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:

1. 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`.

2. Quelle fonction d'OpenCV peut-on utiliser pour calculer la transformation à appliquer à l'image afin de supprimer les effets de perspective ? Quel sont les points source ? Les points cible ? Implémenter cette fonction dans la méthode `PerspectiveCorrection.process()`.

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

3. 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.
4. Enfin, nous allons corriger la perspective afin de créer notre image de référence. Quelle fonction d'OpenCV peut-on utiliser ? Implémenter cette fonction dans `PerspectiveCorrection.process()`.
5. 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 feature 2D (ORB et AKAZE) :

6. Citer d'autres algorithmes de détection/description de features. Pourquoi utiliser ORB et AKAZE ici ?
7. Nous allons dans un premier temps créer nos détecteurs dans la fonction `Tracker.__init__()` du fichier `Tracker.py`.
8. Implémenter la détection et le calcul des descripteurs des 'features' de l'image de référence.
9. Implémenter les fonctions de détection de 'features' (`Feature2D.detectAndCompute()`) et d'appariement (`DescriptorMatcher.knnMatch()`) dans la méthode `Tracker.detectAndMatch()`. À quoi correspondent les 'matchs' ?

10. 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?
11. 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 :

12. Calculer l'homographie entre l'image de référence et l'objet observé dans la fonction `Tracker.computeHomography()`.
13. Implémenter dans `Tracker.drawObjectContours()` une transformation de perspective afin de calculer la position des coins de l'objet dans l'image à partir de l'homographie calculée précédemment.
14. Visualiser les résultats à l'aide de la fonction `Tracker.display()`. Que voit-on s'afficher ? Quel problème apparaît ? Proposer une solution à ce problème.
15. Nous allons 'stabiliser' la détection à l'aide de la fonction `cv.accumulateWeighted()` d'OpenCV. Que fait cette fonction ? L'implémenter dans `Tracker.computeHomography()` et visualiser les nouveaux résultats.
16. 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.
17. (Bonus) Nous voulons pouvoir garder en mémoire la dernière position connue de l'objet pendant quelques frames afin de limiter l'effet des pertes de détection temporaires. Implémenter ce mécanisme.

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.

18. Dans la fonction `PoseEstimator.__init__()` 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 :

19. Rappeler ce qu'est le problème PnP.
20. Calculer les coordonnées des coins de l'objet dans l'image à l'aide de `cv.perspectiveTransform()`.
21. Résoudre le problème PnP à l'aide d'OpenCV, utiliser pour cela les paramètres intrinsèques de la caméra obtenus lors du TP Perception 3D.
22. Comment obtenir la position de la caméra par rapport à l'objet ? Quelles applications pourrait-on envisager ?
23. (Bonus) Implémenter la solution et afficher les coordonnées de la caméra.

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

24. Implémenter la fonction `cv.projectPoints()` dans la méthode `PoseEstimator.drawObjectBox()`.
25. Visualiser les résultats à l'aide de la fonction `PoseEstimator.display()`.