SGN-34006 3D and Virtual Reality

Implementation of ICP and registering point clouds

1 General Instructions

The objective of this exercise is to learn how to register two sets of 3D points by estimating the transformation between the two sets.

The tasks should be completed using synthetic data provided in Moodle.

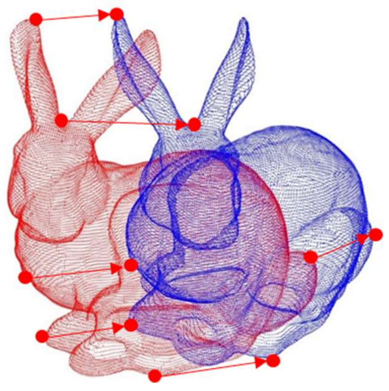
Deliverables
Each group should return:
Matlab script that does the required work. When the script is executed in the directory containing the test data, it should execute without errors and display the requested figures for all tasks. The figures produced by the demo code, which have “Task x:” in the title should be reproduced by your script. Other figures are for guidance only, and do not need to be drawn by you.

The submission should be all Matlab codes and the image data your script needs for functioning in a single **ZIP** file. **See Moodle for the exact deadline.** Possible extensions to the deadline should be negotiated **before** the deadline and may be awarded at the discretion of the assistant if valid arguments are presented.

2 Iterative Closest Point

Iterative Closest Point (ICP) is an algorithm used to align two partially overlapping set of points. The algorithm iteratively minimizes the distance and rotation between two sets of points. The algorithm is widely used for registering the outputs of 3D scanners, which typically only scan an object/scene from one direction/position at a time. The standard ICP starts with two set of points or point clouds and an initial guess for their relative rigid-body transform. The initial guess helps to reach convergence quickly. The algorithm then iteratively refines the transform by repeatedly generating pairs of corresponding points in the clouds and minimizing an error metric.

Among the two point clouds, one would serve as the reference point cloud, which will remain fixed. The second point cloud is transformed towards the reference point cloud. In general, ICP approach can be extended to be used with Line Segment Sets, Implicit Curves, Parametric Curves, Triangle Sets, Implicit Surfaces, and Parametric Surfaces etc. Robust variants of ICP can be used with complex and noisy data to reconstruct 2D or 3D surfaces from different scans, to localize robots and achieve optimal path planning.



3 Exercise Tasks

All results here should be drawn when the Matlab script is executed.

A. Familiarize yourself with the data and transformation [Mandatory]

1. Create a set of 3D points $P = [0\ 0\ 3;\ 5\ 0\ 5;\ 2.5\ 2.5\ 0]$ and visualize with number each point.
2. Create a rotation matrix $R(20{,}50{,}40\ )$ and translation $t = [0\ 2\ 3]$, here

$$R_x(\varphi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\varphi) & -sin(\varphi) \\ 0 & sin(\varphi) & cos(\varphi) \end{bmatrix}, R_y(\Psi) = \begin{bmatrix} cos(\Psi) & 0 & sin(\Psi) \\ 0 & 1 & 0 \\ -sin(\Psi) & 0 & cos(\Psi) \end{bmatrix}, R_z(\theta) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix},$$

   which can be combined as a single rotation matrix, $R(\varphi, \Psi, \theta\ ) = R_z(\theta)\ R_y(\Psi)\ R_x(\varphi)$
3. Create a function `rigidTransform` that takes rotation $R(\varphi, \Psi, \theta\ )$ and translation t as input and transform the points P using the following relation

$$\bar{P} = (P * R)\ + t$$

   The transformed points $\bar{P}$ are now obtained. Note, that here the post transformation convention is adopted. It means that the rotation matrices comes after the 3D points. You are free to use pre-multiplication convention; however, care must be given that the same convention is followed for all tasks.
4. Visualize the results. You can use `pcshow` function to conveniently show the points.

B. Estimate the Transformation R and t [Mandatory]

Find the transformation in terms of the rotation matrix R and translation matrix t to transform the points $\bar{P}$ in task 1 back to P. The estimated transformation would results to be the inverse of the original transformation.

The rotation and translation can be estimated using various methods. Here we will adopt the most popular approach. Note that the points have been transformed, however, we clearly know which point is transformed to which position. Hence, we already know the point correspondence; therefore, we will not bother with finding correspondences for this task.

Create a function `estimateRT_pt2pt` that takes in the inputs and perform the steps explained for this task. The function should output the R and t.

1. Find the centroids of both point clouds.

$$centroid = \frac{1}{N} \sum_{i=1}^{N} P^i$$

2. To find the optimal rotation we first re-centre both dataset so that both point clouds are at the origin. Bring both the point clouds to the origin by subtracting the centroid from each point.

3. Find the covariance between the point clouds. The Covariance matrix gives a measure of similarity between 2 data sets to be matched. One thing to be careful is that you calculate covariance correctly. It should end up being a 3×3 matrix.

$$covariance = transpose(P) * \bar{P}$$

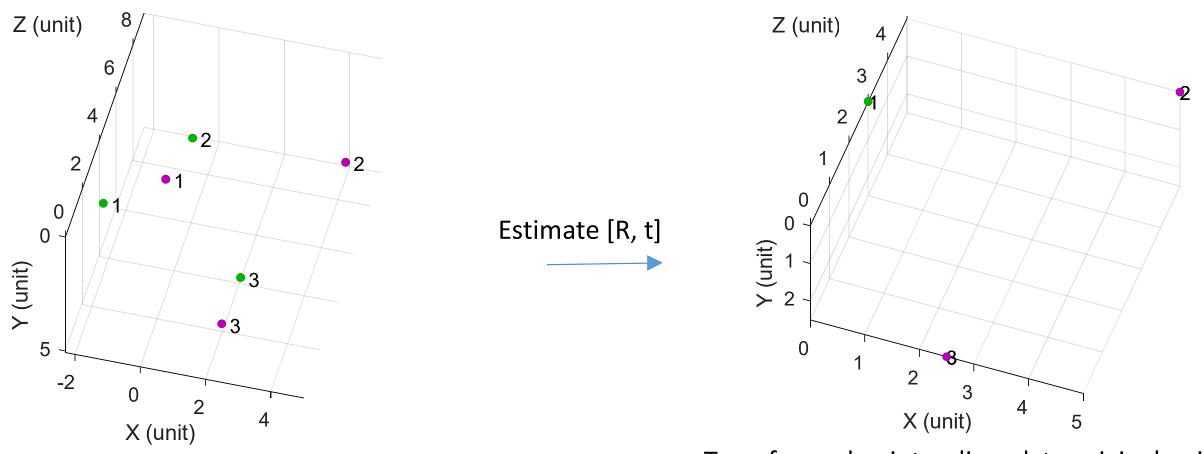
4. Find the optimal rotation, (matrix R). The rotation can be estimated by decomposing the covariance matrix C using singular value decomposition (SVD).The rotation matrix R is then achieved by multiplying the Eigen vectors of the covariance matrix obtained through SVD.

$$[U, S, V] = SVD(Covariance)$$

$$R = U * transpose(V)$$

In some cases when finding the rotation matrix SVD will return a 'reflection' matrix, which is numerically correct but does not equate to real transformation. This can be corrected by placing a simple check. Explore how this can be done.

5. Find the translation t. The translation can be estimated by rotating the centroid of the displaced point cloud and subtracting from the centroid of the reference point cloud. Remember the pre/post multiplication order of rotation matrix.



Original and transformed points from task 1

Transformed points aligned to original points

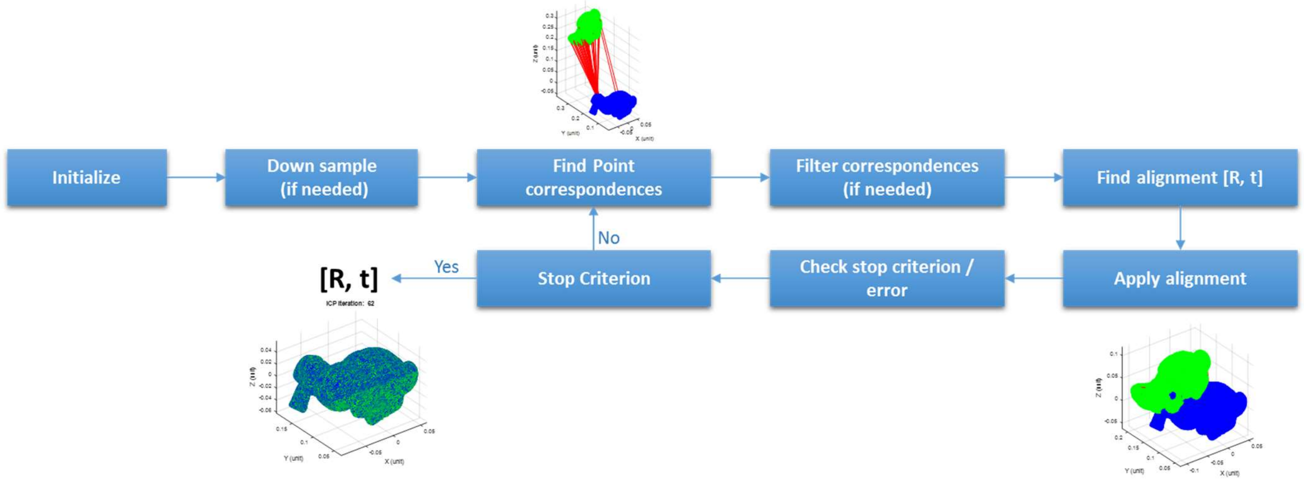
C. Implement ICP [Mandatory]

Implement the ICP algorithm explained in this task. Use the data provided in *bunny.mat* file. Allign *bunnyMoved* point cloud to the reference *bunny* point cloud using ICP. Visualize the update to the point cloud during ICP at each iteration as shown in demo file. Create a function `ICP` that takes in the inputs and perform the steps explained for this task.

Algorithm Pipeline:

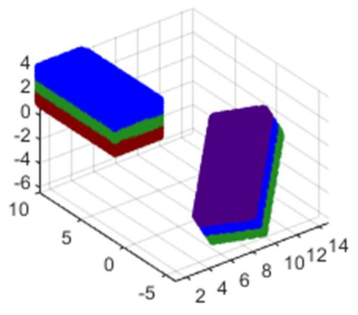
1. Initialize registration parameters (R,t) and down sample (`help pcdownsample`), if needed.
2. For each point in the moved/scene point cloud, find the corresponding closest point in the reference/model point cloud (`help knnsearch`).
3. Filter the correspondences. Sort the correspondences based on distance. Keep a percentage of the matches (e.g. 90%). You can always tune this parameter by giving it as an input to the `ICP` function.

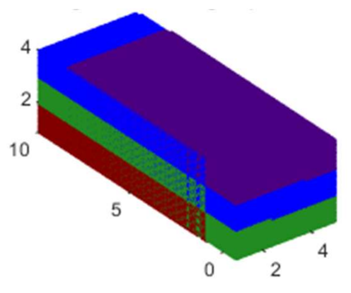
4. Calculate the registration parameters R and t given point correspondences obtained from step You can use the function `estimateRT_pt2pt` created in task 2.
5. Apply the alignment to the moved/scene points using the function `rigidTransform` created in task 1.
6. Check if we have met the criterion to exit the loop. The criterion can be a combination of checks. Here we will set a relative rotation and relation translation change as a check. If the change in rotation and translation that you get from step 3 is less than a threshold for 3 consecutive iterations, then exit the loop, otherwise continue. In addition, set an upper limit on the number of iterations to avoid being stuck in an infinite loop if the system is not able to converge to a solution. *[Tip: convert rotation matrix to axis-angle rotation representation]*
   To summarize, if relative change [delta_R, delta_t] > [threshold_R, threshold_t], return to step 2, else return with , t and the aligned point cloud.

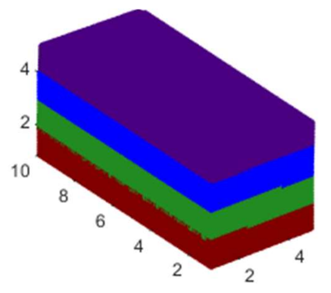


D. Color assisted ICP [Bonus +1]

a- Implement a color assisted ICP. Find the corresponding points using both point distance and color distance of the points from each other. Transform color from RGB to Lab color space for better results `[help rgb2lab]`. Use a weightage parameter alpha [0 > alpha < 1] with the color distance to increase or decrease the effect of color based search. Use the data provided in the `slab.mat` file. Align the slab2 to slab1 using a pipeline similar to task 3 with added color feature.


Original Point Clouds

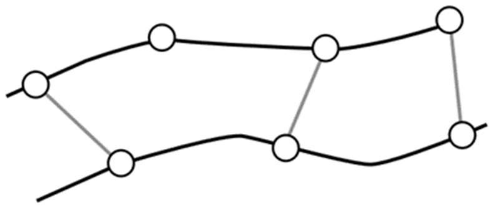

Aligned using point based ICP (Task 3)

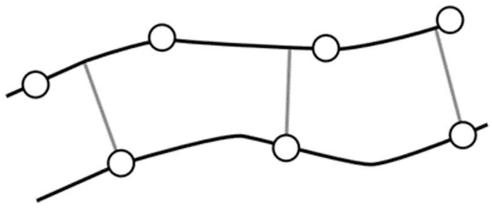

Aligned using color assisted ICP (Task 4)

b- Give a brief answer to the following questions along with the completed task above in the mat file.
   i) Pro and cons of ICP vs other registrations methods
   ii) How to make ICP robust to noise.

E. ICP with point to plane metric [Bonus +1]

Implement the ICP estimating R and t by minimizing point to plane error iteratively. The approach searches the intersection on the destination surface from the normal vector of the reference point. For detailed mathematical relation, read the file tutorial_point2plane.pdf. Create a function `estimateRT_pt2pl`, which will replace the function `estimateRT_pt2pt` in the ICP pipeline.


Point-to-Point Error metric preference


Point-to-Point Error metric preference

Use the data provided in *bunny.mat* file for testing.

Hints:

a- Implement the final linear system equation AX=B from the tutorial. In Matlab you can readily solve a linear relation using x =C\b.

b- The rotation matrix can be obtained as before using $R(\alpha, \beta, \gamma) = R_z(\gamma)\, R_y(\beta)\, R_x(\alpha)$, where α, β and γ are obtained from x.

c- Keep note of the size of matrices explained in the tutorial for C, x, and b.

Comment on the difference in performance of point-to-point and point-to-plane error metric.