La première partie concerne la détection de segments de droites par transformée de Hough. La seconde porte sur la détection de cercles. On se base sur les résultats issus du TP1 sur la détection des contours d'une image.

I. Détection de segments par transformée de Hough

La Transformée de Hough est une technique de reconnaissance de forme appliquée au traitement des images. Inventée en 1962 par Paul Hough, elle permet notamment de détecter des primitives graphiques simples, comme les droites. Sa version généralisée, développée par Richard Duda et Peter Hart en 1972, permet de détecter des formes plus complexes. Nous nous concentrons ici sur sa version simple pour la détection de droites et de cercles.

1. Algorithme

a. Principe général

Le but est de détecter les droites présentes dans une image. Il existe une infinité de droite passant par un point, avec pour seule différence leur orientation. Pour une image donnée, la Transformée de Hough s'attache à déterminer les plus fréquentes. Afin de pouvoir dire que deux points appartiennent à une même droite, on se place dans un espace paramétrique dual permettant de représenter ces droites et les comparer.

Les droites candidates feront partie des points de contours, c'est pourquoi un prétraitement basé sur les résultats du TP précédent est utilisée à savoir : application d'un gradient et extraction des maximaux locaux pour affiner les contours.

Chaque droite va être représentée par l'équation paramétrique suivante à deux paramètres :

$$x \cos(\theta) + y \sin(\theta) = \rho$$

La représentation polaire permet de s'affranchir de cas particulier lié à la représentation cartésienne « $y = a x + b$ », notamment la pente infinie.

Ainsi, à chaque point de l'image, on fait correspondre une courbe de l'espace de Hough basé sur les 2 paramètres « rho » et « theta ». On utilise pour cela une structure de données stockant, pour chaque courbe, le nombre de points associés rencontrés dans l'image. Chaque point va pouvoir « voter » pour une courbe en fonction de si la droite passe ou non par ce point (voir suite, « accumulateur »).

b. Description des paramètres

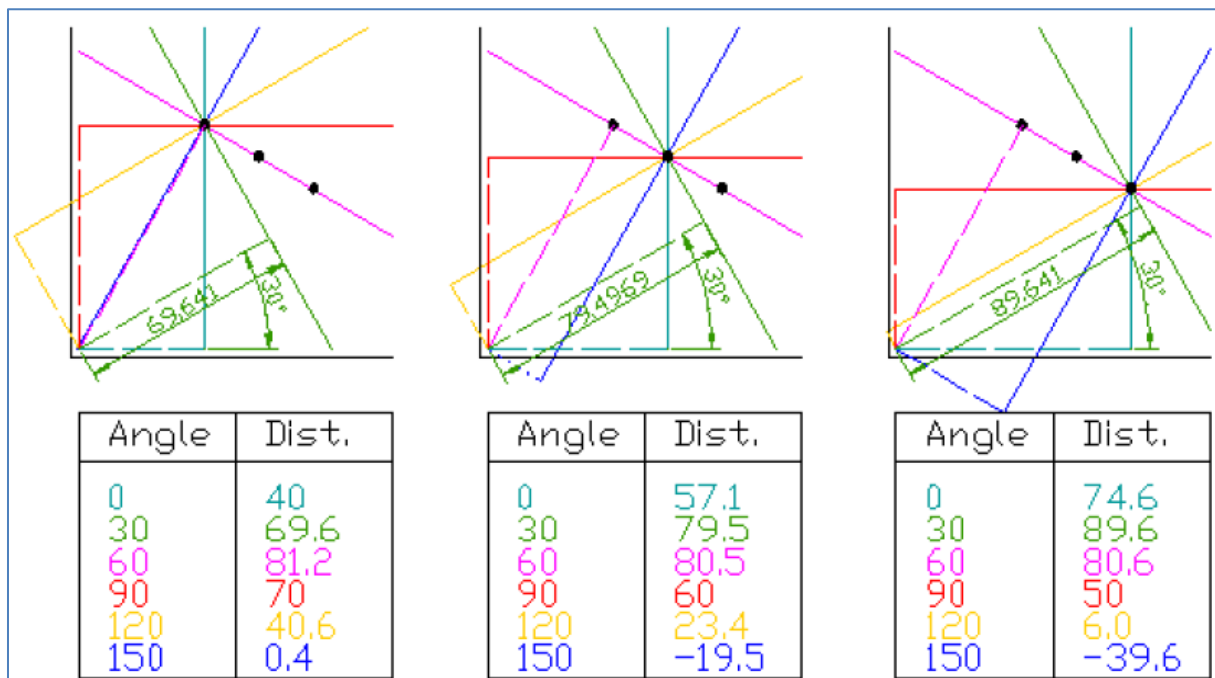
Pour déterminer une droite, on utilise l'équation paramétrique à l'aide de 2 paramètres :

$$x \cos(\theta) + y \sin(\theta) = \rho$$

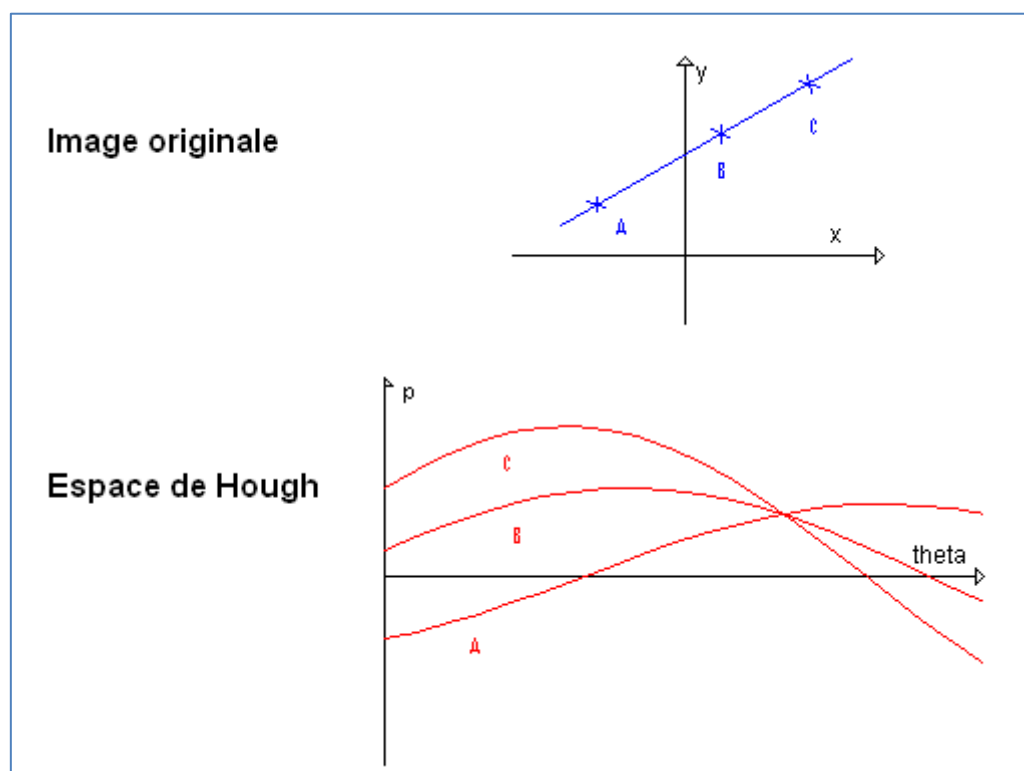
- « Theta » : l'angle de la droite avec l'axe des abscisses.
- « Rho » : la distance de l'origine du repère jusqu'à la droite perpendiculaire à l'angle theta (voir schéma, suite).

Avec « rho » compris entre 0 et la taille de diagonale de l'image (soit $\sqrt{x^2 + y^2}$), et « theta » compris entre $-\pi/2$ et π .

Ci-dessous un exemple illustrant la représentation dans l'espace de Hough (source Wikipedia). Pour points d'un contour de l'image, on détermine l'équation de droites passant par ces points en discrétisant les paramètres « rho » et « theta » :



Voici maintenant un exemple représentant l'espace paramétrique de Hough (source Wikipedia). A chaque point (x,y) de l'image de contours correspond une courbe (theta, rho) dans l'espace de Hough :



Ainsi, par construction, deux courbes se coupant dans l'espace de Hough correspondent à deux points reliés par une droite dans l'image de départ.

2. Structure de données utilisée

Nous souhaitons donc stocker toutes les différentes droites possibles qu'il est possible de tracer dans l'image en fonction de deux paramètres, θ et ρ . Pour pouvoir créer notre accumulateur et ainsi conserver le vote de chaque point, nous utilisons une matrice de « unsigned int » à deux dimensions de taille « ρ » et « θ ».

Pour remplir ce tableau, nous devons donc parcourir chaque point de l'image, et pour chacun de ces points, on doit tester si chacune des droites passe par ce point ou non. Ce qui, au point de vue complexité, peut être très gourmand.

3. Notion de vote

Chaque point va voter ou non pour une droite, en fonction que si celle-ci passe ou non par ce point. Tous les votes sont stockés dans l'accumulateur et ainsi, on peut donc facilement savoir quels sont les droites qui ont le plus de vote.

Dans un premier temps, on va essayer de tracer les droites seulement si elles ont un minimum de vote requis (valeur donnée par l'utilisateur). L'algorithme utilisé est donc très simple, on parcourt l'accumulateur et pour chaque droite, si elle a suffisamment de vote, on la trace.

Dans un deuxième temps, on pourra améliorer les résultats en prenant uniquement les N meilleurs droites.

4. Comment tracer une droite

Pour pouvoir tracer une droite, on procède en deux étapes. La première est de déterminer les deux points d'intersection entre la droite et les bordures de l'image. Puisque la droite est infinie, on est sûr d'obtenir exactement deux intersections. Ensuite, on utilise l'algorithme de « Bresenham » (vu en cours de « synthèse d'images ») pour tracer la continuité entre ces deux points. On répète ce fonctionnement pour toutes les droites qui ont été validées.

5. Limitations des droites

Avec tous ces traitements, on obtient des résultats assez satisfaisant, mais il nous reste encore un traitement à effectuer pour que les résultats soit encore plus lisibles. Pour l'instant, nous voyons simplement plusieurs gros paquets de droites mais nous pouvons difficilement déterminer quelles droites correspondent à quelles droites exactement dans l'image. Pour cela, nous effectuons une limitation des droites pour les restreindre à l'intérieur du contour de l'objet initial.

Pour cela, nous réutilisons la matrice contenant les modules du gradient obtenu dans le TP précédent. Le principe est de partir tour à tour des 4 bordures de l'image, et tant que nous ne rencontrons pas le contour de l'image, on supprime les points de la droite.

Ceci est une solution assez naïve, mais nous permet d'avoir d'assez bons résultats pour les cas simple. Dans les cas un peu plus compliqué, notamment lorsque le contour de l'objet n'est pas fermé, il faudra utiliser un autre algorithme un peu plus poussé.

6. Résultats obtenus

Voici les résultats que nous avons obtenus avec notre programme.

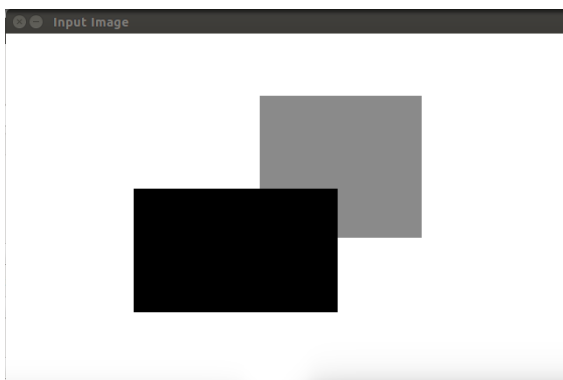
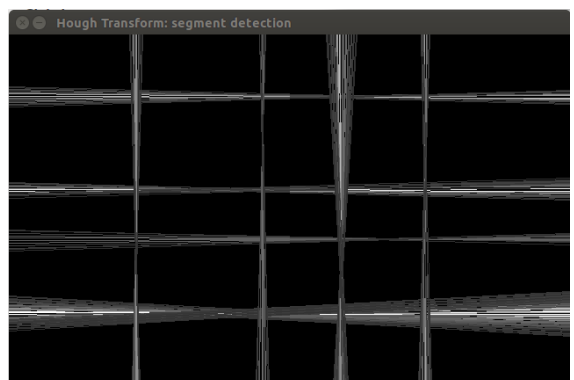
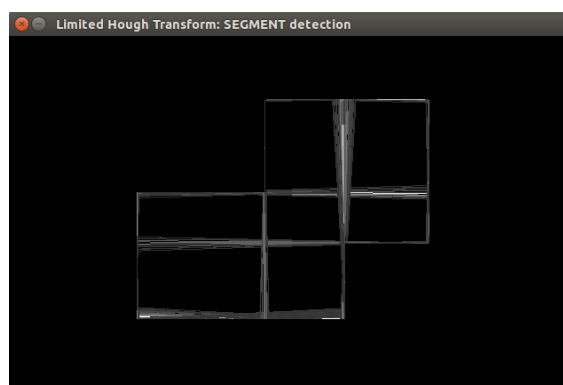


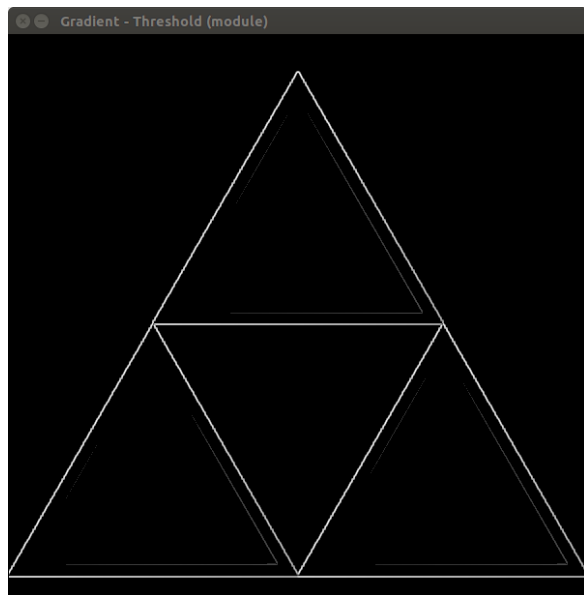
Image en entrée



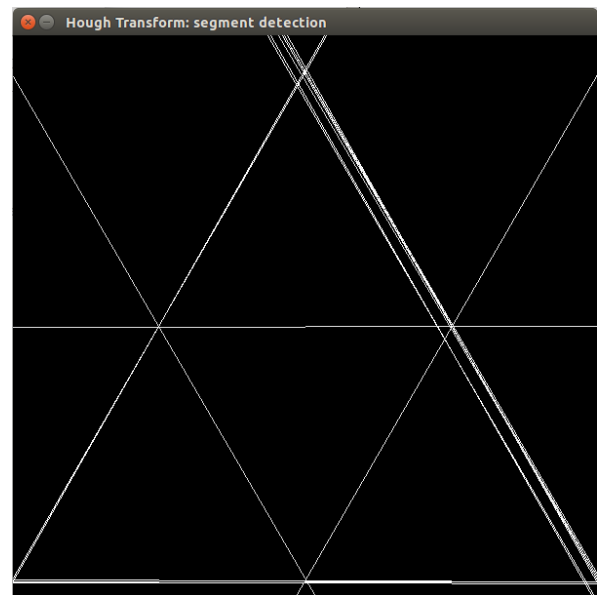
Même Image avec la détection de droites



Traitement pour la limitation des droites appliqué



Résultats du module du gradient obtenu au TP précédent
appliqué sur une autre image



Détection des N meilleurs droites

II. Détection de cercles par transformée de Hough

1. Algorithme

a. Principe général

i. Description

La détection de cercles par Transformée de Hough, ou CHT (Circle Hough Transform), est une spécialisation de la méthode précédente. Comme précédemment, les cercles candidats vont voter dans un accumulateur. Cette fois l'espace paramétrique de Hough est un espace à 3 dimensions basé sur l'équation d'un cercle 2D :

$$(x - a)^2 + (y - b)^2 = r^2 \quad \text{avec } (a,b) : \text{le centre du cercle et « } r \text{ » son rayon}$$

A partir d'un point 2D de l'image, les paramètres a,b et r peuvent être retrouvés par cette équation.

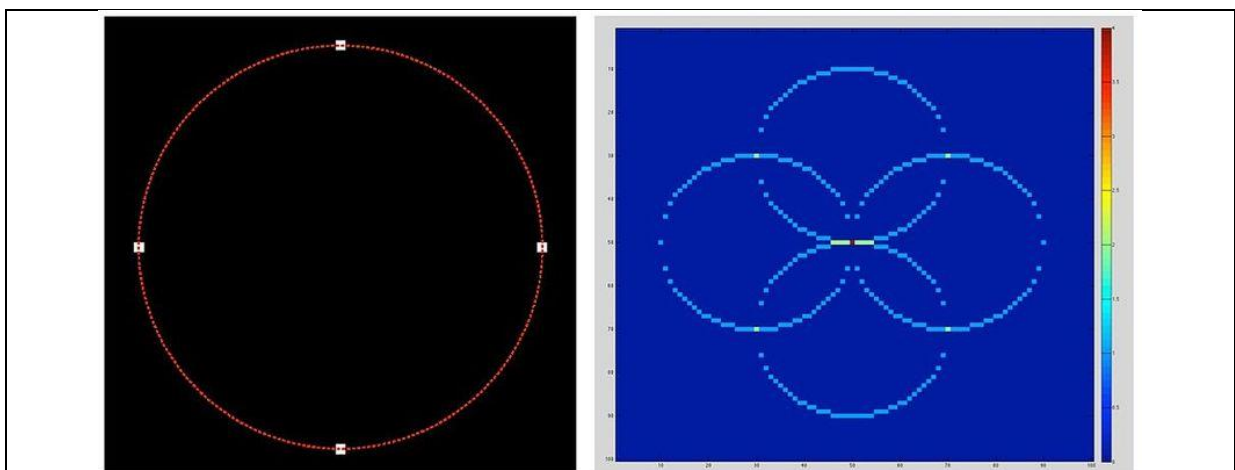
Deux cas sont traités :

- celui où l'utilisateur fixe un rayon
- celui où le rayon varie

RAYON CONNU

Pour simplifier, on peut fixer un des paramètres comme le rayon. L'espace des paramètres est alors réduit à 2 dimensions : la position du cercle (a,b). Pour chaque point de l'image d'entrée (x,y), on définit un cercle centré en (x,y) et de rayon fixe « r ». Les points du cercle sont obtenus en faisant varier les paramètres a et b. Pour chaque configuration de couple (a,b), le point correspondant va voter (incrément d'un contour). On effectue ce travail pour chaque point de l'image d'entrée appartenant aux contours. L'idée est que l'intersection de tous ces cercles dans l'espace paramétrique de Hough donne un point qui serait un bon candidat comme centre du cercle original à déterminer.

Ci-dessous une illustration du principe (source : Wikipédia)



Pour chacun des 4 points du cercle de l'image d'entrée, on génère 4 cercles de (rayon fixé) dans l'espace paramétrique de Hough. En extrayant des maxima locaux, il est possible de retrouver leurs intersections (a,b), correspondant au centre du cercle initial.

Pour cela, on définit une structure de données enregistrant les votes correspondants à tous les cercles candidats de l'image.

2. Structure de données

Comme précédemment, l'accumulateur se définit sous la forme d'une matrice 2D avec un nombre de « b » lignes et « a » colonnes. Les dimensions proviennent de la taille max de l'image d'entrée (on prend la plus grande taille en largeur ou hauteur).

Ci-dessous l'équivalent OpenCV :

```
// Accumulator
// - 2D matrix of size (nbB,nbA) (i.e. nbB rows and nbA columns)
// - initialize accumulator to 0
cv::Mat accumulator = cv::Mat( nbB, nbA, CV_8U/*uchar type*/, cv::Scalar( 0 ) );
```

Accumulateur :

	A = 0	A = 1	A = 2						A = N
B = 0									
B = 1									
B = 2									
B = N									

Si le rayon n'est pas connu, l'accumulateur a une 3ème dimension (profondeur) qui fait varier « r » de 0 à N. Eventuellement, on peut envisager de réduire l'espace des « r ».

L'accumulateur est utilisé pour déterminer le ou les points d'intersection d'un ou plusieurs cercles. En chaque point de l'image de contour (x,y), on génère un cercle centrée en (x,y), de rayon r. En faisant varier « b », on détermine le paramètre « a » à partir de l'équation d'un cercle :

```
float a = x - sqrt( r*r - (y-b)*(y-b) );
```

On veille à ne prendre en compte que des valeurs réelles, dû à la racine carrée.

Chaque point de l'espace de Hough (a,b) ajoute alors son vote dans l'accumulateur. Ci-dessous, l'algorithme:

```
// Iterate through pixels of contour image
for ( int x = 0; x < pImage.rows; x++ )
{
    for ( int y = 0; y < pImage.cols; y++ )
    {
        // Check validity of pixel
        // - valid pixel usually means "is an edge/contour"
        if ( pImage.at< float >( x, y ) < cEPSILON )
        {
            continue;
        }

        // Iterate through parameters in Hough space ([a,b] and r fixed)
        for ( int i = 0; i < nbB; i++ )
        {
            // (x-a)*(x-a)+(y-b)*(y-b)=r*r
            const float tmp = r*r - (y-i)*(y-i);
            if ( tmp >= 0.0f )
            {
                a = x - sqrtf( tmp );
                // Check validity of "a" parameter
                if ( a > 0.0f )
                {
                    // Update accumulare by voting
                    accumulator.at< uchar >( i/*b*/, a ) += 1;
                }
            }
        }
    }
}

...
```

RAYON INDETERMINE

Dans le cas où le paramètre de rayon n'est pas connu, on a un degré de liberté en plus et l'espace paramétrique de Hough comportera 3 dimensions : a, b et r. On conserve la même matrice que précédemment en y ajoutant une 3^{ème} dimension sur le rayon « r » :

```
// Accumulator
// - 3D matrix of size (nbB,nbA,nbR) (i.e. nbB rows, nbA columns with nbR
depth)
// - initiaize accumulator to 0
int accumulatorSizes[] = { nbB, nbA, nbR };
cv::Mat accumulator = cv::Mat( 3, accumulatorSizes, CV_8U/*uchar type*/, cv::Scalar( 0 )
);
```

L'algorithme est identique. Il s'agit juste de rajouter une boucle de parcours du rayon au-dessus des autres paramètres de Hough. Ainsi on parcourt l'espace « r », puis l'espace « b » et on détermine un « a » respectant l'équation d'un cercle centré au pixel de l'image (x,y) :

```

// Iterate through parameters in Hough space (i.e. [a,b,r])
for ( int k = 0; k < nbR; k++ )
{
    for ( int i = 0; i < nbB; i++ )
    {
        // (x-a)*(x-a)+(y-b)*(y-b)=r*r
        const float tmp = static_cast< float >( k*k - (y-i)*(y-i) );
        if ( tmp >= 0.0f )
        {
            a = x - sqrtf( tmp );

            // Check validity of "a" parameter
            if ( a > 0.0f )
            {
                // Update accumulare by voting
                accumulator.at< uchar >( i/*b*/, A, k/*r*/ ) += 1;
            }
        }
    }
}
...

```

3. Résultats

On commence par un exemple simple fait à la main, une image de 128x128 avec 1 seul cercle :

Image en entrée (cercle de rayon 20 pixels)

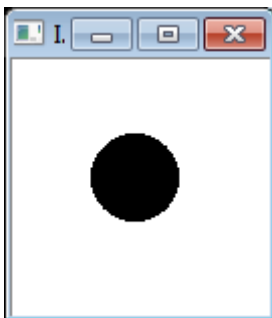
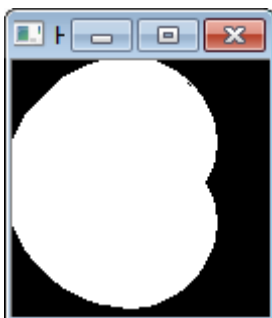


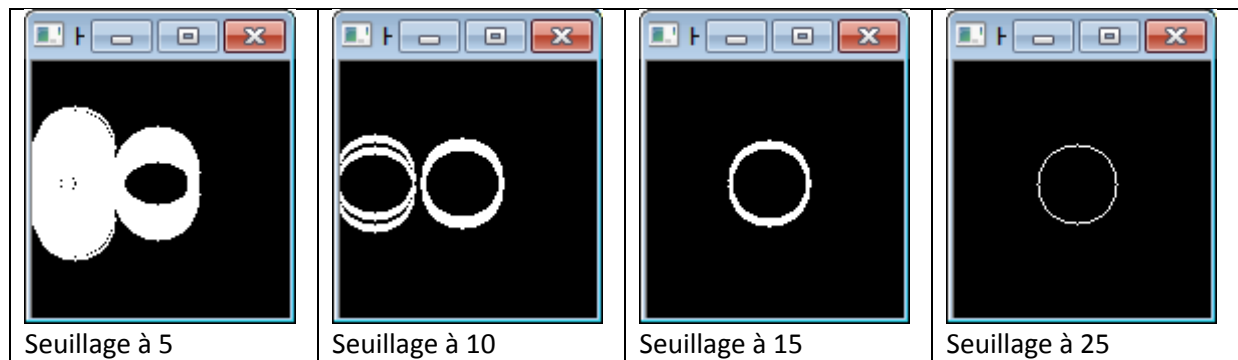
Image de la Transformée de Hough (rayon fixé à 20) :



On remarque les nombreux cercles candidats dans l'espace de Hough

Image de la Transformée de Hough en appliquant un seuil dans l'accumulateur (rayon fixé à 20) :

Ici, on montre le résultat d'un seuillage de l'espace des paramètres de Hough. On ne garde que les cercles ayant un nombre de votes déterminés par l'utilisateur :



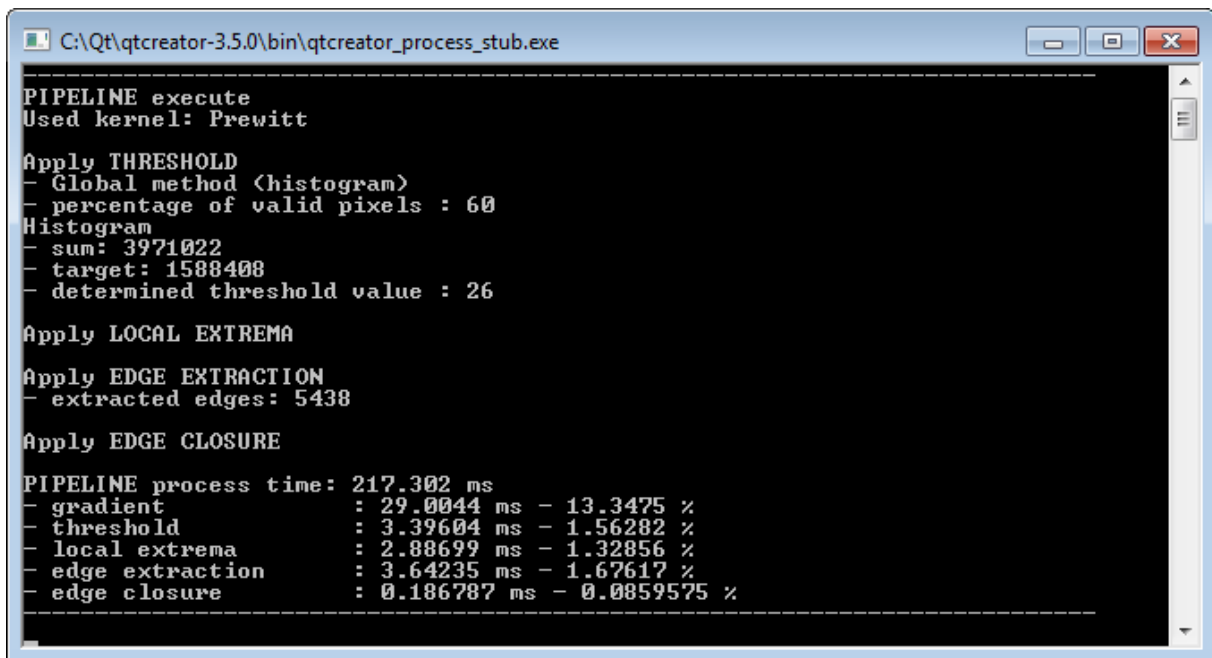
Nous avons ensuite implémenté le cas avec rayon inconnu, soit un 3^{ème} paramètre dans l'espace de Hough. Cependant, nous n'avons pas pu terminer le débogage d'un crash dans le programme.

III. Performances

1. Affichage des résultats

L'API fournit une classe spécialisée dans la gestion des performances en utilisant des timers. Le code multi-plateforme gère le cas de Windows et Linux.

Ci-dessous un exemple de résultats obtenus par notre gestionnaire d'évènements :



```
C:\Qt\qtcreator-3.5.0\bin\qtcreator_process_stub.exe

PIPELINE execute
Used kernel: Prewitt

Apply THRESHOLD
- Global method <histogram>
- percentage of valid pixels : 60
Histogram
- sum: 3971022
- target: 1588408
- determined threshold value : 26

Apply LOCAL EXTREMA

Apply EDGE EXTRACTION
- extracted edges: 5438

Apply EDGE CLOSURE

PIPELINE process time: 217.302 ms
- gradient      : 29.0044 ms - 13.3475 %
- threshold     : 3.39604 ms - 1.56282 %
- local extrema : 2.88699 ms - 1.32856 %
- edge extraction : 3.64235 ms - 1.67617 %
- edge closure  : 0.186787 ms - 0.0859575 %
```

Les informations affichées concernent la durée totale du traitement, puis le détail de chaque étape. Le temps est indiqué en milliseconde et en pourcentage du temps total.

Il est à noter que la somme des étapes est inférieure au temps total. En effet, la bibliothèque OpenCV utilise des méthodes de visualisation qui prennent du temps. Suite à une réorganisation, puis refonte du code, nous n'avons pas pu terminer l'extraction des méthodes de visualisation du pipeline de traitement. Cependant, le gestionnaire d'évènements et performances ne mesure que les temps des fonctions, pas de la visualisation et des éventuelles constructions de matrices intermédiaires.

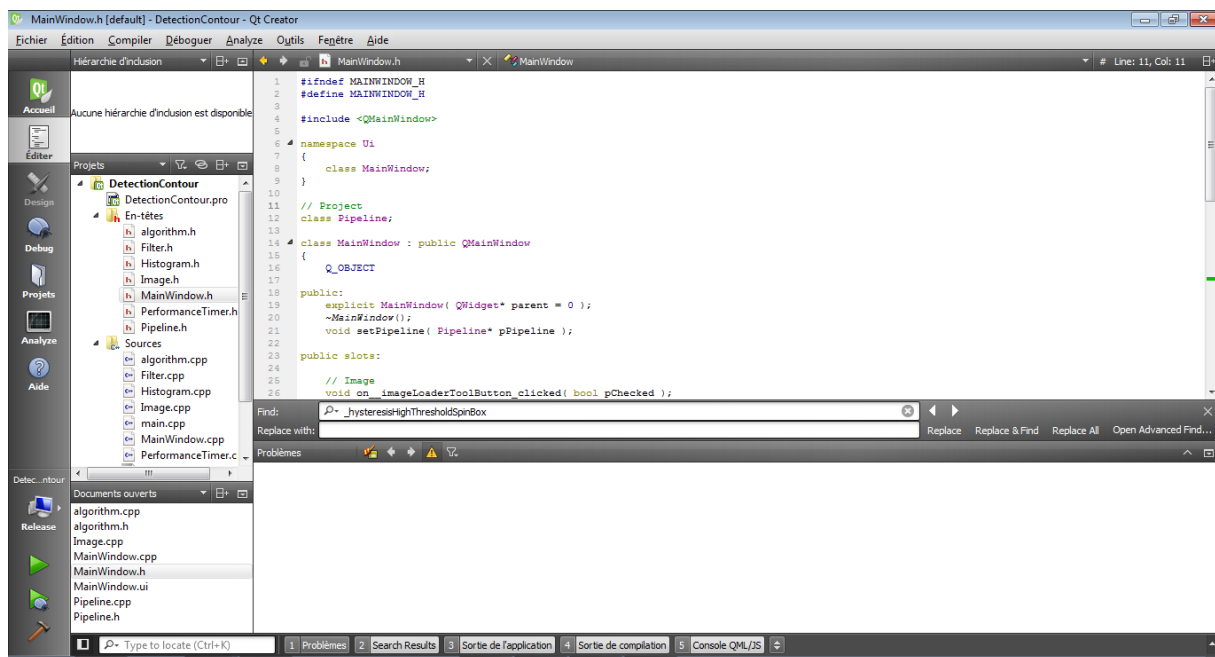
IV. Programme informatique

1. Dépendances

Le programme nécessite l'installation de la librairie OpenCV et de l'environnement de développement qtCreator.

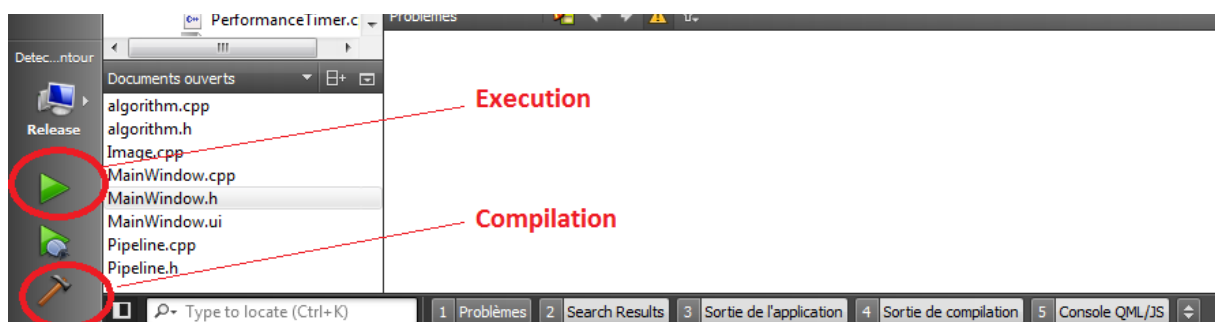
1. Environnement de développement

L'environnement de développement qtCreator permet de se libérer de la contrainte des lignes de commandes et des Makefiles. Il suffit d'ouvrir le fichier projet « DetectionContour.pro » avec qtCreator. L'environnement de développement se configure automatiquement :



2. Compilation / Execution

Pour compiler puis lancer le programme, il suffit de cliquer sur les deux boutons suivants situés en bas à gauche de l'interface :



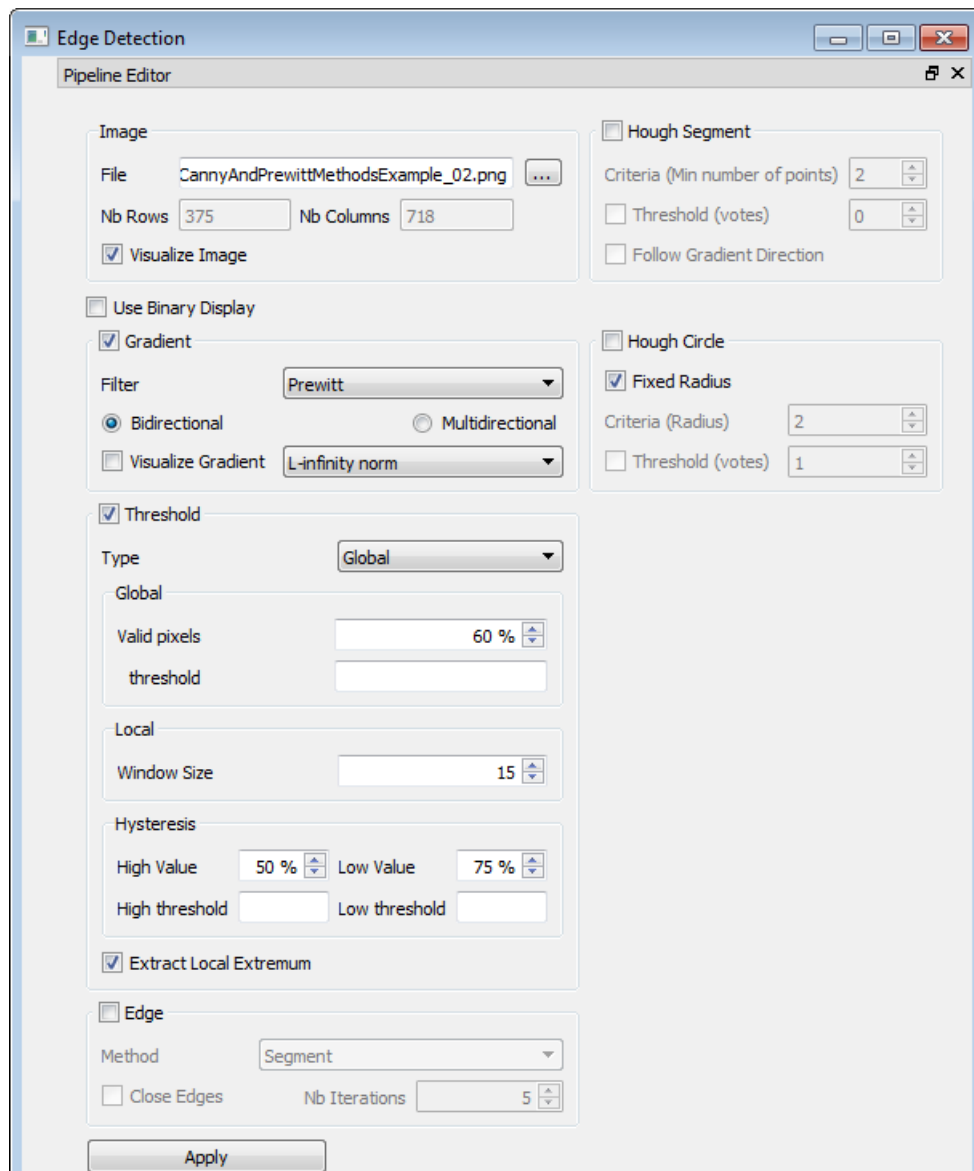
3. Utilisation

Tous les paramètres du pipeline de traitement sont paramétrables via un éditeur Qt :

Un bouton APPLY situé tout en bas permet de lancer les calculs. On peut le lancer autant de fois qu'on le souhaite en modifiant tous les paramètres, les images, etc...

Les performances sont affichées dans une console.

Les images sont affichées dans des images OpenCV.



IMPORTANT :

Pour utiliser la Transformée de Hough, il faut choisir une image, activer les « checkbox » du Gradient, du Threshold et de l' »Extraction des minima locaux », puis ceux associées à la transformée de Hough

NOTES : un soin particulier a été porté sur la documentation du code. Le code a été séparé dans diverses classes C++.

Conclusion

Ce TP nous a permis d'implémenter et mettre en place un pipeline de traitement d'images spécialisé pour la détection de primitives simples par Transformée de Hough.

On obtient des résultats qui sont assez satisfaisant, mais on pourrait encore améliorer ces résultats, notamment pour la détection de droites, en utilisant la direction du gradient pour pondérer le vote apporté par chaque point.

Pour les cercles, on voit bien la complexité de l'algorithme en terme de ressources. Le 3^{ème} paramètre augmente les temps de calculs. De plus, le choix de la discrétisation du « rayon » peut entraîner une difficulté supplémentaire pour déterminer les rayons candidats.