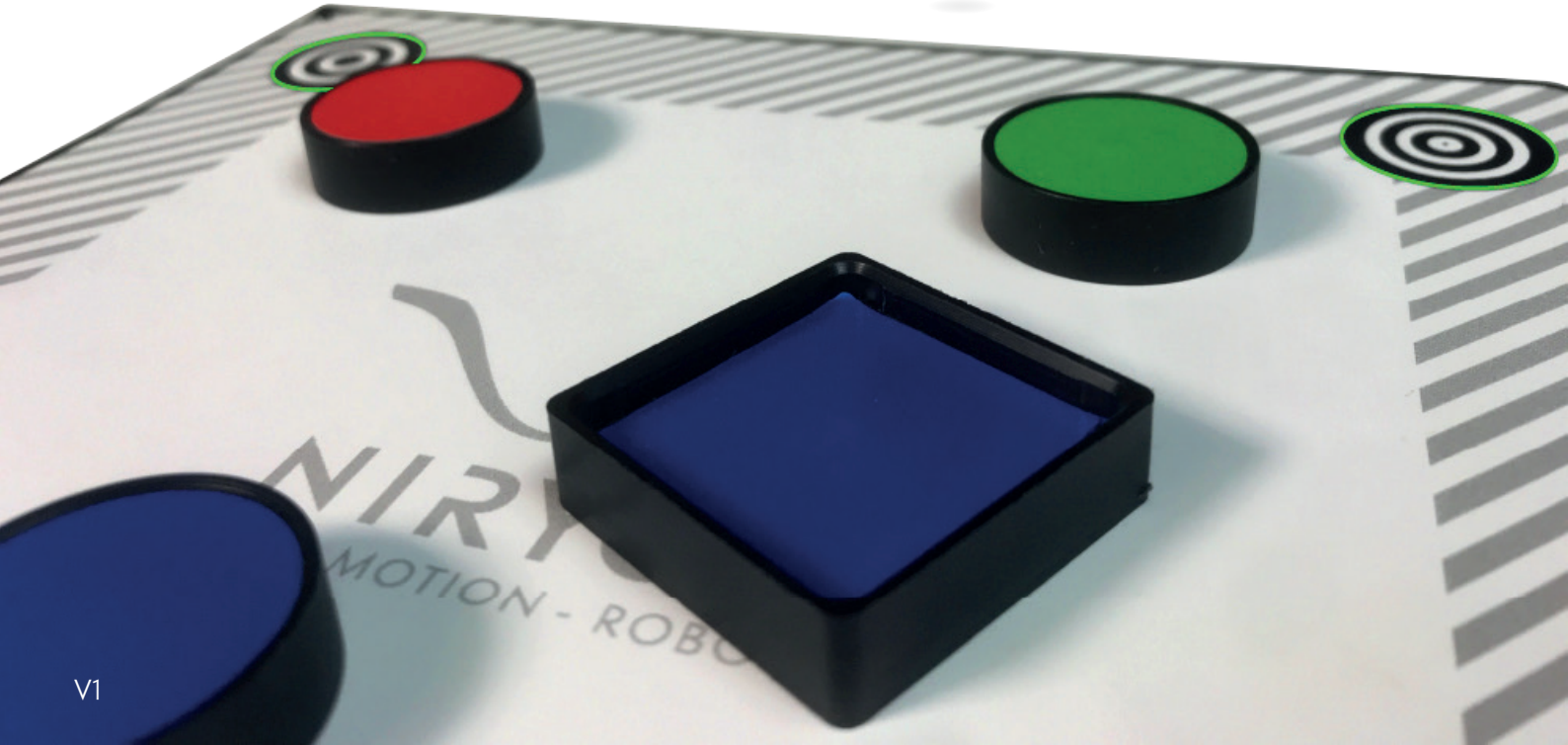
NIRYO

HUMAN - MOTION - ROBOT

VISION 2

Traitement d'images en python™



V1

OBJECTIFS

- Découvrir quelques procédés basiques de traitement d'images
- Apprendre à faire sa propre pipeline de traitement d'images pour contrôler le Niryo One

PRÉREQUIS

- Connaissances de base en Python
- Avoir pris connaissance de :
    - L'*API TCP Python*
    - Le *Manuel utilisateur* du Set Vision
    - Les fonctions disponibles pour le traitement d'images sur *Github*

CE DONT VOUS AUREZ BESOIN

- Niryo One Version ≥ 2.3 (Cette version est installée sur tous les Niryo one achetés après le 30 juin 2020). Le Niryo One peut être mis à jour ici.
- Niryo One équipé d'un Set Vision et attaché à un workspace
- Objets fournis avec le Set Vision (ou autres objets de couleur)

SETUP

Le setup de ce TP est le même que celui de la documentation TCP Python API. Il faudra cependant bien exécuter les étapes qui sont facultatives pour les utilisateurs n'ayant pas de Set Vision.

Avant de vous lancer dans ce TP, veillez à ce que votre robot soit fermement attaché à son workspace. Calibrez ensuite votre workspace via le Niryo One Studio.

Veillez toujours à ce que l'éclairage de votre sujet soit correct. En cas d'éclairage trop faible, ou trop important, les résultats obtenus peuvent être mauvais.

IMPLÉMENTATION D'UNE PIPELINE DE TRAITEMENT D'IMAGES

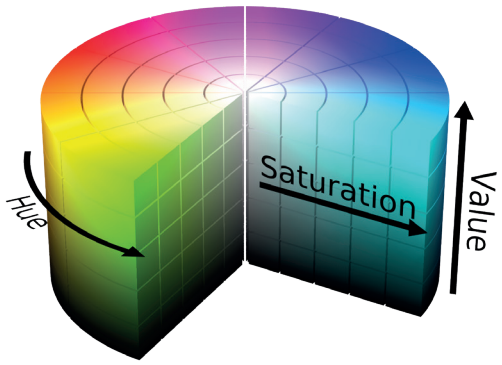
Le but de cette partie est d'extraire les informations de l'image afin d'obtenir la position d'un objet en termes de pixels.

SEUILLAGE COULEUR

Pour seuiller les couleurs, on se placera dans l'espace HSV.

Cet espace décompose une couleur en 3 valeurs :

- **Hue** : Cela correspond à la teinte mise en jeu ;
- **Saturation** : On parle de saturation pour décrire l'intensité d'une teinte. Ainsi, pour la valeur 0, on aura du gris alors que pour la valeur maximale (ici 255), la couleur sera vive ;
- **Value** : Cette valeur correspond à la luminosité. Plus la valeur est faible, plus la couleur est sombre, jusqu'à être noire.



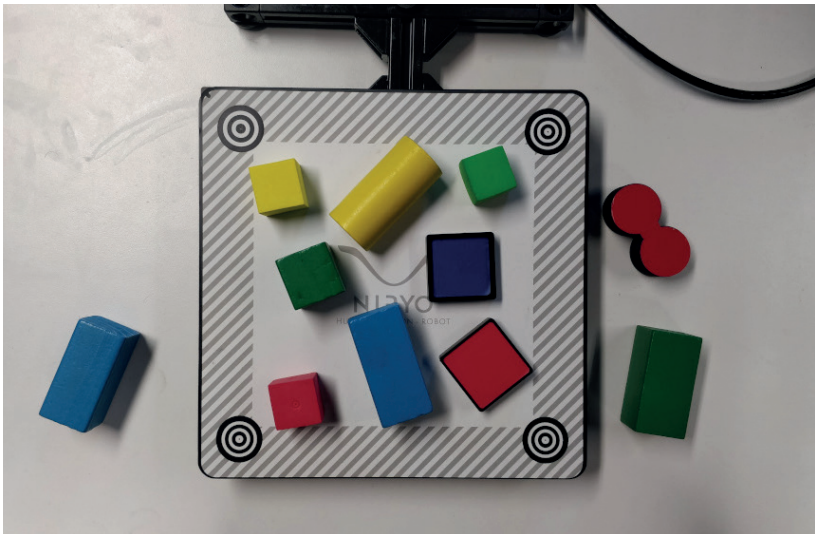
The HSV color model mapped to a cylinder. POV-Ray source is available from the POV-Ray Object Collection.

QUESTIONS THÉORIQUES

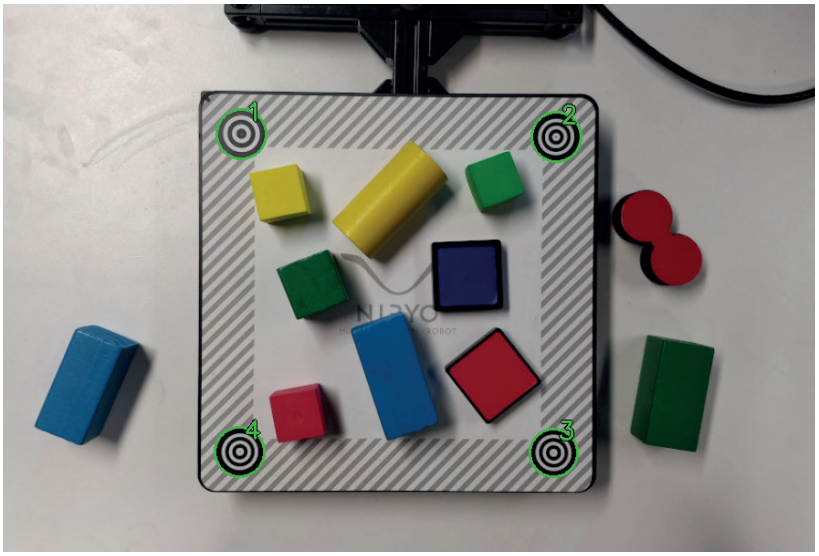
1. **Expliquer en quoi chacun de ces paramètres permettront d'isoler notre objet de prédilection.**
2. **Expliquer l'avantage du HSV par rapport au RGB, dans notre cas.**

APPLICATION

3. **Récupérer l'image compressée du stream vidéo via la fonction get_img_compressed de l'API TCP puis utiliser la fonction de traitement d'images uncompress_image pour décompresser l'image. Vous pouvez afficher cette image avec la fonction show_and_wait_close.**



4. **Le Niryo One utilise des marqueurs pour se repérer dans l'espace. Effectuer l'extraction de l'image délimitée par ces marqueurs via la fonction extract_img_markers.**



5. **Observer le résultat et décrire la pipeline de cette fonction.**

Cette image sera l'image qu'on utilisera dans les traitements suivants.

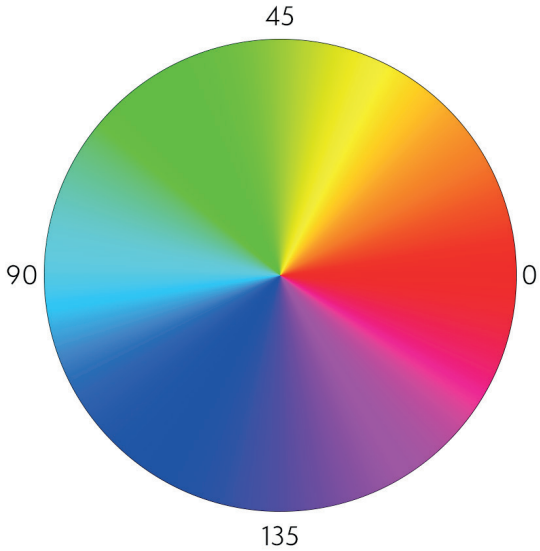
6. **Utiliser ensuite la fonction threshold_hsv pour effectuer un seuillage de couleur et obtenir une image binaire. Tester plusieurs valeurs et afficher les résultats.**
7. **Trouver les paramètres pour isoler les objets de couleur bleue. Faire de même pour le vert.**

Exemple :

```
low_thresh_list, high_thresh_list = [12, 115, 75], [255, 120, 90]

img_thresh = threshold_hsv(im_work,low_thresh_list,high_thresh_list)
```

Dans l'espace de couleur HSV, la Hue est souvent représentée par un cercle.



Valeurs Hue

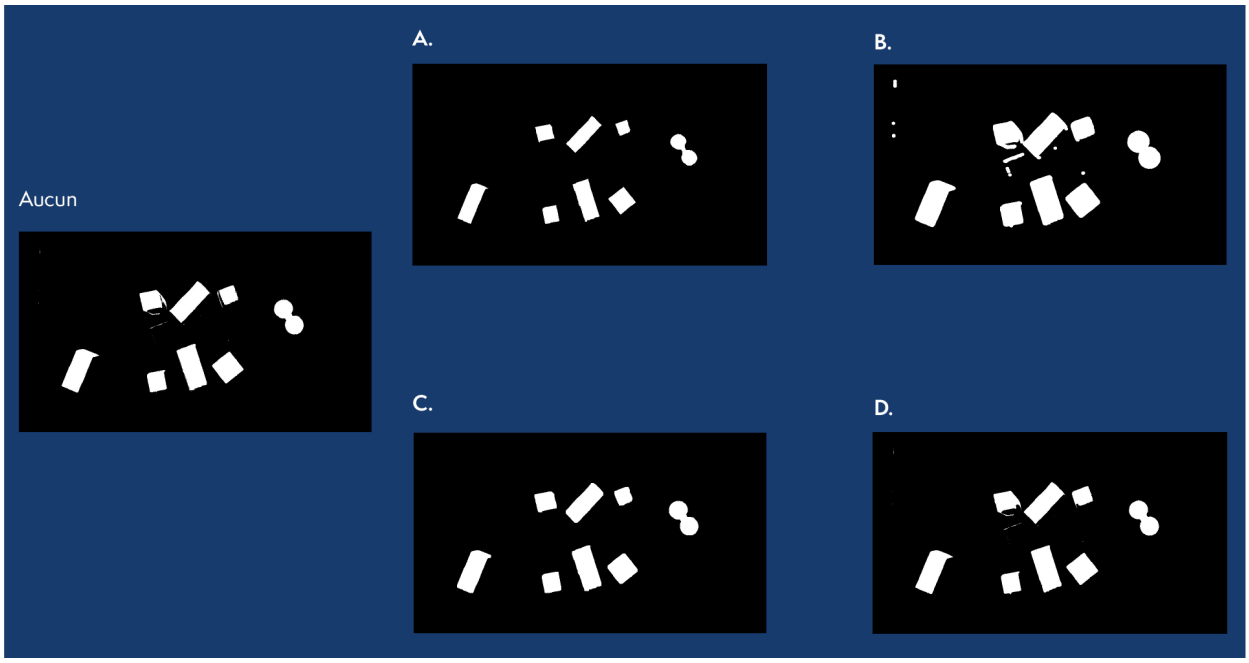
En passant à un vecteur de valeur [0,179], on perd la continuité et ainsi, le rouge se retrouve pour les valeurs maximales et minimales. La fonction que vous utilisez étant déjà écrite, vous ne pouvez pas changer les conditions de seuillage.

8. **Trouver une méthode d'image pour avoir une image comportant tous les rouges.**

Pour la suite, on pourra utiliser la fonction threshold_hsv avec le paramètre invert_hue=True afin d'extraire le rouge.

OPÉRATION MORPHOLOGIQUES

Les opérations morphologiques sont des procédés simples permettant de transformer une image généralement binaire.



QUESTIONS THÉORIQUES

1. **Nommez les quatre transformations (A, B, C, D) ci-dessus.**
2. **Dans notre cas de figure, en quoi les opérations morphologiques sont-elles utiles ? Laquelle / lesquelles utiliser pour améliorer notre résultat ?**

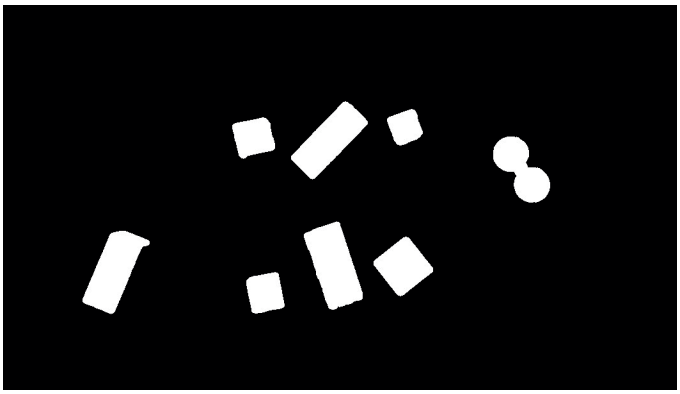
APPLICATION

3. **Ajouter une opération morphologique à votre pipeline de traitement. Pour cela, utiliser la fonction de traitement d'images : morphological_transformations.**

Exemple :

```
im_morpho = morphological_transformations(img_thresh, morpho_type="OPEN")
```

4. **Appliquer les opérations morphologiques qui vous semblent pertinentes.**



L'image ci-dessus est l'image que vous êtes censés obtenir. Toutefois, plusieurs solutions sont possibles. Le résultat peut donc être différent de celui du dessus.

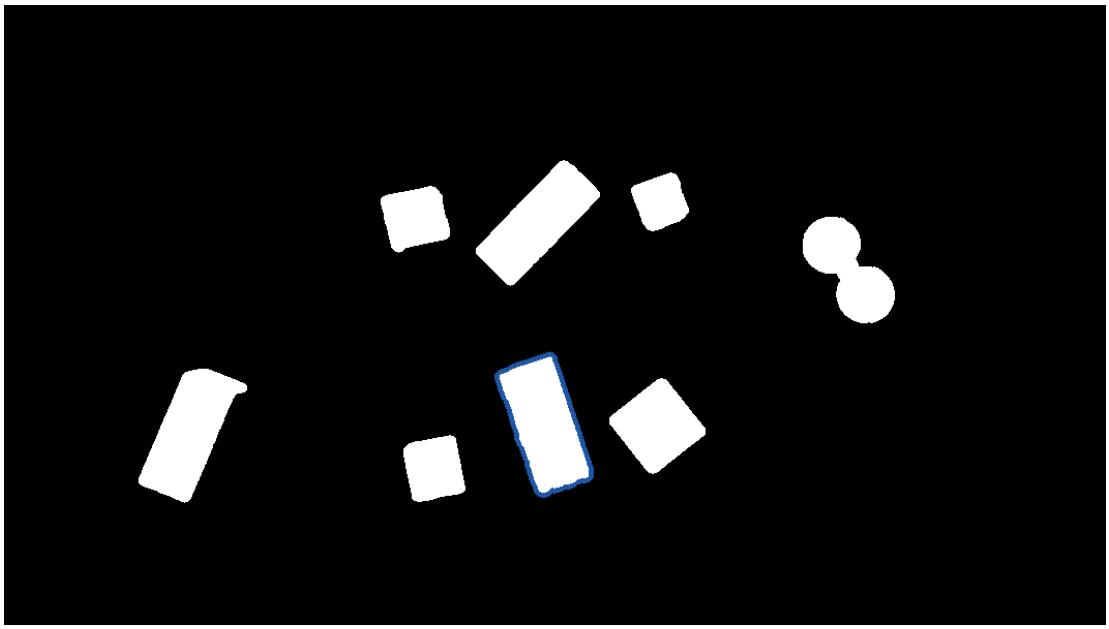
DÉTECTION D'OBJETS

QUESTIONS THÉORIQUES

1. **Comment pourriez-vous isoler les différents objets de l'image initiale grâce à la pipeline de traitement qui vient d'être mise en place ?**
2. **Une fois que ces objets sont détectés, comment obtenir le centre de masse de l'objet ? Comment obtenir son angle de rotation ?**

APPLICATION

3. **Utiliser la fonction biggest_contour_finder pour trouver le plus gros contour dans votre image binaire. Cette fonction renverra une liste de points correspondant au contour. Vous pouvez le dessiner/afficher via la fonction Draw Contours.**



4. **Calculer le barycentre du contour. On cherchera à avoir une position relative par rapport aux bordures de la zone de travail. Vous pouvez utiliser la fonction get_contour_barycenter pour trouver le centre et la fonction relative_pos_from_pixels pour traduire ce centre en termes de position relative.**
5. **Calculer l'angle d'inclinaison du contour. On cherchera à avoir un angle en radians dans le repère trigonométrique. Vous pouvez utiliser la fonction get_contour_angle.**

VALIDATION

QUESTIONS THÉORIQUES

1. **Expliquer comment le robot interprète les coordonnées relatives calculées pour se placer dans l'espace.**
2. **Connaissant les paramètres intrinsèques de la caméra, quelle méthode aurait pu être utilisée pour calculer la position de l'objet par rapport à la caméra ? Et par rapport au robot ?**

APPLICATION

3. **Utiliser la fonction get_target_pose_from_rel pour obtenir la position de l'objet par rapport au robot.**

Exemple :

Si vous souhaitez prendre un objet se situant à 20% selon X, 60% selon Y et un angle de Pi/2, on utilisera :

```
status, obj_pose = niryo_one.get_target_pose_from_rel("workspace_1", 0.0, 0.2, 0.6,
1.57)
```

4. **A partir de cette position, récupérer l'objet se trouvant dans la zone de travail et le ranger à l'extérieur de cette zone.**

Crédits :

- The HSV color model mapped to a cylinder. POV-Ray source is available from the POV-Ray Object Collection - SharkD

IMPLÉMENTATION D'UNE PIPELINE DE TRAITEMENT D'IMAGES - CORRECTION

Le but de cette partie est d'extraire les informations de l'image afin d'obtenir la position d'un objet en termes de pixels.

SEUILLAGE COULEUR

QUESTIONS THÉORIQUES

1. **Expliquer en quoi chacun de ces paramètres permettront d'isoler notre objet de prédilection.**

    - **La Hue permettra de se concentrer sur la couleur (Bleu, Rouge, Vert, …) ;**
    - **La Saturation permettra de supprimer les zones de couleur non intenses (zones grises, blanches, noires) ;**
    - **La Value permettra de supprimer les zones sombres.**

2. **Expliquer l'avantage du HSV par rapport au RGB, dans notre cas.**

Le HSV permet d'isoler une teinte de manière beaucoup plus facile et instinctive que le RGB. En effet, avec le RGB, on ne pourra pas isoler le RGB uniquement avec des valeurs seuils type (200,0,0). En effet, cela correspondrait à un rouge mais très sombre. Pour avoir des couleurs plus claires, il faudra monter les valeurs des channels bleue et verte, mais si elles sont trop hautes, on se retrouvera avec des couleurs grises. Ainsi, il faudrait mettre des valeurs bleue et verte en fonction du Rouge.

APPLICATION

Début du code :

```
#!/usr/bin/env python


# Imports
from niryo_one_tcp_client import *
from niryo_one_camera import *


# Connecting to robot
niryo_one_client = NiryoOneClient()
```

```
niryo_one_client.connect("10.10.10.10")  # =< Replace by robot ip address
```

1. **Récupérer l'image compressée du stream vidéo et affichez-la avec les fonctions get_img_compressed(), uncompress_image(), undistort_image().**

```
# Looping Forever

while True:

    status, img_compressed = niryo_one_client.get_img_compressed()

    if status is not True:

        print("error with Niryo One's service")

        break

    img = uncompress_image(img_compressed)

    key1 = show_img("Image", img, wait_ms=1)
```

2. **Le Niryo One utilise des marqueurs pour se repérer dans l'espace. Effectuez l'extraction de l'image délimitée par ces marqueurs via la fonction extract_img_markers.**

```
im_work = extract_img_markers(img,workspace_ratio=1.0)

if im_work is None:

   print ("Unable to find markers")

   continue
```

3. **Observer le résultat et décrire la pipeline de cette fonction.**

Cette image sera l'image qu'on utilisera dans les traitements suivants.

Pour extraire la zone de marqueurs, la pipeline est la suivante :

- **Détection des marqueurs ;**
- **Identification (Le marqueur 1 est différent des autres + les marqueurs sont ordonnés selon le sens horaire) ;**
- **Extraction de la zone et transformation en une image rectangulaire via warping.**

4. **Utiliser ensuite la fonction threshold_hsv pour effectuer un seuillage de couleur et obtenir une image binaire. Tester plusieurs valeurs et afficher les résultats.**

```
low_thresh_list, high_thresh_list = [90, 115, 75], [115, 255, 255]
```

```
img_thresh = threshold_hsv(im_work,low_thresh_list,high_thresh_list)

key2 = show_img("Thresh", img_thresh, wait_ms=1)
```

5. **Trouver les paramètres pour isoler les objets de couleur bleue. Faire de même pour le vert.**

À l'aide de la fonction calib_hsv, utiliser des curseurs pour trouver des valeurs optimales plutôt que de relancer le programme à chaque fois.

Les résultats varient en fonction de votre éclairage. Voici néanmoins les valeurs de base pour le Niryo One :

```
BLUE = [90, 50, 85], [125, 255, 255]
```

```
GREEN = [40, 60, 75], [85, 255, 255]
```

6. **Dans l'espace de couleur HSV, la Hue est souvent représenté par un cercle. En passant à un vecteur de valeur [0,179], on perd la continuité et ainsi, le rouge se retrouve pour les valeurs maximales et minimales. La fonction que vous utilisez étant déjà écrite, vous ne pouvez pas changer les conditions de seuillage. Trouvez alors une méthode d'image pour avoir une image comportant tous les rouges.**

L'idée est de faire 2 seuillages et d'additionner les 2 images obtenus afin d'avoir une image comportant l'intégralité des rouges.

Soit A et B deux entiers tels que 0 < A < B < 179. On peut faire deux seuillages :

- **1 pour une Hue valant [0,A] permettant d'avoir les rouges de la partie inférieure du vecteur de Hue ;**
- **1 pour une Hue valant [B,179] permettant d'avoir les rouges de la partie supérieure du vecteur de Hue.**

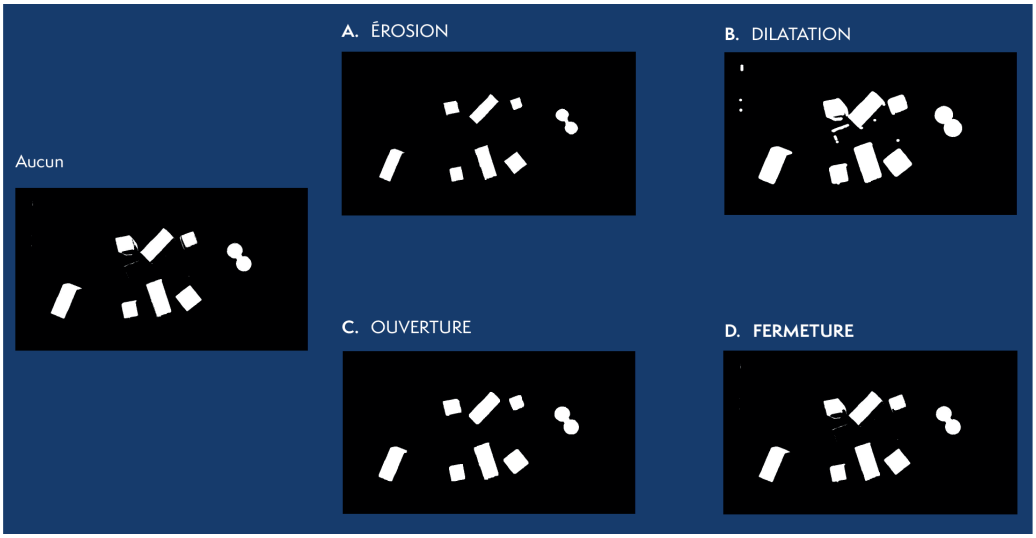
Ensuite, en additionnant les images, on obtiendra le résultat attendu (ici, l'addition se comporte comme un "ou logique").

OPÉRATION MORPHOLOGIQUES

Les opérations morphologiques sont des procédés simples permettant de transformer une image généralement binaire.

QUESTIONS THÉORIQUES

1. **Nommez les quatre transformations ci-dessus.**



2. **Dans notre cas de figure, en quoi les opérations morphologiques sont-elles utiles ? Laquelle / lesquelles utiliser pour améliorer notre résultat ?**

On peut avoir du bruit à cause de l'éclairage. On effectue une ouverture pour supprimer ce bruit. Ensuite, on peut penser à une fermeture pour que les zones d'intérêts de l'image binaire soient bien uniformes.

APPLICATION

3. **Ajouter une opération morphologique à votre pipeline de traitement.**
4. **Appliquer les opérations morphologiques qui vous semblent pertinentes.**

```
im_open = morphological_transformations(img_thresh, morpho_type="OPEN")

im_close = morphological_transformations(im_open, morpho_type="CLOSE")

key3 = show_img("Morpho", im_close, wait_ms=1)
```

DÉTECTION D'OBJETS

QUESTIONS THÉORIQUES

1. **Comment pourriez-vous isoler les différents objets de l'image initiale grâce à la pipeline de traitement qui vient d'être mise en place ?**

Il faut effectuer une détection de zones uniformes sur l'image binaire. Ainsi, en détectant les contours de ces zones, on pourra déduire où sont les objets d'intérêt.

2. **Une fois que ces objets sont détectés, comment obtenir le centre de masse de l'objet ? Comment obtenir son angle de rotation ?**

Pour obtenir le centre de masse, on peut moyenner la position des points de l'ensemble de la zone binaire.

Concernant l'angle, il existe de nombreux moyens. L'un est d'utiliser les moments de l'image : https://en.wikipedia.org/wiki/Image_moment .

APPLICATION

1. **Utiliser la fonction biggest_contour_finder pour trouver le plus gros contour dans votre image binaire. Cette fonction renverra une liste de points correspondant au contour. Vous pouvez le dessiner/afficher via la fonction Draw Contours.**

```
best_contour = biggest_contour_finder(im_close)
if best_contour is None or len(best_contour) == 0:
    print("No blob found")
    continue
img_draw = draw_contours(im_close, [best_contour])
```

2. **Calculer le barycentre du contour. On cherchera à avoir une position relative par rapport aux bordures de la zone de travail. Vous pouvez utiliser la fonction get_contour_barycenter pour trouver le centre et la fonction relative_pos_from_pixels pour traduire ce centre en termes de position relative.**

```
cx, cy = get_contour_barycenter(best_contour)
height, width = img_thresh.shape
x_rel = float(cx) / width
y_rel = float(cy) / height
```

3. **Calculer l'angle d'inclinaison du contour. On cherchera à avoir un angle en radians dans le repère trigonométrique. Vous pouvez utiliser la fonction get_contour_angle.**

```
angle = get_contour_angle(best_contour)
```

VALIDATION

QUESTIONS THÉORIQUES

1. **Expliquer comment le robot interprète les coordonnées relatives calculées pour se placer dans**

l'espace.

Le robot connaissant la position des quatres coins de l'espace de travail, il peut modéliser cet espace de travail en un plan en 3 dimensions. Une fois cela fait, il est capable de déduire la position d'un objet en interprétant ses coordonnées relatives et en les plaçant dans le plan.

2. **Connaissant les paramètres intrinsèques de la caméra, quelle méthode aurait pu être utilisée pour calculer la position de l'objet par rapport à la caméra? Et par rapport au robot ?**

A l'aide des paramètres intrinsèques de la caméra ainsi que la hauteur de la caméra par rapport au plan de travail, on aurait pu déduire la position [x,y] de l'objet dans le plan de vision. Ensuite, à l'aide de transformations cinématiques, on aurait pu passer du repère de la caméra à celui de l'outil.

APPLICATION

1. **Utiliser la fonction get_target_pose_from_rel pour obtenir la position de l'objet par rapport au robot.**

```
ws_name = "markers_1"

height_offset = 0.5

status, obj_pose = niryo_one_client.get_target_pose_from_rel(ws_name, height_offset, x_rel, y_rel, angle)
```

2. **A partir de cette position, récupérer l'objet se trouvant dans la zone de travail et le ranger à l'extérieur de cette zone.**

```
place_pose = PoseObject(
    x=0.0, y=0.20, z=0.3,
    roll=0.0, pitch=1.57, yaw=1.57,
)
obj_pose_high = obj_pose.copy_with_offsets(z_offset=0.05)


# Opening gripper and going to object
tool_used = RobotTool.GRIPPER_2
niryo_one_client.open_gripper(tool_used, 600)
niro one_client.move_pose(*niryo_one_client.pose_to_list(obj_pose_high))
niryo_one_client.move_pose(*niryo_one_client.pose_to_list(obj_pose))
```

```
    # Getting object
    niryo_one_client.close_gripper(tool_used, 300)
    # Leaving and going to place spot
    niryo_one_client.move_pose(*niryo_one_client.pose_to_list(obj_pose_high))
    niryo_one_client.move_pose(*niryo_one_client.pose_to_list(place_pose))
    niryo_one_client.open_gripper(tool_used, 600)
```