COMP0119 : Acquisition and processing of 3d geometry

Oluwatosin Alabi

February 2020

General information

Question attempted/completed

I completed Questions 1, 2, 3, 4, 6. I was able to partially complete question 5, I was able to align 4 out of 6 meshes.

libraries used

Scipy, Numpy, pyrender, trimesh, scipy.spatial.cKDtree, matplotlib

Solutions

Question 1a

The ICP algorithm performs alignment between pairs of "closest points" by minimizing the function $min_{R,t} \sum_i \|P_i - Rq_i - t\|^2$. I implemented the Basic ICP algorithm by

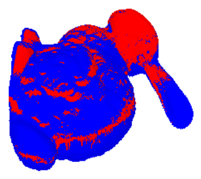
1. Selecting a set of points p_i . I used all the vertices in the bun000_v2(M1).
2. match each point in p_i to the closest point q_i on the other scan. I utilized the cKDTree from scipy.spatial to perform quick nearest neighbour searches between p_i and q_i.
3. Reject bad pairs of (p_i, q_i). I rejected points pairs whose distance between $p_i and q_i$ was larger than two standard deviation above the mean of all distances between $p_i and q_i$.
4. Compute rotation **R** and translation **t** to minimize. $min_{R,t} \sum_i \|P_i - Rq_i - t\|^2$. This can be solved via linear least square method to give
$$A = \sum_{i=1}^{n} \hat{q}_i \hat{p_i}^T \qquad A = U \sum V^T$$
where $\hat{p_i}$ and $\hat{q_i}$ is the difference between a point and its mean for the moving mesh and the destination mesh respectively.
$$R = VU^T \qquad t = \bar{p} - R\bar{q}$$
where $\bar{p}$ and $\bar{q}$ are the mean of p points and q points respectively.

The Fig 1(a) shows my solution for Question 1 and Fig 1(b) shows the graph of errors.
I concluded that for points with a good amount of overlap, ICP is a good algorithm for performing alignment.

Question 1b

Minimize alignment error E(R,t) with associated weights $w_i \in [0, 1]$

$$E(\mathbf{R}, t) = \sum_i w_i \|\mathbf{R}\mathbf{p_i} + \mathbf{t} - \mathbf{q_i}\|^2 \tag{1}$$



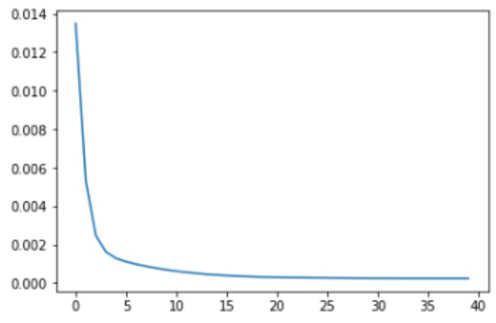


Figure 1: aligned meshes (left), Error plot(right)

The minimum points for 1 is given when $\frac{\partial E((R),t)}{\partial t} = 0$ and $\frac{\partial E((R),t)}{\partial R} = 0$. when $\frac{\partial E}{\partial t} = 0$,

$$\frac{\partial E((R),t)}{\partial t} = 2\sum_i^n w_i(\mathbf{Rp_i} + \mathbf{t} - \mathbf{q_i}) = 0$$

since w_i is a scalar,

$$\frac{\partial E((R),t)}{\partial t} = \sum_i^n \mathbf{R}(\mathbf{w_i p_i}) + \sum_i^n w_i\mathbf{t} - \sum_i^n w_i\mathbf{q_i} = 0$$

dividing by number of points, and rearranging

$$\mathbf{R}\sum_i^n \frac{\mathbf{w_i p_i}}{n} + \sum_i^n \frac{w_i\mathbf{t}}{n} - \sum_i^n \frac{w_i\mathbf{q_i}}{n} = 0$$

let $\bar{p}$ and $\bar{q}$ represent weighted means of the points p and q, hence,

$$\mathbf{R}\bar{p} + t\sum_i^n w_i - \bar{q} = 0 \qquad t = \frac{\bar{q} - \mathbf{R}\bar{p}}{\sum_i^n w_i}$$

which is the value for t for $_{min}E((R),t)$. Substituting this value back to 1, it reduces to

$$E(\mathbf{R}) = \sum_i^n w_i \left\| \mathbf{Rp_i} + \frac{\bar{q} - \mathbf{R}\bar{p}}{\sum_i^n w_i} - \mathbf{q_i} \right\|^2 \tag{2}$$

$$E(\mathbf{R}) = \sum_i^n w_i \left\| \mathbf{R}(\mathbf{p_i} - \frac{\bar{p}}{\sum_i^n w_i}) - (\mathbf{q_i} - \frac{\bar{q}}{\sum_i^n w_i}) \right\|^2$$

let $\tilde{p}$ and $\tilde{q}$ represent normalized weighted mean for p points and q points respectively i.e weighted mean divided by the sum of all weights. This reduces 2 to

$$E(\mathbf{R}) = \sum_i^n w_i \left\| \mathbf{R}(\mathbf{p_i} - \tilde{p}) - (\mathbf{q_i} - \tilde{q}) \right\|^2$$

the difference between each point p_i and q_i and the normalized weighted mean can be represented by $\hat{p}$ and $\hat{q}$ hence,

$$E(\mathbf{R}) = \sum_i^n w_i \left\| \mathbf{R\hat{p}_i} - \mathbf{\hat{q}_i} \right\|^2 \tag{3}$$

hence, equ. 5 can be expanded as

$$E(\mathbf{R}) = \sum_i^n w_i (\mathbf{R\hat{p}_i} - \mathbf{\hat{q}_i})^T (\mathbf{R\hat{p}_i} - \mathbf{\hat{q}_i})$$

which can be expanded and simplified using $R^T R = 1$ to get

$$E(\mathbf{R}) = \sum_i^n w_i(\hat{\mathbf{p}}_\mathbf{i}{}^T \hat{\mathbf{p}}_\mathbf{i} + \hat{\mathbf{q}}_\mathbf{i}{}^T \hat{\mathbf{q}}_\mathbf{i} - 2\hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R}\hat{\mathbf{p}}_\mathbf{i})$$

since we are interested in minimizing R, we can say

$$_{min}E(\mathbf{R}) =_{min} \sum_i^n w_i(-2\hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R}\hat{\mathbf{p}}_\mathbf{i}) =_{min} -\sum_i^n w_i\hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R}\hat{\mathbf{p}}_\mathbf{i} =_{max} \sum_i^n w_i\hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R}\hat{\mathbf{p}}_\mathbf{i}$$

Hence,

$$_{min}E(\mathbf{R}) =_{max} \sum_i^n w_i\hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R}\hat{\mathbf{p}}_\mathbf{i} =_{max} \sum_i^n \hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R} w_i\hat{\mathbf{p}}_\mathbf{i} \text{ since } w_i \text{ is a scalar} \tag{4}$$

Furthermore, equ. 4 can be expanded as

$$\sum_i^n \hat{\mathbf{q}}_\mathbf{i}{}^T \mathbf{R} w_i\hat{\mathbf{p}}_\mathbf{i} = Trace\left\{ \begin{bmatrix} \hat{q_1} \\ \hat{q_2} \\ . \\ . \\ \hat{q_n} \end{bmatrix} [\mathbf{R}] \begin{bmatrix} w_1\hat{p_1} & w_2\hat{p_2} & . & . & w_n\hat{p_n} \end{bmatrix} \right\} = Trace(\hat{\mathbf{Q}}^T \mathbf{R}\mathbf{W}\hat{\mathbf{P}})$$

Which means that equ. 1 further simplifies to

$$E(\mathbf{R}) =_{maxR} Trace(\hat{\mathbf{Q}}^T \mathbf{R}\mathbf{W}\hat{\mathbf{P}}) \tag{5}$$

Since Trace are invariant under cyclic permutation $Trace(ABCD) = Trace(BCDA) = Trace(CDAB) = Trace(DABC)$ Hence, we can write

$$Trace(\hat{\mathbf{Q}}^T \mathbf{R}\mathbf{W}\hat{\mathbf{P}}) = Trace(\mathbf{R}\mathbf{W}\hat{\mathbf{P}}\hat{\mathbf{Q}}^T)$$

We can take Singular value decomposition of $W\hat{\mathbf{P}}\hat{\mathbf{Q}}^T = \mathbf{U\Sigma V^T}$ hence,

$$Trace(\mathbf{R}\mathbf{W}\hat{\mathbf{P}}\hat{\mathbf{Q}}^T) = Trace(\mathbf{RU\Sigma V^T})$$

Since trace products can be switched without changing the results

$$Trace(BA) = Trace(AB)$$

,

$$Trace(\mathbf{RU\Sigma V^T}) = Trace(\mathbf{\Sigma V^T RU})$$

$\mathbf{V^T RU}$ is an orthonormal single as $\mathbf{V^T V} = \mathbf{R^T R} = \mathbf{U^T U} = \mathbf{I}$. We can let

$$\mathbf{M} = \mathbf{V^T RU} \qquad since \mathbf{M^T M} = I \implies 1 = m_i^T m_i = \sum_{i=1}^3 m_{ij}^2 \implies |m_{ij}| \leq 1$$

Equ. 5 further reduces to

$$_{maxR}Trace(\Sigma M) =_{maxR} Trace \begin{bmatrix} \sigma_1 & 0 & 0 \\ 0 & \sigma_2 & 0 \\ 0 & 0 & \sigma_3 \end{bmatrix} \begin{bmatrix} m_{11} & m_{12} & m_{13} \\ m_{21} & m_{22} & m_{23} \\ m_{31} & m_{32} & m_{33} \end{bmatrix}$$

$$_{maxR}Trace(\Sigma M) = \sum_{i=1}^3 \sigma_i m_{ii} \leq \sum_{i=1}^3 \sigma_i$$

This implies that $_{maxR}Trace(\Sigma M) \leq \sum_{i=1}^3 \sigma_i$. It follows that this equality holds when m_{ii} = 1 Hence

$$_{maxR}Trace(\hat{\mathbf{Q}}^T \mathbf{R}\mathbf{W}\hat{\mathbf{P}}) \implies m_{ii} = 1 \implies \mathbf{M} = \mathbf{I} \implies \mathbf{V^T RU} = \mathbf{I}$$

and by left multiplying by V and right multiplying by U^T

$$_{maxR}Trace(\hat{\mathbf{Q}}^T \mathbf{R}\mathbf{W}\hat{\mathbf{P}}) \implies \mathbf{R} = \mathbf{VU^T}$$

Note that, if

$$det(VU^T) = -1, then, \qquad R = \mathbf{V}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & -1\end{bmatrix}\mathbf{U}$$

The final solution can be written by these sets of equations

$$\bar{p} = \sum_{i}^{n} \frac{w_i \mathbf{p_i}}{n} \qquad \bar{q} = \sum_{i}^{n} \frac{w_i \mathbf{q_i}}{n}$$

$$W\hat{\mathbf{P}}\hat{\mathbf{Q}}^T = \mathbf{U\Sigma V^T}$$

$$R = \mathbf{V}\begin{bmatrix}1 & 0 & 0\\ 0 & 1 & 0\\ 0 & 0 & det(VU^T)\end{bmatrix}\mathbf{U}$$

$$t = \frac{\bar{q} - \mathbf{R}\bar{p}}{\sum_i^n w_i}$$

Question 2

To bring my mesh(M1) to the origin, I subtracted all vertices from its centroid(given by trimesh mesh.centroid property). After that, I rotated around the z axis by multiple angles name [5,10,15,20,25,30,40,50,70,90] using trimesh.applyTransform(R). where R is

$$R = \begin{bmatrix}cos(\theta) & -sin(\theta) & 0 & 0\\ sin(\theta) & cos(\theta) & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{bmatrix}$$

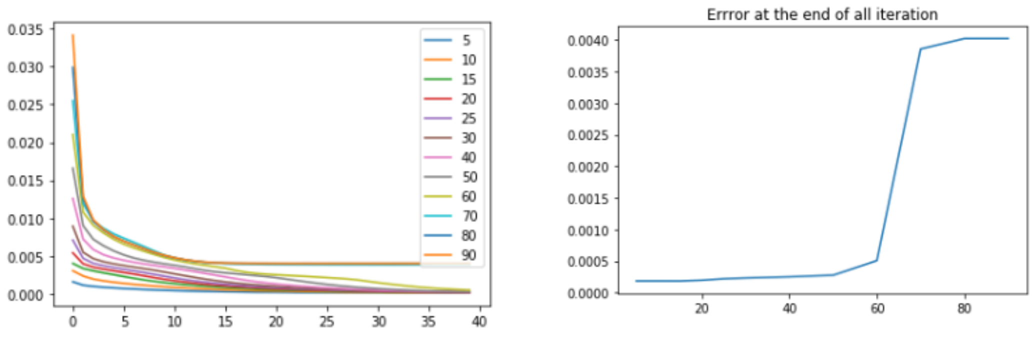


Figure 2: convergence rate of different angles(left), Errors at the end of iterations for all angles(right)

From fig. 2(a) and fig. 2(b), I noticed that ICP algorithm converges to the global minimum for rotations up to a particular angle. In other words, when the misalignment was 0 to about 60 degrees, the ICP algorithm converged to global maximum. But from around 70 degree and above, The Basic ICP algorithm Implemented does not converge to the global maximum. It can also be noticed from fig. 2(a) that the larger the misalignment angle the longer it took to achieve convergence. for example, a misalignment angle of 10 degree converged in less than 5 epochs, while that of 60 degrees converged at close to 40 epochs.

Question 3

To add Gaussian zero of mean noise zero to M2, I used an array of random numbers with Gaussian mean zero and standard deviation equal to a noise level I set. I reshaped the array of values to the shape of the vertices of the mesh. Then I defined a constant k which I used along with the bounding box dimensions to scale the noise to a value that can be added to the Mesh Vertices. The bounding box dimensions were gotten using trimesh.mesh.extents

$$boundedNoise = noise * (bounding_box_dimensions/K)$$

As noise level increases, the value of errorthat ICP converges to also increases. This is expressed directly in fig. 4(b) in which the convergence points has a linear relationship with the amount of noise in the system. I also noticed that even with noise, as long as the points have a reasonable amount of overlap, the ICP algorithm aligns the meshes quite well.

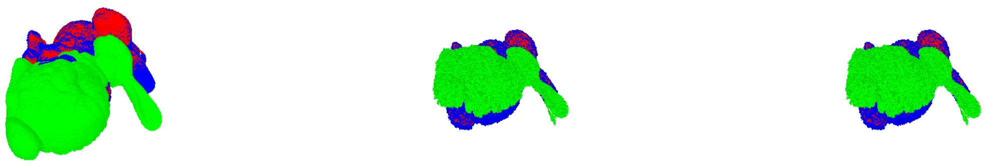


Figure 3: Solution at noise level equal to 2 (left), noise level 5 (middle) and noise level 20(right)

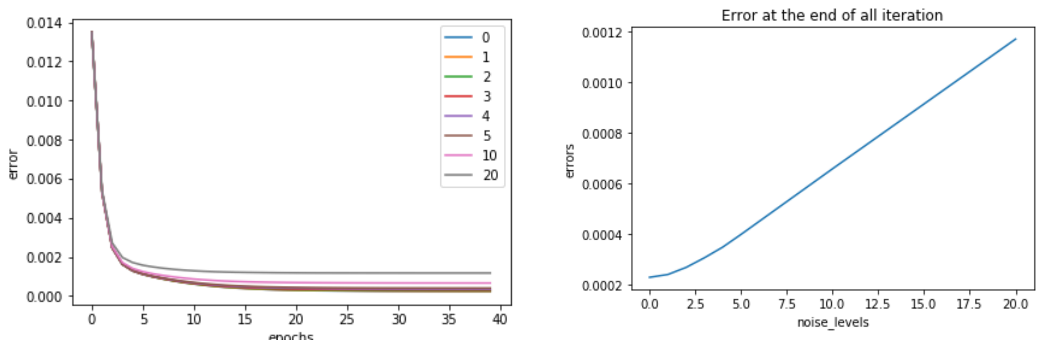


Figure 4: convergence rate of noise levels(left), Errors at the end of iterations for noise levels(right)

4 Solution

I used subsampled versions of M1 and M2 to estimate the transforms. I performed subsampling on the mesh vertices by using numpy.random.choice without replacement which performs uniform sampling. There are other ways of making optimal sampling, but since the algorithm(including the sampling) would run for more than 10 iterations, I am confident that uniform random sampling would give good results.

I observed that sampling had little effect on convergence rate and accuracy. A sampling ratio of 5,10,20,50 (reducing the points used for ICP from 40,000 to 8k, 4k, 2k, 800) had very little effect on convergence. When sampling ratio brought the amount of points used for for minimizing error below 20 points, accuracy dropped. I also experimented with different meshes[M1,M6] and found that the same phenomenon was observed. I conclude that as long as there is a **LARGE OVERLAP** between the meshes that are to be aligned via ICP, sampling should be done whenever possible as it has little effect on accuracy and convergence.

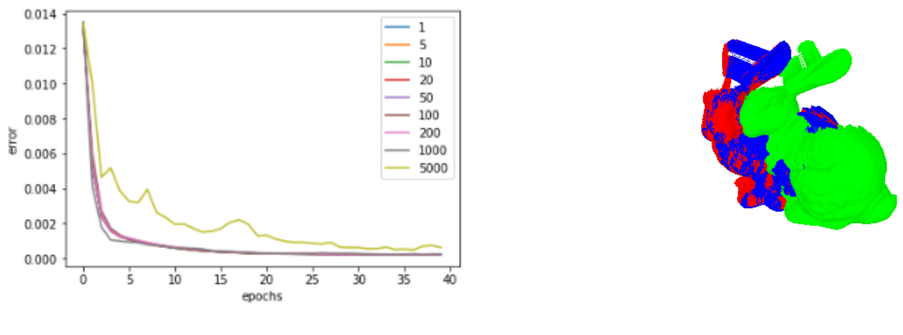


Figure 5: convergence rate of subsampling levels(left), Solution at subsampling rate equals to 1000, using about 38 points for ICP(right)

5 Solution

I did this question with trial and error method as seen in the jupyter notebook submitted. I was able to get 4 meshes aligned by trying multiple pairs with ICP but I was not able to get bunny90(M3) and bunny180(M4) aligned. The steps I took that were successful were as follows:

1. I aligned bunny45(M2) to bunny00(M1). With M2 been the source and M1 been the destination. shown in Fig 6(a)

Figure 6: M1 and the aligned M2(left), M1, aligned M2 and old M2

2. I aligned bunny315(M6) and bunny270(M5). With M5 been the source and M6 been the destination. shown in Fig. 7(a) and Fig. 7(b)

Figure 7: M6 and the aligned M5(left). M6, aligned M5 and old M5

3. I aligned M1 and M6. With M6 been the source and M1 been the destination. as seen in Fig. 8(a) and fig. 8(b)
4. Next, I used trimesh overload + operation to join all Meshes aligned in M1 namely M1, aligned M2 (new_ M2) and aligned M6(new_M6) results are shown in Fig. 9(a) and fig. 9(b)
5. Next, Align M1M2M6 gotten from previous step 4 to new_M5 gotten from step 2 with new_M5 as the destination and M1M2M6 as the source. The results are shown

These were the steps I took to get 4 of the meshes aligned with ICP. I realize that this is a brute force method which has a lot of con's such as the error is not distributed among meshes, the method is not feasible if there are plenty meshes to be aligned, and the method requires a lot time and human effort. On the pro side, I was able to really understand how these meshes fit together and make a solution specifically to this problem that would work.

Figure 8: M1 and aligned M6(left). M1, aligned M6 and old M6

Figure 9: M1 M2 and M6 aligned M6 multiple views

6 Solution

I implemented the solution as describe in **"Linear Least-Squares Optimization for Point-to-Plane ICP Surface Registration" by Kok-Lim Low**. The method involves derivation of a linear system whose least-squares solution is a good approximation to that obtained from a nonlinear optimization of point-to-plane algorithm.
point to plane ICP iterations aim to solve the equation

$$M_{opt} = argmin_M \sum_i ((M.s_i - d_i) \cdot n_i)^2 \tag{6}$$

Where M is a 4x4 3D homogenous transformation matrix. M can be said to be comprised of 3 rotations $R(\alpha, \beta, \gamma)$ and a translation components $t_x, t_y and t_z$ normally moving from angles involves sin's and cos's which makes this operation non-linear. The author provides a method for linearizing 6 for meshes that are quite close to each other, by assuming that at an $\theta \approx 0$ $sin\theta = 0$ and $cos\theta = 1$ using this assumption,

$$M_{approx} = \begin{bmatrix} 1 & -\gamma & \beta & t_x \\ \gamma & 1 & -\alpha & t_y \\ - & \alpha & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

This is substitiuted back int equ. 6 and solved to get a form $Ax - b = 0$, which is then minimized and the optimal x is given by

$$x_{opt} = A^{+}b$$

where A^{+} is the pseudoinverse of matrix A gven in the paper and b is also given in the paper.

I implemented the paper explained above and also performed normal shading and got good convergence rate as shown. I eliminated bad points with the normals as well by using the dot product between the normals. Since the vertex normal

Figure 10: aligned M1M2M6 and M5 only(left, middle-left) unaligned M1M2M6, aligned M1M2M6 and M5

magnitude given by trimesh is 1, I took the cosine of the dot product directly and had a threshold of about 120 degrees. I also implemented normal shading for this question.

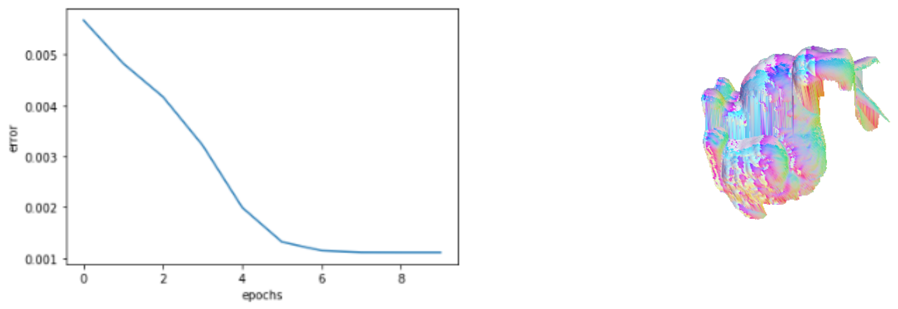


Figure 11: convergence rate of point to plane (left), Solution of point to plane for M6 and M5(right)