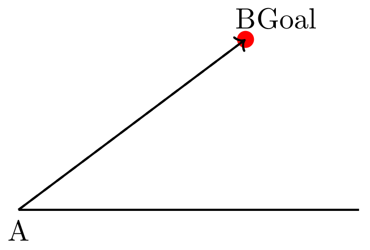
EECS 465/ROB 422: Introduction to Algorithmic Robotics
Fall 2024
Homework Assignment #4

Jinjia Guo

1. Explain why it is more difficult to define a distance metric that leads to good performance for a non-holonomic motion planning problem than a holonomic one. Use a car as an example that illustrates this difficulty and include diagrams.

Ans: Consider a car trying to reach a target location B from A. Its initial direction is not towards B.
For Holonomic Metric, the Euclidean distance metric can directly calculate the distance from the car's current position to the target (a straight path).
But for Non-Holonomic Metric, the car cannot drive along a straight path. It must turn and align with the target, taking a curved path. This makes the cost of the car's driving not only the distance but also the turns, etc.



Defining a good distance metric for nonholonomic motion planning is difficult because: it must take into account direction, speed, and turning constraints; it must reflect the feasibility of following the shortest path within the constraints of the system.

2. a Jacobian-based Inverse Kinematics Solver for the seven joints of the PR2's left arm. The code for this problem is in the *ik* folder. To simplify this problem, we will ignore the end-effector rotation and use only the position variables in the Jacobian. We will also ignore collisions.

   (a) Assume the robot has been placed in a certain configuration and you would like to compute the Jacobian at that configuration. Write a function to compute the Jacobian numerically, using the method described in class. Do this by filling in the code in the get translation jacobian function in *ik template.py* in the ik folder. You will need to use the *get joint axis*, *get joint position*, and *get ee* *transform* functions defined in the template. You may not use any functions to generate Jacobians directly from pybullet.

(b) Now write a function to compute the pseudo-inverse of the Jacobian. Do this by filling in the *get jacobian pinv*function in the template. Because the robot may get into singularities, use the damped least-squares method with a small λ.

(c) Implement the Iterative Jacobian Pseudo-Inverse Inverse Kinematics algorithm from lecture in *ik template.py*. You will need to tune the parameters. In the template you can select one of five end-effector position targets. Try to make your algorithm achieve all the targets when starting from $q = \{0, 0, 0, 0, 0, 0, 0\}$. Do not allow q to exceed joint limits; if the algorithm goes past a joint limit, set that joint's value to be the limit. For each target you can achieve, save the configuration that places the end-effector at that position and a screenshot of the robot in that configuration in your pdf.

(d) Make a copy of your code called *ik template nullspace.py*. Modify your algorithm in this file to use the left null-space of the Jacobian pseudo-inverse to repel the configuration away from joint limits while achieving the desired end-effector position. You will need to tune β so that the secondary task of repeling from joint limits does not interfere with task of reaching the target.For each target you can achieve, save the configuration that places the end-effector at that position and a screenshot of the robot in that configuration in your pdf.

Ans:

(a) see codes in the zip file

(b) see codes in the zip file

(c) The red spheres represent the targets. The blue spheres are the path of the end-effector.
For target 0: The configuration is: [[4.45276564e-01 -2.42458215e-01 -1.36561963e-01 -6.62551165e-01 -2.77546181e-02 -3.53612865e-01 1.63465823e-17]]

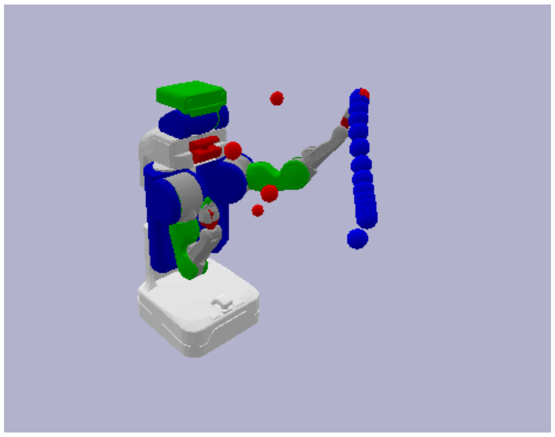


Figure 1: The configuration for target 0

For target 1: The configuration is: [[-6.74281183e-18 1.06537592e+00 -2.20079580e-17 -1.83796365e+00 2.51140131e-17 -1.30720365e+00 1.52125960e-17]]

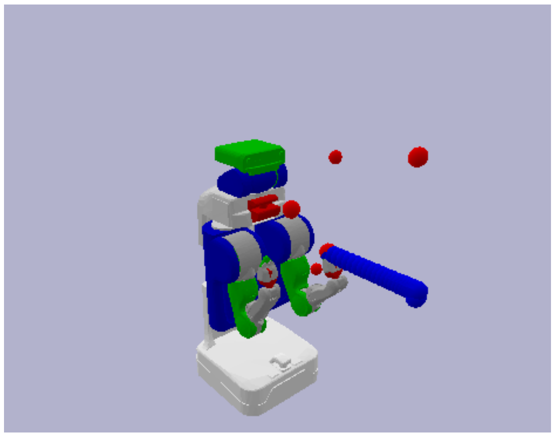


Figure 2: The configuration for target 1

For target 2: The configuration is: [[-2.33028170e-01 3.14247224e-01 1.24318501e-01 -2.04991497e+00 4.80552827e-02 -1.13283959e+00 -6.82568798e-18]]

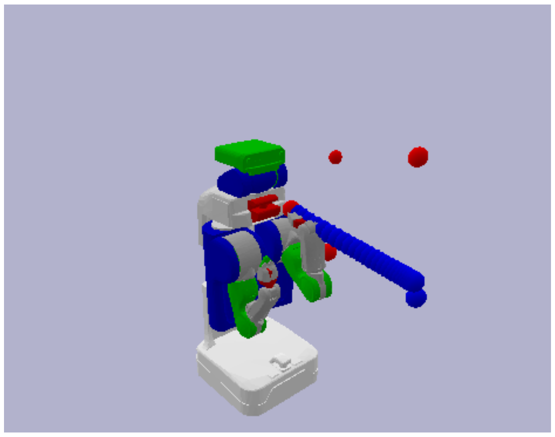


Figure 3: The configuration for target 0

For target 3: The configuration is: [[1.93593411e+00 9.69504928e-01 -8.40472305e-06 -3.57105911e-01 -1.52773093e-06 -2.50829349e-01 1.21943465e-14]]

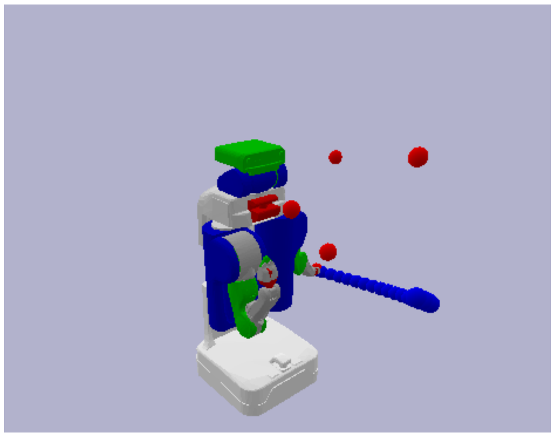


Figure 4: The configuration for target 0

For target 4: The configuration is: [[1.72566804e+00 -2.86011238e-02 -5.95382250e-01 -5.37015112e-01 -2.27900495e-01 -2.73243952e-01 -8.36841689e-17]]

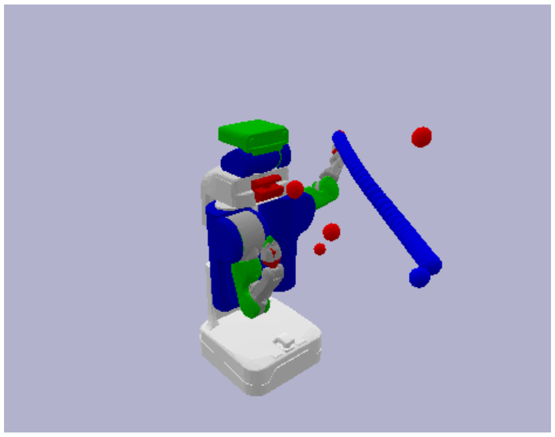


Figure 5: The configuration for target 0

(d) The red spheres represent the targets. The blue spheres are the path of the end-effector.
For target 0: The configuration is: [[0.44553 -0.24243 -0.13602 -0.66249 -0.02777 -0.35372 0.]]

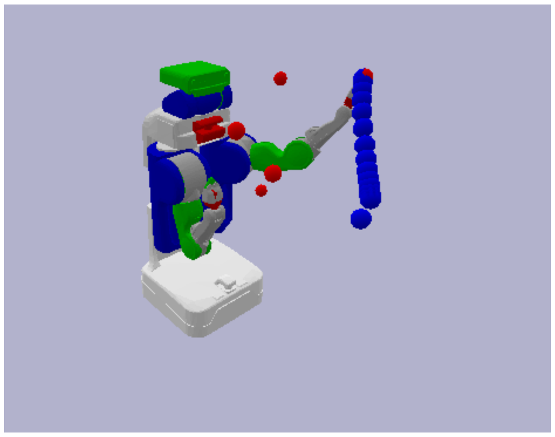


Figure 6: The configuration for target 0 with left null-space of the Jacobian pseudo-inverse

For target 1: The configuration is: [[0.00051 1.0651 0.00071 -1.83779 0.00001 -1.30795 0.]]

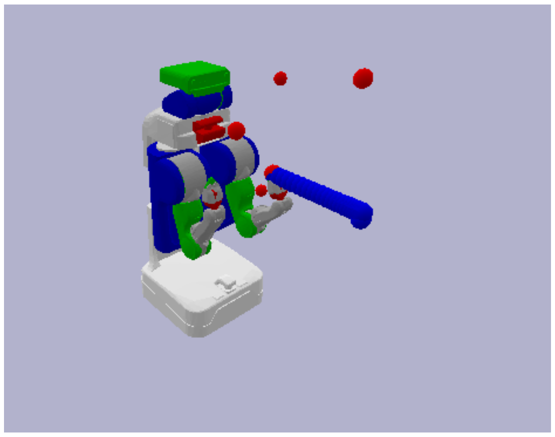


Figure 7: The configuration for target 0 with left null-space of the Jacobian pseudo-inverse

For target 2: The configuration is: [[-0.23259 0.31408 0.12479 -2.04985 0.04799 -1.13294 -0.]]

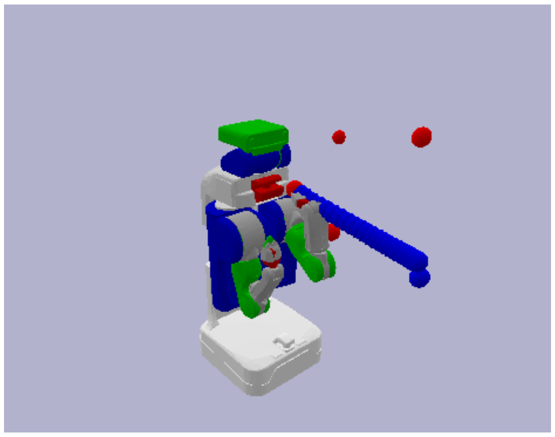


Figure 8: The configuration for target 0 with left null-space of the Jacobian pseudo-inverse

For target 3: The configuration is: [[1.93622 0.96918 0.00103 -0.35548 -0.00001 -0.25382 0.]]

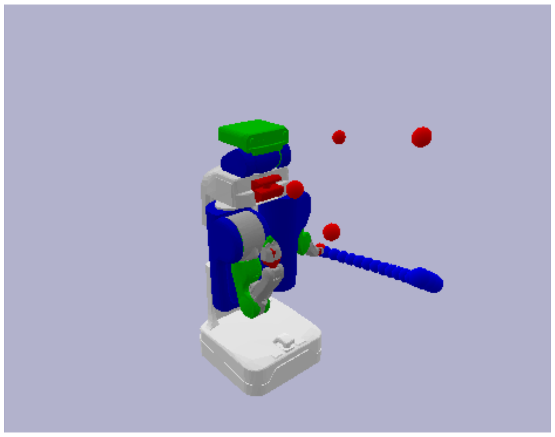


Figure 9: The configuration for target 0 with left null-space of the Jacobian pseudo-inverse

For target 4: The configuration is: [[1.72586 -0.02847 -0.5947 -0.53693 -0.22793 -0.27341 0.]]

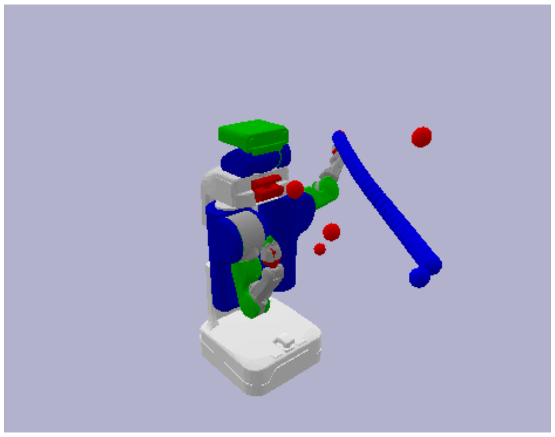


Figure 10: The configuration for target 0 with left null-space of the Jacobian pseudo-inverse

3. Force closure evaluation: The closure template.py script executes grasps using a 3-finger Robotiq gripper and a 7-DOF Kuka iiwa arm simulated in PyBullet. The code for this problem is in the grasping folder. You will be implementing friction cone calculation and a method that uses a convex hull to determine if grasps are in force closure.

   (a) In the pdf, include the volume of the true cone, 4-vector discretization, and 8-vector discretization for contact point 1 from Grasp 1. Round the volumes to 4 decimal places. Explain how changing the number of vectors changes the error of the discretization.

(b) In the pdf, include an explanation of your method for checking force closure and computing the radius of the hypersphere. Also include the force closure output for each grasp and the hypersphere radius if the grasp is in force closure. Explain why knowing the hypersphere radius is useful.

Ans:

(a) Grasp 1 Contact point 1 Volumes:
True volume: 0.1227
4 edge volume: 0.0781
8 edge volume: 0.1105
Increasing the number of vectors can improve the accuracy of discretization, thereby reducing the error. However, this comes at the cost of computational complexity, as more vectors require additional calculations. Therefore, the more vectors there are, the less the difference between the true value and the true value.

(b) **Method Explanation for checking fore closure and computing the radius**
Grasp 1 : not in force closure
Grasp 2: In force closure. Maximum radius: 0.0097
Force closure occurs when the convex hull of the wrench formed by all contact points completely contains the origin. This means that any wrench can be resisted by the contact points, thus ensuring stability. Wrench Calculation: For each contact point, a set of wrenches is computed by combining the forces (from the friction cone vectors) and torques (calculated using cross products of the radius vector and force). The convex hull of the wrenches is created using the *scipy.spatial.ConvexHull* library. By inspecting the equations of the convex hull, we check if all equations (plane normals) point away from the origin. If true, grasp force closure.
The hypersphere radius is the distance from the origin to the nearest plane in the convex hull of the wrench. It provides a measure of the robustness of force closure. After the convex hull is computed, the minimum distance to any bounding plane is computed (using the negative of the constant term in the plane equation). This value represents the maximum radius of a hypersphere that is completely contained within the convex hull.

Why useful?
A larger hypersphere radius indicates a higher robustness of the grip, meaning it can resist greater external perturbations before losing force closure.

When selecting a gripper, a larger radius can be given priority as it provides a higher stability of the grip.

4. pca template.py in the point clouds folder. Here a point cloud of a randomly-sampled planar surface has been rotated and noise has been added to it. Rotate the plot to get a sense of how the points are distributed. Edit the template to accomplish the following.
   (a) Use the Principle Component Analysis (PCA) algorithm to compute a rotation for the point cloud that aligns the surface in the point cloud with the XY plane as best as possible. Save a picture of the rotated point cloud in your pdf along with the V^T matrix you applied at the last step. Adjust the view in the plot so it is clear that the point cloud aligns with the XY plane.
   (b) Use PCA to do the same rotation while also eliminating the noise in the data. The resulting point cloud should be two-dimensional. You will need to set the threshold for variances to properly filter out the noise. Save a picture of the point cloud after PCA is applied in your pdf along with the matrix V_s^T you applied at the last step. Adjust the view in the plot so it is clear that the point cloud aligns with the XY plane and all the z values are 0.
   (c) Use PCA to fit a plane to the cloud and draw that plane in green in a plot with the point cloud. Include the plot in your pdf.

Ans:

(a) Transformation matrix V^T:

$$\begin{bmatrix} -0.52935 & -0.00254 & 0.8484 \\ -0.05809 & -0.99754 & -0.03923 \\ 0.84641 & -0.07005 & 0.5279 \end{bmatrix}$$

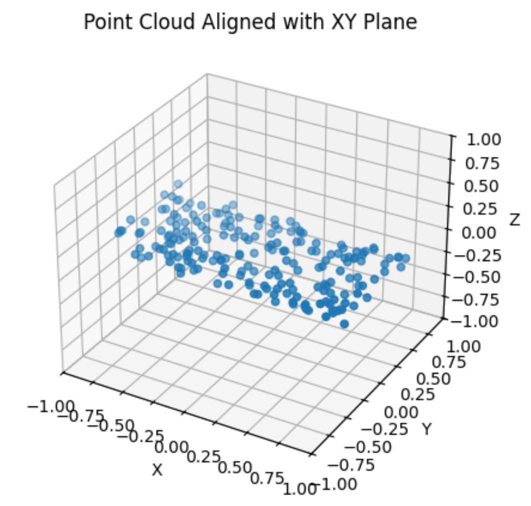


Figure 11: Point Cloud Aligned with XY Plane

(b) Transformation matrix V^T after elimination:

$$\begin{bmatrix} -0.52935 & -0.00254 & 0.8484 \\ -0.05809 & -0.99754 & -0.03923 \\ 0 & 0 & 0 \end{bmatrix}$$

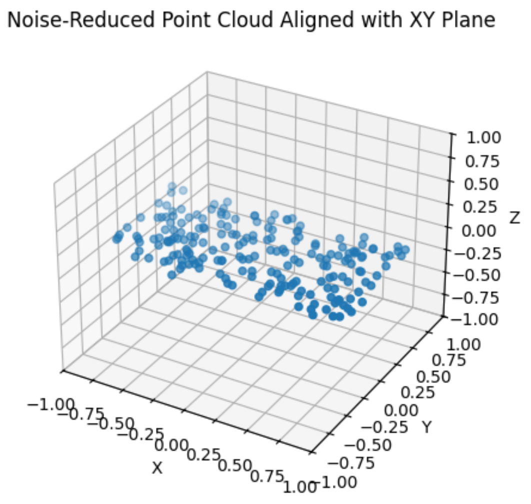


Figure 12: Noise-Reduced Point Cloud Aligned with XY Plane

(c) Here is the plot:

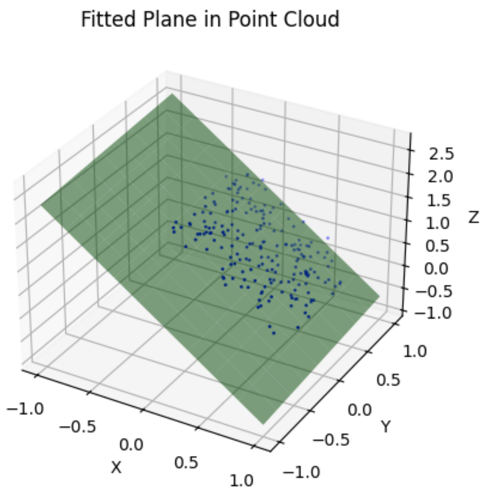


Figure 13: PCA Fitted Plane in Point Cloud

5. Run ransac template.py in the pointclouds folder. Here a point cloud of a randomly-sampled planar surface has been rotated and noise has been added to it. We have also added some outlier points to the cloud. Rotate the plot to get a sense of how the points are distributed. Edit the template to accomplish the following.

   (a) Use the RANSAC algorithm from lecture to find the equations of a plane that best fits the surface in the point cloud. You should select three points at each iteration to fit the model of the plane. The Error function for inliers should encode least-squares error, i.e. "the sum of squared residuals (a residual being the difference between an observed value, and the fitted value provided by a model)." (Wikipedia) You will need to set the number of iterations and the other parameters.
   Once you have computed the plane that best fits the data, include the equations for that plane in your pdf. Also include an image showing the plane in green fitting to the data, the inliers for that plane in red, and all other points in blue. Rotate the plot to clearly show the plane fitting to the data.

Ans:

(a) The equation is:

$$-0.83x + -0.18y + -0.53z = 0.59$$

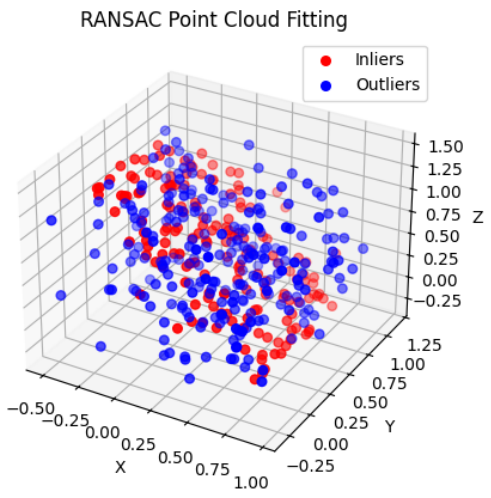


Figure 14: RANSAC Point Cloud Fitting

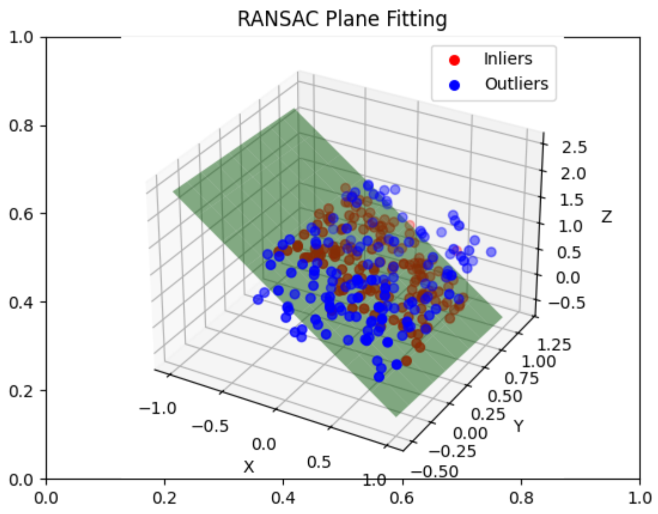


Figure 15: RANSAC Plane Fitting

6. Now we will compare RANSAC and PCA with varying amounts of outliers. Run pca vs ransac.py in the pointclouds folder to see a series of point clouds with added outliers. At each iteration,we add 10 outliers to the cloud. At each iterations where outliers are added, run your PCA and RANSAC algorithms from above so that each produces a plane that fits to the cloud. Use the Error function from RANSAC to compute the error of each algorithm's plane to its inliers. Determine the inliers for PCA the same way as was done in the RANSAC algorithm.

   (a) For the last iteration, generate two plots: 1) Showing PCA's plane fitting to the data in green, the inliers for that plane in red, and all other points in blue; 2) Likewise for RANSAC.Clearly label the plots so it is clear which one corresponds to which algorithm.

   (b) Generate a plot of Error vs. Number of Outliers for both algorithms and include it in your pdf. Also record the computation times of PCA and RANSAC at each iteration.

   (c) Discuss the performance difference between the algorithms. Was there a clear performance difference in terms of computation time and error? If so, which algorithm performed better and why? If not, explain why not

Ans:

(a) Here are the plots:

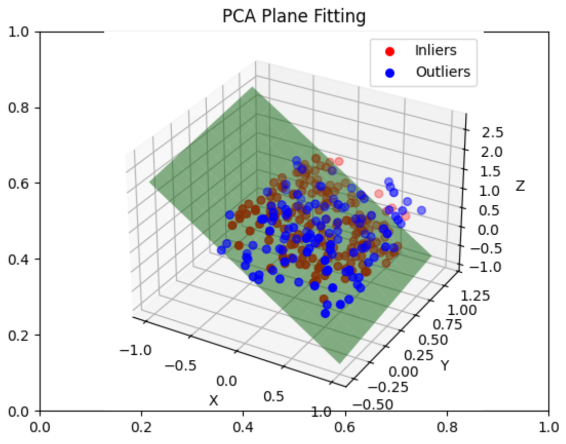


Figure 16: PCA Plane Fitting

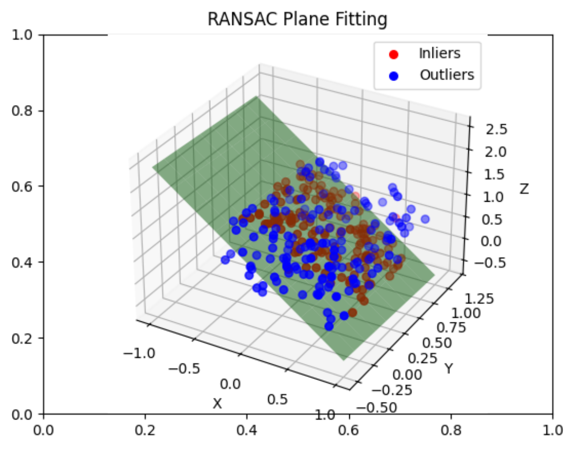


Figure 17: RANSAC Plane Fitting

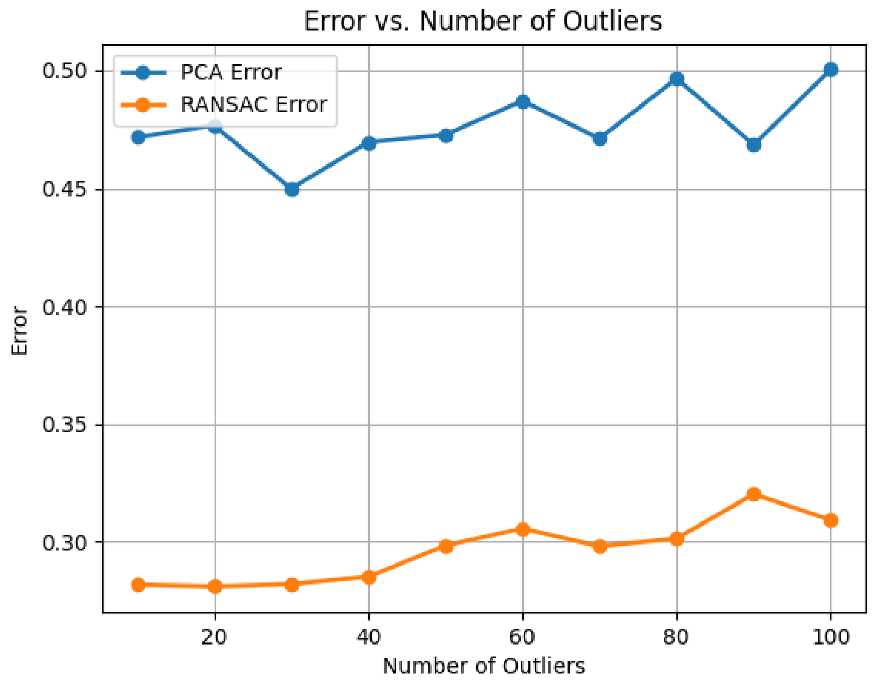


Figure 18: Error vs. Number of Outliers for both algorithms

(b) PCA Computation Times: [0.0008027553558349609, 0.0006880760192871094, 0.0014798641204833984,

0.0007290840148925781, 0.0015630722045898438, 0.0008168220520019531, 0.0008130073547363281, 0.0008559226989746094, 0.0018150806427001953, 0.0009131431579589844]
Average PCA Computation Time: 0.0010476827621459961
RANSAC Computation Times: [0.1859912872314453, 0.09840083122253418, 0.10109829902648926, 0.09494709968566895, 0.11116600036621094, 0.11530494689941406, 0.09122776985168457, 0.10346794128417969, 0.10794329643249512, 0.09831619262695312]
Average RANSAC Computation Time: 0.11078636646270752

(c) PCA is much faster to compute than RANSAC, about two orders of magnitude faster. The computation time of PCA is very short and almost negligible, while the computation time of RANSAC is relatively long. Because PCA has relatively high computational complexity, PCA only needs to calculate the covariance matrix and perform singular value decomposition once. This process is very efficient in most cases. RANSAC requires iterative random sampling, model fitting, and interior point checking. The computational overhead of RANSAC increases significantly with a large number of iterations, especially when the data contains many outliers. Therefore, PCA significantly outperforms RANSAC in terms of computation time.
RANSAC is more robust when there are many outliers, and the error is usually smaller than PCA, so RANSAC performs better when dealing with data containing outliers.
If there are no or only a few outliers in the data, and computational speed is the primary consideration, PCA is a better choice. If there are a large number of outliers in the data and accurate plane fitting results are required, RANSAC is more suitable, although its computational time is longer.

7. Run icp template.py in the pointclouds folder to see a source and target point cloud of a mug. Implement the ICP algorithm to align a source point cloud to a target point cloud. Use Euclidean distance to determine correspondences at every iteration. You will need to tune the parameters and determine a reasonable termination condition.
   (a) In the template you will see a way to load in one of four target point clouds. Run ICP on each target cloud separately. You may tune parameters separately for each target. You may not be able to produce a good alignment for all the targets but try to make the source align to as many targets as you can.
   (b) For each target, generate 1) A plot of Error vs. Iteration of ICP; 2) A plot showing the target point cloud and the source point cloud at the final iteration (in different colors). Include these in your pdf.

(c) Discuss the results; which targets were more difficult for ICP and why?

Ans:

(a) see codes in zip file

(b) Target 0:

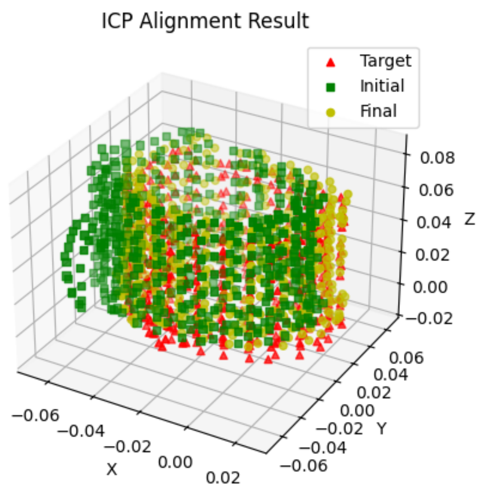


Figure 19: icp alignment result cloud target0

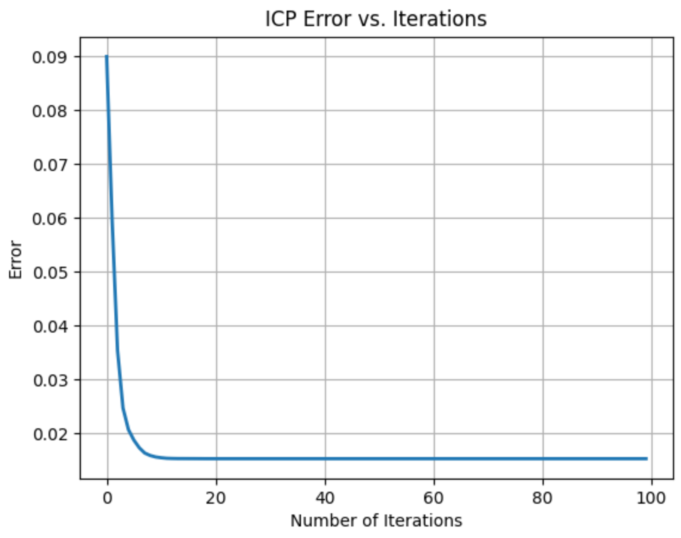


Figure 20: error vs iterations cloud target0

Target 1:

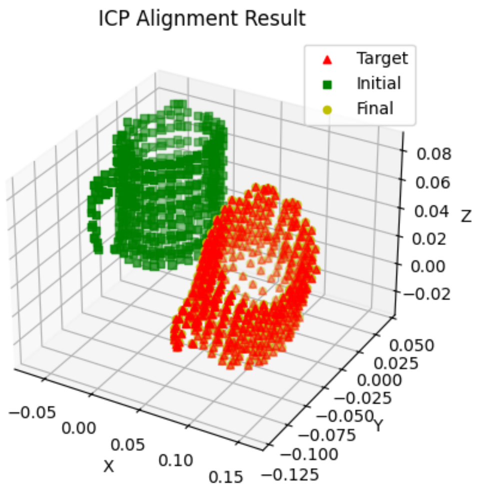


Figure 21: icp alignment result cloud target1

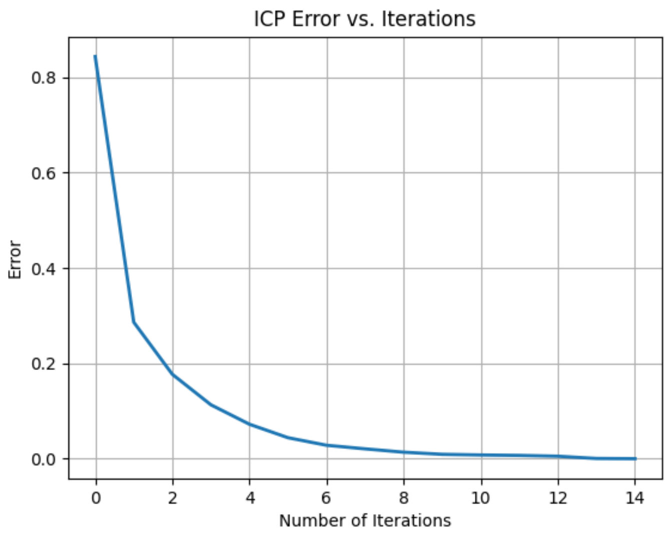


Figure 22: error vs iterations cloud target1

Target 2:

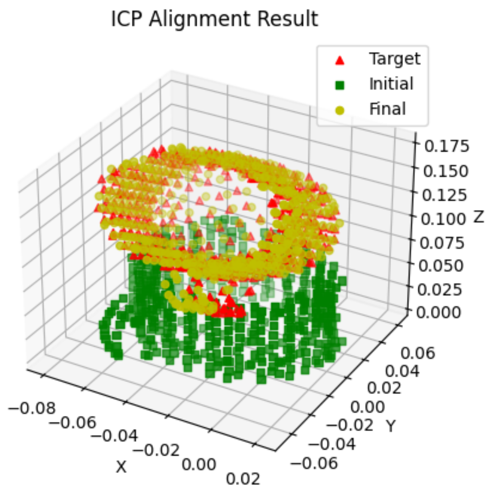


Figure 23: icp alignment result cloud target2

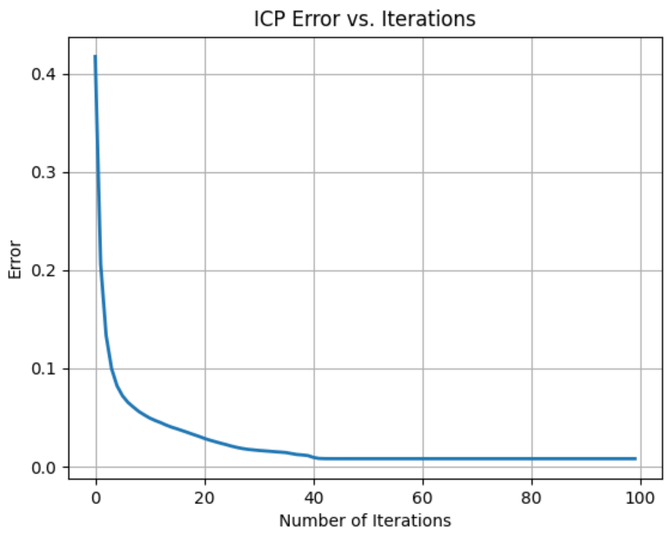


Figure 24: error vs iterations cloud target2

Target 3:

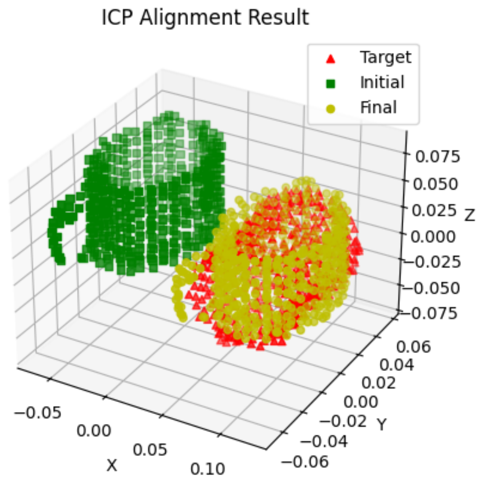


Figure 25: icp alignment result cloud target3

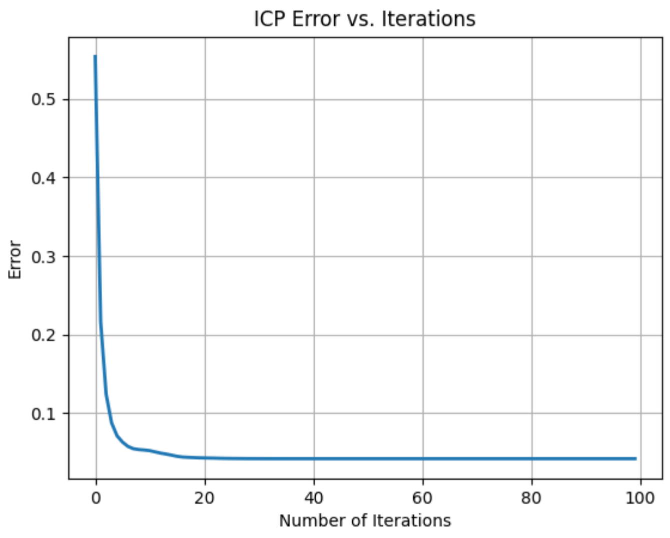


Figure 26: error vs iterations cloud target3

(c) Target 2 was the most difficult.Target 0 was the easiest.
Poor initial alignment, resulting in fewer corresponding points in early iterations, and complex or irregular surface features can cause alignment issues due to fewer clear correspondences.