# Première partie : Filtres

Dans cette première partie, nous allons implémenter les filtres moyen, médian ainsi que des filtres morphologiques.

## Filtre Moyen :

Pour chaque pixel de l’image nous calculons la moyenne de ses voisins qui appartiennent à la matrice 3x3.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  | X |  |
|  |  |  |

La valeur de chaque pixel alentour est stockée dans un tableau, puis, la moyenne de ce tableau est affectée en tant que nouvelle valeur du pixel. Ainsi de suite pour toute l’image.

Nous avons également implémenté ce filtre en utilisant la fonction prédéfinie d’open CV :

CvMat \*masque = cvCreateMat(3,3,CV\_32FC1);

cvSet(masque,cvScalar(0));

cvAddS(masque,cvScalar(1.0),masque);

cvNormalize(masque,masque,1.0,0.0,CV\_L1);

cvFilter2D(image\_nvg,image\_nvg,masque);

## Filtre Médian :

Pour chaque pixel de l’image, nous récupérons la valeur de chaque pixel alentour en considérant une matrice 3x3 comme précédemment puis, nous trions le tableau obtenu pour en récupérer la valeur médiane. Cette valeur sera affectée au pixel courant.

Nous avons également implémenté ce filtre en utilisant la fonction prédéfinie d’open CV :

cvConvertImage(image,image\_nvg,flip);

cvSmooth(image\_nvg, image\_nvg, CV\_MEDIAN, 3);

## Filtres morphologiques :

Cette partie contient les filtres :

* Erosion
* Dilatation
* Ouverture
* Fermeture
* Gradient
* Top hat
* Black hat

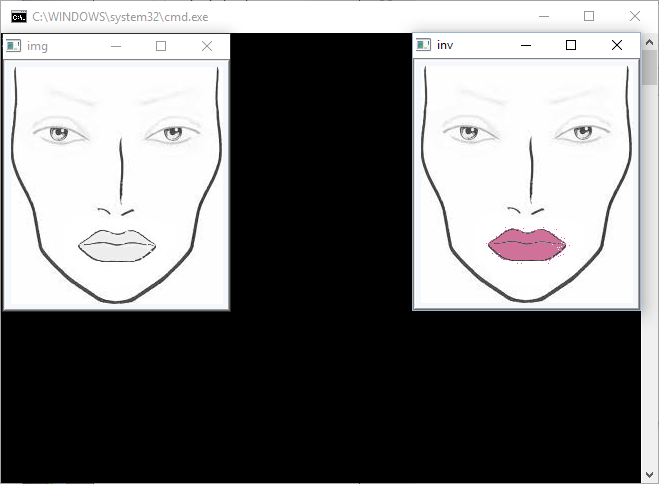
Ces filtres sont implémentés à l’aide des instructions prédéfinies d’open CV comme indiqué dans le TP6.

# Deuxième partie : Traitement supplémentaire

Nous nous intéressons dans cette partie au domaine du maquillage professionnel et des face chart.

Les face chart sont des figures humaines sur lesquelles les maquilleurs conçoivent de nouveaux looks avant de les appliquer sur des modèles réels.

Notre traitement supplémentaire consiste à appliquer du rouge à lèvres à un face chart comme montré dans l’image :



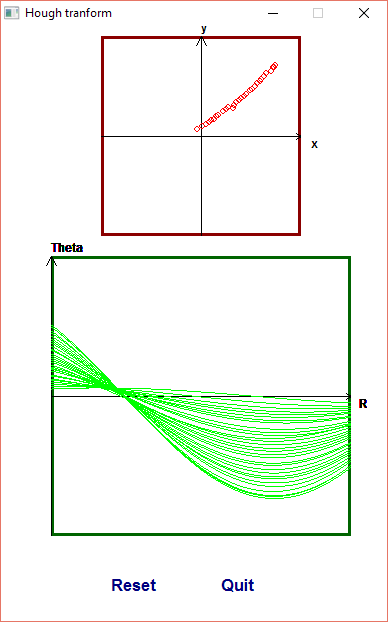
# Troisième partie : Transformée de Hough

## Générale :

La transformée de Hough est une technique de d’imagerie numérique inventée en 1962 par Paul Hough qui permet initialement dans sa forme la plus simple, la détection de droites dans une image, mais aussi après certaines modification basés sur le même principe elle permet la détection de diverses formes géométriques. La transformée de Hough généralisée quant à elle fut inventé quelques années plus tard par 1972 par R. Duda et P. Hart et permet d’étendre le principe de la transformée de Hough à la détection de formes quelconques.

## Détection de droite :

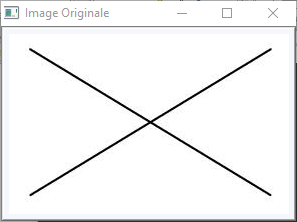
Le principe est simple chaque point représenté dans un espace 2d sur un plan (X,Y) est représenté par une droite dans l’espace de Hough (R,Theta) aussi appelé accumulateur, si tous les points de cette droite sont représenté par une certaine droite dans l’espace de Hough avec un certain ongle theta alors toutes les droites vont se croiser en un seule point qui représentera la droite initiale du plan (X,Y). Malheureusement cette représentation présente quelques problème en pratique dû au fait que le point (0,0) dans une image se trouve en haute à gauche et non en bas à gauche comme dans un repère classique, pour palier à ce problème une nouvelle représentation est apparu qui consiste à représenter un point du plan (X,Y) en une sinusoïde dans l’espace de Hough, ce principe est présenté très facilement par l’application fournis avec l’application du TP.



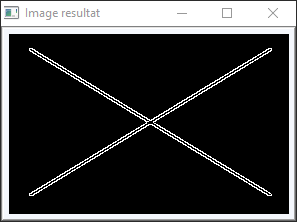
Chaque points de la droite dans le repère (X,Y) est représenté par une sinusoïde dans le repère (R,Thera) et tous les points de cette droite se croiserons en un seul point dans l’espace de Hough.

Coté implémentation l’accumulateur n’est autre qu’une matrice de la même taille que l’image à chaque fois on incrémente chaque point de la sinusoïde (système de votes) la case qui aura reçu le plus de vote ne sera autre que la droite de l’image initiale, bien sûr avant chaque détection l’image est lissé par un filtre gaussien pour diminuer le bruit puis on applique la fonction cvCanny pour la détection de contours.

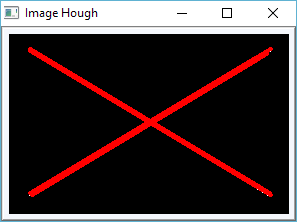
Image initiale :



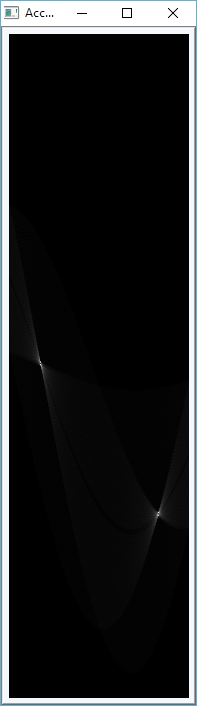
Application des filtres et détections de contours :



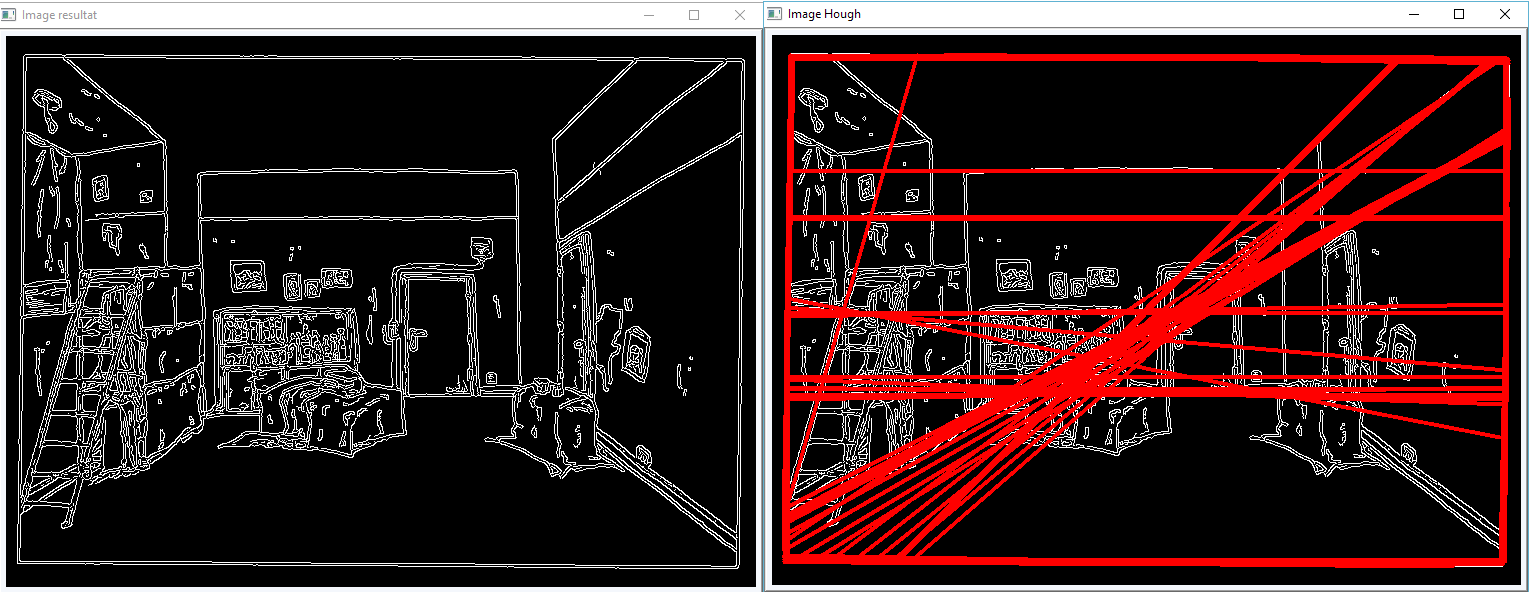
Application de Hough :



Accumulateur :



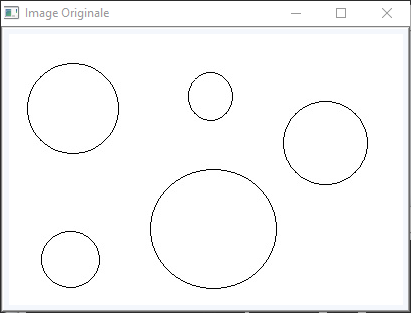
Autre image plus compliqué :



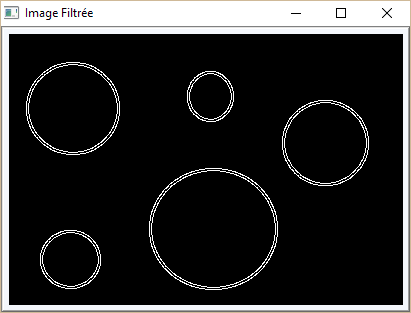
## Détection de cercles :

Le même principe est appliqué pour la détection de cercles chaque point dans l’image initiale est représenter dans un espace de Hough, chaque point du cercle initiale est représenté par un autre cercle dans l’espace de Hough le plan est représenté en 3 dimension la 3ème fera varier le rayon ce qui faut qu’au final nous aurons un cône représentant tous les cercles possible selon le rayon qui sera un seuil max à définir, cette représentation étant beaucoup trop gourmande chaque point du cercle initiale est représenté par un cône dans l’espace de Hough, il a fallu trouvé une alternative, ce qui est le cas de l’utilisation de la tangente chaque point sera représenter par sa tangente.

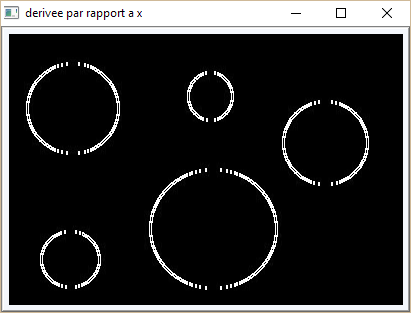
Image initiale :

.

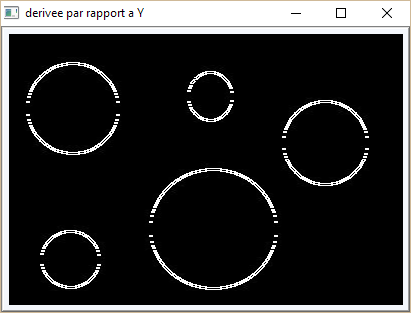
Application des filtres :



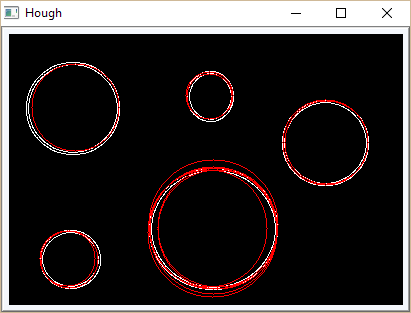
Dérivée par rapport à x :



Dérivée par rapport à y :

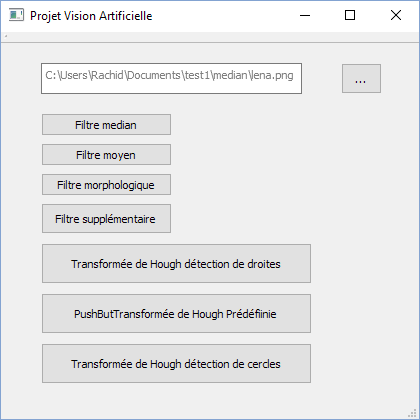


Résultat Hough :



# Interface :

Cette première partie est regroupée dans une interface créée en Qt



Il suffit de choisir le fichier sur lequel tester les filtres.

Tous les filtres y sont représenté avec les fonctions prédéfinies et créer par nos soins, lors du clique sur un bouton le filtre sélectionné est appliqué à l’image sélectionnée.

# Quatrième partie : Reconnaissance de textures

Dans cette partie du projet, nous allons concevoir et implémenter une méthode de reconnaissance de texture sur la base d’un ensemble de cinq images relevant du domaine de la métallurgie. Notre objectif final est de pouvoir décider si une image donnée en entrée appartient ou non à cet ensemble initial.

Pour ce faire, nous utiliserons l’outil MATLAB pour les traitements et Java FX pour l’interface.

## Méthode de fonctionnement

Chaque image est lue puis représentée sous forme de matrice grâce à l’instruction

>> img1 = imread('D:\\images\_test\\test1.tif');

Les images étant en niveaux de gris, chaque cellule de la matrice contient un nombre entre 0 et 255 qui représente un pixel de l’image.

Après la lecture des cinq images, nous calculons les distances entre elles et ce, deux à deux en utilisant la distance euclidienne.

>> tab(1)=sqrt(sumabs((img1-img2).\*(img1-img2)));

Il en résulte un tableau de 10 éléments suivant :

|  |  |
| --- | --- |
| Distance 1-2 | 1.1462e+04 |
| Distance 1-3 | 1.3284e+04 |
| Distance 1-4 | 1.1221e+04 |
| Distance 1-5 | 1.1367e+04 |
| Distance 2-3 | 1.8923e+04 |
| Distance 2-4 | 1.5485e+04 |
| Distance 2-5 | 1.5970e+04 |
| Distance 3-4 | 1.3493e+04 |
| Distance 3-5 | 1.3182e+04 |
| Distance 4-5 | 1.6240e+04 |

Par la suite, nous calculons :

* La moyenne des distances = 1.4063e+04
* Le maximum des distances= 1.8923e+04
* Le minimum des distances= 1.1221e+04

Le nombre d’images étant restreint, nous décidons d’évaluer si une image est reconnue de la manière suivante :

A partir de la moyenne des distances, nous récupérons la distance qui s’y rapproche le plus :

>> moyenne = 1.4063e+04

>> distance 3-4 = 1.3493e+04

Nous choisissons l’image 3 comme un « centre » à partir duquel nous allons calculer la distance des nouvelles images.

Nouvelle image : égale

>> imgEg = imread('D:\\images\_test\\egale.tif');

>> distance 3- imgEgale = 1.8384e+04

Si la distance calculée entre l’image centre et la nouvelle image est comprise entre le minimum et le maximum de notre ensemble alors l’image est identifiée comme étant correcte.

>> imgDif= imread('D:\\images\_test\\differente.tif');

>> distance 3- imgDif = 6.5517e+03

Au contraire, si la valeur de la distance n’est pas comprise entre le minimum et le maximum comme pour imgDif, cette image est rejetée : elle n’appartient pas à l’ensemble.

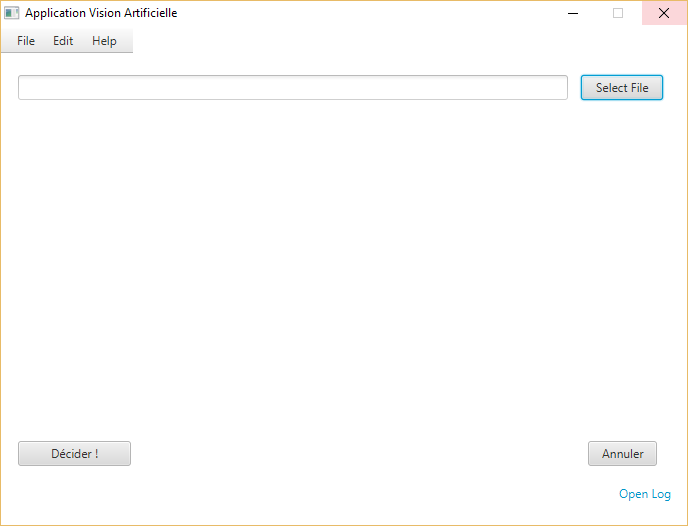
Pour améliorer cette méthode, nous avons choisi d’utiliser de nouvelles matrices calculées à partir des matrices initiales auxquelles la transformée de Fourier a été appliquée et dont les fonctions sont disponibles sur MATLAB ( fft() et fft2() ). Nous calculons les distances de la même manière entre ces nouvelles matrices et nous obtenons un tableau des distances avec de nouvelles valeurs.

|  |  |
| --- | --- |
| Distance 1-2 | 9.9719e+06 |
| Distance 1-3 | 1.1460e+07 |
| Distance 1-4 | 1.1243e+07 |
| Distance 1-5 | 1.1884e+07 |
| Distance 2-3 | 1.1018e+07 |
| Distance 2-4 | 1.0756e+07 |
| Distance 2-5 | 1.0998e+07 |
| Distance 3-4 | 1.1713e+07 |
| Distance 3-5 | 1.0253e+07 |
| Distance 4-5 | 1.0738e+07 |

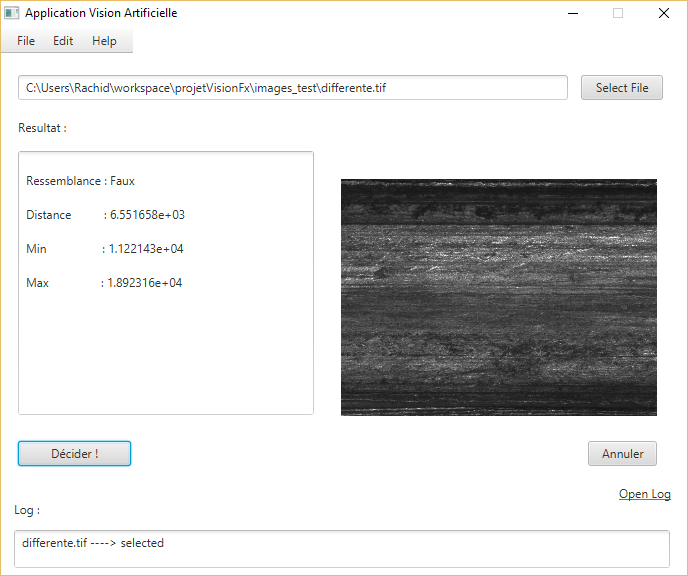
La distance entre le minimum et le maximum étant plus importante, l’espace couvert est donc plus large.

## Interface

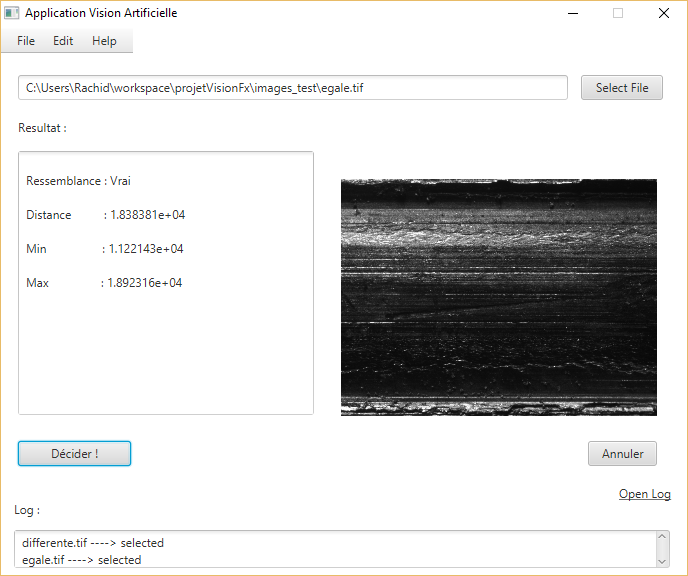
L’interface développée nous permet d’introduire une image et d’effectuer la décision selon le raisonnement expliqué plus haut, elle se présente comme suit :



Après le choix d’un fichier grâce au sélecteur de fichiers l’image est affichée dans un cadre sur la droite de la fenêtre dans le cadre de droite, une fois que le bouton « décider » est cliqué le traitement est lancé puis une fois terminé le résultat de la décision ainsi que la distance calculée le minimum et le et le maximum sont affichés dans un nouveau champ à gauche de la fenêtre de l’application dans le cas où la décision est positive ressemblance est mis à faux sinon a vrai. Un dernier cas est celui ou la taille de l’image sélectionnée ne correspond pas un message est directement affiché dans la section « résultat » alors. Aussi un log détaillant tout ce qui se passe en arrière plant est ouvrable/fermable grâce au lien « Open Log »

Cas différent :

Cas égale :



Cas taille différente :

