RAPPORT VISION PAR ORDINATEUR TP1

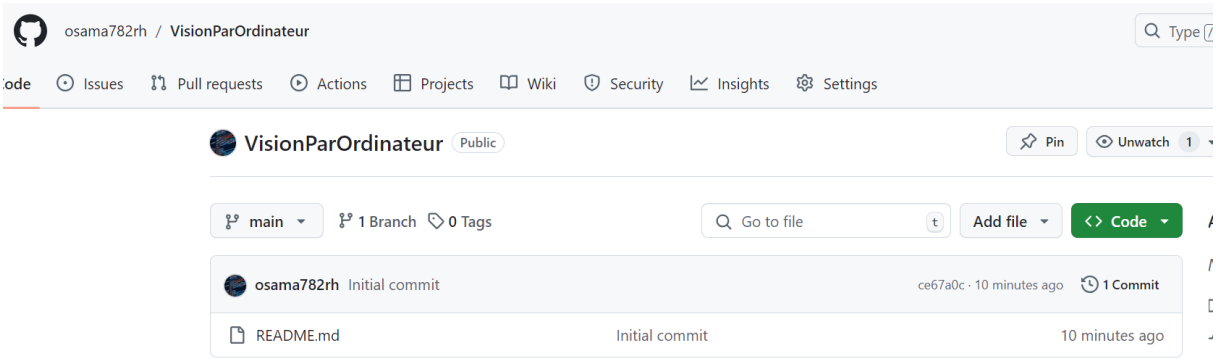
Prérequis :

Utilisation d'un menu pour afficher les réponses, exécuter le fichier main.py:

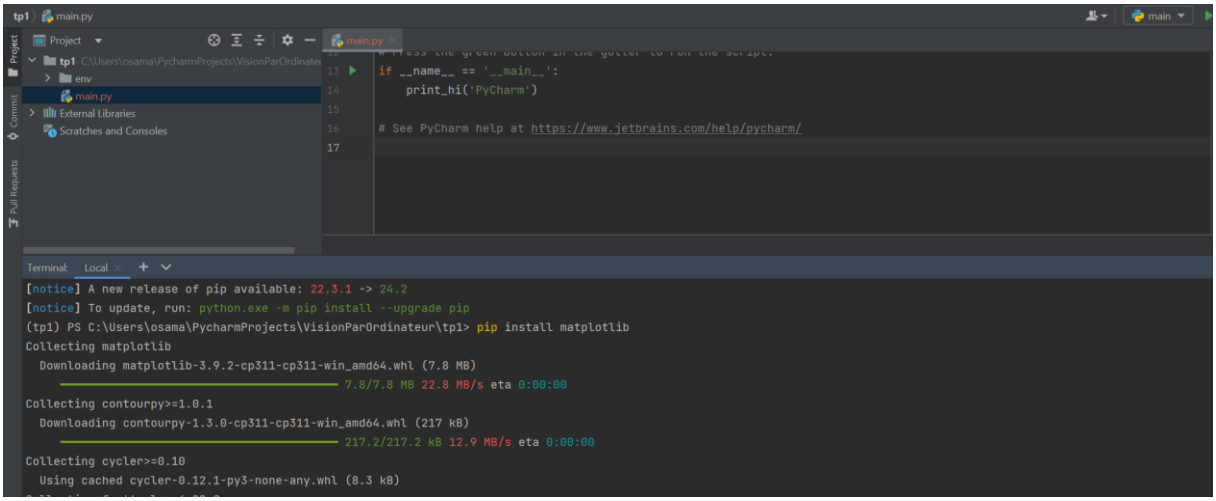
```
(env) PS C:\Users\osama\PycharmProjects\VisionParOrdinateur\tp1> python .\main.py
Sélectionnez une question du TP à exécuter :
1. Manipulation numpy (Question 7)
2. Manipulation matplotlib (Question 8)
3. Manipulation opencv (Question 9)
4. Seuillage (Partie B - Question 1)
5. Détection de contour (Partie B - Question 2)
6. Transformation par point (Partie B - Question 3)
7. Distance entre images (Partie B - Question 4)
8. Quitter
Entrez votre choix :
```

1,2) Création du répertoire dans GIT

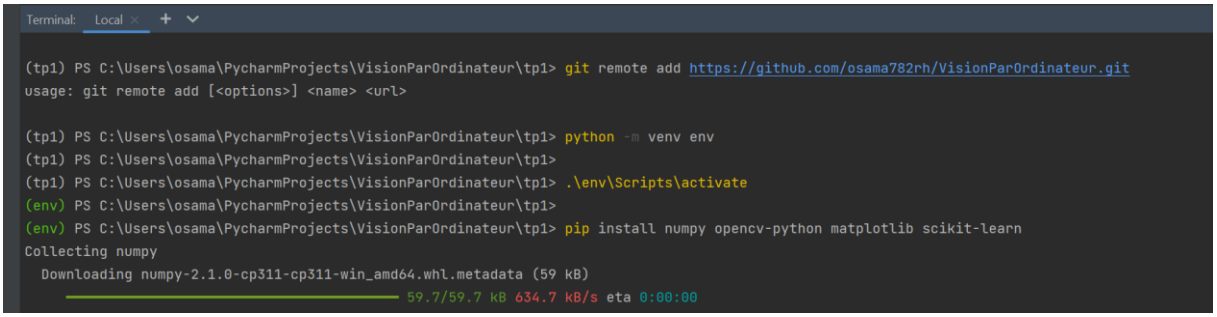
Repos : https://github.com/osama782rh/VisionParOrdinateur



3) Importation des différentes bibliothèques et création de l'espace de travail



4,5, 6)Création de l'environnement virtuel



7) Manipulation numpy

```
(env) PS C:\Users\osama\PycharmProjects\VisionParOrdinateur\tp1> python manipulation_numpy.py
Moyenne de X : 1.47
Écart type de X : 0.88
Médiane de X : 1.49
Moyenne de X_bis : 1.52
Écart type de X_bis : 0.88
Médiane de X_bis : 1.56

Comparaison des résultats :
Différence de Moyenne : 0.050000000000000044
Différence d'Écart type : 0.0
Différence de Médiane : 0.07000000000000006

Quelques valeurs de y :
[0.81387361 0.20273686 0.78820668 1.01149068 0.54251061 0.37076968
 0.32263922 0.48964942 0.97094546 0.77601004]
```

 Création de l'array X :

- Une array X de 1000 points a été créée avec des valeurs aléatoires dans l'intervalle [0,3][0, 3][0,3].

 Calcul de la moyenne, de l'écart type et de la médiane de X :

- Moyenne de X : **1.47**
- Écart type de X : **0.88**
- Médiane de X : **1.49**
- Les valeurs ont été arrondies au centième.

 Création de l'array X_bis :

- Une deuxième array X_bis de 1000 points a été créée avec des valeurs aléatoires dans l'intervalle [0,3][0, 3][0,3].

 Calcul de la moyenne, de l'écart type et de la médiane de X_bis :

- Moyenne de X_bis : **1.52**
- Écart type de X_bis : **0.88**
- Médiane de X_bis : **1.56**
- Les valeurs ont été arrondies au centième.

 Comparaison des résultats de X et X_bis :

- Différence de Moyenne : **0.05**
- Différence d'Écart type : **0.0**
- Différence de Médiane : **0.07**
- Les deux listes montrent des statistiques similaires, avec des différences minimes.

Fixation de l'aléa (seed) :

- L'aléa (seed) a été fixé avec la valeur 42 pour garantir la reproductibilité des résultats.
- Fixer la seed permet de reproduire les mêmes résultats lors de chaque exécution, ce qui est essentiel pour des expériences reproductibles en apprentissage machine.

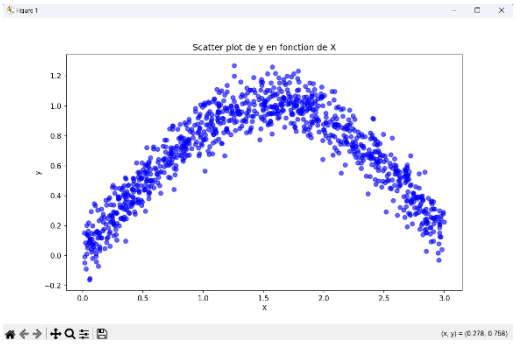
Utilité de la fixation de l'aléa (seed) :

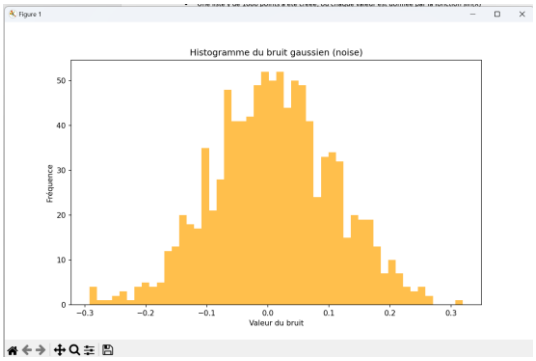
- Fixer l'aléa permet de garantir que les résultats aléatoires générés (comme les listes X et X_bis) restent constants d'une exécution à l'autre. Cela est particulièrement important en machine learning pour valider les modèles et comparer les résultats de manière fiable.

Création de la liste y :

- Une liste y de 1000 points a été créée, où chaque valeur est donnée par la fonction sin(X) avec l'ajout d'un bruit gaussien aléatoire ayant une amplitude de 10 % (0.1).
- Les premières valeurs de y sont : [0.8137361, 0.20273686, 0.78820618, 1.01149068, 0.54251061, 0.37076968, 0.32263922, 0.48964942, 0.97094546, 0.77601004].

8)





Visualisation de y en fonction de X sous forme de graph 'scatter' :

- Un graphique de type "scatter plot" a été créé pour visualiser y en fonction de X.
- La relation non linéaire entre X et y est bien représentée, avec une courbe de type sinus, perturbée par un bruit gaussien.

Changement de la taille de la figure :

- La taille de la figure a été modifiée à l'aide de la ligne de code suivante :

  plt.figure(figsize=(10, 6))

- Cette commande définit la taille de la figure à 10 pouces de large et 6 pouces de hauteur.

Visualisation du bruit gaussien, noise, sous forme d'histogramme :

- Le bruit gaussien noise a été visualisé sous forme d'histogramme avec un nombre de bins fixé à 50.

- L'histogramme montre que la distribution du bruit est symétrique et en forme de cloche, ce qui est typique d'une distribution normale.

Fonction à laquelle la distribution de noise fait penser :

- La distribution de noise rappelle une **distribution normale** (ou distribution gaussienne).
- Cette distribution est caractérisée par une courbe en cloche où la majorité des valeurs sont concentrées autour de la moyenne.

9)

Je lis l'image 'image_1.jpg' et je la mets dans la variable img. Quelle méthode est utilisée ?

- La méthode utilisée pour lire l'image et la stocker dans la variable img est cv2.imread(). Cette méthode permet de charger une image depuis un fichier en spécifiant son chemin, ici source/image_1.jpg.

Je visualise l'image dans le bon espace couleur, i.e., je vois la même chose que lorsque j'ouvre l'image JPG. Je ferme l'image (avec 'entrée' et non avec la croix de la fenêtre).

- L'image a été visualisée en utilisant la fonction cv2.imshow(). Cette fonction affiche l'image dans une fenêtre OpenCV. Pour fermer la fenêtre, la commande cv2.waitKey(0) a été utilisée, permettant à l'utilisateur de fermer la fenêtre en appuyant sur la touche "Entrée".

Afficher l'image en noir et blanc. Que faut-il ajouter à la ligne de code précédente ?

- Pour afficher l'image en noir et blanc, il est nécessaire d'ajouter la conversion de l'image en niveaux de gris avec la méthode cv2.cvtColor(img, cv2.COLOR_BGR2GRAY). Cette conversion change l'image de son espace de couleur BGR (Bleu, Vert, Rouge) en niveaux de gris.

Créer une méthode display_image qui prend en argument une image img et de façon optionnelle un nom name (str) de fenêtre et qui affiche l'image avec OpenCV. Cette méthode sera mise dans un fichier utils.py.

- La méthode display_image a été créée directement dans le même fichier manipulation_opencv.py. Cette méthode prend en argument l'image img et un nom de fenêtre name, puis utilise cv2.imshow() pour afficher l'image. La fenêtre reste ouverte jusqu'à ce que l'utilisateur appuie sur "Entrée".

Importer et appeler la méthode display_image précédemment créée pour afficher l'image image_1.jpg.

- La méthode display_image a été utilisée dans le même fichier pour afficher l'image image_1.jpg à la fois en couleur et en noir et blanc.

Partie B : Manipulation d'image

Dans cette partie, nous travaillons avec l'image image_1.jpg.

Chaque pixel d'une image est entre 0 et 255.

- **Quelle couleur est représentée par le 0 ?**
  - Le 0 représente la couleur **noire**.
- **Quelle couleur est représentée par le 255 ?**
  - Le 255 représente la couleur **blanche**.

---

1. Seuillage

Explication du seuillage manuel :

- Le seuillage manuel consiste à fixer un seuil spécifique. Tous les pixels dont l'intensité est inférieure au seuil sont convertis en noir (0), et ceux dont l'intensité est supérieure au seuil sont convertis en blanc (255). Cela permet de séparer les objets clairs des objets sombres dans une image.

Application du seuillage manuel :

- **Quels sont les seuils min et max qui optimisent la segmentation ?**
  - Cela dépend de l'image, mais après plusieurs tests, les seuils optimaux pour segmenter le chat blanc de cette image sont généralement autour de 200 pour le seuil minimum. Les pixels ayant une intensité supérieure à 200 seront considérés comme appartenant au chat blanc.
- **Enregistrer l'image en noir et blanc, l'intégrer au Readme.**

Application du seuillage d'OTSU :

- **Quels sont les seuils optimaux trouvés par ce seuillage pour chacun des canaux HSV de l'image ?**
    - Le seuillage d'OTSU trouve le seuil optimal en minimisant la variance intra-classe. Pour chaque canal HSV, OTSU calculera un seuil différent pour segmenter l'image de manière optimale.

Fusion des images avec les opérations logiques OR et AND :

- Les images résultantes des canaux HSV après le seuillage d'OTSU peuvent être fusionnées pour affiner la segmentation du chat blanc en utilisant les opérations cv2.bitwise_or et cv2.bitwise_and.

Optimisation de la segmentation du chat blanc :

- Une combinaison de plusieurs seuils et opérations logiques peut être utilisée pour affiner la segmentation du chat blanc. La version finale de l'image segmentée sera enregistrée et intégrée au Readme.

Comparaison des deux techniques de seuillage :

- Le seuillage manuel est simple mais nécessite une bonne intuition pour choisir le seuil.
- Le seuillage d'OTSU est plus automatique et calcule le seuil optimal pour une segmentation plus précise, bien qu'il puisse être moins flexible pour certaines images.

---

2. Détection de contour

Application du filtre de Sobel :

- **Détection des contours du clavier :**
    - Le filtre de Sobel est utilisé pour détecter les bords en calculant les dérivées en x et y de l'image. Les paramètres clés incluent dx et dy, qui déterminent l'ordre de dérivation dans les directions x et y respectivement, et ksize, qui spécifie la taille du noyau Sobel. Pour le clavier, dx=1, dy=0 pour détecter les contours verticaux, et dx=0, dy=1 pour les contours horizontaux. Une valeur optimale pour ksize est souvent 3 ou 5.
- **Détection des contours du chat blanc :**
    - Les mêmes paramètres que pour le clavier peuvent être utilisés, mais l'accent est mis sur la capture des contours doux du pelage du chat. Par exemple, dx=1 et dy=0 peuvent être utilisés pour détecter les contours verticaux du chat.

Application du filtre de Canny :

- **Détection des contours du clavier :**
    - Le filtre de Canny utilise deux seuils (threshold1 et threshold2) pour détecter les contours. Il est sensible aux variations d'intensité des pixels. Des valeurs typiques pour la détection des contours du clavier pourraient être threshold1=50 et threshold2=150.

- **Détection des contours du chat blanc :**
    - Des seuils plus faibles pourraient être utilisés pour capter les contours plus fins du pelage du chat, par exemple threshold1=30 et threshold2=100.

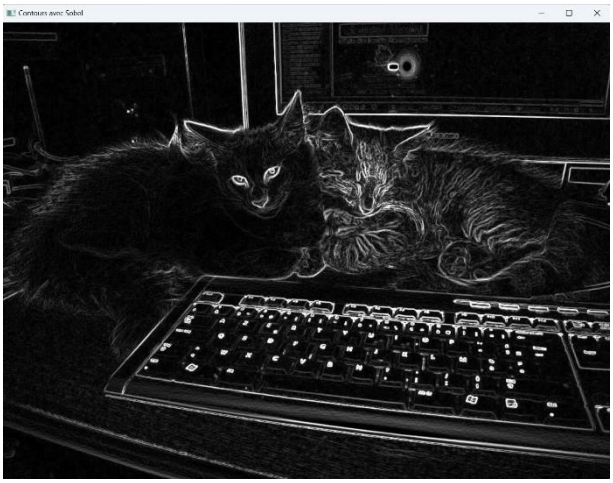
Comparaison des deux techniques de détection de contours :

- **Sobel :** Produit des contours plus doux et est utile pour détecter les changements d'intensité graduels.
- **Canny :** Est plus précis et peut détecter des contours plus nets, mais il est plus sensible au bruit.

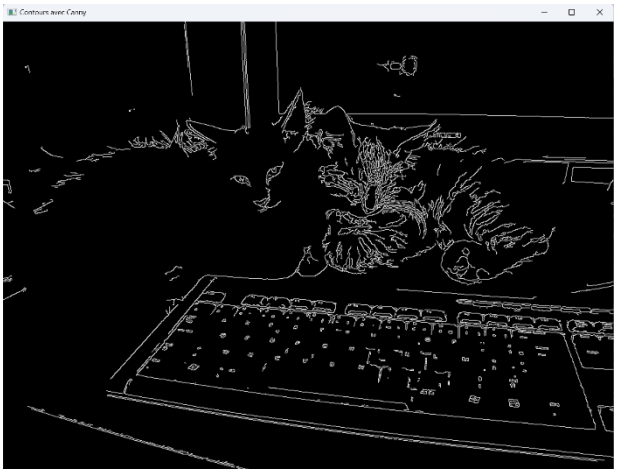
Seuillage manuel

Seuillage d'OTSU

Contours avec Sobel

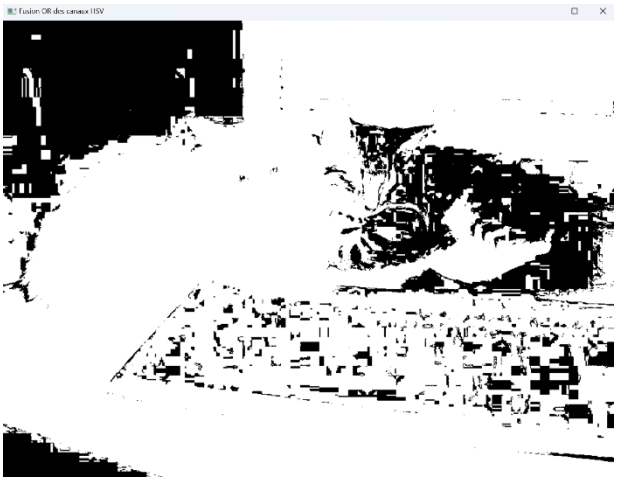


Contours avec Canny

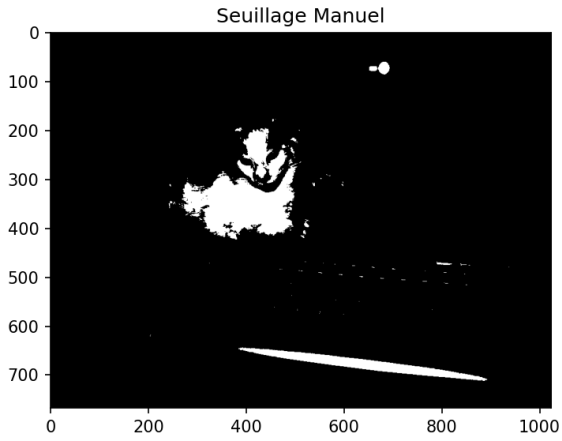


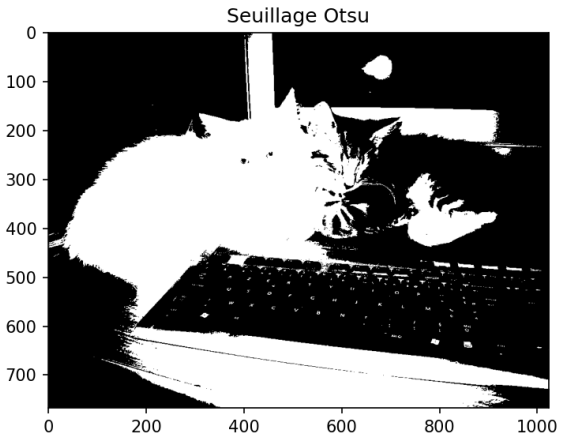
Vision par ordinateur TP1 : RAHIM Osama

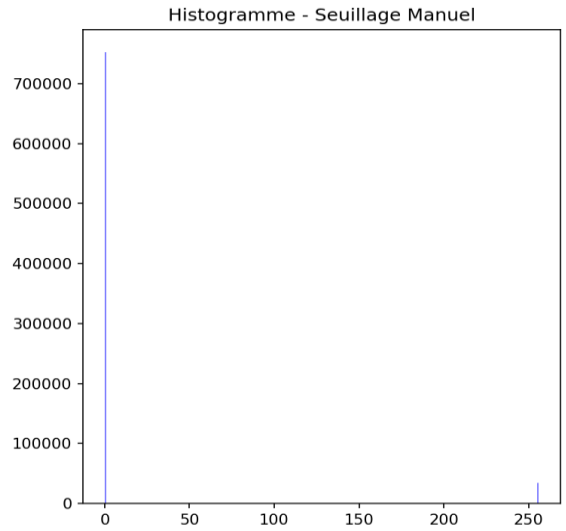
Fusion OR des canaux HSV

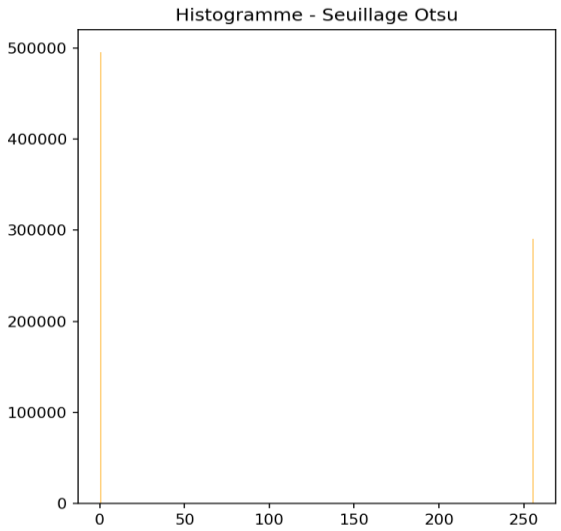


- Comparaison seuillage









Détection de Contour

1. Filtre de Sobel pour détecter les contours du clavier

Explication des paramètres :

- **ddepth (Profondeur de l'image de sortie)** : cv2.CV_64F est utilisé pour préserver les valeurs de dérivation avec des signes (positif/négatif).
- **dx (Ordre de dérivation en x)** : 1 détecte les changements d'intensité horizontaux, ce qui est crucial pour capturer les contours verticaux.
- **dy (Ordre de dérivation en y)** : 1 détecte les changements d'intensité verticaux, ce qui permet de capturer les contours horizontaux.
- **ksize (Taille du noyau)** : 3 est utilisé pour capturer des détails fins tout en lissant légèrement les variations.

Valeurs optimales pour le clavier :

- **dx = 1, dy = 0, ksize = 3** pour les contours verticaux.
- **dx = 0, dy = 1, ksize = 3** pour les contours horizontaux.

---

2. Filtre de Sobel pour détecter le contour du chat blanc

Valeurs optimales :

- Pour capturer les contours plus doux du pelage du chat, une taille de noyau légèrement plus grande peut être utilisée pour lisser les détails fins :
  - **dx = 1, dy = 0, ksize = 5** pour les contours verticaux.
  - **dx = 0, dy = 1, ksize = 5** pour les contours horizontaux.

---

3. Filtre de Canny pour détecter les contours du clavier

Explication des paramètres :

- **threshold1 (Seuil inférieur)** : Détermine les gradients considérés comme non contours. Les gradients plus faibles que ce seuil seront ignorés.
- **threshold2 (Seuil supérieur)** : Détermine les gradients considérés comme contours forts. Les gradients plus forts que ce seuil seront acceptés.
- **apertureSize (Taille du noyau Sobel)** : Détermine la précision du calcul du gradient utilisé pour la détection des contours.

Valeurs optimales pour le clavier :

- **threshold1 = 50**, **threshold2 = 150** sont souvent efficaces pour capturer les contours nets du clavier.

---

4. Filtre de Canny pour détecter le contour du chat blanc

Valeurs optimales :

- Pour le pelage du chat, des seuils plus faibles peuvent être utilisés pour capturer les contours plus doux :
    - **threshold1 = 30**, **threshold2 = 100** pour détecter les contours fins du pelage du chat.

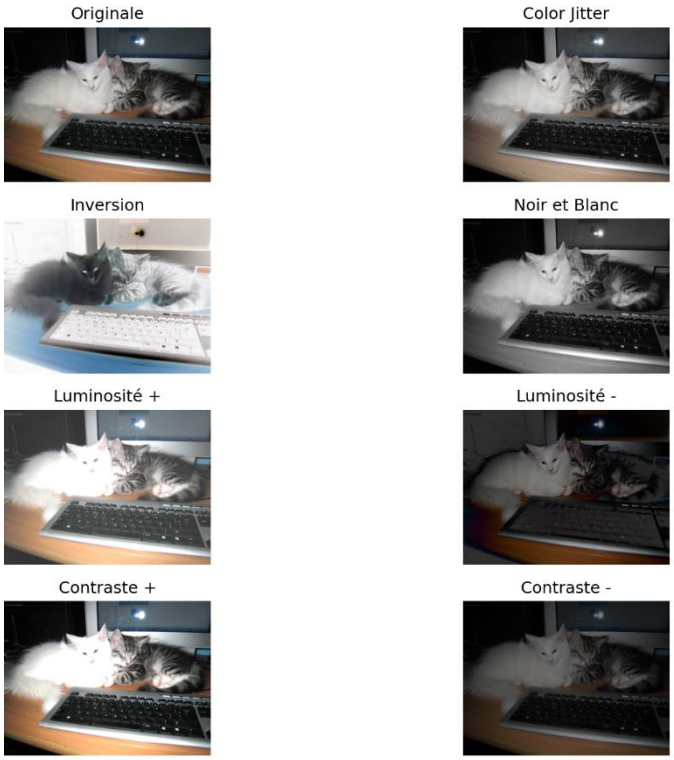
---

5. Comparaison des deux techniques de détection de contour pour le chat blanc

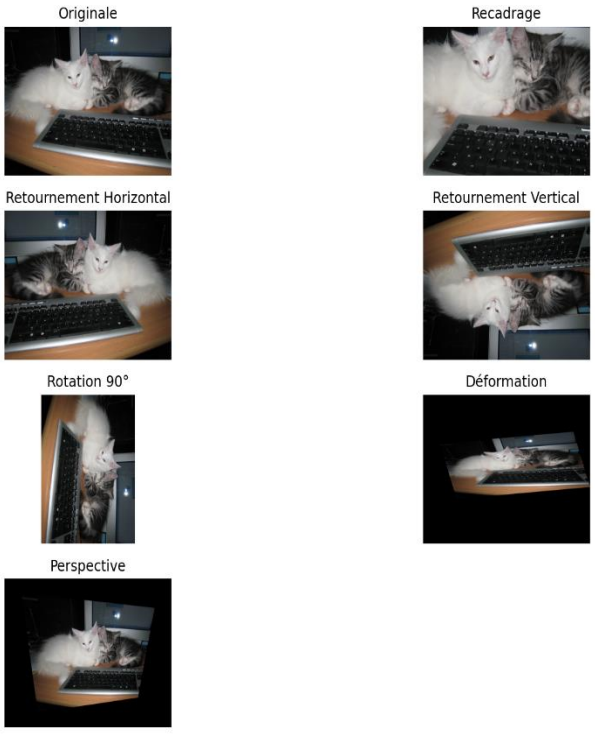
- **Sobel :** Le filtre de Sobel est efficace pour capturer les gradients doux et les contours non nets. Il est utile pour détecter les transitions graduelles, comme les contours du pelage du chat blanc.
- **Canny :** Le filtre de Canny est plus adapté pour des contours nets et distincts. Il détecte les bords précis et est souvent plus sensible aux variations rapides d'intensité, ce qui peut être plus ou moins adapté pour les contours flous du pelage du chat.

3) Transformation par point

Transformation valeurs



Transformation position



4) Distance entre images

```
(env) PS C:\Users\osama\PycharmProjects\VisionParOrdinateur\tp1> python .\main.py
La distance euclidienne entre les deux images est : 155113.8785634606
```