ROB312-TP1

Pan Mengyu

December 2020

1 Introduction

Dans ce TP, on va implémenter la méthode ICP et ses variants pour associater les objets dans deux images et on va tester les résultats obtenus. Les donées sont fournis par U2IS(Unité d'Informatique et d'Ingénierie des Systèmes)

2 Méthode

La méthode ICP s'appelle 'Iterated Closest Point'[1]. Il y a trois étapes principales. La premiere étape est de supprimer les points très proches. Ensuite, il recherche les paires de points entre deux nuage de points. On peut achiever cet étape soit par les voisions plus proches ou soit par 'normal shooting' par exemple. À la fin, il associate les deux nuages de points par les paires obtenues.

3 Code et Résultat

3.1 Q1

Dans la premiere étape, donc je calcule la distance entre chaque deux points et supprime les points très proches.

```
dat_filtX = []
dat_filtY = []
dat_filt = []
for i in range(len(dat[1]) - 1):
    Choose = True
    for j in range(i+1,len(dat[1])):
        Distance = np.sqrt(pow(dat[0][i] - dat[0][j],2) +
            pow(dat[1][i] - dat[1][j],2))
        if Distance < MinDistance:
            Choose = False
    if Choose:
        dat_filtX.append(dat[0][i])
        dat_filtY.append(dat[1][i])
dat_filtX.append(dat[0][len(dat[0])-1])
```

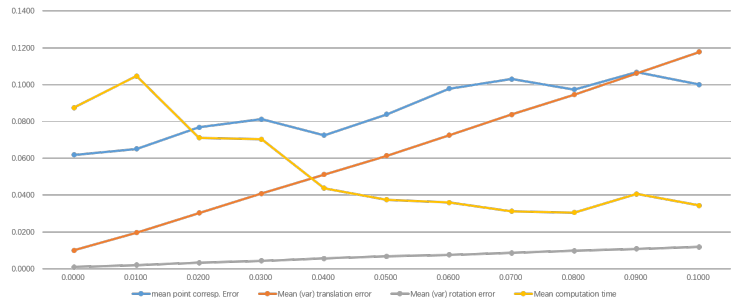

Figure 1: Résultat de paramètre Q1

```
dat_filtY.append(dat[1][len(dat[1])-1])

dat_filt.append(dat_filtX)
dat_filt.append(dat_filtY)
dat_filt = np.array(dat_filt)
```

J'ai testé la distance entre 0.01 et 0.1 dans la figure 1. J'ai choisi 0.02 comme la distance minimale car il y a moins des errreurs que la distance 0.01 et il est plus rapide.

3.2 Q2

Dans cette partie, je choisis les points plus proches. D'abord, je calcule la distance entre deux nuage de points par kd-trees. Ensuite, je choisis les paires de points qui est plus proche.

```
    Index = 0.5
    dat_matchedx = []
    dat_matchedy = []
    dat_matched = []
    best_matched = []

    length = math.ceil(len(dat_filt[0]) * Index)

    ##nearest shooting
    Max = np.max(distance)
    for i in range(length):
        j = np.argmin(distance)
        best_matched.append(j)
        distance[j] = Max

    for i in range(length):
        dat_matchedx.append(dat_filt[0][best_matched[i]])
        dat_matchedy.append(dat_filt[1][best_matched[i]])
```

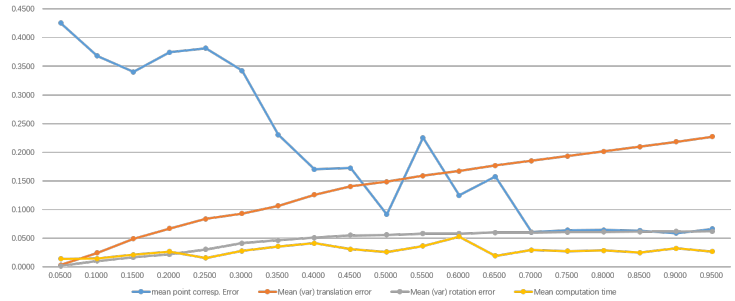

Figure 2: Résultat de paramètre Q2

```
dat_matched.append(dat_matchedx)
dat_matched.append(dat_matchedy)
dat_matched = np.array(dat_matched)
distance, index = tree.query(dat_matched.T)
```

Pour choisir les paires plus proches, j'utilise une variable Index. J'ai testé l'Index entre 0.05 et 0.95 dans la figure 2. L'Index 0.5 est le meilleur car les erreurs et le temps du calcul sont plus petits.

3.3 Q3

Dans cette partie, je le fait avec la variable step = [1,3,5,10,15,20]. Les résultats sont dans la figure 3. Avec l'augmentation du step, les informations d'images et les temps d'éxecution diminuent.

step	Temps(minute)
1	24
3	8
5	4
10	3
15	2
20	1

Table 1: relation entre step et temps d'éxecution

3.4 Q4

Pour faire 'normal shooting', je calcule la vecteur de point entre deux nuage de points et la ligne tangente de ce point. Donc, je peux calcule le cosinus entre ces deux vecteur et choisis le minimal.

```
Index = 0.4
```

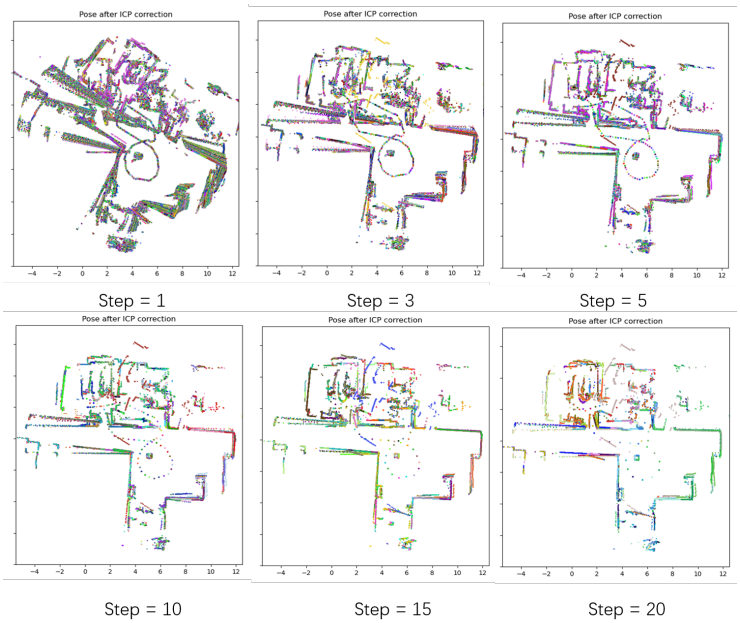

Figure 3: Résultat de performance Q3

```
dat_matchedx = []
dat_matchedy = []
dat_matched = []
best_matched = []

length = math.ceil(len(dat_filt[0]) * Index)

##normal shooting
index = np.zeros(length, dtype = np.int16)
distance = np.zeros(length)
CosDistance = np.zeros(len(ref))
point = np.zeros(2)

for i in range(length):
    point = np.array([dat_filt[0][i],dat_filt[1][i]])
    vector =
        np.array([(dat_filt[0][i+1]-dat_filt[0][i]),(dat_filt[1][i+1]-dat_filt[1][i])])
    vector1 =
        np.array([(ref.T[i][0]-dat_filt[0][i]),(ref.T[i][1]-dat_filt[1][i])])
    for j in range(len(ref)):
        CosDistance[j] =
            abs(vector1[0]*vector[0]+vector1[1]*vector[1])/np.sqrt(pow(vector[0],2)
            + pow(vector[1],2))
    index[i] = int(np.argmin(CosDistance))
```

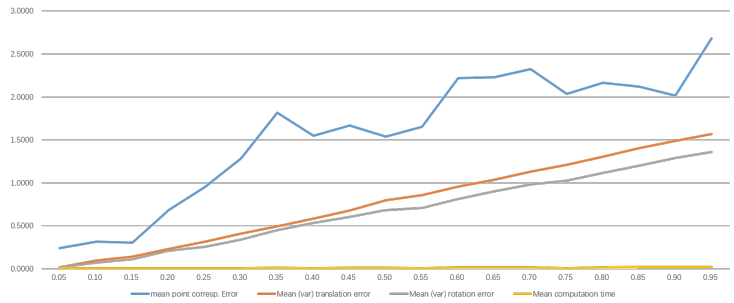

Figure 4: Résultat de paramètre Q4

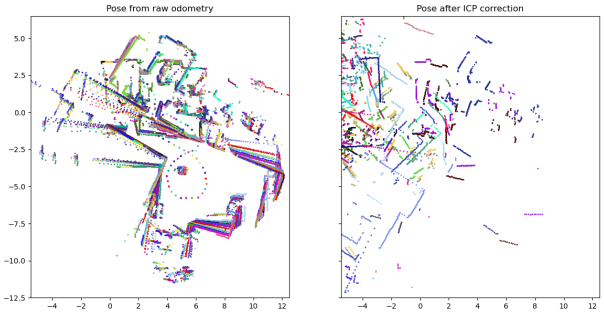

Figure 5: Résultat de performance Q4

```
    #print(index[i])
    distance[i] = np.min(CosDistance)
meandist = np.mean(distance)
for i in range(length):
    dat_matchedx.append(dat_filt[0][int(index[i])])
    dat_matchedy.append(dat_filt[1][int(index[i])])

dat_matched.append(dat_matchedx)
dat_matched.append(dat_matchedy)
dat_matched = np.array(dat_matched)
distance, index = tree.query(dat_matched.T)
```

Je refais le Q2 et choisis Index = 0.4 car il est meilleur dans la figure 4.Mais le résultat obtenu dans la figure 5 n'est pas bon. Je pense que la méthode 'normal shooting' est moins capable de treater les points discrète.

References

[1] Paul J.; N.D. McKay Besl. A method for registration of 3-d shapes. *EEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2)(239–256), 1992.