ROB 498/599 3D Robot Perception: HW2

Term: Winter 2025
Instructor: Bernadette Bucher, EECS, University of Michigan
Due Date: 6th April 2025
Version: 2 (March 12, 2025 release)

Constraints: This is an open-world homework assignment. You can search for resources on the web; talk to your friends about the assignment; do anything you would do in a real-world setting; but you must do your own work; you may not copy from anyone or any LLM.

Goals: Dive into the world of point clouds! Understand what they represent, how they can be used and understand and implement some famous algorithms to derive various results from point cloud data.

Data: You need to download the supplemental data to complete this assignment. Unzip and upload it to your google drive in a folder called `Homework2`. Make sure all files are stored in a folder called `data` inside the folder `Homework2`. Link for data

General Instructions We will be using Google Colab for the assignment. Problems 1 and 2 are required for all students. Students enrolled in ROB599 are additionally required to complete both problems 3 and 4, whereas students enrolled in ROB498 may complete *either* problem 3 or problem 4 - whichever is more interesting to them. Like the bonus problem in HW1, ROB498 students are encouraged to understand the bonus problem as its content may still be relevant for exams/quizzes, etc., and completing its contents may be a useful exercise.

The starter code can be found here. Save a copy of the script to your drive in the folder `Homework2`. Remember to start the assignment early as you need to use a GPU runtime to train the models for problem 3 and 4 which has a daily use limit.

Doubts and Discussions: There is a section/tag in Piazza for HW2. Kindly direct all your questions in that group, and consult the staff in office hours as required.

Submission Instructions: Rename the notebook as `uniquename_homework2.ipynb`. Run all cells and make sure all outputs are visible as we will not run your code. Submit the link to your solution on Canvas. Make sure to give us viewer access to the file.

Problem 1 (10): Generating a Point Cloud

- **a.** You are given a RGB image, a depth image and a mask as shown in fig(1). Your task is to generate point cloud data from the given depth image and the corresponding colour from the RGB image. Recall we used projection equations to project 3D points to a 2D plane. Here, given depth, we need to reproject the points back onto a 3D plane. You are also given the camera data (focal length and camera center). Use the data from file `generate_point_cloud.npz`
- **b.** Use the point clouds and RGB data obtained from Q1a and plot them to generate at least 3 different views of the object. Note: You will need to visualize a lot of point clouds in this assignment. It would help if you generalize this function for future tasks. Feel free to modify input args as needed.

Iterative Closest Point Algorithm

When working with real world point cloud data, one often encounters 2 sets points that are unaligned but represent the same scene or object. These point clouds might be slightly rotated, translated, or even have non-uniform scaling due to various factors such as sensor noise, calibration errors, or deformations in the object itself. In such cases, accurately aligning these point clouds becomes essential to make meaningful comparisons or perform further analysis. The Iterative Closest Point (ICP) algorithm is a powerful tool that addresses this challenge, providing a systematic and efficient way to find the optimal transformation that aligns two point clouds or shapes.

In this section, we are going to develop an in-depth understanding of the ICP algorithm by implementing it and computing a rigid transformation between two point clouds.

Problem 2 (30): Implementing the ICP algorithm

- **a. Estimate Correspondences between the two point clouds:** You are given two point clouds X and Y, an initial guess of transformation T and rotation R and a threshold for maximum distance between two points. Return a list of estimated point correspondences. Procedure to follow is given in the pseudocode in fig(2).
- **b. Estimating Optimal Rigid Transform:** Now that we have computed the correspondences, it is time to compute the transform between them. The algorithm to accomplish this task is mentioned in fig(3)
- **c. Stitching Everything Together:** Use the functions you defined in the previous questions and implement the ICP algorithm. Test the algorithm by using the point clouds in the files `data/point_cloud_X.txt` and `data/point_cloud_Y.txt`. Set initial estimate of t as 0, R as identity, d_{max} as 0.25 and run the code for 30 iterations. Report the RMSE error computed

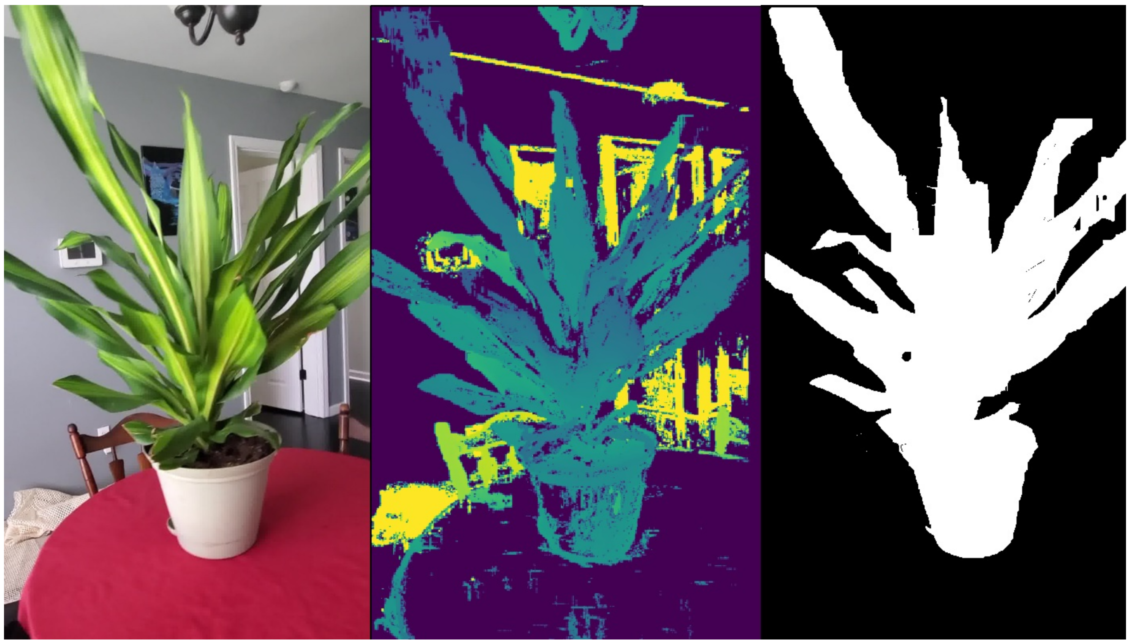

Figure 1: Data for Problem1

from the following equation:

$$\text{RMSE} = \sqrt{\frac{1}{K}\sum_{k=1}^{K}\left\|y_{jk} - (\hat{R}\cdot x_{ik} + \hat{t})\right\|^2} \tag{1}$$

d. Visualizing the point clouds: Write a function to visualize the two point clouds before and after transformation to witness the might of the ICP algorithm. You are free to use the function you wrote previously for Problem 1

Pointnets

PointNet is a groundbreaking neural network architecture tailored for the classification and segmentation of point cloud data. Introduced in the seminal paper ”PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation” by Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. PointNet represents a significant advancement in the field of 3D deep learning.

Processing point clouds poses unique challenges compared to traditional 2D image data. Point clouds are unordered, meaning that the order of the points within the cloud does not inherently convey information. Furthermore, point clouds may have varying numbers of points, making it challenging to apply standard convolutional neural networks (CNNs) designed for regular grids.

In the next few sections, we are going to implement the classification and segmentation network.

Problem 3 (30): Classification

In this question, you are given point clouds of objects corresponding to 3 different classes: Vase, Chair and Lamp . You are to implement the pointnet neural network and train it to differentiate between different objects.

Note: Use of a GPU is recommended. You can switch to a GPU runtime by going to `Runtime>Change Runtime Type`

a. Implementation of Pointnet Architecture for classification: Refer to the fig(4) to implement the structure of Pointnet for classification. The values in brackets correspond to layers. After every layer there is a `1D Batch normalization` layer and a `Relu` activation layer. Refer to pytorch documentation on how to implement these layers. MLP layers can also be implemented as 1D Convolutions with Kernel size 1. Remember to use a `softmax layer` to get the output scores for different classes.

b. Training the network: To train the network the following steps need to be followed:

i. Check and convert the dimensions into proper order. Recall, the model needs the data to be in $B \times C \times N$ format where B is the batch, C is the channels and N is the number of points

ii. The dataset contains 10000 points per point clouds. To ease the computation, we are only using 1000 points per point cloud. Slice the data so that you are only using 1000 points.

iii. Compute Forward pass

```
Algorithm 1 Iterative closest point
Input: Pointclouds $X = \{x_i\}_{i=1}^{n_X} \subset \mathbb{R}^d$ and $Y = \{y_i\}_{i=1}^{n_Y} \subset \mathbb{R}^d$, initial guess $(t_0, R_0) \in \mathrm{SE}(d)$
    for a rigid registration aligning pointcloud $X$ to pointcloud $Y$, maximum admissible distance
    $d_{\max} > 0$ for associating points.
Output: An estimated set $C = \{(i_k, j_k)\}_{k=1}^{K}$ of correspondences between points in $X$ and $Y$,
    and the rigid registration $T = (\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ that optimally aligns corresponding points of
    $X$ and $Y$ in the least-squares sense (1).
 1: function ICP$(X, Y, t_0, R_0, d_{\max})$
 2:     Initialization: $\hat{t} \leftarrow t_0$, $\hat{R} \leftarrow R_0$.
 3:     repeat
 4:         Initialize list of empty correspondences: $C \leftarrow \varnothing$, $K \leftarrow 0$.
 5:         for $i = 1, \ldots, n_X$ do:                    ▷ Estimate point correspondences
 6:             Find closest point $y_j$ in $Y$ to image of $x_i$ under transformation $(\hat{t}, \hat{R})$:

                    $y_j = \underset{y \in Y}{\operatorname{argmin}} \|y - (\hat{R}x_i + \hat{t})\|_2^2.$                (2)

 7:             if $\|y_j - (\hat{R}x_i + \hat{t})\|_2 < d_{\max}$ then
 8:                 Add $x_i$ and $y_j$ to the list of point correspondences: $C \leftarrow C \cup \{(i, j)\}$.
 9:             end if
10:         end for
11:         Compute optimal registration given the point correspondences $C$:

                $(\hat{t}, \hat{R}) \leftarrow$ ComputeOptimalRigidRegistration$(X, Y, C)$                (3)

12:     until convergence return $(\hat{t}, \hat{R}, C)$
13: end function
```

Figure 2: Algorithm for estimating correspondences between the two point clouds

iv. Calculate the Loss. We are using the `CrossEntropyLoss` for classification.

v. Perform back propogation and optimize

c. Testing the network:

i. Perform the same steps as `Problem 3b` to prepare the data for the network.

ii. Get the predictions from your network. Remember to not compute gradients or the loss!

iii. Calculate the accuracy of the model by comparing how many predictions correctly match the labels and dividing the number by total number of datapoints

d. Inference:

i. Visualize few point clouds that have been classified correctly

ii. Visualize few point clouds that have been classfied incorrectly

You can re-use functions defined previously

Problem 4 (30): Segmentation

In this question, you are given a point cloud of different types of chairs with labels or classes for different parts of the chair: back rest, arm rest, seat etc. Your task is to create a PointNet segmentation network and train it to perform part segmentation.

Note: Use of a GPU is recommended. You can switch to a GPU runtime by going to `Runtime>Change Runtime Type`

a. Implementation of the Pointnet architecture for segmentation: The architecture of pointnet for segmentation is given in fig(5). There are a total of 6 classes that the points can be segmented into. For segmentation, we store the local features extracted after the first set of MLP layers and concatenate each point in the local feature tensor with the global feature vector, making the input to the segmentation network $n \times (64 + 1024) = n \times 1088$. The values in brackets correspond to layers. After every layer there is a `1D Batch normalization` layer and a `Relu` activation layer. Refer to pytorch documentation on how to implement these layers. Your implementation should **NOT** contain the input transform and the feature transform. MLP can also be implemented as 1D Convolutions with Kernel size 1.

```
Algorithm 2 Optimally aligning pointclouds with known point correspondences
Input: Pointclouds $X = \{x_i\}_{i=1}^{n_X} \subset \mathbb{R}^d$ and $Y = \{y_j\}_{j=1}^{n_Y} \subset \mathbb{R}^d$, list of point correspondences
    $C = \{(i_k, j_k)\}_{k=1}^{K}$.
Output: Rigid transformation $T = (\hat{t}, \hat{R}) \in \mathrm{SE}(d)$ that optimally aligns corresponding points of
    $X$ and $Y$ in the least-squares sense (1).
1: function ComputeOptimalRigidRegistration($X, Y, C$)
2:     Calculate pointcloud centroids:
           $\bar{x} \triangleq \frac{1}{K} \sum_{k=1}^{n} x_{i_k}, \qquad \bar{y} \triangleq \frac{1}{K} \sum_{k=1}^{K} y_{j_k}.$    (4)
3:     Calculate deviations of each point from the centroid of its pointcloud:
           $x'_{i_k} \triangleq x_{i_k} - \bar{x}, \qquad y'_{j_k} \triangleq y_{j_k} - \bar{y}.$    (5)
4:     Compute cross-covariance matrix $W$:
           $W \triangleq \frac{1}{K} \sum_{k=1}^{K} y'_{j_k} (x'_{i_k})^{\mathsf{T}}.$    (6)
5:     Compute singular value decomposition: $W = U \Sigma V^{\mathsf{T}}$.
6:     Construct optimal rotation:
           $\hat{R} \triangleq U \,\mathrm{Diag}(1, \ldots, 1, \det(UV))) V^{\mathsf{T}}.$    (7)
7:     Recover optimal translation: $\hat{t} \triangleq \bar{y} - \hat{R}\bar{x}$.
8:     return $(\hat{t}, \hat{R})$.
9: end function
```

Figure 3: Algorithm for estimating optimal rigid transform

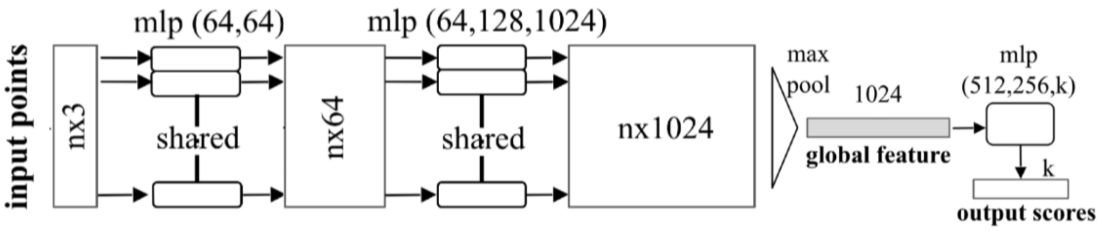

Figure 4: Model architecture for Classification

b. Implementing the loss function: In this case, we need a special loss function because our data is not uniformly distributed (Refer fig(6)). We observe a significant imbalance in the dataset. Penalizing the network equally for misclassifying categories of varying frequencies wouldn't be logical. If we penalize high-frequency categories the same as low-frequency ones, the model might excel at capturing common patterns but struggle with rare features. For this reason, we define a few weights for the loss function that correspond to the different classes. These are learnable parameters and we are initializing them manually as per the visualization. You are free to experiment with them as you please. Initialize the cross entropy loss with the weights and perform the forward pass. Refer to the pytorch documentation on `Cross Entropy loss` for more information.

c. Training the Network To train the network the following steps need to be followed:

i. Check and convert the dimensions into proper order. Recall, the model needs the data to be in $B \times C \times N$ format where B is the batch, C is the channels and N is the number of points

ii. The dataset contains 10000 points per point clouds. To reduce computation, we are only using 2000 points per point cloud. Slice the data so that you are only using 2000 points.

iii. Compute Forward pass

iv. Calculate the Loss. Here the `PointNetSegLoss` is passed as the argument criterion in the function

v. Perform back propogation and optimize

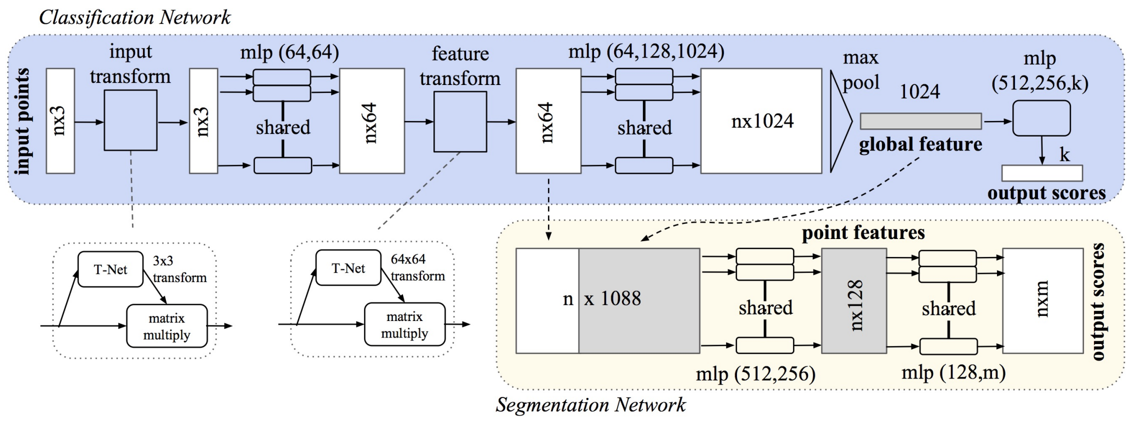

Figure 5: Model architecture for segmentation

d. Testing the Network:

i. Perform the same steps as `Problem 4c` to prepare the data for the network.

ii. Get the predictions from your network. Remember to not compute gradients or the loss!

iii. Get the prediction with highest probability from the output by using `torch.softmax`

iv. Calculate the accuracy of the model by comparing how many predictions correctly match the labels and dividing the number by total number of datapoints

e. Inference

i. Visualize a few point clouds that have been segmented correctly. Make sure to use different colours for different classes

ii. Visualize few point clouds that have been segmented incorrectly. Make sure to use different colours for different classes

You can re-use functions defined previously

References and Credits:

Parts of this homework are from the CMU 16-825: Learning for 3D vision.

Figures and content for Pointnet comes from the paper: ”PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation” by Charles R. Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas

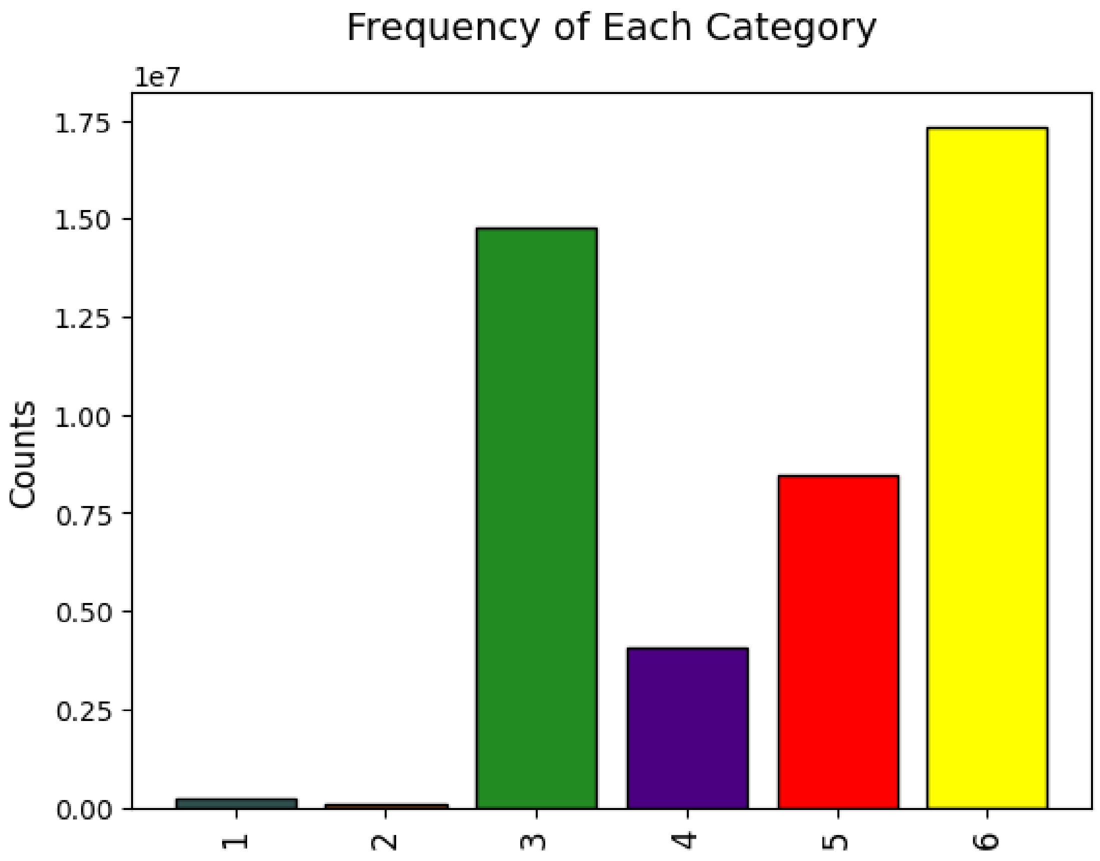

Figure 6: Distribution of classes over the data set