Temporal Consistent 2D-3D Registration Project Report

Peking University School of Physics 刘浚哲 1500011370

1.Problem Statement

In this project, we're dealing with image registration. Image registration is the process of transforming different sets of data into one coordinate system. Data may be multiple photographs, data from different sensors, times, depths, or viewpoints. Registration is necessary in order to be able to compare or integrate the data obtained from these different measurements.

When handling volumetric Image representation, we introduce the cubic B-spline-based interpolation and PCA techniques. The cubic B-spline-based interpolation defines a nonrigid deformation M, then after applying PCA algorithm, the volumetric image could be represented by the low dimensional subspace coordinate α, and the associated projection matrix P:

$$V = f(\bar{V}, M(\alpha P)) \tag{1}$$

To estimate 3D volumetric images of a series of LCRs captured at a different time, temporal constraints were introduced to refine the registration, the objective function is defined as :

$$E(\alpha) = E_c + \gamma E_r \tag{2}$$

the first term is :

$$E_c(\alpha) = \sum_{i}^{K} d^2\left(I_i, I_i'\right) \tag{3}$$

denoting the image distance between I_i and the deformed volumetric image $I_i^{'}$ using the L_2 metric. The second regularization term E_r aims at keeping the consistency of the images at different times. It can be expressed in the following way:

$$E_r(\alpha) = \sum_{i,j=1}^{K} s_{ij} \left\| U \circ \left(M\left(\alpha_i P\right) - M\left(\alpha_j P\right)^2 \right.\right. + \mu \sum_{i=1}^{K} \left\|\alpha_i\right\|_1^2 \tag{4}$$

where α_i denotes the deformation parameters of the i-th LCR, and s measures the similarity of the input LCRs:

$$s_{ij} = \exp(-d^2(I_i, I_j)) \tag{5}$$

U is a mask represented by a matrix of the same size as the reference volume Vr, it is set to 1 when the corresponding voxel is inside the stable structure, and 0 otherwise. We set the constant $\mu = 10^4$. And the Gauss-Newton method is used to solve the optimization problem, the residual function F_r is defined as follows:

$$F_r\left(\alpha_i\right) = \begin{pmatrix} \sqrt{s_{i,j}}\left[U \circ \left(M\left(\alpha_i P\right) - M\left(\alpha_j P\right)\right)\right] \\ \left\|\alpha_i\right\|_1 \end{pmatrix} \tag{6}$$

In each iteration, the subspace coordinate is updated by the newly estimated $\delta\alpha$, and it equals to:

$$\delta\alpha = \left(J'J\right)^{-1} J'F(\alpha) \tag{7}$$

where J is the Jacobian matrix of F_r with partial derivative defined as follows:

$$\frac{\partial F_r}{\partial \alpha_i} = \begin{pmatrix} \dfrac{\sqrt{s_{i,j}}\left[U \circ \left(M\left(\alpha_i + \delta\alpha_i\right) P\right) - M\left(\alpha_i P\right)\right)}{\delta a_i} \\ \operatorname{sign}\left(\alpha_i\right) \end{pmatrix} \tag{8}$$

2.Algorithm

In solving this problem, we mainly use the Gauss-Newton method. The Gauss–Newton algorithm is used to solve non-linear least squares problems.

Given m functions $r = (r_1, \ldots, r_m)$ (often called residuals) of n variables $\beta = (\beta_1, \ldots, \beta_n)$, with $m \geq n$, the Gauss–Newton algorithm iteratively finds the value of the variables that minimizes the sum of squares:

$$S(\boldsymbol{\beta}) = \sum_{i=1}^{m} r_i^2(\boldsymbol{\beta}). \tag{9}$$

Starting with an initial guess $\boldsymbol{\beta}^{(0)}$

for the minimum, the method proceeds by the iterations:

$$\boldsymbol{\beta}^{(s+1)} = \boldsymbol{\beta}^{(s)} - \left(\mathbf{J_r}^{\mathsf{T}}\mathbf{J_r}\right)^{-1}\mathbf{J_r}^{\mathsf{T}}\mathbf{r}(\boldsymbol{\beta}^{(s)}), \tag{10}$$

where, if r and β are column vectors, the entries of the Jacobian matrix are:

$$(\mathbf{J_r})_{ij} = \frac{\partial r_i(\boldsymbol{\beta}^{(s)})}{\partial \beta_j}, \tag{11}$$

and the symbol $^{\mathsf{T}}$ denotes the matrix transpose.

3.Numerical Solution

3.1.Given image $I_i, I_i^{'}$, compute image distance using L2 and L1. Plot the difference image.

We use Pillow to read two images in python, and by subtracting two matrix, we will obtain the difference matrix, and then we'll be able to compute the distance and plot the image.the image distance are:

$$\begin{aligned} D_{L2} &= 4640.006538494603 \\ D_{L1} &= 13392.0 \end{aligned} \tag{12}$$

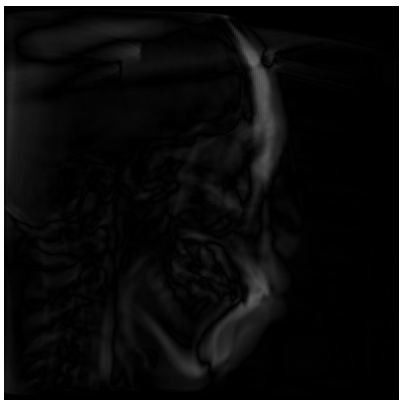


figure 1: difference image

The difference image is shown in figure 1. The following is the code.

```
In [3]: from PIL import Image
        import numpy as np
        import os

In [4]: path = os.getcwd()
        im1=Image.open(path+'/2D Images/1.png')
        im2=Image.open(path+'/2D Images/2.png')
        print(im1.format,im1.size,im1.mode)
        print(im2.format,im2.size,im2.mode)

        PNG (400, 400) L
        PNG (400, 400) L

In [5]: image1 = np.array(im1)
        image2 = np.array(im2)

In [9]: difference = np.zeros_like(image1)
        for i in range(400):
            for j in range(400):
                difference[i][j] = abs(int(image1[i][j])-int(image2[i][j]))

In [10]: norm2 = np.linalg.norm(difference, ord=2)
         print(norm2)

         4640.006538494603

In [11]: norm1 = np.linalg.norm(difference, ord=1)
         print(norm1)

         13392.0

In [12]: diff_image = Image.fromarray(difference)
         diff_image.show()
         diff_image.save('diff_image.png')
```

3.2.Show the derivatives of Eq. (6) with the form of Eq. 8).

Proof: From Eq.(6), we have :

$$\frac{\partial F_r}{\partial \alpha_i} = \begin{pmatrix} \frac{\partial}{\partial \alpha_i}\sqrt{s_{i,j}}\left[U \circ \left(M\left(\alpha_i P\right) - M\left(\alpha_j P\right)\right)\right] \\ \frac{\partial}{\partial \alpha_i}\left\|\alpha_i\right\|_1 \end{pmatrix} \tag{13}$$

and then:

$$\frac{\partial F_r}{\partial \alpha_i} = \begin{pmatrix} \sqrt{s_{i,j}}\left[U \circ \left(\frac{\partial}{\partial \alpha_i} M\left(\alpha_i P\right)\right)\right] \\ \operatorname{sign}\left(\alpha_i\right) \end{pmatrix} \tag{14}$$

for a ministep $\delta\alpha$, we have:

$$\frac{\partial}{\partial \alpha_i} M\left(\alpha_i P\right) = \frac{M\left(\alpha_i + \delta\alpha_i\right) P - M\left(\alpha_i P\right)}{\delta a_i} \tag{15}$$

therefore, at last we have:

$$\frac{\partial F_r}{\partial \alpha_i} = \begin{pmatrix} \frac{\sqrt{s_{i,j}}\left[U \circ \left(M\left(\alpha_i + \delta\alpha_i\right) P\right) - M\left(\alpha_i P\right)\right)}{\delta a_i} \\ \operatorname{sign}\left(\alpha_i\right) \end{pmatrix} \tag{16}$$

3.3.Show how to use Gauss-Newton method to solve the objective function $E(\alpha) = E_c + \gamma E_r$.

Our target function statisfies the criteria of Gauss-Newton method. Since $K = 1$, therefore the first term of $E_r = 0$, according to Gauss-Newton method, the solution can be acquired using the iteration method:

$$\alpha = \alpha^{(0)} - (J'J)^{-1} J'F(\alpha) \tag{17}$$

The Jacobian Matrix element also contains $\delta\alpha$. Therefore, we can initialize $\alpha^{(0)}$ as a zero vector, and $\delta\alpha^{(0)}$ as a non-zero vector, and conduct the above iteration. The solution will converge to the optimal α^*.

3.4.Given α_1 and α_2 regarding volumetric images of two ages, compute the regularization loss E_r

We first use Matlab and BSplineTransform.m to process "Ref.mha" Using "COEF50.mat" and "mean$_X$Matrix50.mat" as input parameters. We then get "1.mha" and "2.mha" respectively representing the images generated by α_1 and α_2. We then use the Eq.(4) to compute the loss. U can be acquired from "Mask.mha", the result shown as follows:

$$E_r = 75534074482.98254 \tag{18}$$

In [48]:
```
import numpy as np
import math
from PIL import Image
```

In [49]:
```
import SimpleITK as sitk
import os
```

In [50]:
```
path = os.getcwd()
alpha1 = np.loadtxt(path+"/Alpha/1.txt")
alpha2 = np.loadtxt(path+"/Alpha/2.txt")
```

In [51]:
```
Alpha1P = sitk.ReadImage(path+"/CheckV/1.mha")
Alpha2P = sitk.ReadImage(path+"/CheckV/2.mha")
Mask = sitk.ReadImage("Mask.mha")
```

In [52]:
```
M_alpha1P = sitk.GetArrayFromImage(Alpha1P)
M_alpha2P = sitk.GetArrayFromImage(Alpha2P)
mask = sitk.GetArrayFromImage(Mask)
```

In [53]:
```
print(M_alpha1P.shape)
for i in range(M_alpha1P.shape[0]):
    alpha1_i = Image.fromarray(M_alpha1P[i])
    alpha1_i = alpha1_i.convert("L")
    alpha1_i.save(path+"/CheckV/alpha1_"+str(i+1)+".jpg")
    alpha2_i = Image.fromarray(M_alpha2P[i])
    alpha2_i = alpha2_i.convert("L")
    alpha2_i.save(path+"/CheckV/alpha2_"+str(i+1)+".jpg")
```

```
(128, 128, 128)
```

In [54]:
```
ref = sitk.ReadImage("Ref.mha")
Ref = sitk.GetArrayFromImage(ref)
print(Ref.shape)
for i in range(Ref.shape[0]):
    ref_i = Image.fromarray(Ref[i])
    ref_i = ref_i.convert("L")
    ref_i.save(path+"/CheckV/ref_"+str(i+1)+".jpg")
```

```
(128, 128, 128)
```

In [55]:
```
X = np.zeros_like(Ref)
for i in range(128):
    for j in range(128):
        for k in range(128):
            X[i][j][k] = abs(mask[i][j][k] * (int(M_alpha1P[i][j][k])-int(M_alpha2P[i][j][k])))
            X[i][j][k] = pow(X[i][j][k], 2)
Er1 = np.sum(X)
print(Er1)
```

```
349444825
```

In [58]:
```
mu = 1e4
a1 = np.linalg.norm(alpha1, ord=1)
a2 = np.linalg.norm(alpha2, ord=1)
Er2 = mu*(pow(a1, 2)+pow(a2, 2))
print(Er2)
```

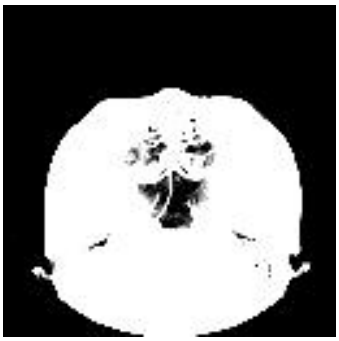
```
75184629657.98254
```

In [59]:
```
Er = Er1 + Er2
print(Er)
```

```
75534074482.98254
```

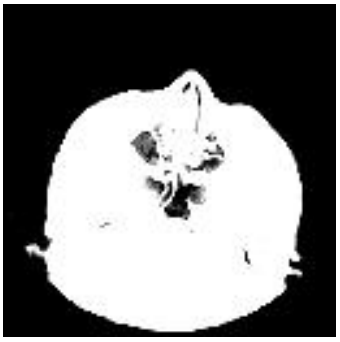
5.plot one slice of ACB regarding α_1 and α_2 side by side.

3 pictures : Ref, α_1 generated, α_2 generated shown respectively:



Reference

Reference



Reference

Obviously, 3 pictures are similar to each other, therefore are consistent.

4.Conclusion

In this project, I proved that Gauss-Newton method could be introduced into the Image Registration process and computed the image distance given two images. Using cubic spline interpolation, I also obtained the images generated by α_1 and α_2 respectively, and calculated the loss E_r. Finally, I compared the 3 images between the original image, α_1-generated, and α_2-generated, and confirmed that they're consistent.

References:

[1] Munsky B, Neuert G, van Oudenaarden A (2012) Using gene expression noise to understand gene regulation. Science 336(6078):183-187.