

# CR 6 : Musée Sécurisé Virtuel

Semaine 29 novembre au 5 décembre 2021

## Contexte

On cherche à chiffrer puis acquérir et déchiffrer des images, correspondant à des œuvres dans un musée.

01	Chiffrement / Déchiffrement	<ul style="list-style-type: none"><li>• Définition de la clé</li><li>• Chiffrement par permutation (par bloc de pixel et carte du boulanger)</li><li>• Déchiffrement avec comparaison</li></ul>
02	Traitement de la photo : Recalage	<ul style="list-style-type: none"><li>• Acquisition de l'image</li><li>• Définition de la projection</li><li>• Recalage</li></ul>
03	Analyse des résultats sur images réelles	<ul style="list-style-type: none"><li>• Analyse de la performance recalage</li><li>• Analyse de la performance chiffrement/déchiffrement</li></ul>
04	Réalité augmentée	<ul style="list-style-type: none"><li>• OpenCV : Module AR</li><li>• Test et Performances</li></ul>

## Création de l'oeuvre affichée

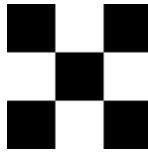
Il faut automatiser la création de l'oeuvre chiffrée en déterminant le pattern exact nécessaire à l'oeuvre. On décide de créer l'oeuvre en [page](#).

## Détection de l'image

Notre œuvre telle que décrite dans le point 6, une fois chiffrée sera très texturée. La recherche de points d'intérêts directement sur notre image risque de ne pas donner des points d'intérêt spécifiques, voire pire des points de l'image.

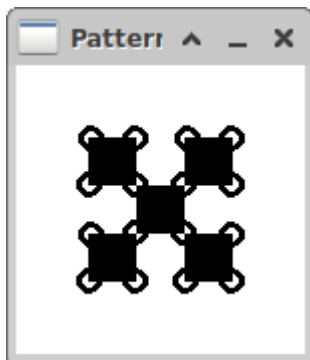
Pour détecter l'image on cherche donc à détecter ses coins, pour cela on ajoute un pattern en damier aux 4 coins de l'image. Ce pattern permettra d'avoir des corners points plus robustes.

*Pattern*



### Détection du pattern :

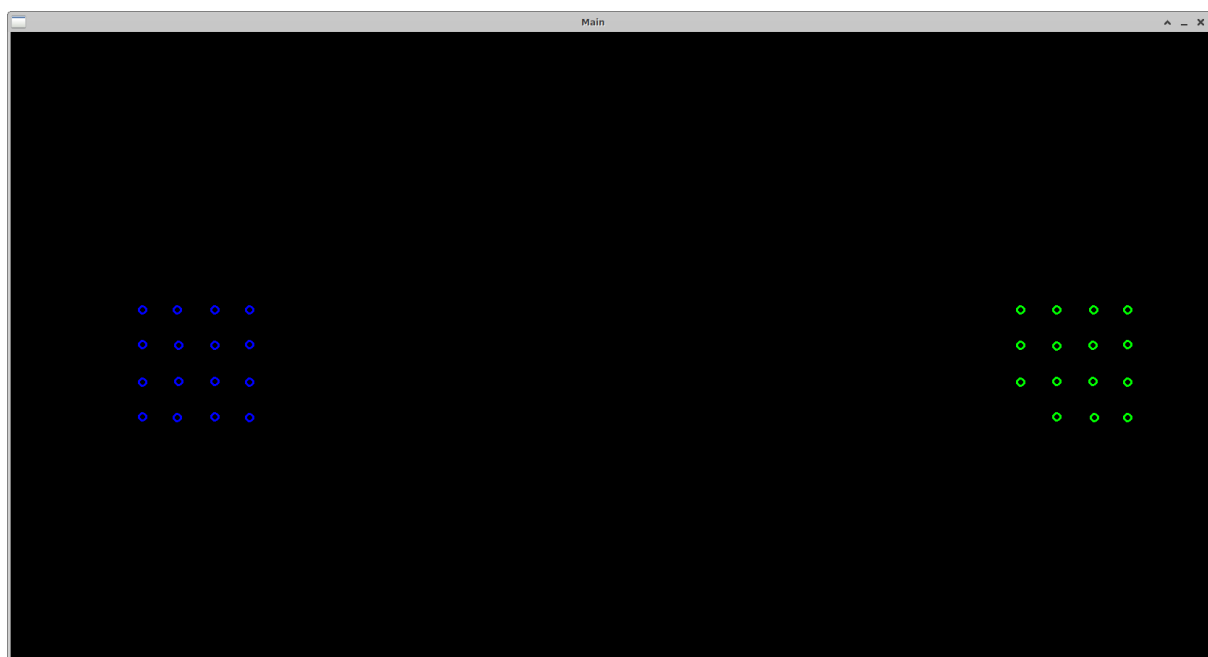
On détecte les coins de Harris sur notre pattern, ainsi que sur l'image globale. Puis on filtre afin de supprimer les doublons.



*exemple détection cv::cornerHarris sur le pattern*

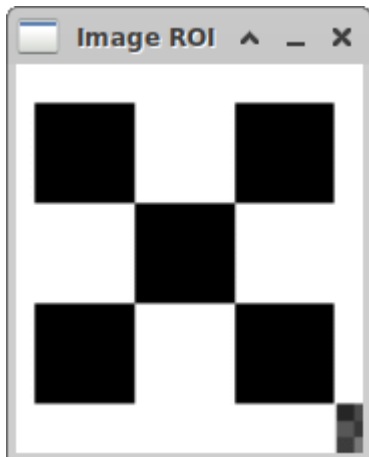
Mon implémentation du hierarchical clustering ne donnant pas des classes cohérentes. Je décide de classifier les données de manière "brute". On sépare l'image en 4 (en divisant par 2 sur la longueur et la largeur). On classe les points par appartenance à leur coin. On affine ensuite la méthode en définissant un radius autour du centre de l'image pour ne pas classifier les données qui appartiendraient à l'image.

Avec notre image précédente (sans déformation de l'image par une caméra) on obtient cette classification. *Exemple sur le début de l'image*



On définit ensuite la region of interest sur laquelle on souhaite matcher nos points détectés sur l'image principale versus nos points détectés sur le pattern (étendu).

*Exemple ROI pour la classe 0*

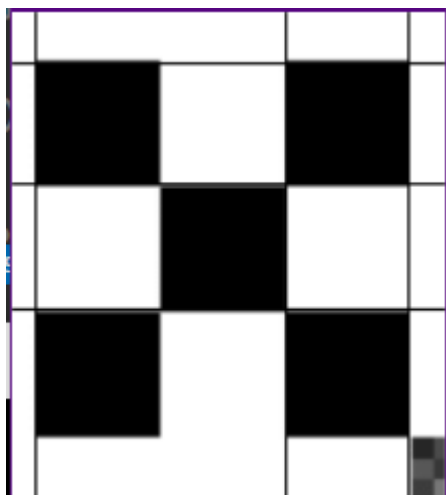


## Implémentation de la transformation de Hough

Objectif: retrouver l'équation des droites passant par nos points en exprimant cette droite selon l'équation (A)  $\rho = x_k * \cos(\theta) + y_k * \sin(\theta)$

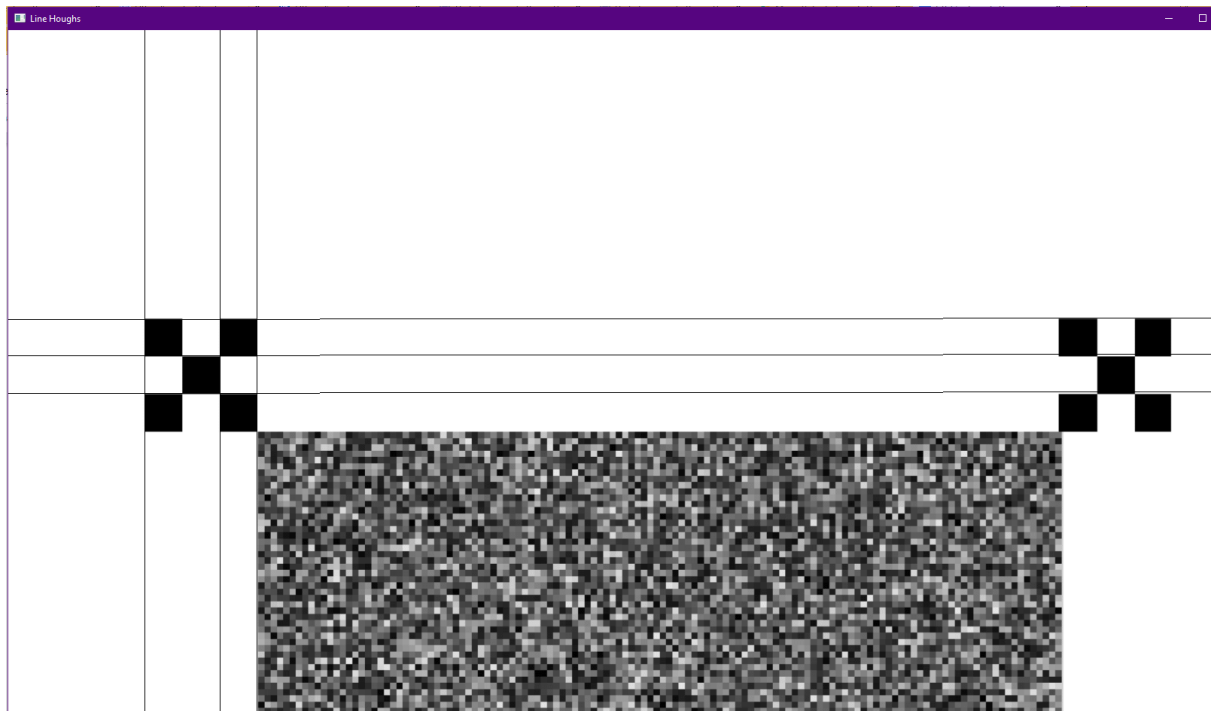
On cherche à trouver les lignes de l'image à partir des points extraits via la recherche des edges. En exprimant chacun des points d'une classe on obtient les equations de droites suivantes. On trace ensuite ces droites dans l'espace de l'image.

*Exemple: Droites dans le ROI*



*Exemple: Hough accumulator array*



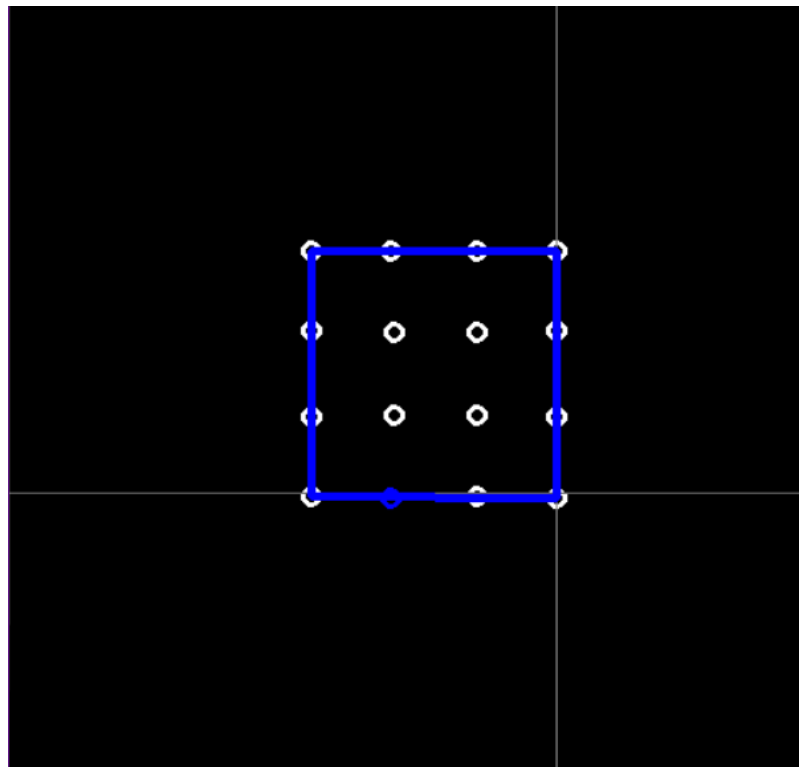


Pour la classe 0, on trouve pour  $d\theta = 2.0^\circ$ , et  $d\rho = 6.5 \times 10^{-3} \times \text{diagonale}(\text{image})$ , des droites dont la précision semble bonne et dans une quantité suffisante pour détecter notre image.

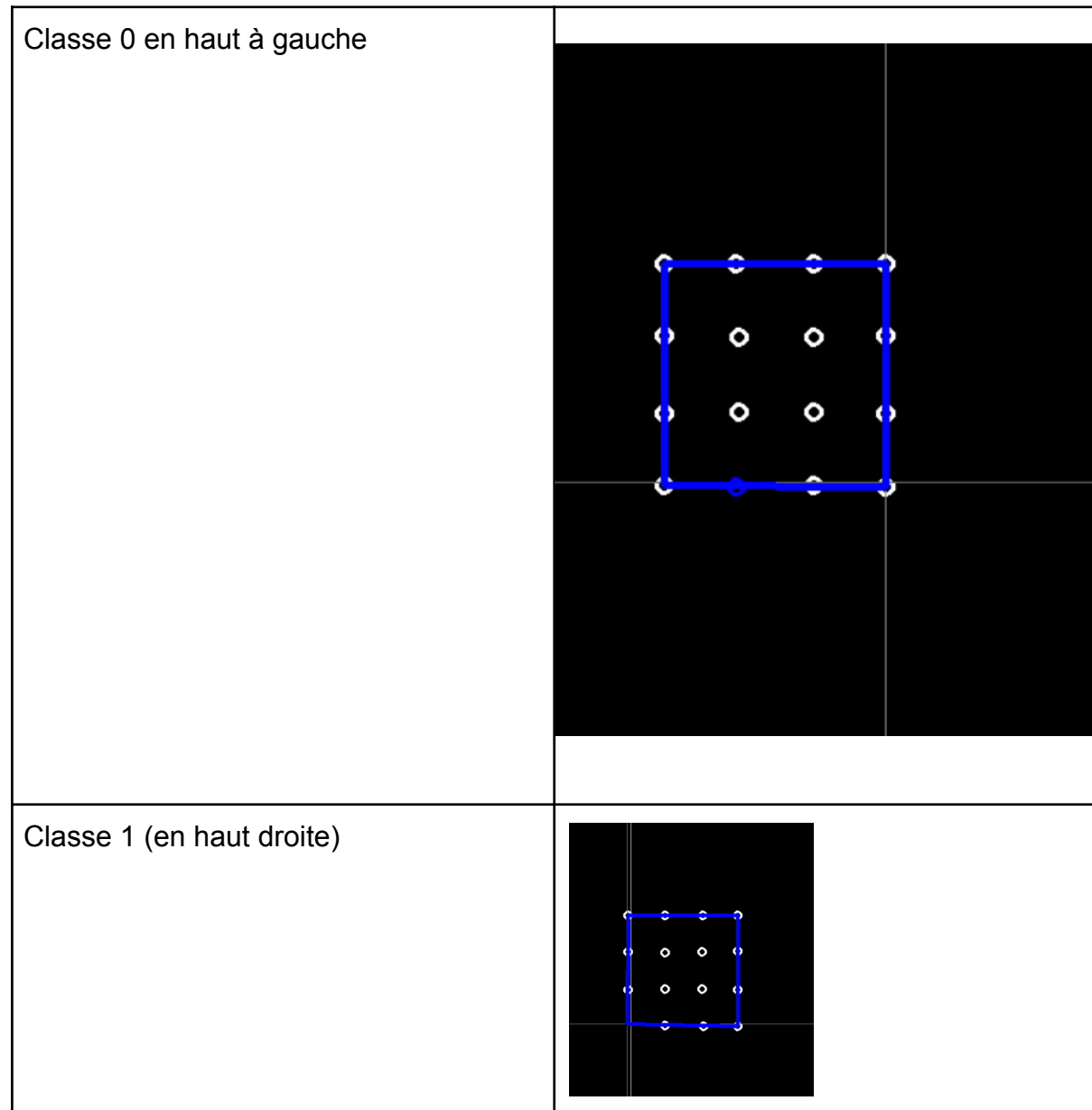
On souhaite ensuite extraire les limites du pattern qui sont accolées aux coins de notre image.

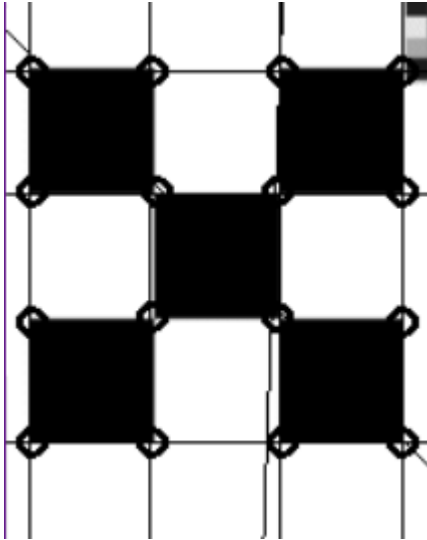
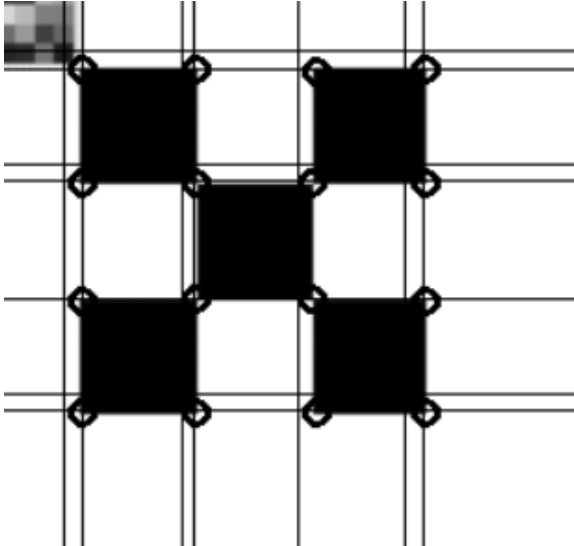
Chaque point vérifiant l'équation d'une droite valide peut servir à la reconstruction du rebord de notre pattern. Une droite est dite valide si elle a été construite avec au moins 3 points. Ces points sont en blanc sur notre image.

En gris une définition simple des bords qui nous intéressent en prenant en compte chacun des points détectés (minima / maxima). En bleu la définition d'un carré représentant la détection du pattern.



Lorsque nous n'avons pas de distortion induite par une caméra et/ou de la perspective, notre algorithme permet de retrouver l'image chiffrée.

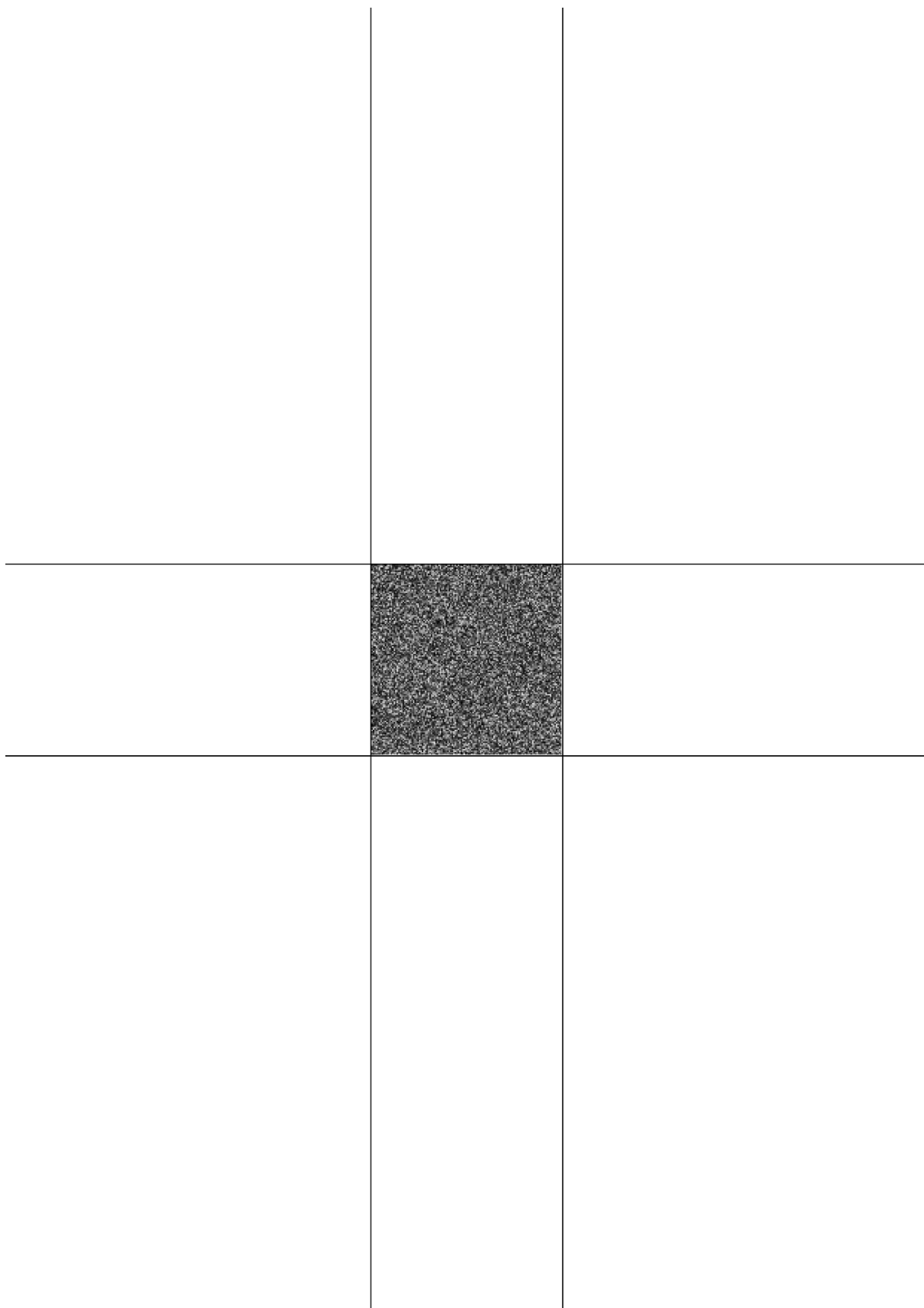


Classe 2	
Classe 3	

Cependant dès lors qu'une distorsion sous forme de perspective est induite, alors l'algorithme ne parvient pas à retrouver des droites suffisamment précises. La définition des droites "simples" de rebord ne fonctionne plus, il faut alors reposer entièrement sur les droites de Hough, pour détecter l'image et identifier la transformation appliquée à notre œuvre chiffrée.

Les premiers résultats ne sont pas satisfaisants, pour remédier à cela nous pouvons, premièrement soit changer de méthode de détection (1) (prendre une méthode simple avec détection de lignes plutôt qu'un pattern complexe), ou changer de détecteur de features et prendre un détecteur de type Canny / Sobel pour avoir les lignes entières de nos patterns.

(1) Potentielle nouvelle oeuvre







## Oeuvre 1

