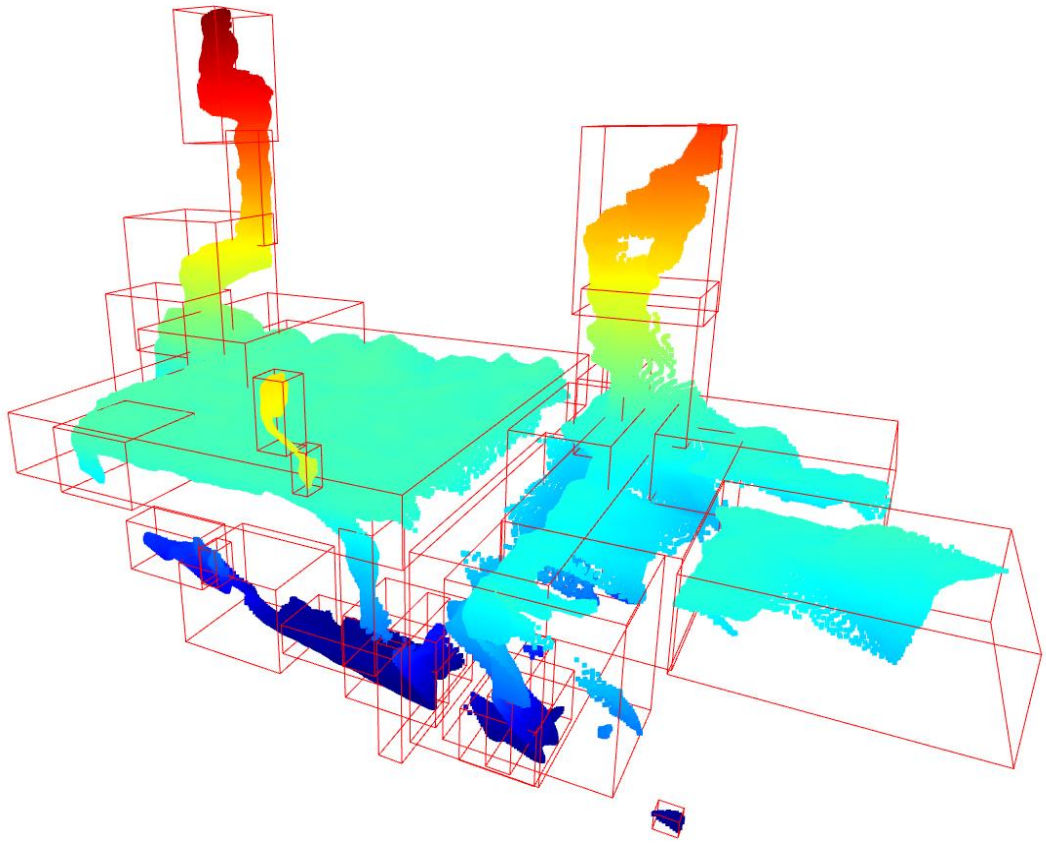
Notice d’utilisation et de modification de DyVision



Natanael Jamet

Sommaire

Vue d’ensemble

L’ensemble des livré a été réalisé dans le cadre d’un stage de 10 semaines au sein du laboratoire Dynamium. Le but de DyVision est de fournir des informations sur les obstacles présents sur une zone de travail cobotique.

Ces informations prennent la forme de boite AABB (Axis aligned Bounding Boxes) et des points contenus dans chacune de ces boites. La détection se fait avec une caméra RGB-D et le traitement est basé en python.

Le livré contient les éléments suivants :

- DyVision 2.0 : les trois scripts permettant les traitements et visualisation.
- Une archive des versions.
- Différentes ressources permettant l’utilisation et la modification de l’outil.

L’utilisation et la modification de ces codes sont détaillées dans ce document.

I – Informations générales - Materiel requis

I.1 – Langage et librairies requises

Ces scripts ont été écrit en python, j'ai utilisé l'IDLE disponible directement sur le site : Python.org.

La version utilisée est la version ***3.11***

Les librairies suivantes ont été utilisée avec les versions affichées :

- NumPy : 1.26.2
- Open3D : 0.18.0
- SciPy : 1.13.0
- Scikit-Learn : 1.5.1
- OpenCV : 4.5.5.64
- Pandas : 2.2.2
- Tkinter : 8.6

La version de OpenCV est importante, des versions plus récentes ne fonctionne pas avec le code réalisé.

Pour installer la bonne version de cette librairie, on peut utiliser les commandes suivantes (pour *windows* avec *pip*) :

pip uninstall opencv-python opencv-python-headless

pip uninstall opencv-python opencv-contrib-python

pip install opencv-python==4.5.5.64 opencv-python-headless==4.5.5.64

pip install opencv-contrib-python==4.5.5.64

Les fonctions d'affichage présentes dans les versions plus récentes ne sont pas disponibles, il faudra donc utiliser une autre librairie pour la détection des marqueur Aruco ou pour l'affichage si jamais un affichage est souhaité.

I.2 – Materiel requis

Caméra :

La caméra utilisée pour le développement est une **RealSense D435** de chez Intel. L'utilisation d'un autre modèle de la gamme RealSense ne devrait pas poser de problèmes. D'autres caméras ne fonctionneront pas sans modification du code.

Je conseil de télécharger le SDK d'Intel : https://www.intelrealsense.com/sdk-2/ . Celui-ci permet d'avoir accès à un logiciel de visualisation (RealSense Viewer) et à un outil de Calibration (Depth Quality Tool). Ils permettent de visualiser les sorties de la caméra facilement, de faire des réglages et la calibration au besoin.

Dans le cadre du projet, la caméra a été montée sur un pied photo Manfrotto, j'ai pour cela imprimé un adaptateur, les fichiers de celui-ci se trouvent dans le livré.

La caméra n'a pas besoin d'être positionnée précisément, le positionnement se faisant grâce à des marqueurs.

Attention lors de l'utilisation de ces logiciels : s'ils ne sont pas fermés au moment de l'utilisation des scripts, ceux-ci considèreront que la caméra n'est pas connectée.

Marqueurs :

Le positionnement de la caméra se fait avec des marqueurs ArUco. Il faut donc en placer un dans l'espace de travail.

Le fichier pour imprimer le capteur utilisé est disponible dans le livré. En cas de changement de marqueur, se référer à la section ***Modifications***.

Il faut placer le marqueur à une position connue du robot.

Il est préférable d'utiliser des marqueurs imprimés sur un support mat sans plastification afin d'éviter les reflets et de s'assurer d'une détection fiable.

Attention : les fichiers exportés expriment les coordonnées des boites et des points dans le repère du marqueur, il faudra donc appliquer une transformation pour les utiliser dans le repère du robot.

II – Utilisation

II.1 – Reconstruction

Le premier script permet de reconstruire la scène statique à partir de plusieurs points de vue. Assurez-vous que la caméra est connectée sur un port USB 3.1 avec un câble adapté.

Il sera demandé à l'utilisateur le nombre de points de vue souhaités :

```
Début de la prise des photos pour reconstruire la scéne statique.
Entrez le nombre de Points de vue souhaités.
Assurez-vous que le marqueur soit visible à chaque prise de vue.

Combien de prises de vue souhaitez-vous effectuer ? 1
Prise de vue 1/1
Appuyez sur 'Enter' pour capturer une image.
```

Figure-x : légende

Après avoir rentré le nombre voulu de prises de vue, l'utilisateur pourra appuyer sur 'entrée' pour capturer un nuage de points.

Si aucun marqueur n'est détecté, il lui sera redemandé de capturer un nuage :

```
Aucun marqueur détecté. Veuillez reprendre la photo.
Appuyez sur 'Enter' pour capturer une image.
```

Figure-x : légende

Une fois qu'un marqueur est détecté le message suivant apparait et le pré-traitement du nuage commence, une fois que celui-ci est fait, l'image rgb avec la détection du marqueur et le nuage apparaissent :

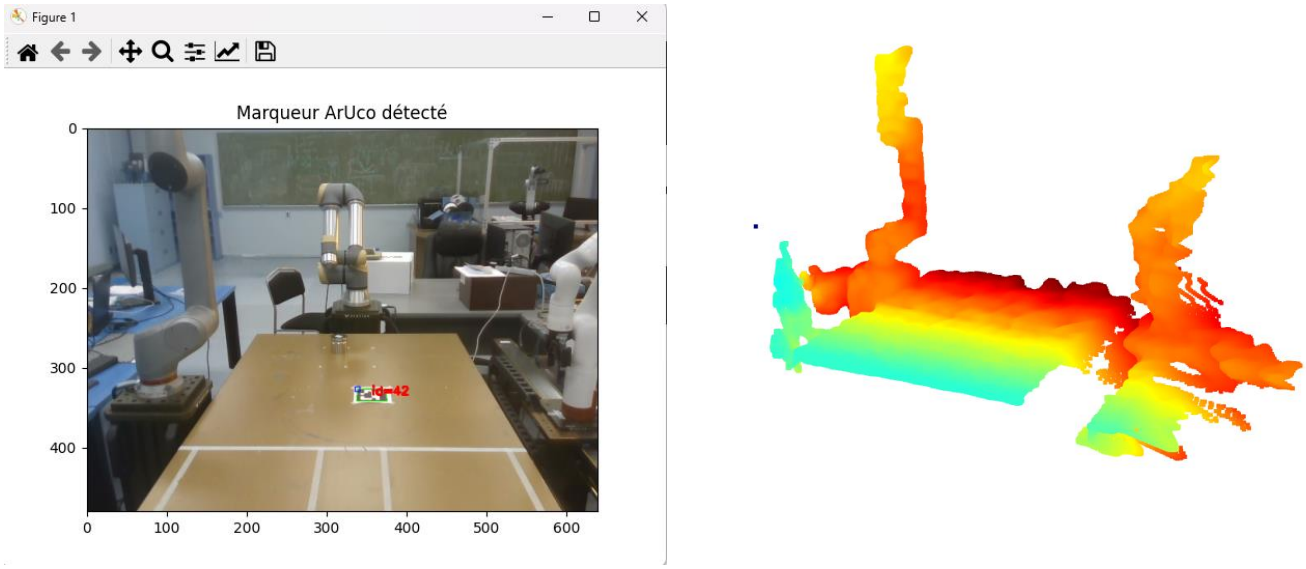


Figure-x : images affichées pour vérification

L’utilisateur peut alors vérifier que la prise de vue lui convient et la reprendre si besoins.

Il faut vérifier, surtout, que la détection du marqueur se soit bien faite : le numéro d’id du marqueur, ses contours et le coin Haut-Gauche apparaissent, ces informations doivent correspondre au marqueur utilisé.

Après un temp d’attente, la définition des boites AABB est faite, une interface de sélection s’ouvre. Celle-ci permet de sélectionner les boites à supprimer de la scène statique. Les boites restantes définiront les objets statiques et donc les zones à éviter :

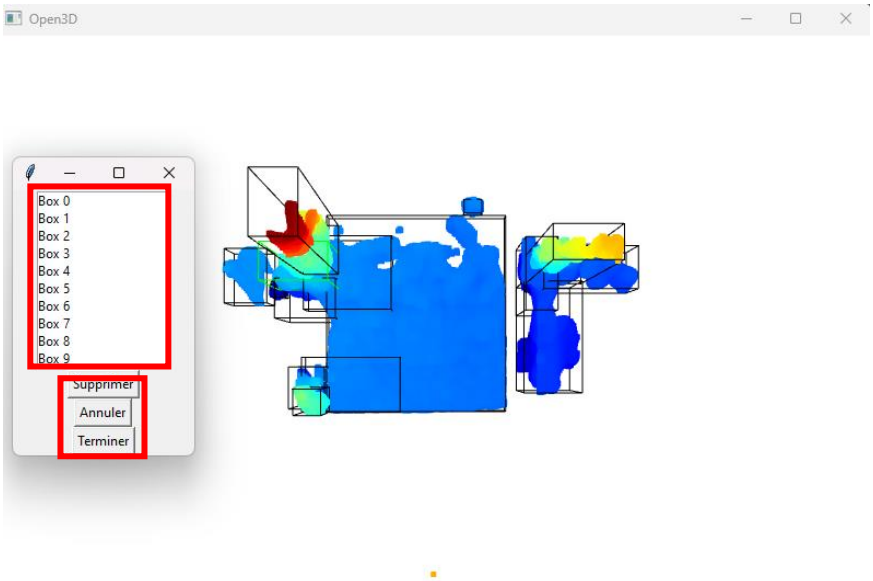


Figure-x : Interface de sélection des boites

Eléments de l’interface :

1 : Les boites et les points sont affichés. Une boite noire veut dire qu’elle est gardée, rouge qu’elle est supprimée.

2 : La liste des boites permet de sélectionner les boites, lorsque la souris est passée au-dessus du nom d’une des boites, celle-ci est affichée en vert.

3 : Le bouton ‘Supprimer’ permet de supprimer une boite sélectionnée. Le bouton ‘Annuler’ permet d’annuler la dernière suppression. Le bouton ‘Terminer’ permet de mettre fin à la sélection.

Une fois cette sélection faite, le résultat est affiché :

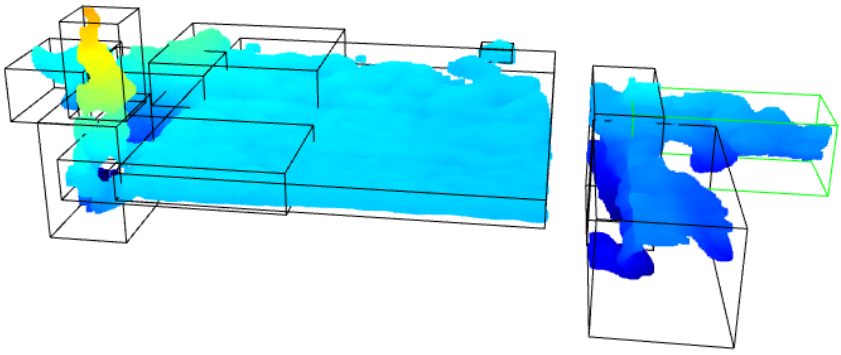


Figure-x : résultat après sélection des boites

Une fois la fenêtre fermée, les boites restantes seront enregistrées au format. PARQUET ainsi que la liste des points restant organisée par appartenance aux boites. Un fichier. ply contenant les points combinés est également enregistré.

Ces trois fichiers sont utilisés par le traitement dynamique, il est donc important de laisser les deux scripts dans le même fichier et de ne pas supprimer les fichiers PARQUET et ply crées dans le répertoire.

Attention : si la configuration de la zone de travail change ou que la marqueuse est déplacée, il faudra refaire la reconstruction. Sinon il y a seulement besoins de la faire une fois.

II.2 – Traitement dynamique

Une fois la scène statique traitée, l'utilisateur peut lancer le traitement dynamique.

Il est important encore une fois que les fichiers générer par la reconstruction soient dans le même répertoire.

Une fois ce script lancé, la bonne récupération des données de la reconstruction est affichée. Il est demandé à l'utilisateur de prendre une capture qui permet de positionner la caméra dans le repère du marqueur :

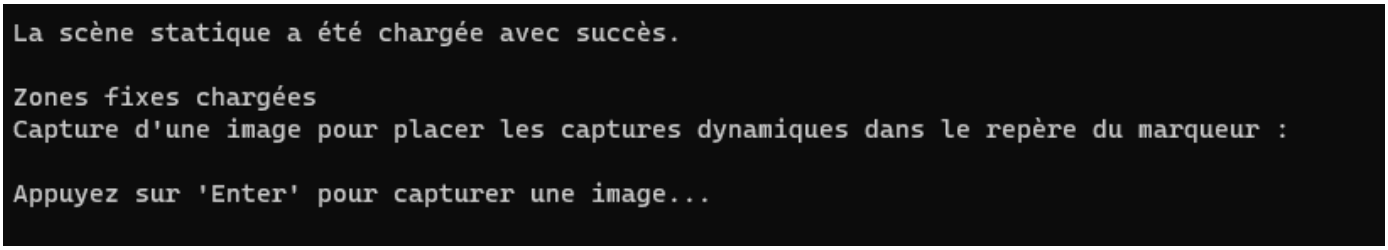


Figure-x : légende

Si aucun marqueur n'est détecté, une autre image sera demandée. Une fois que le marqueur est détecté, il ne faut plus bouger la caméra.

Une fois le positionnement fait, la détection se fera alors. La fréquence de rafraichissement peut varier selon les ordinateurs utilisés.

Deux fichier PARQUET au même format que ceux issus de la reconstruction sont créés, ils seront écrasés à chaque itération et contiendront les dernières informations calculées.

II.3 – Données exportées

Il y a deux types de données exportées utilisables pour l'optimisation de trajectoire :

- La liste des boites
- La liste des points restants

Ces données sont au format PARQUET, qui organise en colonne.

Les listes de boites sont sous la forme suivante :

Idx	Centre x	Centre y	Centre z	Dimension x	Dimension y	Dimension z

Avec :

- *Idx* : le numéro d'identification de la boite
- *Centre_xyz* : les coordonnées du centre de la boite
- *Dimension_xyz* : les demi-longueurs de la boite selon chaque axe

La liste des points est, elle, sous la forme :

Idx	X	Y	Z

Avec :

- *Idx* : le numéro d'identification de la boite à laquelle le point appartient
- *X , Y, et Z* : les coordonnées du point dans le repère du marqueur

III – Vue détaillée du code

Les arguments et sorties des fonctions sont détaillés en V.1, ici seulement les corps des programme seront expliqué.

III.1 – Reconstruction

Le tableau des variables est donné en V.2 pour plus de détails.

Récupération

```
try:
    #-------------------------------------------------------------------------------------
    # Récupératioin des nuages
    #-------------------------------------------------------------------------------------

    print('Début de la prise des photos pour reconstruire la scéne statique. \nEntrez le nombre de Points de vue souhaités. \nAssu
    n = int(input("\nCombien de prises de vue souhaitez-vous effectuer ? "))
    point_clouds = []
    color_images = []
    for i in range(n):
        print(f"Prise de vue {i+1}/{n}")
        color_frame, depth_frame, color_image, gray, corners, ids = capture_image()
        color_images.append(color_image)

        if ids is not None:
            cv2.aruco.drawDetectedMarkers(color_image, corners, ids)
            rvecs, tvecs, _ = cv2.aruco.estimatePoseSingleMarkers(corners, taille_marqueur, camera_matrix, dist_coeffs)
            for j in range(len(ids)):
                cv2.aruco.drawAxis(color_image, camera_matrix, dist_coeffs, rvecs[j], tvecs[j], 0.1)

                position = tvecs[j].flatten()
                rotation_matrix, _ = cv2.Rodrigues(rvecs[j])

                point_cloud, _ = depth_to_point_cloud(depth_frame, color_frame)
                rotation_matrix = rotation_matrix.T
                translation = -rotation_matrix @ position
                transformed_points = (rotation_matrix @ point_cloud.T).T + translation

                filtered_points = filter_points_by_density(transformed_points, radius, threshold)
                point_clouds.append(filtered_points)
```

Figure-x : récupération des points

Les variables *point_clouds* et *color_images* permettent de stocker respectivement les nuages de points et les images rgb obtenue grâce à la fonction *capture_image().*

ids et *corners* sont le numéros d'identification du marqueur est les coordonnées des coins dans le repère de la caméra. *rvecs* et *tvecs*, les vecteur de rotation et de translation exprimant la position du marqueur dans le repère caméra, sont calculés à partir de *corners* et des paramètres de la caméra.

rotation_matrix et *translation* sont les matrice de passage d'un point de caméra vers marqueur

transformed_points est la liste des points dans le repère marqueur

Les points sont ensuite filtrés par densité puis stockés dans *point_clouds,* qui contient alors tous les nuages capturés et traités.

Combinaison :

```
#-------------------------------------------------------------------------------------------
# Combinaison et enregitrement des nuages
#-------------------------------------------------------------------------------------------
if point_clouds:
    combined_points = np.concatenate(point_clouds, axis=0)

    pcd = o3d.geometry.PointCloud()
    pcd.points = o3d.utility.Vector3dVector(combined_points)

    output_file = "combined_point_cloud.ply"
    o3d.io.write_point_cloud(output_file, pcd)
    print(f"nuage sauvegardé dans : {output_file}")
    print('Fin de la récuperation des points de vue - Début de l\'optimisation')
```

Figure-x : combinaison et enregistrement des nuages

Combined_points contient tous les points en une seule liste, ce nuage est enregistré au format *.ply* dans le répertoire où se trouve les scripts.

Optimisation :

```
#-------------------------------------------------------------------------------------------
# Optimisation
#-------------------------------------------------------------------------------------------
optimized_bounding_boxes = optimisation(combined_points, alpha, beta, K)

print('L\'optimisation est finie. Veuillez sélectionner les boîtes à supprimer dans l\'interface graphique')
```

Figure-x : définition par optimisation des AABB

Des boites AABB sont calculées à partir d'une optimisation, voici les détails de la fonction :

Cette fonction génère un premier ensemble de K boites, puis fusionne les boites en collision en en vérifiant que C(x) baisse avec la fusion :

$$C(x) = \alpha N + \beta V \quad (1)$$

Avec :

- $x=(N, V)$
- N : le nombre de boite total
- V : le volume total des boites
- α, β Les paramètres de l'optimisation
- C(x) est appelé le coût

Voici la structure de la fonction :

```python
def optimisation(combined_points, alpha, beta, K):
    kmeans = KMeans(n_clusters=K)
    kmeans.fit(combined_points)
    labels = kmeans.labels_
    clusters = [[] for _ in range(K)]
    for point, label in zip(combined_points, labels):
        clusters[label].append(point.tolist())

    bounding_boxes = []
    AABB = []
    for cluster in clusters:
        dimensions, center = id_cube(np.array(cluster))
        AABB.append((dimensions, center))
        bbox = create_bounding_box(center, dimensions)
        bounding_boxes.append(bbox)
    C = Cout_sa(AABB, alpha, beta)
```

Figure-x : initialisation des AABB

Le nuage est divisé en *K* clusters avec un clustering K-Means, Les clusters sont stockés dans une liste : *clusters.*

Les AABB sont ensuite calculées pour chaque cluster, *AABB* contient les dimensions et centre de chaque boite tandis que *bounding_boxes* contient des objets de type AABB de open3d. Ce sont les mêmes boites sous des formats différents.

Un calcul du cout de l'ensemble initial est stocké dans *C*.

```python
rtree_idx = build_rtree(AABB)

continuer = True
c = 0
while continuer:
    c += 1
    rtree_idx = build_rtree(AABB)
    Nouvelles_boites = AABB.copy()
    fusion = [False] * len(AABB)
    nf = 0
    for i, boite_i in enumerate(AABB):
        dimensions_i, center_i = boite_i
        min_coords_i = np.array(center_i) - np.array(dimensions_i) / 2
        max_coords_i = np.array(center_i) + np.array(dimensions_i) / 2

        possible_collisions = list(rtree_idx.intersection(min_coords_i.tolist() + max_coords_i.tolist()))

        fusion = [False for _ in range(len(AABB))]
```

Figure-x : détection des collisions

Une R-arbre est calculé pour simplifier la recherche de collision.

On met à jour et réinitialise les variables au début de la boucle.

Fusion permet de garder une trace des boites ayant été fusionnées pour être sûr de fusionner chaque boite une seule fois.

Nf compte le nombre de fusion fait à chaque itération.

La boucle *for* génère une liste des indices des boites avec lesquelles la boite *i* est en collision.

```
            for j in possible_collisions:
                if i != j and not fusion[j]:
                    boite_j = AABB[j]
                    dimensions_j, center_j = boite_j
                    min_coords_j = np.array(center_j) - np.array(dimensions_j) / 2
                    max_coords_j = np.array(center_j) + np.array(dimensions_j) / 2

                    cout_avant = Cout_sa(Nouvelles_boites, alpha, beta)

                    Jean = Nouvelles_boites.copy()
                    Jean = [lst for lst in Jean if lst != boite_j]
                    Jean = [lst for lst in Jean if lst != boite_i]

                    nouvelle_boite = fusion_boite(boite_i, boite_j)
                    Jean.append(nouvelle_boite)

                    cout_apres = Cout_sa(Jean, alpha, beta)

                    if cout_avant >= cout_apres:
                        Nouvelles_boites = Jean.copy()
                        fusion[i], fusion[j] = True, True
                        nf += 1

                        rtree_idx.delete(j, min_coords_j.tolist() + max_coords_j.tolist())
                        rtree_idx.delete(i, min_coords_i.tolist() + max_coords_i.tolist())
```

Figure-x : fusion des boites

Pour la première fusion possible uniquement, on calcul la boite issue de la fusion des boites i et j. Pour vérifier la pertinence de la fusion, on calcul le cout avant fusion(*cout_avant*).

Puis on calcul ce que donnerait l'ensemble dans la liste temporaire *Jean.* A partir de cette liste, on calcul *cout_apres*, le coût si la fusion est appliquée.

Si le coût est bien réduit, on applique la fusion, on informe que les boites *i* et *j* ont déjà été fusionnées et on incrémente *nf*, puis on met à jour le R-arbre.

```
        AABB = Nouvelles_boites
        if nf == 0:
            continuer = False
            optimized_bounding_boxes = []

    optimized_bounding_boxes = []
    for dimensions, center in Nouvelles_boites:
        bbox = create_bounding_box(center, dimensions)
        optimized_bounding_boxes.append(bbox)



    return optimized_bounding_boxes
```

Figure-x : fin de la fonction Optimisation

On vérifie qu'une fusion ou plus ont été faites pour cette itération, si ce n'est pas le cas l'optimisation est finit et on arrête la boucle. On renvoie alors une liste d'objets AABB de open3d.

Sélection :

```
#-------------------------------------------------------------------------------
# Sélection des boîtes à garder
#-------------------------------------------------------------------------------
selected_boxes, remaining_boxes_indices = interactive_box_selection(combined_points, optimized_bounding_boxes)

#-------------------------------------------------------------------------------
# Obtenir les boîtes sélectionnées et les points correspondants
#-------------------------------------------------------------------------------
selected_bounding_boxes, points_per_selected_box = get_selected_boxes_and_points(combined_points, remaining_boxes_indices, optimized_bounding_boxes)

# Visualiser les points restant après sélection
remaining_points = filter_points_in_boxes(combined_points, optimized_bounding_boxes, remaining_boxes_indices)
visualize_point_cloud(remaining_points, [optimized_bounding_boxes[i] for i in remaining_boxes_indices])
```

Figure-x : Sélection des boites

On récupère la liste des indices des boites supprimées et gardées dans les listes *selected_boxes* et *remaining_boxes_indices*. On peut ensuite récupérer la liste des boites restantes et une liste contenant les points appartenant à une même boite restante : *selected_bounding_boxes* et *points_per_selected_box.* Les index correspondent entre les deux listes, une boite *selected_bounding_boxes[i] contiendra* les points *: points_per_selected_box[i]*

III.2 – Traitement dynamique

Le tableau des variables est donné en V.3

Récupération de la scène statique :

```
#-------------------------------------------------------------------
#------- Recuperation du nuage -------------------------------------
#-------------------------------------------------------------------

# Chemin du fichier .ply
ply_file_path = "combined_point_cloud.ply"

# Charger le nuage de points depuis le fichier .ply
point_cloud = o3d.io.read_point_cloud(ply_file_path)

# Vérifier si le nuage de points est chargé correctement
if point_cloud.is_empty():
    print("Le nuage de points est vide.")
else:
    print("La scène statique a été chargée avec succès. \n")
# Convertir le nuage de points en un tableau NumPy de type float32
points_np = np.asarray(point_cloud.points, dtype=np.float32)
points_cp = cp.asarray(points_np)

Zones_fixes = read_bounding_boxes_from_parquet('selected_bounding_boxes.parquet')
print('Zones fixes chargées ')
```

Figure-x : Récupération des éléments statiques

La première étape est de récupérer les éléments statiques exportés par le script précèdent.

Le nuage de point combiné est stocké dans la variable *: point_cloud* puis convertit en format *cp.asarray* pour optimiser le temps de calcul.

Les boites identifiées sont stockées dans *Zones_fixes.*

Boucle principale :

C'est elle qui capture les nuages dynamiques et identifie les boites de collision sous forme de AABB.

```
centers = np.array([zone[0] for zone in Zones_fixes])
kdtree = cKDTree(centers)
```

Figure-x : Calcul de l'arbre KD

Un arbre KD est construit avant la boucle, il va permettre d'identifier rapidement les boites auxquelles appartiennent les points.

```
#-------------------------------------------------------------
#------- Detection de la position de la camera  ------------------
#-------------------------------------------------------------
# Capture de l'image


color_frame, depth_frame, color_image, gray, corners, ids = capture_image()

start_time1 = time.time()  # Démarrer le chronométrage - optionel

color_images = [color_image]

if ids is not None:
    cv2.aruco.drawDetectedMarkers(color_image, corners, ids)
    rvecs, tvecs, _objPoints = cv2.aruco.estimatePoseSingleMarkers(corners, taille_marqueur, camera_matrix, dist_coeffs)

    position = tvecs[0].flatten()
    rotation_matrix, _ = cv2.Rodrigues(rvecs[0])

    rotation_matrix = rotation_matrix.T
    translation = -rotation_matrix @ position

rotation_matrix_cp = cp.asarray(rotation_matrix)
translation_cp = cp.asarray(translation)

end_time1 = time.time()
elapsed_time1 = end_time1 - start_time1
print(f"La detection de la position de la caméra a pris {elapsed_time1:.2f} secondes.")
```

Figure-x : détection de la position de la caméra

De manière identique à la détection faite pour la scène statique, on identifie les matrices de passage pour changer les points de repère. Ceci est fait une seule fois avant le début de la boucle.

Les matrices *rotation_matrix_cp* et *translation_cp* sont les matrices rotation et translation déclarées sur le GPU pour optimiser el temps de calcul.

```
while True :

    #-------- Capture d'un nouveau point

    points_np2 = capture_point_cloud(pipeline)
    if points_np2 is None:
        print("Échec de la capture d'un nouveau nuage de points.")
        continue

    # Transformation des points
    points_cp2 = cp.asarray(points_np2)
    points_cpT = cp.dot(rotation_matrix_cp, points_cp2.T).T + translation_cp
    points_cpT_downsampled = downsample_point_cloud_random(points_cpT, 8)

    start_time = time.time()
    # Filtrage des points
    filtered_points_cp = filter_points(points_cpT_downsampled, centers, Zones_fixes)
```

Figure-x : début de la boucle

La première étape est de passer les points dans le repère du marqueur puis de réduire d'un facteur 8 le nombre de points et de filtrer par densité.

La fonction *filter_points()* permet de retirer tous les points qui appartiennent à une des boites identifiées dans la scène statique.

Les points gardés sont dans le tableau cp : *filtered_points-cp*

```python
#-------- Clustering ------------------------------------------------------------------
if len(filtered_points_cp) > 0:
    filtered_points_np = cp.asnumpy(filtered_points_cp)
    cluster_labels2 = birch_model.fit_predict(filtered_points_np)
    num_clusters2 = len(set(cluster_labels2)) - (1 if -1 in cluster_labels2 else 0)
    clusters2 = [[] for _ in range(num_clusters2)]
    for i, label in enumerate(cluster_labels2):
        if label != -1 and label < num_clusters2:
            clusters2[label].append(filtered_points_np[i])


    #------- Calcul des bounding box  ----------------------------------------------
    Zones_dynamique = []
    for cluster in clusters2:
        if len(cluster) > 0:
            dimensions, center = id_cube(cluster)
            Zones_dynamique.append((center, dimensions))
```

Figure-x : Clustering et identification des AABB

Un clustering de type BIRCH est appliqué, puis pour chaque cluster une boitte AABB est calculée.

Ces données sont ensuite exportées au format PARQUET.

III.3 – Influence des paramètres et modifications

III.3.1 Paramètres liés à la caméra

Ceux-ci sont communs aux deux scripts.

camera_matrix, dist_coeffs : ce sont des paramètres intrinsèques de la caméra, ceux-ci devront êtres changés si la caméra utilisée n'est plus un RealSense D435. Ils sont obtenables soit par un code de calibration, soit avec le SDK Intel, ils peuvent également être obtenue directement avec la librairie Pyrealsense qui permet l'accès a ces informations. Ils permettent la bonne construction du nuage de points et la bonne détection des marqueurs ArUco.

taille_marqueur : c'est la taille en m du marqueur placé dans la scène, il est important de changer cette valeur si le marqueur n'est plus le même.

config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30) : cette fonction permet d'initialiser les paramètres de la caméra, en changeant 640 et 480, il est possible de changer la résolution.

III.3.2 – Paramètres de la reconstruction

radius : ce paramètre est utilisé dans le filtrage par densité, il quantifie la taille du voisinage d'un point. Augmenter cette valeur réduira le caractère local du filtre.

threshold : ce paramètre est également utilisé dans le filtrage par densité, c'est le nombre de voisin que le point doit avoir pour ne pas être supprimé, une plus grande valeur rendra le filtre plus discriminant.

K : c'est le nombre de clusters fait pour la première étape de l'optimisation. Une réduction de cette valeur donnera une approche moins précise du volume.

Alpha et beta : ce sont les paramètres de l'optimisations, alpha est fixé à 1. Beta quantifie l'importance que l'on donne à la réduction du volume par rapport à la réduction du nombre de boites. Augmenter beta permet d'avoir un volume plus réduit donc une meilleure approche mais plus de boites. Réduire beta réduira le nombre de boite mais le résultat sera moins précis. Pour déterminer la valeur de beta qui convient, un script : « identification de beta » est disponible dans le livré.

III.3.3 – Paramètres du traitement dynamique

Th2 : c'est le seuil du clustering BIRCH, c'est un paramètre qui quantifie la distance entre deux points pour qu'ils soient considéré comme voisins. Augmenter cette valeur donnera moins de clusters, donc une identification des AABB plus grossière mais un temps de traitement plus faible.

III.3.3 – Modifications possibles

En cas de changement de marqueur AruCo, il faudra penser à modifier le paramètre *taille_marqueur* en conséquence et que le marqueur choisit est présent dans le dictionnaire importé.

Dans la version DyVision 2.0, le marqueur utilisé est le premier détecté, si plusieurs marqueurs sont présents il faudra ajouter une sélection du marqueur à partir duquel on veut se repérer.

IV – Perspectives d'améliorations

Ces scripts ayant été fait sur une période relativement courte il reste plusieurs points à améliorer, voici quelques pistes qui permettraient de rendre ceux-ci plus agréable ou plus efficaces.

IV.1 – Reconstruction

L'interface de sélection des boites pourrait être modifiées pour être plus intuitive :

En gardant le format actuel :

- Faire en sorte que les points des boites supprimées soient supprimés en direct.
- Ajouter les couleurs réelles des points

En changeant le format :

- Mettre en place une interface qui permet de pointer directement les boites avec le curseur de la souris
- Permettre d'ajouter ou d'ajuster des boites

Il serait également intéressant d'améliorer l'interaction avec l'utilisateur en ajoutant :

- L'image vue par la caméra et la détection du marqueur en temps réel pour permettre de mieux cadrer
- Des boites de dialogues au lieu des *print()* et des *input()*

IV.2 – Traitement dynamique

Bien que le temps de traitement ait été déjà réduit, il devrait être possible de la réduire en passant le clustering sur le GPU, pour cela il existe une librairie : CuML qui devrait permettre ceci. Le temps de clustering étant le majoritaire dans le temps de traitement, cela permettrait d'augmenter considérablement la fréquence d'envoi des données.

Afin de permettre l'attribution de label aux obstacles et ainsi de permettre un suivi de la position de celui-ci, la segmentation peut être envisagée. La segmentation permet de séparer des zones de l'image, et donc d'isoler différents objets. Meta a sorti le modèle SAM-2 (Segment Anything Model) qui permet une segmentation sur n'importe quel objet, même si le modèle ne le connait pas.

Une piste serait d'effectuer une segmentation sur l'image RGB puis avec les données en profondeur, isoler les points appartenant à un même objet.

L'utilisation d'une seconde caméra serait pertinente afin d'éviter les angles morts, l'ajout d'une caméra ne devrait pas poser de problème en se basant sur les exemples disponibles en C++.

Pour améliorer le positionnement, plusieurs solutions sont envisageables :

- Ajout de plusieurs marqueurs : cela permettrait de repérer la position de la caméra même si certaines parties de la zone sont cachées par le robot. Pour cela il faudrait repérer les marqueurs par leur id (un numéro propre à chaque marqueur) et appliquer une transformation, différente en fonction du marqueur choisit, afin de placer les points directement dans un repère commun.
- Repérer le ou les marqueurs à chaque itération : cela permettrait de pouvoir déplacer la caméra au cours du fonctionnement. Pour cela il faut être sûr qu'un marqueur soit toujours visible ou si aucun marqueur n'est repéré, garder en mémoire la dernière position connue.

V – Tables des variables et des fonctions

V.1 Table des fonctions

Fonction	Arguments	Renvoi	Action
depth_to_point_cloud(depth_frame, color_frame, max_distance=2.0)	***Depth_frame*** : liste de données RGB ***Color_frame*** : liste des données RGB ***max_distance*** : float par défaut =2	Une liste de coordonnées [x, y, z] issue de la depthmap tronquée au-dessus de 2m	
build_rtree(aabbs)	aabbs : une liste de coordonnées de boite	Un objet Rtree construit à partir des aabb	
capture_image()		***color_frame,*** ***depth_frame,*** ***color_image,*** : l'image RGB sous forme de liste ***gray*** : l'image en niveau de gris ***corner*** : la liste des coordonnées des coins du marqueur détecté ***ids*** : un entier identifiant le marqueur trouvé	Demande à l'utilisateur d'appuyer sur entrée pour capturer l'image et demande la validation.
count_points_within_radius(points, center, radius)	***Points*** : liste de coordonnées [x, y, z] ***Center*** : cordonnées d'un point ***Radius*** : float, rayon du voisinage	Un float étant le nombre de points dans le rayon ***radius*** autour de ***center***	
Cout(AABB, alpha, beta)	***AABB*** : liste de coordonnées de boites AABB ***Alpha*** : float ***Beta*** : float	Un float étant le cout de l'ensemble de boite ***AABB***	Affiche le volume et le nombre de boites de l'ensemble de boite ***AABB***
Cout_sa(AABB, alpha, beta)	***AABB*** : liste de coordonnées de boites AABB ***Alpha*** : float ***Beta*** : float	Un float étant le cout de l'ensemble de boite ***AABB***	
create_bounding_box(center, dimensions)	***Center*** : liste de taille 3 ***Dimensions*** : liste de taille 3	Un objet AABB de centre ***center*** et de dimensions ***dimensions***	
downsample_point_cloud_random(points_cp, factor)	***Points_cp*** : tableau cp de coordonnées [x, y, z] ***Factor*** **:** int	Un tableau cp	
filter_points(points, centers, zones_fixes)	***Points*** : une liste de coordonnées ***Centers*** : une liste de coordonnées ***zones_fixes*** : une liste d'objet AABB	Une liste de coordonnées, ce sont les point de ***points*** n'appartenant pas aux boites de ***zones_fixes***	
filter_points_by_density(points, radius, threshold)	***Points*** : une liste de coordonnées ***Radius*** : float ***Threshold*** : int	Une liste de coordonnée des points ayant plus de ***threshold*** voisins dans un rayon de ***radius***	

filter_points_in_boxes(points, aabbs, indices)	***Points*** : une liste de coordonnées ***AABB*** : liste de coordonnées de boites AABB ***Indices*** : liste d'int	Une liste de coordonnée des ***points*** appartenant aux boite d'indice dans ***indices***	
fusion_boite(boite1, boite2)	***boite1 , boite2*** : une liste avec les coordonnées d'une boite	Les coordonnées d'une boite	
get_selected_boxes_and_points(combined_points, selected_boxes_indices, bounding_boxes)	***combined_points*** : une liste de coordonnées ***selected_boxes***_indices : liste d'entier ***bounding_boxes :*** liste de boite	***selected_bounding_boxes*** : liste d'objet AABB ***points_per_selected_box*** : liste de liste de coordonnées [x, y, z]	
id_cube(cluster)	***Cluster*** : liste de coordonnées [x, y, z]	Les coordonnées d'une boite	
interactive_box_selection(points, aabbs)	***points*** **:** liste de coordonnées [x, y, z] ***aabbs*** **:** liste d'objet aabb	Liste des indices des boites gardés Liste des indices des boites supprimées	Affiche une interface de sélection
optimisation(combined_points, alpha, beta, K)	***combined_points :*** liste de coordonnées [x, y, z] ***Alpha : float*** ***Beta : float*** ***K : int***	Une liste d'objet AABB	Affiche le numéro de la génération
points_in_boxes(points, aabbs)	***points*** **:** liste de coordonnées [x, y, z] ***aabbs*** **:** liste d'objet aabb	Une liste de liste de coordonnées [x, y, z]	
read_bounding_boxes_from_parquet(parquet_file)	***parquet_file :*** str chemin vers le fichier	Une liste de coordonnée de AABB	
visualize_point_cloud(points)	***points*** **:** liste de coordonnées [x, y, z]		Ouvre une fenêtre open3D qui affiche les points de ***points***
visualize_point_cloud(points, bounding_boxes=None, color_boxes=False)	***points*** **:** liste de coordonnées [x, y, z] ***bounding_boxes :*** liste de boite non renseignée par défaut ***color_boxes*** **:** coordronnées rgb d'une couleur, non renseignées par défaut		Ouvre une fenetre open3D qui affiche les points de ***points*** et les boites de ***bounding_boxes*** avec la couleur ***color_boxes***

V.2 – Variables et paramètres de reconstruction

Nom	type	Variable/Paramètre	Utilisation
alpha	float	paramètre	coefficient du calcul du cout de l'optimisation
beta	float	paramètre	Coefficient du calcul du cout de l'optimisation
boxes_data	liste	Variable	Donnée exportées
camera_matrix	Tableau np	Paramètre	Matrice intrinsèque de la caméra
color_frame	Tableau np	Variable	
color_images	Tableau np	Variable	Image RGB
combined_points	Tableau np	Variable	Points de toutes les vues combinées
corners	liste	Variable	Coordonnées des coins du marqueur repéré dans le repère caméra
depth_frame		Variable	
dist_coeffs	Tableau np	Paramètre	Matrice de déformation de la caméra
filtered_points	Tableau np	Variable	Points après déplacement
gray	Tableau np	Variable	Image en niveaux de gris pour la détection du marqueur
ids	int	Variable	Numéro d'identification du marqueur
K	int	paramètre	Nombre de boites initial de l'optimisation
n	int	paramètre	Nombre de points de vue (demandé à l'utilisateur)
optimized_bounding_boxes	Liste d'objet aabb	Variable	AABB déterminées par l'optimisation
output_file	str	Variable	Nom du fichier ou est stocké le nuage statique
pcd	o3d.geometry. PointCloud	Variable	Objet open3d à partir de combined_points
point_cloud	liste	Variable	Variable discrète de la boucle for stockant le nuage en cour de capture
point_clouds	liste	Variable	Ensemble des nuages capturés
points_data	liste	Variable	Données envoyées
points_per_selected_box	Liste[liste]	Variable	Point restant organisés par appartenance à une même boite
position	liste	Variable	Position du marqueur
radius	float	paramètre	Taille du voisinage pour le filtrage par densité
remaining_boxes_indices	Liste[int]	Variable	Indices des boites restantes
remaining_points	Liste	Variable	Points restant après la selection
rotation_matrix	Cv2.rodrigues	Variable	Matrice de rotation pour passer de caméra à marqueur
rvecs	liste	Variable	Vecteur de rotation du marqueur dans le repère caméra
selected_bounding_boxes	Liste d'objets aabb	Variable	Boites restantes après sélection
selected_boxes	Liste[int]	Variable	Indices des boites supprimées
taille_marqueur	float	Variable	Taille du marqueur ArUco en m
threshold	int	paramètre	Nombre de voisin minimal pour le filtrage par densité
transformed_points	Tableau np	Variable	Points après filtrage par appartenance
translation	Cv2.rodrigues	Variable	Matrice de translation pour passer de caméra à marqueur
tvecs	liste	Variable	Vecteur de translation du marqueur dans le repère caméra

V.3 – Variables et paramètres du traitement dynamique

Les variables suivantes sont communes à celles de reconstruction, elle ne seront pas dans ce tableau :

camera_matrix, color_frame, color_images, depth_frame, corners, ids, gray, position, rotation_matrix, translation, rvecs, tvecs, taille_marqueur

Nom	**type**	**Variable/Paramètre**	**Utilisation**
bbox_data	liste	variable	Données exportées
birch_model	Objet sklearn		Modele de clustering
centers	liste	Variable	Liste des centres des boites statiques
cluster_labels2	liste	Variable	Liste des numéro de cluster associés à chaque point
clusters2	liste	Variable	Liste des points organisé par appartenance aux clusters
data	liste	variable	Données exportées
df_bboxes	Data frame panda	variable	
df_clusters	Data frame panda	variable	
filtered_points_cp	Tableau cp	Variable	Liste des points après filtrage
filtered_points_np	Tableau np	Variable	Liste des points après filtrage
kdtree	Arbre kd	Variable	Arbre Kd utilisé pour organiser la recherche de voisin proche
num_clusters2	int	Variable	Nombre de clusters identifiés
ply_file_path	str	Paramètres	Chemin vers le fichier ply
point_cloud	Objet point coud open3d	variable	Nuage récupéré de la scène statique
points_cp	Tableau cp	variable	Nuage récupéré de la scène statique
points_cpT	Tableau cp	variable	Nuage dynamique déplacé
points_cpT_downsampled	Tableau cp	variable	Nuage dynamique réduit
points_np	Tableau np	variable	Nuage récupéré de la scène statique
points_np2			
th2	float	Paramètre	Paramètre du clustering, taille du voisinage d'un point
Zones_dynamique	Liste	Variables	Boites de la zone statique
Zones_fixes	Liste	Variables	Boites de la zone dynamqiue

VI – Historique des versions

1.0 :

- Reconstruction envoi seulement le nuage de points sans traitements
- Traitement dynamique : à 5s de traitement, exporte 1 csv/itération

1.1 :

- Ajout d'un down sampling
- Passage au GPU du maximum de variables
- 1 seul CSV écrasé à chaque itération
- Préparation des données avant l'écriture au lieu de tout écrire d'un coup
- Temps de traitement à 0.30s

1.2 :

- Ecriture en PARQUET
- Filtrage sur GPU
- Ajout de l'affichage des photos prises lors de la reconstruction
- Calcul des Bounding Boxes dynamiques et exportation en PARQUET

1.3 :

- Filtrage par densité sur la reconstruction

2.0 :

- Reconstruction avec primitive par optimisation puis interface de sélection
- Demande de satisfaction lors de la capture
- Reconstruction exporte un .ply et 2 PARQUET
- Le traitement dynamique utilise les boites du traitement statique