Cours « Numérisation et Modélisation 3D »

Travaux Pratiques Informatiques

Visualisation de nuages de points, algorithme ICP

Logiciels Python et CloudCompare

F. Goulette

Version : septembre 2020

1/ Préambule

Objectifs du TD :

1/ Visualiser des nuages de points sous CloudCompare / Python

2/ Modifier et sauvegarder des nuages de points
- Décimation
- Translation, Rotation

3/ Implémenter le calcul de meilleure transformation rigide entre nuages de points appariés

4/ Tester la méthode ICP

Logiciels à installer

CloudCompare
OpenSource – disponible sous Windows 64, MacOs 64, Linux 64 – Ubuntu et « universel »

Python / Spyder
- Interpréteur Python > 3.6 ;
- **IDE Python conseillée : Spyder > 4 (Anaconda)**
- Modules Python publics nécessaires : **numpy, matplotlib, mpl_toolkits, sklearn**
- Module Python fourni : **utils.ply**

Annexe : données, tableaux et fonctions Numpy

1/ Visualisation de nuages de points

CloudCompare

Lancer CloudCompare.

Charger le fichier « bunny.ply » (lapin de Stanford) (File->Open)
Tourner (clic-gauche), translater (clic-droit), zoomer (molette).

Fermer le fichier (File->Close all) et charger le fichier « Notre_Dame_Des_Champs_1.ply »
(boulevard du Montparnasse et église Notre Dame des Champs »
Visualisation : activer le bouton EDL (Eye Dome Lighting) pour donner un rendu plus lisible.

Python

Ouvrir le fichier TP_ICP.py dans Spyder.

Le fichier est un « code à trou ». Il peut être exécuté mais doit être complété pour le TP.

Le début du fichier charge les modules nécessaires. Ensuite, des fonctions sont définies.
Enfin, il y a une partie principale « Main »

Certaines parties du code sont mises en veille par une condition « if False » qu'il suffit d'activer en la modifiant en « if True ».

Tester l'exécution du fichier (flèche verte dans Spyder). Le fichier peut être copié-collé (en totalité ou en partie) et exécuté dans la fenêtre d'exécution de Spyder.

Dans le « Main » activer la partie de visualisation du nuage de points en indiquant « if True ».
Exécuter le fichier.

Aide en ligne sur Python :
`help()`
Puis taper la commande recherchée, ou un mot-clé

Voir rappels sur tableaux et fonctions Numpy en Annexe.

2/ Décimation de nuages de points

Décimation

Il s'agit de sous-échantillonner le nuage de points d'un facteur d'échelle k_ech (par exemple 10).

Sous-échantillonnage avec boucle « for »

Ecrire une première méthode dans la fonction « decimate », en utilisant une boucle « for ».
Astuce : regarder l'annexe sur l'utilisation des fonctions de concaténation vstack et hstack.
Dans le « Main » activer cette partie en indiquant « if True ».
Exécuter le fichier.

Visualiser le résultat sous python et sous CloudCompare.

Sous-échantillonnage avec fonction avancée de Numpy array

Ecrire une 2e version de l'échantillonnage en utilisant les sous-tableaux de Numpy array (voir Annexe).

Visualiser le fichier de points échantillonnés avec Python :

Sauvegarder dans un nouveau fichier le fichier échantillonné et le visualiser sous CloudCompare.

Appliquer le sous-échantillonnage d'un facteur 1000 au fichier "Notre Dame des Champs" et le visualiser sous CloudCompare.
Activer la partie correspondante de la fonction « main » par « if True: »

3/ Translation, Rotation

Translation

Effectuer une translation d'un vecteur [0, -0.1, 0.1] (utiliser np.array) et visualiser le résultat.

Centrer

Centrer le nuage de points (utiliser np.mean) et visualiser le résultat.
Aide en ligne : help(np.mean)

Echelle

Diviser par une echelle de 2 et visusaliser le résultat.

Rotation

Définir une matrice de rotation R de dimension 3x3, d'un angle theta = pi/3 autour de l'axe Z (utiliser np.pi, np.cos, np.sin, np.array).

```
R =  [cos(theta),   -sin(theta),   0],
     [sin(theta),   cos(theta),    0],
     [0,            0,             1]])
```

Appliquer la rotation (fonction .dot) sur le nuage de points centré, puis re-translaté à sa position d'origine et visualiser le résultat.

4/ Transformation rigide entre nuages de points appariés

Il s'agit de trouver la rotation et translation permettant de recaler deux nuages de points appariés, meilleure transformation rigide au sens des moindres carrés.

On recherche la transformation (R,t) qui minimise :

$$f(R,t) = \frac{1}{n}\sum_{i=1}^{n}\left[\vec{p}_i - \left(R\vec{p}'_i + t\right)\right]^2$$

Fonction à écrire : **best_rigid_transform(data, ref)**

Pour l'exécuter, activer ensuite « if True » dans le code principal (main) correspondant.
Le code renvoie les erreurs quadratiques moyennes avant et après transformation rigide.

Etapes :

Déterminer les barycentres p_m et p'_m
Calculer les points qi et q'i

Calculer la matrice H

$$H = \sum_{i=1}^{n} q_i^{'} q_i^{T}$$

Décomposer H en valeurs singulières

$$H = U\Sigma V^T$$

Utiliser la fonction SVD (Numpy : `np.linalg.svd`)

Calculer R puis t

$$R = VU^T$$

$$t = p_m - Rp_m^{'}$$

5/ Recalage par ICP

Il s'agit de trouver la rotation et translation permettant de recaler deux nuages de points en recouvrement partiel.

On recherche la transformation (R,t) qui minimise :

$$f(R,t) = \frac{1}{n} \sum_{i=1}^{n} \left[\vec{p}_i - \left(R\vec{p}'_i + t \right) \right]^2$$

Attention : le nombre de points peut varier à chaque itération.
La fonction des moindres carrés est normalisée.

Pseudo-code ICP

- Recalage initial (NP, NP')
- Repéter :
  - Association de données →(P, P')
  - Calcul de la transformation (R,T)
  - Application de la transformation au nuage NP'
  - Calcul de la distance entre nuages
- Tant que :

(distance normalisée entre nuages > seuil)
et (nombre d'itérations < nb_max)

Entrée : Jeux de n points (P, P')

Sortie : matrice de rotation R, vecteur t

Etape 1 : Appariement des nuages de points

Détermination des points appariés (critère de distance minimale)
[X1_ap,X2_ap,n_ap,f]=appariement[X1,X2,dmin]

X1_ap et X2_ap ont le même nombre de points (n_ap)
X1 et X2 n'ont pas nécessairement le même nombre de points.
f est la valeur de la fonction à minimiser

Etape 2 : fonction de calcul de R et T à partir de points 3D appariés – voir fonction best-rigid-transform

Etape 3 : Calcul itératif de R et T

Calcul itératif : appariement, calcul de f, R, T
Critère d'arrêt pour une valeur faible ou variant peu.

Tests

Tracer l'évolution de f en fonction des itérations

La fonction **icp_point_to_point** est déjà rédigée et sera fonctionnelle quand best_rigid_transform le sera.
Activer alors simplement la partie correspondante du code principal (main).
Le code renvoie une figure de la variation de l'erreur quadratique moyenne en fonction des itérations.

Faire les tests sur le nuage bunny.

Ainsi que sur les fichiers Notre Dame des Champs 1 (référence) et 2 (modifié), éventuellement échantillonnés.

Annexe : données, tableaux et fonctions Numpy

Visusalisation de données, type de données, forme des données :

```
print(decimated_points)
print(type(bunny_o))
print(decimated_points.shape)
```

Vecteurs de Numpy Array :
np.array([1, 5, 12]).reshape((-1,1))

Sous-tableaux de Numpy Array :

Voir : https://jakevdp.github.io/PythonDataScienceHandbook/02.02-the-basics-of-numpy-arrays.html

1ère colonne :
`bunny_o[ :, 0]`

(i+1)ème colonne :
`bunny_o[:, i]`

3 premières colonnes :
`bunny_o[:,0:3]`

Sélection des colonnes avec un incrément régulier :
`bunny_o` [:, ::indent]

Concaténation :

Voir : https://docs.scipy.org/doc/numpy/reference/generated/numpy.vstack.html

```
np.vstack((bunny_o[3, :], bunny_o[0, :]])
```