MINI PROJECT 1: PERCEPTION

Due: Monday, September 22 11:59pm EST

The objective of MP 1 is to learn Iterative Closest Point (ICP) algorithm for point cloud registration and tracking. In this Mini Project, we will use MuJoCo to simulate known objects moving through a predefined trajectory. Your goal is to implement an ICP algorithm to continuously track the object's pose.

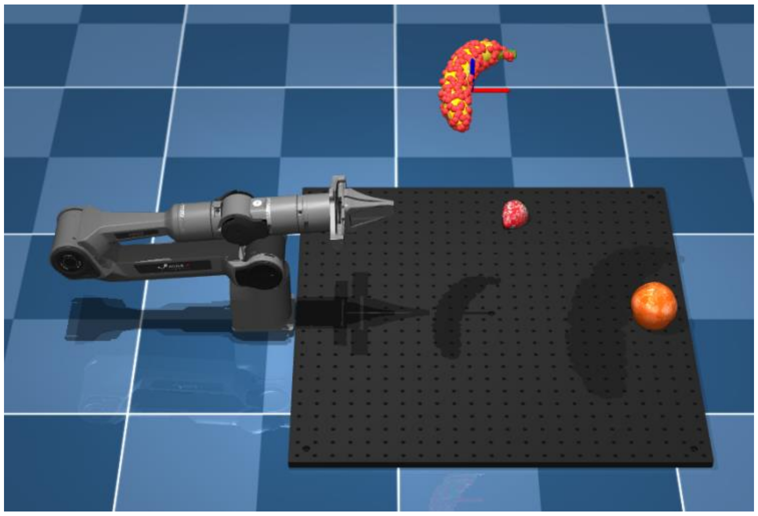

Figure 1. MuJoCo simulation of tracking a banana with ICP. The point cloud with an estimated object pose is shown as a collection of red dots on the banana.

A. Point Cloud Registration with Known Correspondences (5 pts)

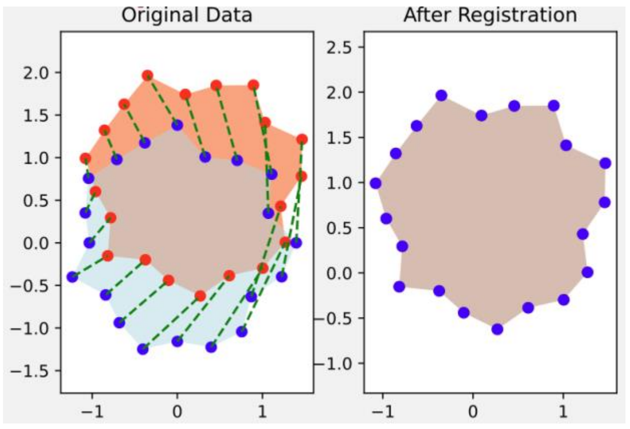

Figure 2. Point cloud registration with perfect correspondence. Source: Robotic Manipulation, Russ Tedrake, 2024

As a first step to implement an ICP algorithm, you will work on a point cloud registration function.

Let us denote a set of model points ${}^{O}p^{m} \in \mathbb{R}^{N_m}$ in object frame where $N_m \in \mathbb{N}$ and a set of scene points in world frame $p^{s} \in \mathbb{R}^{N_s}$ where $N_s \in \mathbb{N}$. Point cloud registration is the process of determining the rotation matrix $R \in \mathbb{R}^{3\times 3}$ and the translation vector $p \in \mathbb{R}^{3}$ that effectively transforms the model points to scene points. Given a correspondence vector $c \in [1, N_m]^{N_s}$ where the i^{th} element $c_i = j$ indicates that a scene point s_i corresponds with the model point m_j, the point cloud registration finds the rotation matrix and the translation vector which altogether best transform the model points to scene points. A 2D version of this process is described in Figure 2 where blue points are model points, and red points are scene points. With perfect correspondences, as described in the figure, you will be able to find the transformation that perfectly transforms the model points to the scene points. For specific details on how point cloud registration works, please refer to lecture notes and Section 4.3 in these notes.

Your task is to complete `register` method in `PointCloud` class in `PointCloud.py`.

B. Nearest Neighbor (3 pts)

In reality, you do not know the correspondences between model points and scene points. Therefore, you need to find the correspondence for each point in the model point cloud to a point in the scene point cloud. To this end, we will use "nearest neighbor" approach where the correspondence of a model point is determined to be the closest point in the scene point cloud in Euclidean distance. For specific details on nearest neighbor, please refer to lecture notes and Section 4.4 in these notes.

Your task is to complete `nearest_neighbor` method in the `PointCloud` class.

Note: You cannot use any libraries other than numpy.

C. Iterative Closest Point (ICP) (7 pts)

The ICP algorithm performs point cloud registration by iteratively alternating between computing correspondences through nearest neighbor and using those correspondences for updating the estimated pose of a model point cloud. For specific details on ICP, please refer to lecture notes and Section 4.4 in these notes.

Your task is to complete `icp` method in the `PointCloud` class.

D. Testing your implementations in `PointCloud.py`

We provide a testing script that checks your implementations in the `PointCloud.py`. If you haven't installed pytest, please do so by running

```
pip install pytest
```

Then, run

```
python3 -m pytest test_point_cloud.py
```

This test script does sanity checks on your implementation. **Passing the sanity checks does not guarantee a successful implementation as these tests are minimal, but they should indicate that you are on the right track!** Later this week, we will also upload an autograder on Gradescope.

E. Testing your implementation in MuJoCo (Linux):

Note: If you have not completed MP0, we strongly recommend you to do so before starting MP1. The rest of the instructions assume that you have successfully completed MP0.

# Download and unzip `MP1.zip` from Canvas and place it under

```
CS4803ARM_Fall2025/user_data
```

Note: Any files placed in this directory will be accessible from the Docker container

# Change the directory

```
cd CS4803ARM_Fall2025/docker
```

# Important: You will not see GUI if you don't run this command before running the shell script.

```
xhost +
```

# Run the convenience script to run the container

```
./docker_run.sh
```

or

```
./docker_run_gpu.sh
```

(If your computer has an Nvidia GPU and you have installed Nvidia container toolkit. MuJoCo will be much smoother and faster.)

Change the directory to MP1 inside the Docker container.

```
cd user_data/MP1
```

# Run the MuJoCo simulation.

```
python3 test_mujoco.py
```

You can try a more challenging trajectory by changing line 103 in `test_mujoco.py`

`diff_case = "easy"` to `diff_case="medium"`

F. Testing your implementation in MuJoCo (Mac):

# Download and unzip MP1.zip from Canvas and place it under

```
CS4803ARM_Fall2025/user_data
```

# Change the current directory to the MP1 folder

```
cd CS4803ARM_Fall2025/user_data/MP1
```

# Run the MuJoCo simulation.

```
mjpython test_mujoco.py
```