CS 4610/5335: Robotic Science and Systems (Spring 2023) Robert Platt
Northeastern University

HW 4: Point Cloud Analysis

Please remember the following policies:

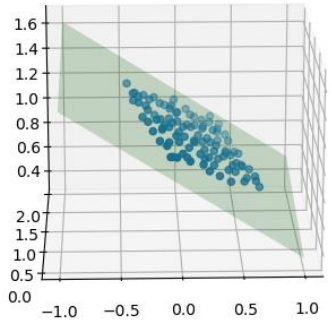
- Submissions should be made electronically via the Canvas. Please ensure that your solutions for both the written and/or programming parts are present and zipped into a single file.
- Solutions may be handwritten or typeset. For the former, please ensure handwriting is legible.
- You are welcome to discuss the programming questions (but *not* the written questions) with other students in the class. However, you must understand and write all code yourself. Also, you must list all students (if any) with whom you discussed your solutions to the programming questions.

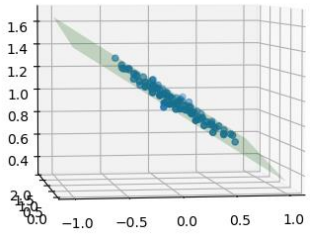
All of the code for this assignment is in Python, and located within 'hw4.zip'. A guide for how to install necessary libraries is found in 'README.md'. The only file where you need to make edits is 'questions.py'. **Please add comments to your code to help the graders understand what you are doing. If you make a mistake, you are much more likely to receive partial credit if the code is understandable.**

Q1. Plane fitting (**5 points**).

In this question, you are given a set P of 100 points and asked to fit a plane to these points. The output of these functions should be a point intercept of the plane (a vector called "center") and the plane surface normal (a vector called "normal").

(a) Implement the function `q1 a` in questions.py: fit a plane by calculating the sample mean and covariance matrix of the points. You will need to obtain the Eigen values and vectors of the covariance matrix in order to complete this question. You can test your implementation by running the command `$python q1 a` in your terminal.



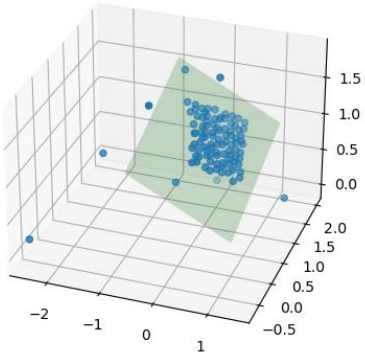


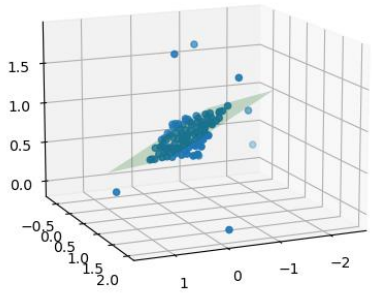
(b) Test your plane fitting on an example with outliers by running the command `$python q1 b`. How is this different from the result in part (a) and why?

- The result of the plane fitting in part (b) with outliers is likely to be different from the result in part (a) without outliers. This is because in part (b), the input point cloud contains outliers that deviate significantly from the underlying plane. As a result, the sample covariance matrix computed in part (b) will be biased by the presence of outliers, and the eigenvectors and eigenvalues obtained from this matrix may not accurately represent the underlying plane.

- In contrast, in part (a), the input point cloud is assumed to be noise-free, and the sample covariance matrix computed from this cloud accurately represents the underlying plane. The eigenvectors and eigenvalues obtained from this matrix accurately represent the orientation and scale of the plane, respectively.

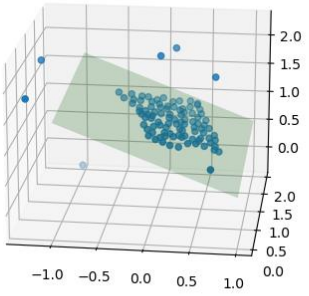
- Therefore, in the presence of outliers, the result of plane fitting obtained from the sample covariance matrix may not accurately represent the underlying plane, and alternative methods such as RANSAC may be more appropriate to robustly estimate the plane parameters.

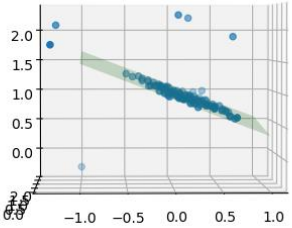




(c) Implement the function q1_c in questions.py: fit a plane using a ransac based method. You can test your implementation by running $python hw4.py q1 c in your terminal. What are the strengths and weaknesses of each approach?

- Strengths:
  - RANSAC is more robust to outliers compared to the other methods because it only considers a subset of points to fit the model and ignores the rest.
  - RANSAC can handle non-linear models and does not require the assumption of linearity.
- Weaknesses:
  - RANSAC requires more computation time because it samples points and fits the model multiple times to obtain the best result.
  - The parameters used in RANSAC, such as the number of iterations and the inlier threshold, need to be carefully tuned to obtain a good result.





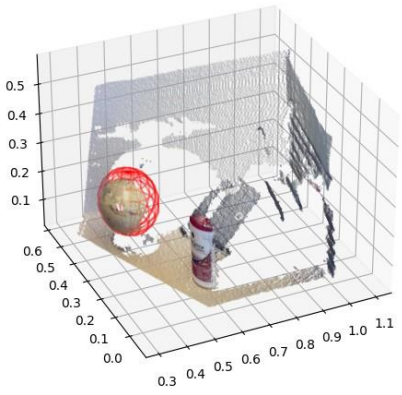


Figure 1: (a) the sphere localized in Q2.

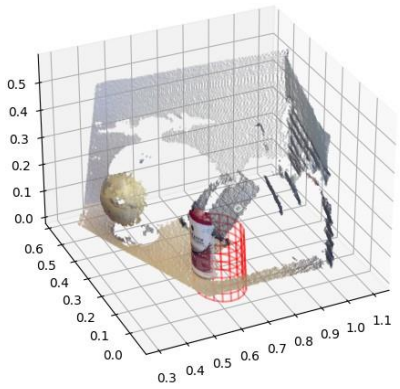


Figure 2: (b) the cylinder localized in Q3.

Q2. Sphere fitting (**5 points**).

In this question, you must use RANSAC to localize a sphere given point cloud data. The point cloud contains a ball that is only partially visible to the sensor. The position of the center of the ball is unknown. The radius is unknown, but between 5cm (0.05m) and 11cm (0.11m). Write the function q2 in questions.py that calculates these two quantities. You can test your code using the cropped point cloud (see commented hw4.py). But, you should submit code that works on the entire point cloud without cropping. Important: please do NOT use any external libraries for plane fitting.

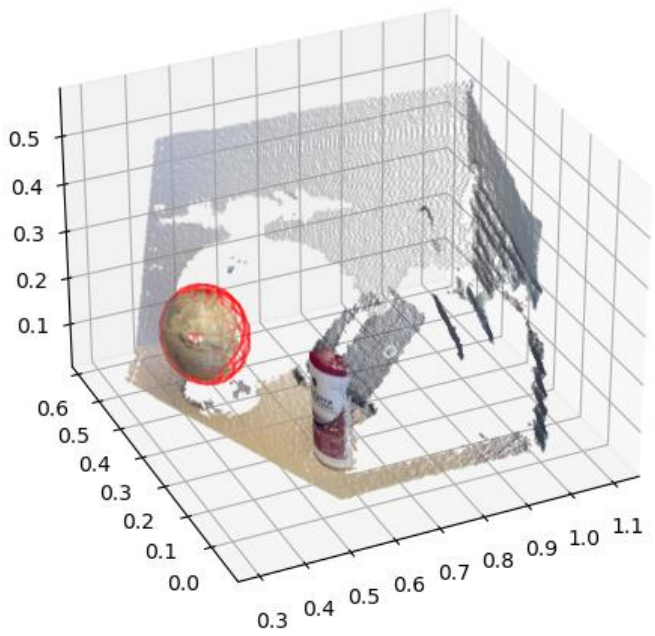
Hint: one way to generate sphere hypotheses is as follows:

(a) sample a point from the cloud;

(b) sample a radius of the candidate sphere between 5 and 11cm;

(c) project a vector from the sampled point in the direction of the associated surface normal for a distance equal to the sampled radius. This point would be at the center of the candidate sphere.

- After a long and arduous empirical analysis I have settled for **2000 samples** and **epsilon (inlier threshold) of 0.1**. This is the figure we come up with given the ransac algorithm.



Q3. Cylinder fitting (**5 points**).

Same as Q3 except for a cylinder. This question is harder because you need to calculate the center, the orientation, and the radius (between 0.05m and 0.1m). The orientation should be returned in the form of a unit vector pointing along the axis of the cylinder (either direction is fine). You only need to solve this problem for the segmented cloud, as implemented in the code. The function you should implement is `q3` in `questions.py`.

Hint: one way to generate cylinder hypotheses is as follows:

(a) Sample a radius for the candidate cylinder between 5 and 10 cm.

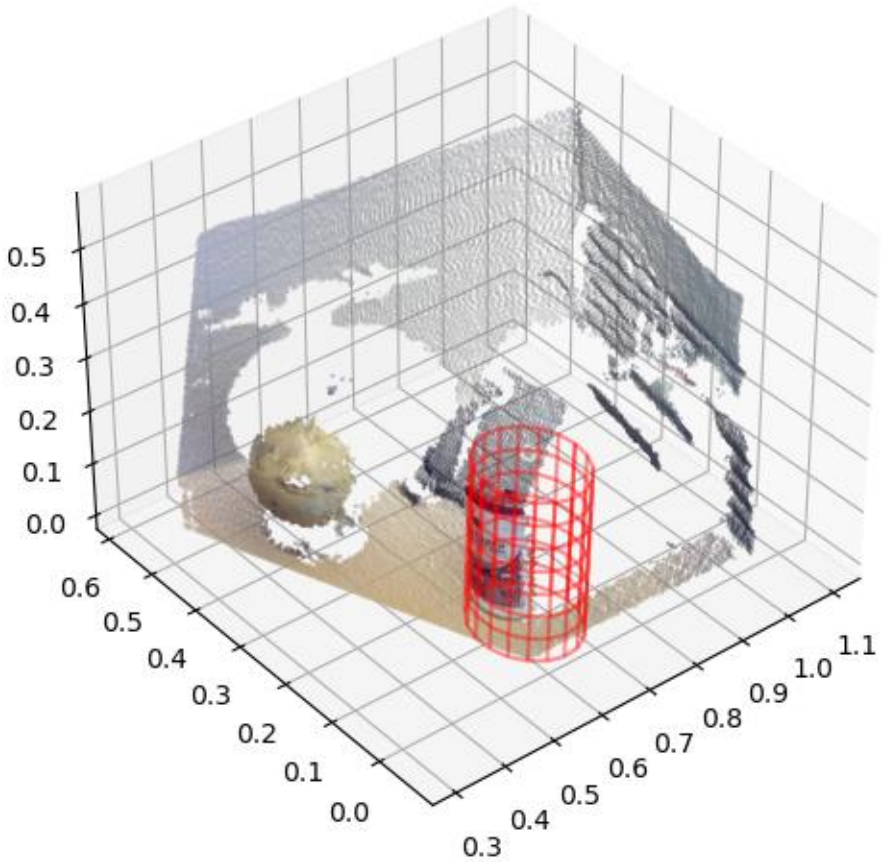
(b) Sample two points from the cloud

(c) Set the cylinder axis direction equal to the direction of the cross product between the surface normal associated with the two sampled points.

(d) Pick one of the sampled points from the cloud and use it to estimate a candidate center, just as you did in Q2.

(e) Project the points in the cloud onto the plane orthogonal to the axis you just calculated. You can do this projection by multiplying the points in the cloud by this matrix: $I - \hat{a}\hat{a}^T$, where $\hat{a}$ is equal to the axis of the cylinder. Also project the candidate center into this plane in the same way.

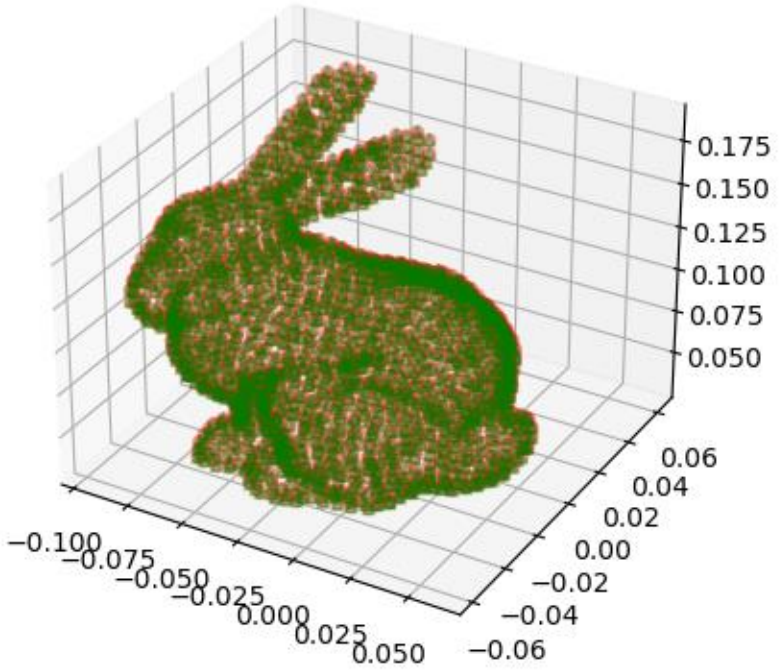
(f) Evaluate number of inliers (i.e. $\|\bar{P} - \bar{c}\|_2 < \epsilon$ where $\bar{P}$ is the projected point cloud, $\bar{c}$ is the projected candidate center, and ϵ is the noise threshold)



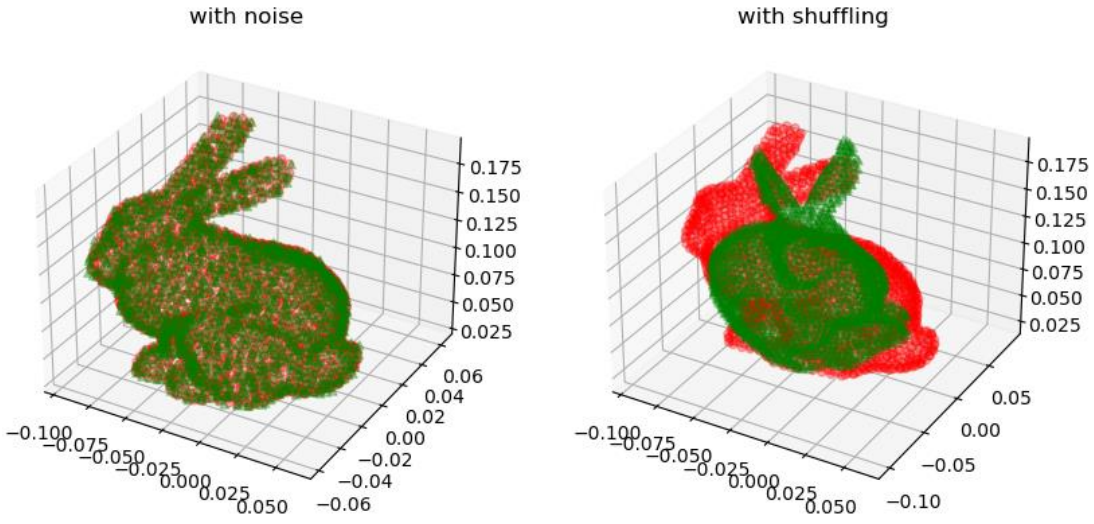
Q4. Iterated closest point (ICP) (**5 points**).

Given a set of points M, and another a set of points D with a different pose, ICP algorithm could be used to find a 4-by-4 transformation matrix T so that $D = TM$.

(a) Implement the function q4 a to find the transformation matrix that aligns two point clouds given full correspondences between points in the two clouds. In other words, D and M are the same point cloud but in different poses. You can test your implementation by running: $python hw4.py q4 a



(b) Run $python hw4.py q4 _b to test your implementation from part (a) on noisy data. Explain why the algorithm still works when gaussian noise is added to one of the point clouds, but does not work when the order of the points is shuffled.



- When Gaussian noise is added to one of the point clouds, the algorithm still works because it is based on finding the closest points between the two clouds. The noise will cause some points in one cloud to be slightly offset from their corresponding points in the other cloud, but the majority of the points will still be close to their corresponding points. Therefore, the algorithm can still find a good alignment between the two clouds.

- However, when the order of the points is shuffled, the correspondence between the points in the two clouds is lost. This means that the algorithm cannot find the closest points between the two clouds, and as a result, it cannot find a good alignment.

(c) Implement the function q4 c to perform iterative closest point (ICP). Your implementation should get reasonably alignment on shuffled and noisy data: run $python hw4.py q4 c to test this.