Détection et Localisation

I. Introduction

L'objectif de ce TP est de détecter un objet en 2D puis de le localiser dans l'espace 3D à l'aide d'un système mono-caméra supposé étalonné au préalable.

Nous proposons de décomposer cette tâche en trois étapes :

- La création d'une image de référence pour la détection de l'objet,
- La détection et le tracking de la surface de l'objet dans l'image,
- Le calcul de la position de l'objet par rapport à la caméra

II. Création d'une image de référence

Nous allons dans un premier créer une image de référence de notre objet planaire :

- Acquérir une image de l'objet en le prenant en photo.

Afin de pouvoir utiliser cette image comme plan de référence, nous allons devoir corriger la perspective. Nous utiliserons pour cela la classe `PerspectiveCorrection` du fichier `Homography.py`.

Nous allons d'abord développer une interface permettant à l'utilisateur de sélectionner les quatre coins du plan de référence :

- Compléter la 'callback' `mouseCallback`. Il faut ici récupérer un clic gauche sur la souris, dessiner un point à l'endroit sélectionné avec `cv.circle()`, afficher l'image, puis rajouter le point à la liste des points sélectionnés.

Ensuite, nous allons corriger la perspective afin de créer notre image de référence :

1. Quelles fonctions d'OpenCV peut-on utiliser pour calculer l'homographie et corriger la perspective ? Implémenter ces fonctions dans `PerspectiveCorrection.process()`. Quel sont les points source ? Les points cible ? Sauvegarder l'image obtenue.

III. Tracking

Dans cette partie nous allons essayer de détecter notre objet de référence dans un flux vidéo et de suivre son mouvement.

Pour cela, on se propose de détecter et de matcher des features 2D (ORB et AKAZE) :

- Nous allons dans un premier temps créer nos détecteurs dans la fonction `Tracker.__init__()` du fichier `Tracking.py`.
2. Implémenter les fonctions de détection de 'features' (`Feature2D.detectAndCompute()`) et d'appariement (`DescriptorMatcher.knnMatch()`) dans la méthode `Tracker.detectAndMatch()`. À quoi correspondent les 'matchs' ?
3. Filtrer ces 'matchs' en implémentant le test du ratio de Lowe dans la fonction `Tracker.getCorrespondancePoints()`. Comment les matchs sont-ils filtrés par ce test?
4. Tester l'algorithme en utilisant la fonction `Tracker.display()`. Que voit-on s'afficher ? Faire varier le paramètre « Matching Ratio », que se passe-t-il ? Comment l'expliquer ?

Nous allons maintenant détecter notre objet et suivre son mouvement :

- Calculer l'homographie entre l'image de référence et l'objet observé dans la fonction `Tracker.computeHomography()`.
- Implémenter dans `Tracker.computeObjectCorners()` une transformation de perspective afin de calculer la position des coins de l'objet dans l'image à partir de l'homographie calculée précédemment.

5. Visualiser les résultats à l'aide de la fonction `Tracker.display()`, il faut pour cela implémenter le calcul de l'homographie et l'affichage du contour de l'objet. Que voit-on s'afficher ? Quel problème apparaît ? Proposer une solution à ce problème.

- Nous allons 'stabiliser' la détection à l'aide de la fonction `cv.accumulateWeighted()` d'OpenCV. L'implémenter dans `Tracker.computeHomography()` et visualiser les nouveaux résultats.

6. Faire varier les différents paramètres et commenter leurs effets. Faire varier la taille de l'image de référence à l'aide de la fonction `resize()` d'OpenCV, commenter.

IV. Détection de pose

Enfin, nous voulons calculer la position de l'objet en 3D dans la scène et projeter sa 'bounding box' sur l'image pour la visualiser.

- Dans le constructeur de `PoseEstimator` du fichier `PoseEstimation.py`, remplir la variable `boxPoints` à l'aide des dimensions de l'objet à détecter.

Nous allons implémenter la fonction `PoseEstimator.computePose()` qui doit permettre de calculer la position de l'objet par rapport à la caméra :

7. Qu'est-ce que le problème PnP ? Le résoudre à l'aide d'OpenCV, utiliser pour cela les paramètres intrinsèques de la caméra obtenus lors du TP Perception 3D.
8. Comment obtenir la position de la caméra par rapport à l'objet ? Quelles applications pourrait-on envisager ?

Enfin, nous allons projeter les points 3D de l'objet sur l'image afin de dessiner sa 'bounding box' :

- Implémenter la fonction `cv.projectPoints()` dans la méthode `PoseEstimator.drawObjectBox()`.

9. Visualiser les résultats à l'aide de la fonction `PoseEstimator.display()`. Ajouter une capture d'écran dans le rapport.