Promotion 2020/2023

MERKLE Théo

SCHAUER Alexis

FIP MIK5

Projet innovation et recherche FIP MIK 5 S9

Titre du projet : Smart Robot

Table des matières

1) Objectif

Le but du projet est de contrôler un robot par commandes vocales. Le processus se traduit comme suit :

1) Plusieurs pièces ou objets sont posés sur une table, dans le champ de vision du robot.
2) L'utilisateur donne vocalement « l'appellation » d'un objet, c'est-à-dire une forme et une couleur dans le cas des jetons (précisions en section 2.4).
3) Le robot scanne l'environnement et détecte l'objet demandé par l'intermédiaire d'une reconnaisse d'images.
4) Le robot localise l'objet à un point A dans l'espace, le saisit puis le dépose à un point B donné

2) Moyens mis à disposition

a) Le robot Niryo Ned 2

i) Le robot

Le Niryo Ned 2 est un robot collaboratif six axes destinés à l'éducation et à la recherche. Ce robot a été conçu par la société française Niryo basée à Lille. Le robot est principalement constitué d'aluminium et de plastique imprimé en 3D. Ned2 est basé sur Ubuntu 18.04 et ROS Mélodic. Il est également équipé d'un Raspberry PI 4. Il pèse 7 kg et peut porter jusqu'à 300 grammes. Sa répétabilité est de ± 0.5 mm.

Le robot est commandable soit par l'intermédiaire d'une connexion wifi, soit par réseau Ethernet. L'adresse IP associé est alors à renseigner au moment de la connexion.

ii) Les préhenseurs

Le robot est livré avec deux préhenseurs différents :

- Le premier est un gripper électrique classique qui permet d'attraper les objets en exerçant une pression sur deux points opposés.

- Le second préhenseur est une pompe à vide permettant d'attraper les objets à surface plane et non-poreuse, à l'aide de sa ventouse. Elle est reliée à une pompe à vide qui transmet l'aspiration d'air.

Figure 1 : Préhenseur de type pince et préhenseur de type pompe à vide

Pour la préhension des jetons du kit, nous avons remarqué que le système à ventouses est plus fiable. En effet, la surface de contact des pinces avec l'objet est relativement molle et ne permet pas une préhension fiable pour ce type d'objet.

iii) Le kit de vision

Figure 2 : Caméra du Set Vison vissé sur le robot

Le Set Vision est un module fourni par Niryo qui permet d'ajouter la reconnaissance d'image au robot. Ce set contient une caméra qui peut se visser sur le cinquième axe du robot, un plan de travail et une pointe pour la calibration pour les réglages. Au début du projet, nous avons défini un espace de travail et appliqué la méthode de calibration énoncée. Cela permet par la suite d'avoir une vue fiable des distances.

Des blocs préprogrammés permettent ensuite de récupérer la vision de la caméra, mais aussi d'utiliser la reconnaissance d'image.

iv) Les objets standards du kit

Ces objets sont réalisés en matière plastique et pèse quelques dizaines de grammes. Deux formes d'objets sont fournies, des cercles et des carrés. Chacun est disponible en 3 couleurs différentes, soit le rouge, le vert et le bleu.

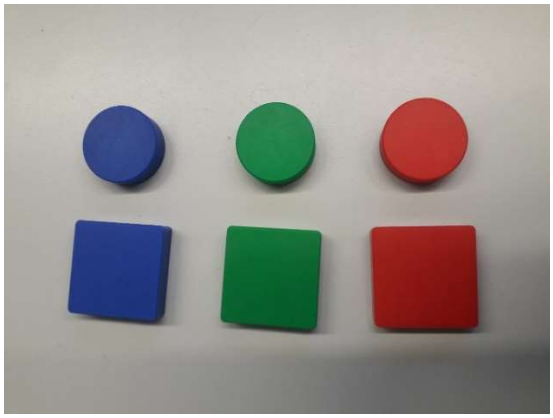

Figure 3: Les différents objets à attraper

v) Le logiciel Niryo Studio

Le robot peut être piloter soit manuellement en mode apprentissage soit à l'aide d'un logiciel fourni nommé « Niryo Studio ».

Ce logiciel permet de programmer facilement et sans connaissance en programmation dans le langage « Blockly ». Ce langage proche du scratch est constitué de blocs préprogrammés qui sont à assembler les uns avec les autres

Le Python peut aussi être utilisé. Nous avons choisi de programmer en Python car il offre davantage de souplesse et de possibilités pour notre application. Le package PyNiryo 2 basé sur la bibliothèque roslibpy.

Il est également possible de piloter le robot via une télécommande virtuel. Le robot peut être bouger soit avec des commandes d'angles d'axes (mode Joint), soit selon un repère x, y, z. Lors de son mouvement, le jumeau numérique visible dans l'interface du logiciel permet de visualiser les mouvements du robot. La vitesse de déplacement et les paramètres de l'outil peuvent être directement modifiés.

Dans l'onglet réservé au Set Vison dans le logiciel, on peut visualiser le rendu de la caméra et régler quelques paramètres liés à celle-ci, ce qui est pratique lors de l'utilisation à distance du robot.

b) Le microphone/haut-parleur

Le microphone et le haut-parleur à la disposition pour ce projet est un Jabra Speak 410 Haut-parleur de la marque Jabra. Cette enceinte se branche en USB sur un ordinateur et permet de fonctionner à la fois comme un hautparleur et à la fois comme un microphone. Le microphone a la particularité d'être omnidirectionnel.

Figure 4: Le Jabra microphone/haut-parleur

c) La Jetson Nano Developer kit

La Jetson Nano Développer Kit est un mini-ordinateur permettant le déploiement de l'intelligence artificiel sur une carte à faible coût et a basse consommation.

En termes de spécifications techniques, la carte est équipée d'un GPU architecture NVIDIA Maxwell™ avec 128 cœurs NVIDIA CUDA® et un CPU quad-core ARM® Cortex®-A57 MPCore. La carte a une mémoire vive de 4Go 64-bit LPDDR4.

Figure 5 : La Jetson Nano Developer Kit

d) Les bibliothèques Python

Pour le besoin de notre projet, différentes bibliothèques ont été nécessaires en plus de celle déjà disponibles lors de la création de l'image. Sur la Jetson Nano, nous avons dans un premier temps construit un container avec les bibliothèques de base (voir figure). Puis, au fur et à mesure de nos essais, nous avons ajouté de nouvelles bibliothèques pour répondre à nos besoins spécifiques au projet. Finalement, nous avons pu réaliser une nouvelle image complète, qui contient donc toutes les bibliothèques nécessaires à la compilation du code.

Installed libraries:
PyTorch v1.9.0 (base image)
torchvision v0.10.0 (base image)
torchaudio v0.9.0 (base image)
onnx 1.8.0 (base image)
CuPy 9.2.0 (base image)
numpy 1.19.5 (base image)
numba 0.53.1 (base image)
OpenCV 4.5.0 (base image)
pandas 1.1.5 (base image)
scipy 1.5.4 (base image)
scikit-learn 0.23.2 (base image)
JupyterLab 2.2.9 (base image)
TensorFlow 2.7.0
Jetson.GPIO 2.0.17
pyaudio 0.2.11
simpleaudio 1.0.4
librosa 0.8.1
transformers 4.15.0
netron 6.0.9
torch2trt 0.3.0
trt-pose 0.0.1
jetcam 0.0.0
jetson-inference

Figure 6 : Liste des bibliothèques de base

Ci-après un descriptif des bibliothèques spécifiques que nous avons utilisées :

PyNiryo est une bibliothèque développée par la marque de robot Niryo et qui permet de contrôler le robot, le convoyeur ainsi que le set de vision. La bibliothèque permet de contrôler de manière simple et complet à travers un script python.

FLAC (Free Lossless Audio Codec) est un package. Le format FLAC est similaire au mp3 mais avec un meilleur taux de compression et sans perte en qualité. Le FLAC est le plus rapide et le plus largement supporté des codecs.

SpeechRecognition est une bibliothèque qui permet de réaliser de la reconnaissance vocale à l'aide des moteurs ou des API tels que google ou sphinx.

CMU Sphinx est une bibliothèque qui regroupe notamment la bibliothèque pocketsphinx utilisé dans ce projet. Elle permet de réaliser une reconnaissance vocale sans avoir besoin d'une connexion internet.

e) Répertoire Github

Dans le cadre de ce projet, nous avons à notre disposition un répertoire Github faisant office d'archive de nos travaux. D'une part, il contient, il contient toutes les spécifications en termes de librairies additionnelles à importer sur le container pour la compilation du programme. D'autre part, il contient l'intégralité des codes utiles pour le fonctionnement du projet, ainsi que les fichiers additionnels liés au projet (fichier texte, image, fichier son, etc.). Des indications en relation avec le projet sont ajoutées pour assister l'utilisateur.

Le lien d'accès est le suivant : https://github.com/INSA-FIPMIK/SmartRobot

Ci-après les instructions d'utilisation selon la logique d'enregistrement définie :

- Le fichier DockerFile comprend le code à exécuter pour créer le container associé au projet.
- Le fichier requierements.txt comprend toutes les librairies installables avec pip3 nécessaires au projet.
- Le dossier src regroupe tous les codes en langage Python associés au projet.
- Le dossier data regroupe tous les fichiers utiles au projet mais qui ne sont pas des codes Python.

3) Le projet

a) Première prise en main du robot et premier essai

N'ayant jamais travaillé avec ce type de robot, nous avons essayé de nous familiariser avec celui-ci en utilisant le Logiciel Niryo Studio. Pour de la programmation simple, le logiciel propose une série de « Block » qui permettent de réaliser une certaine action. Comme par exemple, se déplacer jusqu'à un point donné.

En lien avec notre projet, il existe un block qui permet d'indiquer au robot de prendre un objet avec une certaine forme et une certaine couleur. Evidemment, cette reconnaissance est réalisée avec la caméra et le traitement d'image pré-entrainé du robot. Après avoir choisi

par exemple la couleur bleue et la forme ronde par exemple, le robot va prendre une image et puis se diriger vers l'objet choisie pour le prendre. Jusqu'à le lâcher à une certaine position.

Ce programme a très bien fonctionné et nous a permis de comprendre comment le robot fonctionné avec le langage « Blockly ». Néanmoins notre projet est fait en langage Python.

b) Structure globale du projet

i) Prérequis

Le pilotage du robot se fait au moyen du microcontrôleur Jetson Nano. La carte doit tout d'abord être alimentée et disposer d'un dongle wifi ou d'une antenne. Puis, la connexion avec le PC se fait au moyen du port micro SD vers un port USB du PC. Enfin, la connexion sur la carte est réalisée via le protocole SSH avec la ligne de commande ci-après :

ssh -X jetson« numéro de Jetson »@192.168.55.1

Il est recommandé d'utiliser MobaXterm pour la connexion SSH car la carte est adaptée à l'environnement Linux et non Windows. Une fois connecté sur la Jetson, il est recommandé d'utiliser un container, car il présente l'avantage de ne pas endommagé le contenu de la carte en cas de fausse manipulation. Pour créer un container, il suffit de suivre la procédure accessible au lien suivant : . Pour créer un container adapté au projet, il est nécessaire d'utiliser les fichiers DockerFile et requirements.txt dans le Github du projet SmartRobot. Cela permet d'éviter l'installation des librairies à chaque redémarrage de container. Une fois le container créé, il peut être ouvert :

drun -c « nom_du_projet »

L'ouverture d'un container rend accessible une interface Jupyter selon le mode de connexion écrit dans le terminal de commande. Le dossier « app » de Jupyter est ensuite libre d'accès pour modifier et exécuter des programmes.

Dans le cadre du projet, il est à noter qu'une ligne de commande doit tout de même être entrée à la main au démarrage du projet. Elle concerne la lecture des codes ASCII spécifiques, notamment les lettres avec accent, dans le cadre de la reconnaissance vocale. Elle est stockée dans le Github en tant que ReadMe.

README.md

SmartRobot

Created from https://github.com/nlpTRIZ/jetson_docker_X_forwarding

Avant de lancer le programme : 1)Brancher le Jabra 2)Connecter le robot au réseau wifi via le logiciel NiryoStudio
3)Taper la commande suivante dans le terminal
export LC_CTYPE="en_US.UTF-8"

Pour lancer le programme taper dans le terminal:

1. Se placer à l'endroit où le fichier "main.py" se trouve
2. Taper dans le terminal : python3 main.py

Figure 7 : Aide au démarrage du programme dans Github

Pour l'utilisation du programme, il est nécessaire de connecter la Jetson au robot Niryo sur un réseau WiFi identique. La connexion de la Jetson se fait directement dans le terminal Linux une fois connecté sur la Jetson :

- Se connecter à un réseau WiFi : sudo nmcli dev wifi
- Scanner les réseaux WiFi accessibles : nmcli dev wifi connect « nom_du_réseau » password « mot de passe »

Pour connecter le robot à ce réseau WiFi, il faut accéder au logiciel Niryo Studio puis choisir d'enregistrer un nouveau réseau en entrant les paramètres de connexion.

Enfin, pour exécuter le code via Python sur la Jetson, il est nécessaire d'intégrer au programme l'adresse IP du robot et le workspace utilisé. Créer un nouveau workspace peut se faire directement dans le logiciel Niryo Studio.

ii) Structure du programme

Le programme s'articule autour de deux branches. La première traite de l'objectif même du projet, à savoir la préhension d'un objet donné par l'utilisateur en commande vocale, détectable à l'aide d'une reconnaissance d'image. La seconde traite quant à elle d'une partie induite à savoir l'enregistrement de nouveaux objets dans la base de données du projet. Cette étape est nécessaire à l'utilisation de la première.

Une problématique est apparue, liée au besoin d'enregistrer les objets scannés d'une compilation à une autre. Nous avons ajouté une fonctionnalité qui permet d'enregistrer dans un dictionnaire toutes les formes et couleurs déjà entraînées précédemment pour palier ce problème. Le dictionnaire est également accessible dans le répertoire « data » pour consultation et est mis à jour en temps réel. Il est enregistré sous l'extension json.

Finalement, au démarrage du programme, l'utilisateur a le choix entre les deux possibilités précédemment évoquées. Grâce au stockage dans le dictionnaire, il est informé à l'aide d'un affichage dans le terminal du contenu actuel du dictionnaire, ce qui permet d'orienter son choix sur la nécessité ou non de réaliser un nouvel enregistrement d'objet.

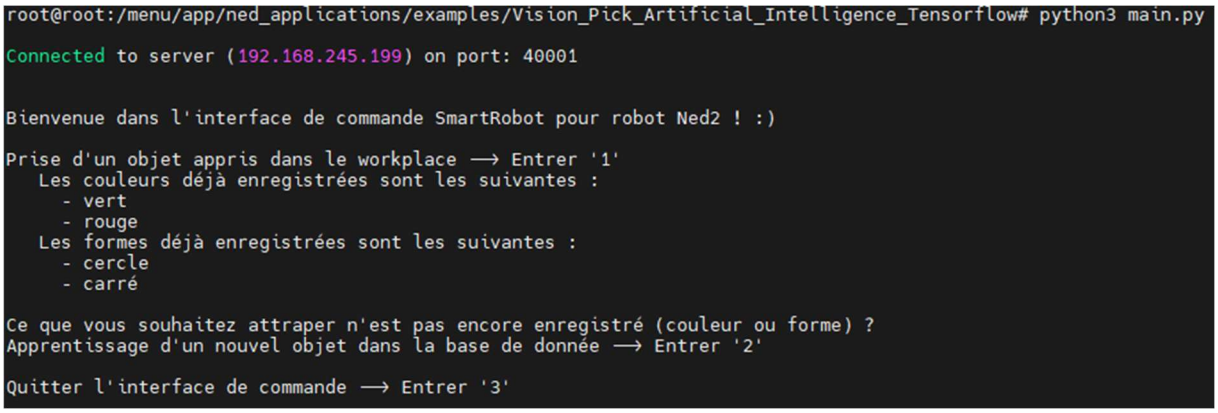

Figure 8 : Vue de démarrage du programme dans le terminal

Chaque partie fait l'objet d'un fichier distinct pour une meilleure compréhension à la lecture du programme. Certaines fonctions notamment liées au dictionnaire et à la reconnaissance d'images sont réimportées dans un autre programme car utiles dans plusieurs cas distincts. Le projet comporte donc 3 fichiers Python (main.py, new_objects.py et detection.py), un fichier json qui correspond au dictionnaire de couleurs et de formes

(Dictionnary database.json) et un fichier audio comportant la retranscription audio de la commande vocale (voc.wav).

Au démarrage de chaque programme, le programme se met en relation avec le robot à l'aide de son adresse IP. La librairie PyNiryo contient la classe NiryoRobot permettant d'affecter l'adresse IP au robot. La classe PoseObject permet quant à elle d'affecter des coordonnées de position au robot pour la position d'attente (ou observation) et la position de dépose.

Le programme s'exécute en boucle en continu tant que l'utilisateur n'a pas fait la demande dans le terminal de vouloir le quitter en tapant « 3 » dans le menu principal.

c) La reconnaissance vocale

iii) Version 1 : Google Recognizer

Nous avons commencé à créer une reconnaissance vocale à l'aide d'un script Python inspiré d'Internet en utilisant uniquement l'entrée microphone et la sortie haut-parleur d'un ordinateur. Le script fonctionne et est adapté à nos attentes. Le code est capable de retranscrire à l'écrit ce que nous disons au micro, de manière assez fiable.

Nous avons ensuite essayé d'utiliser ce script avec le robot mais cela n'a pas directement fonctionné. Nous nous sommes rendus compte à ce moment-là que le wifi du robot n'était pas connecté à Internet et que donc notre programme avec Google Recognizer en a besoin pour fonctionner.

iv) Version 2 : Sphinx Recognizer

Pour pallier ce problème nous avons décidé d'utiliser la bibliothèque Pocketsphinx qui permet de faire la même chose mais tout en étant hors ligne. Cette bibliothèque de SpeechRecognition est uniquement disponible en anglais c'est pourquoi nous avons dû télécharger un dictionnaire français pour l'ajouter dans le dossier des langages de Pocketsphinx. Après nos essais la bibliothèque semble correspondre à nos besoins mais elles présentent notamment quelques désavantages. Elle a le désavantage d'être notamment beaucoup moins rapide mais aussi moins précise dans sa compréhension. Après essai, nous pouvons affirmer qu'elle reconnait le texte énoncé environ une fois sur deux, à condition d'être dans un environnement très calme.

v) Utilisation concrète dans le programme

En fin de projet, nous avons finalement découvert la possibilité d'utiliser malgré tout d'utiliser le Google Recognizer qui est bien plus efficace. Tout cela est dû au fait que nous avons trouvé la possibilité de connecter le robot à un réseau wifi qui dispose d'une connexion Internet selon la méthode expliquée en préambule.

Dans le programme, la commande vocale se fait en deux temps. Dans un premier temps, l'entrée microphone du Jabra est attribuée dans la variable MIC_INDEX. Elle est active pendant 5 secondes, et l'utilisateur est informé de ce laps de temps dans le terminal. Le son est stocké dans un fichier wav. Dans un second temps, le fichier son est analysé à l'aide de Google Recognizer, qui se charge de traduire chaque mot de l'enregistrement sonore. Le résultat de la commande vocale est ensuite affiché à l'écran pour informer l'utilisateur.

```
rec_vocale = sr.Recognizer()
fichier = "../data/voc.wav", "fr-FR"

with sr.AudioFile(fichier[0]) as src: # open file
    audio = rec_vocale.record(src)
    texte = rec_vocale.recognize_google(audio, language=fichier[1]) # translate speech into text in French
    print(texte)

return texte
```

Figure 9 : Reconnaissance vocale à partir d'un fichier son avec Google Recognizer

d) Reconnaissance d'image

L'intelligence du robot permet de centrer la vue de la caméra sur un carré délimité par des symboles particuliers. Le robot est ensuite capable de détecter tous les objets qui se situent à l'intérieur de cette zone. C'est ce qu'on appelle le « workspace ». Le format de l'image associé à ce carré est de 200 sur 200 px.

Notre première piste en ce qui concerne la reconnaissance d'images a été d'utiliser les commandes associées issues de la bibliothèque PyNiryo. Nous sommes parvenus à tester cette éventualité, mais nous nous sommes confrontés à des problèmes de fiabilité. Bien que les recommandations données par Niryo garantissent une reconnaissance fiable à partir d'un entraînement sur une vingtaine d'images, nous nous sommes rendu compte que même cinquante images ne sont pas suffisantes. La reconnaissance de couleur est fiable à 100%, mais la reconnaissance de forme à l'inverse ne fonctionne pas avec les pièces simples. L'intelligence du robot n'arrive pas à faire la distinction entre un rond et un carré.

Pour palier ce problème, nous avons décidé de créer nous-même notre reconnaissance d'images à partir des exemples du cours de Machine Learning. Pour cela, nous avons scindés la reconnaissance en deux branches : la reconnaissance de couleur au moyen du code RGB et la reconnaissance de formes au moyen du nombre de pixels de la couleur de l'objet. Ce choix reste réducteur car il permet de différencier uniquement des objets aux couleurs unies et de taille différente.

Dans le cas où l'utilisateur souhaite enregistrer un nouvel objet inconnu dans la base de données, il doit tout d'abord placer uniquement cet objet précis dans la zone de travail. La détection de la couleur de l'objet s'effectue au moyen d'une méthode K-means effectué avec comme paramètre le code RGB des pixels. Par observation, nous pouvons remarquer que toutes les couleurs de la photo sont des nuances de gris, hormis la couleur de l'objet. La particularité de ces couleurs est qu'elles ont un code RGB de proportion égale, c'est-à-dire équilibrée entre le rouge, le vert et le bleu. 4 couleurs se dégagent donc : le noir, le blanc, le gris et la couleur de l'objet, ce qui correspond à un besoin de 4 centroids pour le K-means. Au moyen d'un écart type entre les valeurs RGB de chaque centroid, nous pouvons affirmer que la dispersion la plus grande correspond au code RGB associé à la couleur de l'objet. Cette méthode est très fiable mais présente le désavantage de ne pas pouvoir détecter les objets noirs, blancs ou gris avec certitude. La deuxième partie de l'entraînement est lié à la détection de forme. Le but ici est seulement de compter le nombre de pixels contenus dans le centroid de la couleur détectée précédemment. Pour une meilleure fiabilité, il est demandé à l'utilisateur de changer à 3 reprises la position de l'objet sur l'espace de travail. Après cela effectué, un code RGB moyen ainsi qu'un nombre de pixels moyen est déterminé et stocké en tant que référence pour l'objet.

```
# number of clusters (K)
k = 4 # white, grey, black and the color of the object
_, labels, (centers) = cv2.kmeans(pixel_values, k, None, criteria, 10, cv2.KMEANS_RANDOM_CENTERS)

# convert back to 8 bit values
centers = np.uint8(centers)
print(centers)

# detect which centroid correspond to the color of the object thanks to standard deviation method
sigma_max = 0
for each in range (len(centers)):
    temp = np.std(centers[each])
    if temp > sigma_max:
        sigma_max = temp
        pos = each

# count the number of pixels attached to the centroid of the object color
nb_pixels_centroid = int((labels == pos).sum())

return centers, nb_pixels_centroid, pos
```

Figure 10 : Méthode Kmeans pour déterminer la couleur et le nombre de pixels de l'objet

Finalement, ces deux données sont ensuite stockées dans le dictionnaire évoqué précédemment pour permettre de les conserver.

```
def update_dict(dict_color, dict_shape):
    # update the content of the two dict
    tf = open("../data/Dictionnary database.json", "w")
    json.dump((dict_color, dict_shape), tf)
    tf.close()
```

Figure 11 : Sauvegarde dans le dictionnaire

La seconde partie dans laquelle la reconnaissance d'images est utilisée est la détection d'objets prédéfinis dans l'espace de travail. Après l'entraînement préalable de l'objet selon la méthode prédéfinie précédemment, l'utilisateur peut désormais tester la détection d'objets. Les méthodes de reconnaissance de couleurs et de forme sont similaires à celles définies pour l'enregistrement d'un nouvel objet puisqu'elles reposent sur le principe de comparaison des valeurs.

En réalité, le code RGB entre la couleur de l'objet sur le workspace et celui contenue dans le dictionnaire pour cette même couleur sont comparées. Chacune des 3 valeurs du code RGB de l'un sont soustraits à chacune des 3 valeurs du code RGB de l'autre, puis sommées. On obtient en résultat un certain écart correspondant à la différence de couleur. Etant donné la différence de luminosité d'un moment à un autre de la journée et de la qualité de reconnaissance de couleur par la caméra, nous avons attribué la tolérance à 100. Si l'écart est en-dessous, cela signifie qu'aucun objet de cette couleur est présent sur le workspace et l'utilisateur est informé. Si la couleur donnée par l'utilisateur vocalement n'est pas disponible dans le dictionnaire, cela signifie que la couleur n'a pas encore été enregistrée, et l'utilisateur est également informé.

```python
def test_color(dict_color, texte, centers, pos):
    # check the link between recorded data from training and data from actual picture for the color
    color_detect = False
    error_color = False
    for i_color in dict_color.keys():
        if i_color in texte:
            color_detect = True
            sum_diff = 0
            for j_color in range(len(centers[pos])):
                sum_diff = sum_diff + abs(dict_color[i_color][j_color] - centers[pos][j_color])
            if abs(sum_diff) < 100: # gap until 100 is accepted
                print("Couleur détectée !")
            else:
                print("Pas d'objet de couleur", i_color, "sur le workplace !")
                error_color = True
    if color_detect == False:
        print("Couleur non-détectée dans la base de données !")
        error_color = True
    return error_color
```

Figure 12 : Exemple de la fonction de test sur la reconnaissance de couleur

Pour ce qu'il en est de la reconnaissance de formes, la logique est la même ; le nombre de pixels de référence contenu dans le dictionnaire pour la forme est soustrait au nombre de pixels de la couleur détectés sur l'image. Une tolérance de l'ordre de 200px est ici attribuée, et a été définie sur base de l'expérience. Le comptage de pixels n'est pas toujours extrêmement précis car il peut persister des ombres d'une couleur proche de la couleur à détecter.

e) Détection de position et commande du robot

Une fois la confirmation que l'objet détecté est conforme aux attentes de l'utilisateur, la phase de détection de position se déclenche. Le principe repose autour de la génération d'un masque sur l'image afin de colorer d'une couleur différente l'objet reconnu. Nous avons récupéré certaines fonctions d'un exemple de reconnaissance d'images du robot pour réaliser cette opération. La fonction de création de masque permet ensuite de déterminer les coordonnées du centre de la forme colorée (en jaune) suivant les deux axes du plan, ainsi que l'angle d'orientation de la forme. L'angle n'est pas nécessaire dans le cas du cercle mais davantage dans celui du carré attrapé par la pince car il permet d'orienter l'axe de rotation de l'outil pour réussir la prise.

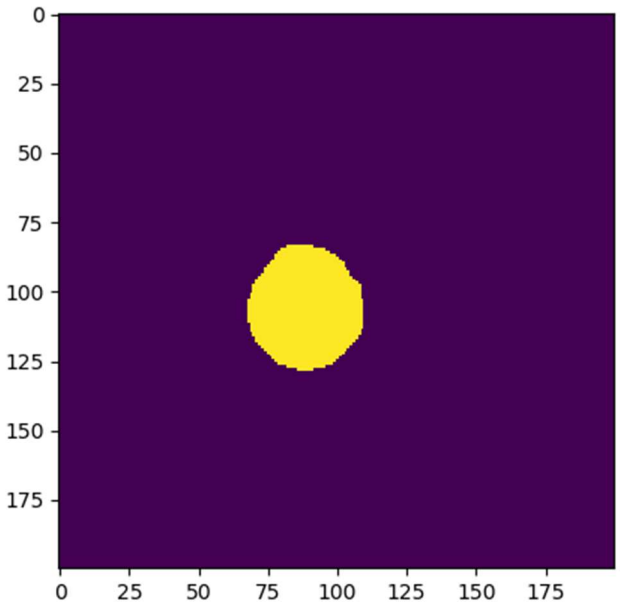

Figure 13 : Résultat du traitement de l'image avec au centre l'objet à récupérer

Enfin, les coordonnées et informations du point trouvé est récupérée pour générer ensuite les commandes de prise et de dépose. Ces dernières opérations sont réalisées grâce aux fonctions de la librairie PyNiryo qui sont directement adaptées et donc utilisables pour les commandes de mouvement notamment.

```
z_offset = 0.01 # offset for the vacuum pump
obj_ = client.get_target_pose_from_rel(workspace, z_offset, (x / 200), (y / 200),
                                       angle)
print("Position objet : ", obj_)
client.pick_from_pose(obj_) # take the object
client.place_from_pose(*drop_pose.to_list()) # place it in the chosen drop zone
#client.close_gripper() # in case of use of the gripper instead of vacuum pump
client.move_pose(*observation_pose.to_list()) # robot is again in observation pose for a new round
```

Figure 14 : Commandes de mouvement du robot à partir des coordonnées

4) Conclusion et pistes d'amélioration

En conclusion, ce projet permet au robot de récupérer la pièce demander oralement par l'utilisateur. La fonction principale est remplie mais la reconnaissance marche uniquement en présence d'un seul objet sur le plateau, et à condition que l'objet est d'une couleur unie différente du noir, blanc ou gris. Il est par conséquent viable à la détection des jetons du kit Niryo, présentés en introduction.

Les points à améliorer sur ce projet sont la vision dans le cas où plusieurs pièces se trouvent sur le plateau. Pour résoudre cela, plusieurs pistes sont possibles mais le temps ne nous a pas permis de mettre en place ces méthodes.

Pour la reconnaissance d'image, un réseau de neurone pourrait être utilisé. Cette méthode est plus efficace qu'un simple K-means mais va nécessiter un nombre d'image colossal pour l'entrainement. Elle sera également plus modulable étant donné que l'on pourra l'entrainer pour tous les objets désirés à condition d'avoir assez d'images.

La deuxième serait de réaliser deux K-means à la suite. Le premier K-means permettrait de réaliser un tri des données pour la couleur des pixels. Puis, un second K-means permettrait de déterminer les coordonnées des pièces de même couleur et de compter les pixels pour déterminer de quelle forme il s'agit.

Ces deux solutions pourraient être des pistes qui amélioreraient ce projet.

5) Sources

a) Table des illustrations

b) Liens internet

Codigo (Réalisateur). (2021, mars 29). *Reconnaissance Vocale avec Python*. https://www.youtube.com/watch?v=kxmchcEb44E

GitHub—NiryoRobotics/ned_applications : Applications or tutorials code hosting for Ned. (s. d.). Consulté 16 janvier 2023, à l'adresse https://github.com/NiryoRobotics/ned_applications

Kit de développement NVIDIA Jetson Nano. (s. d.). NVIDIA. Consulté 7 janvier 2023, à l'adresse https://www.nvidia.com/fr-fr/autonomous-machines/embedded-systems/jetson-nano/education-projects/

Le set vision pour robot éducation. (s. d.). Niryo. Consulté 7 janvier 2023, à l'adresse https://niryo.com/fr/product/set-vision-ned2/

Ned2 : Bras robotique pour l'éducation. (s. d.). Niryo. Consulté 7 janvier 2023, à l'adresse https://niryo.com/fr/product/ned2-cobot-education/

nlpTRIZ. (2022). *Jetson Nano : Build a docker container with X forwarding* [Python]. https://github.com/nlpTRIZ/jetson_docker_X_forwarding (Original work published 2022)

PyNiryo Documentation—PyNiryo v1.1.2 documentation. (s. d.). Consulté 7 janvier 2023, à l'adresse https://docs.niryo.com/dev/pyniryo/v1.1.2/en/index.html

Shmyrev, N. (s. d.). *Building an application with PocketSphinx*. CMUSphinx Open Source Speech Recognition. Consulté 7 janvier 2023, à l'adresse http://cmusphinx.github.io/wiki/tutorialpocketsphinx/

SpeechRecognition : Library for performing speech recognition, with support for several engines and APIs, online and offline. (3.9.0). (s. d.). [Python; MacOS :: MacOS X, Microsoft :: Windows, Other OS, POSIX :: Linux]. Consulté 7 janvier 2023, à l'adresse https://github.com/Uberi/speech_recognition#readme

Titre : Flac [Wiki ubuntu-fr]. (s. d.). Consulté 7 janvier 2023, à l'adresse https://www.google.com/imgres

Vision set pour robot éducation. (s. d.). Niryo. Consulté 7 janvier 2023, à l'adresse https://niryo.com/fr/product/vision-set-fr/

Rockikz, A. (s. d.). *How to Use K-Means Clustering for Image Segmentation using OpenCV in Python—Python Code*. Consulté 20 janvier 2023, à l'adresse https://www.thepythoncode.com/article/kmeans-for-image-segmentation-opencv-python