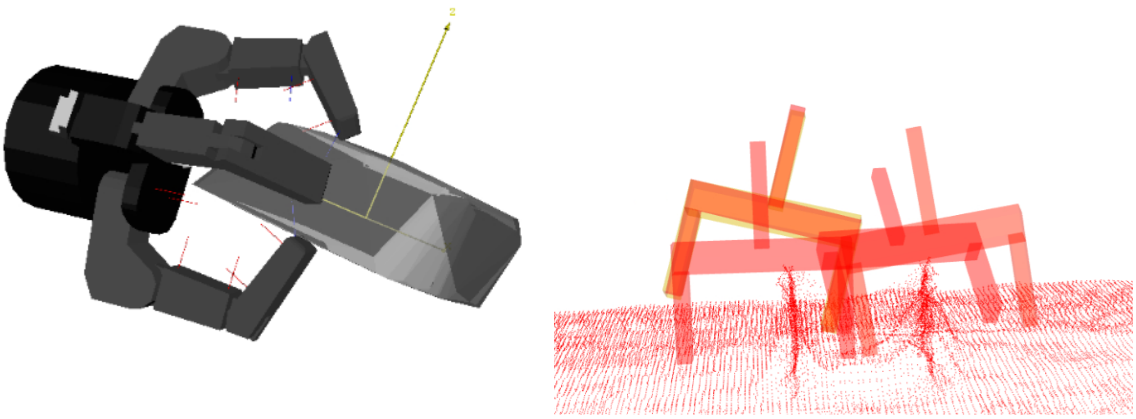
B3M33HRO HW3
Grasping

1 Introduction

You are provided with point clouds from noisy real depth cameras. The individual point clouds are different views on an object on a table. Your task is to combine them to get a full view of a scene and use it to get grasp from two pipelines: GraspIt! and GPD. Examples of grasps can be seen in Figure 1.



(a) Example grasp from GraspIt!

(b) Example grasp from GPD.

Figure 1: Examples of grasps.

2 Assignment

- Download the assignment from the course website.
- The first part of the assignment is a python script that will allow you to read, manipulate, and process point clouds. You are give three point clouds of the same object.
- Combine the provided point clouds in one. You can process them as you want and as needed—downsampling, removing outliers, cropping with a bounding box.
  - Set limits for the *z-axis* for the bounding box of the workspace. Different values may be needed for GraspIt! and for GPD.
  - The processed point cloud should have the right number of points to balance betwen too much data and not enough data.,and should not contain unnecessary holes.
  - Decide whether to use the processing on the final point cloud, or on individual samples.
  - See Open3D Point Cloud Class and Open3D Point Cloud Tutorial.
- Prepare point cloud for GPD.
  - This point cloud **must have** “a table” under the object.
  - It is better to translate it into (0,0,0) otherwise you will have to zoom out in the GPD output.
  - save the point cloud as a .pcd file.
- Prepare point cloud for GraspIt!. It must contain only the object, without the table.

- Create a mesh from the point cloud, translate it to position (0,0,0), and save it to file.
  - Select the appropriate method that will work in GraspIt!
  - See Surface Reconstruction Tutorial and Open3D Triangle Mesh Class.
  - **Note:** The item in the point cloud is an opened box, *i.e.,* it is concave and has a hollow part. However, it is hard to obtain a concave mesh for this point cloud, so it is fine if your mesh looks like a closed box.
- Complete the code. Please, **pay attention to code quality and performance.**
- The second part of the assignment will take place on GitPod.
  - Create an account (you can log in with GitHub).
  - Create a workspace by clicking here.
  - Upload your .pcd and .ply files, and the Jupyter notebook for part 2, inside the B3M33HRO-gitpod folder. These files should be available in the environment in the /workspace folder. Alternatively you can email them to yourself and download them inside the environment.
  - Run the Jupyter notebook. You might have to run it with the `--allow-root` option.
- Open the GraspIt interface and:
  - Clear the World;
  - Import *Barrett* as a robot;
  - Import your mesh as a graspable body;
  - See GraspIt! commander API.
  - **Note:** if you see only black/grey after you load the robot and the body, zoom-out in the GraspIt GUI.
- Run the Eigengrasp planner and sort the grasp by ϵ-quality.
  - ϵ-quality: the closer to 1, the better. **Note:** if you close the GraspIt interface, you will probably need to restart the kernel in the notebook before you run it again.
- Check if the grasp looks like you would assume and take a picture of it.
- Run the GPD and take a picture of the output.
  - Make it run as fast as possible.
    - The run-time can vary on different computers, but if the GPD runs for more than 5 seconds, it is too much even on a slow computer.
    - Right processing of the point cloud can help you to reduce time, or you can play with the values in *eigen_params.cfg* (in Docker located in */home/docker/gpd/cfg/eigen_params.cfg.*

3 Points

- Correct GraspIt! output - 3 points
  - Screenshot of the grasp
  - Correct code
- Correct GPD output - 2 points
  - Screenshot of the grasp
  - Correct code