COMP0129
Pick and Place Under Constraint
Group Assignment 3

Chiang Hou Nam
15055142
`hou.chiang.20@ucl.ac.uk`

Renjie Zhou
17000505
`renjie.zhou.20@ucl.ac.uk`

Mohamed Ali Shire
17019507
`mohamed.shire.17@ucl.ac.uk`

15th Apr, 2021

Module Lead:	Dimitrios Kanoulas
Contact e-mail:	`d.kanoulas@ucl.ac.uk`
Department:	Department of Computer Science
Submission Date:	15th of April 2021

Contents

Question 1

1.1

A function `filterEnvironment` is implemented to use `VoxelGrid` filter from `pcl` library, which downsamples the point cloud from This is where a 3D voxel grid is created over the input point cloud, all the points in each voxel are then approximated using their centroid. This reduces the number of points to work with, while still retaining the underlying shape of the surfaces. By reducing the number of points that are processed, using `VoxelGrid` filter speeds up the segmentation process. When the filter is applied by pressing "f", the number of points reduces from 83,484 after the original `PassThrough` filter, to 5,687, while still preserving the shape of the cylinder. The time taken by the cylinder extraction process goes from taking 4,278 microseconds to 490 microseconds.

1.2

Assuming that we know the approximate central position of the cylinder and the estimation error of the area in every direction. We can implement a `PassThrough` filter from `pcl` library, setting the filter limits in each axis to be plus and minus the error estimation. This way any points outside of the field will be discarded. Reducing the number of points that are being worked with increases the speed of cylinder extraction. The filter is implemented in function `fastFilterEnvironment`. By pressing "p", applying the fast filter reduced the number of points from 83,483 to 43,122. With the fast filter, the execution of cylinder extraction can only take 3,384 microseconds.

However, after comparison, the time taken for the cylinder extraction after applying the filter from question is much shorter than the fast filter implemented in question , probably due to the fact that VoxelGrid has reduced the number of points throughout the scene, including the region of interest, whereas the fast filter keeps all the points in the region where the cylinder is approximately at, resulting in much more points left for cylinder extraction.

1.3

The publisher `cylinder_pose_pub` was initialised in the `CylinderSegment` object constructor. In the `CloudCB` function, If the cylinder has been found, the cylinder parameters are stored in a geometry pose message called `cylinder_pose`. This is then published to the `cylinder_pose` topic.

Question 2

2.1

To ensure the modularity and readability of the code, a helper function is written to publish the boolean messages to the three publishers for the bumper sensor data, namely "`bumper1_pub`", "`bumper2_pub`" and "`bumper3_pub`", which are initialised in the "`test_cw3.cpp`". Global variables are initialised in the `cw3.cpp` as the bumper sensor which can be toggled based on whether object or cylinder is put or removed from the table. The bumper sensor of the empty table publishes "false" boolean value whereas the bumper sensor of the occupied table publishes "true" boolean value. Hence at the beginning, `bumper1_pub` and `bumper3_pub` publish "false" whereas `bumper2_pub` publishes "true" because the object is place on table2 at the beginning.

2.2

In this case, the code with I/O is implemented in the "`test_cw3.cpp`" as such:

```
// press 1 to move object from ground to table1
// centre in pick-and-place without constraint
      if (ch == '1')
      {
        std::string fromTable = "";
        std::string toTable   = "table1";
        bool isCylinder = true;
        cw3_obj.pick(move_group, fromTable, isCylinder);
        ros::WallDuration(1.0).sleep();
        cw3_obj.place(move_group, toTable, isCylinder);
      }
```

Where `pick` and `place` are the functions written in file `cw3.cpp` to perform pick-up and place-down of the objects based on where the objects are. There are two other arguments apart from `move_group`: `fromTable` or `toTable`, which indicate which table to pick up from or place to, changing the positions of pre-grasping and post-grasping for the robot arm automatically based on the stored locations of the three tables; `isCylinder`, which indicates whether the current target is the cylinder or the object and the exact coordinate on z-axis is then changed accordingly, because the cylinder and the object have different heights. This is to pave the way for the implementation in question 3.

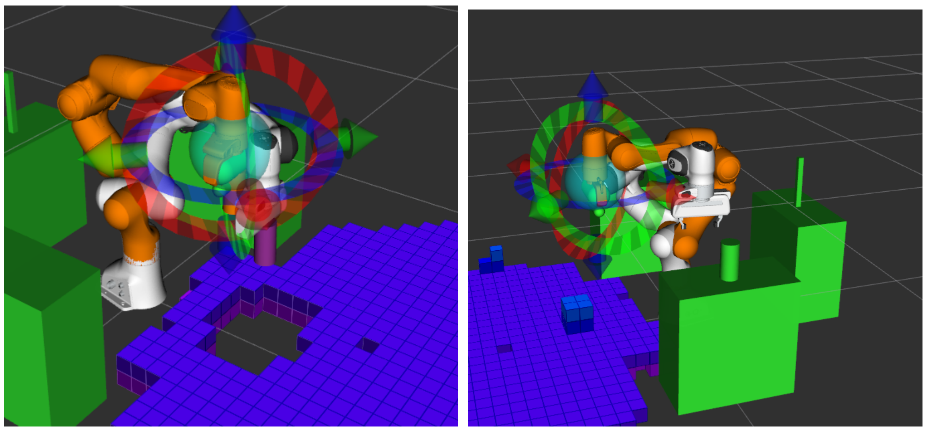

Figure 1: *Grasping and placing onto table1 without constraint*

As shown in the figure 1, upon pressing "1", the robot automatically detects the robot at the ground location and moves the robot from the ground location to table1.

Question 3

This question has posed extra constraints to the pick and place of the object, e.g. the tables and the hindering object (stick) that occupies table 2 at the beginning. Hence we need to use the information published into `bumper1_pub`, `bumper2_pub` and `bumper3_pub` to detect whether the designated table to move the cylinder to is empty. Also, even if the bumper sensor detects that the designated table is occupied, we need to distinguish whether the occupant is cylinder (in which case nothing needs to be done) or the obstruction objects (in which case the object should be moved away to other free tables).

In this case, we defined various local variables together with the latest pose information of the cylinder to help the robot decide what to do automatically. Firstly, we check whether the designated table to move to is occupied by checking the information published by the bumper sensors. If the designated table is occupied, we then check whether the occupant is cylinder by checking the latest published pose of the cylinder from `ROS` topic `cylinder_pose`. If the x and y coordinates of the latest cylinder pose match the x and y coordinates of the designated table, we don't need to do anything; else, we need to move the stick object to other free table.

Locally, we created a variable `isCylinder` to indicate whether the current object to be picked and placed is cylinder or not, because the grasping pose changes accordingly. We also defined `fromTable` and `toTable` to notify the robot which is the table to pick from and to place the object onto. If the `toTable` is occupied based on the bumper sensor data, the robot then checks for the occupant identity as described above before moving the cylinder.

The work flow for pressing either 1, 2, or 3 are the same. However, the tables and bumpers that we are checking are different subject to the destination table. Below shows the case when we want to move the cylinder to table1:

```
if (ch == '1')
{
  std::string fromTable = "";
  std::string toTable   = "";
  // Check table 1 if there is an object via bumper
  if (cw3_obj.bumper1HasObject)
  {
    if (cw3_obj.objPosX == 0.0 && cw3_obj.objPosY == 0.5)
    {
      // Cylinder already on table one, do nothing.
      ROS_INFO("Cylinder already on table 1");
    }
    else
    {
      // Stick object on table one, move it to any empty table.
      fromTable = "table1";
      if (!cw3_obj.bumper2HasObject)
      {
        toTable = "table2";
      }
      else if (!cw3_obj.bumper3HasObject)
      {
        toTable = "table3";
      }

      // Move stick
      bool isCylinder = false;
      cw3_obj.pick(move_group, fromTable, isCylinder);
      cw3_obj.place(move_group, toTable, isCylinder);

      // Update sensor value
      if (toTable.compare("table2") == 0)
      {
        cw3_obj.bumper2HasObject = true;
      }
      else if (toTable.compare("table3") == 0)
      {
        cw3_obj.bumper3HasObject = true;
      }
      cw3_obj.bumper1HasObject = false;
    }
  }

  if (!cw3_obj.bumper1HasObject)
  {
    // Now find the location of the cylinder
    if (cw3_obj.objPosX == -0.5 && cw3_obj.objPosY == 0.0)
    {
      fromTable = "table2";
    }
    else if (cw3_obj.objPosX == 0.0 && cw3_obj.objPosY == -0.5)
    {
      fromTable = "table3";
    }
    else
    {
      fromTable = "";
    }
    toTable = "table1";

    // Move cylinder
    bool isCylinder = true;
    cw3_obj.pick(move_group, fromTable, isCylinder);
    cw3_obj.place(move_group, toTable, isCylinder);
    cw3_obj.updateCylinderPose(cylinder_pose_pub, toTable);
    cw3_obj.bumper1HasObject = true;

    // Update bumper
    if (fromTable.compare("table2") == 0)
    {
      cw3_obj.bumper2HasObject = false;
    }
    else if (fromTable.compare("table3") == 0)
    {
      cw3_obj.bumper3HasObject = false;
    }
  }
}
```

For full section of the code please refer to the code in `test_cw3.cpp`.

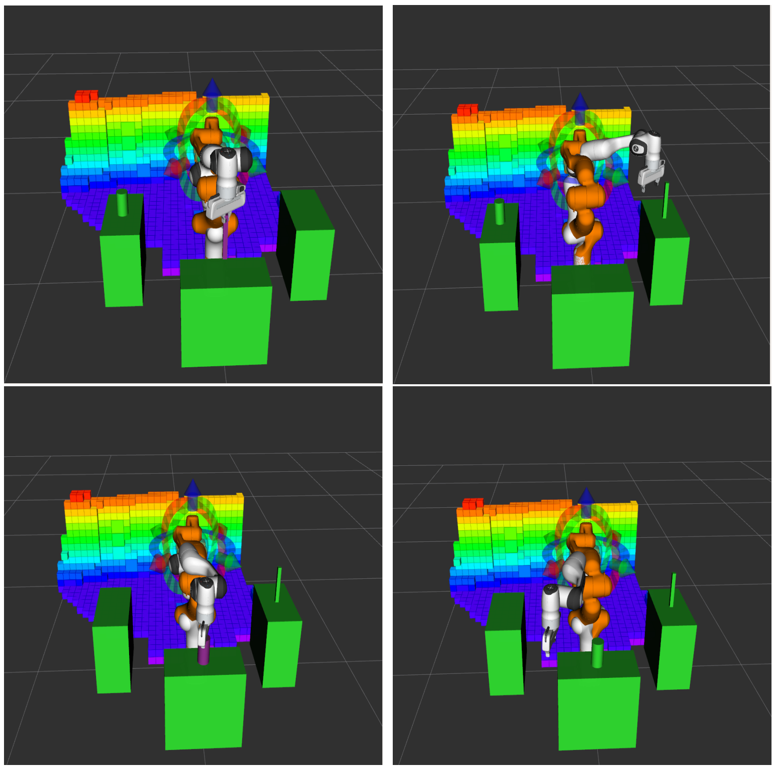

Figure 2: *Robot picks stick from table 2 and places stick onto table 3, then pick the cylinder from table 1 and places it on table 2*

The I/O implementation is done in `test_cw3.cpp` for the key pressing. As shown in figure 2, when pressing "2", the robot moves the stick object from table 2 to the empty table 3, before moving the cylinder from table 1 to table 2.

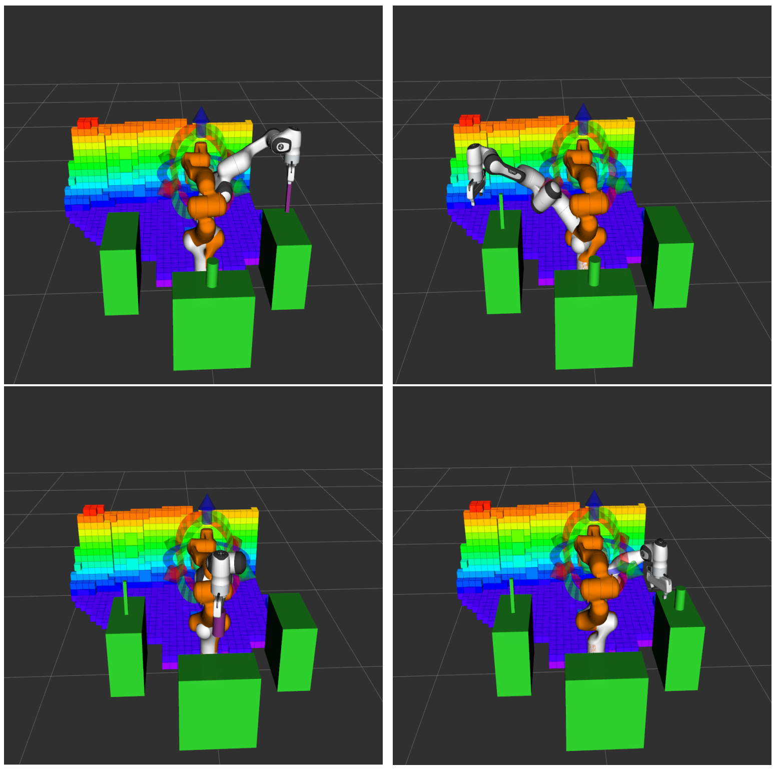

Figure 3: *Robot picks stick from table 3 and places stick onto table 1, then pick cylinder from table 2 and places it on table 3*

Similarly, as shown in figure 3, when pressing "3", the robot moves the stick object from table 3 to the empty table 1, before moving the cylinder from table 2 to table 3.

During the development, we have discovered that we also need different parameters for pre-grasp approach and post-grasp retreat. We have tried different numbers and found the following worked. These are implemented in the function `void CW3::place()` at `cw3.cpp`:

```

  if(isCylinder)
  {
    // Approach
    place_location[0].pre_place_approach.min_distance     = 0.05;
    place_location[0].pre_place_approach.desired_distance = 0.15;

    // Retreat
    place_location[0].post_place_retreat.min_distance     = 0.1;
    place_location[0].post_place_retreat.desired_distance = 0.25;
  }
  else
  {
    // Approach
    place_location[0].pre_place_approach.min_distance     = 0.05;
    place_location[0].pre_place_approach.desired_distance = 0.35;

    // Retreat
    place_location[0].post_place_retreat.min_distance     = 0.05;
    place_location[0].post_place_retreat.desired_distance = 0.55;
  }
```