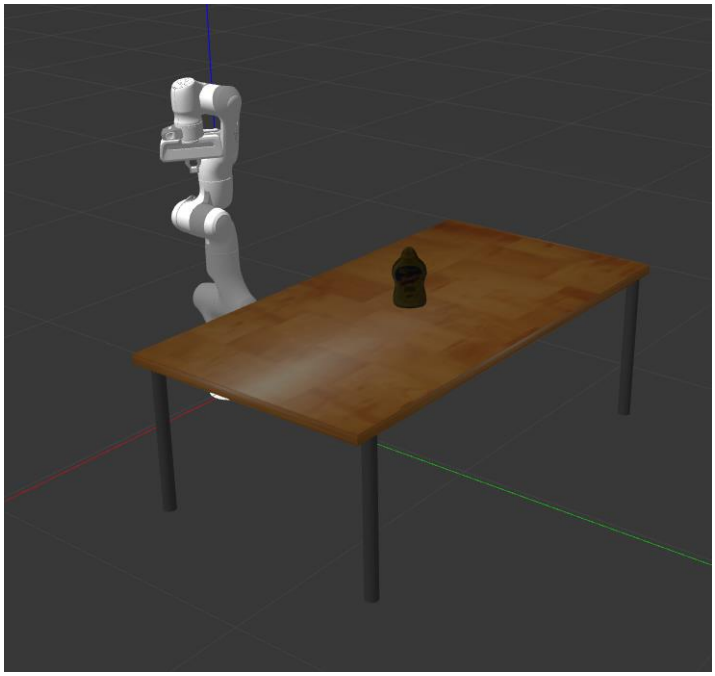
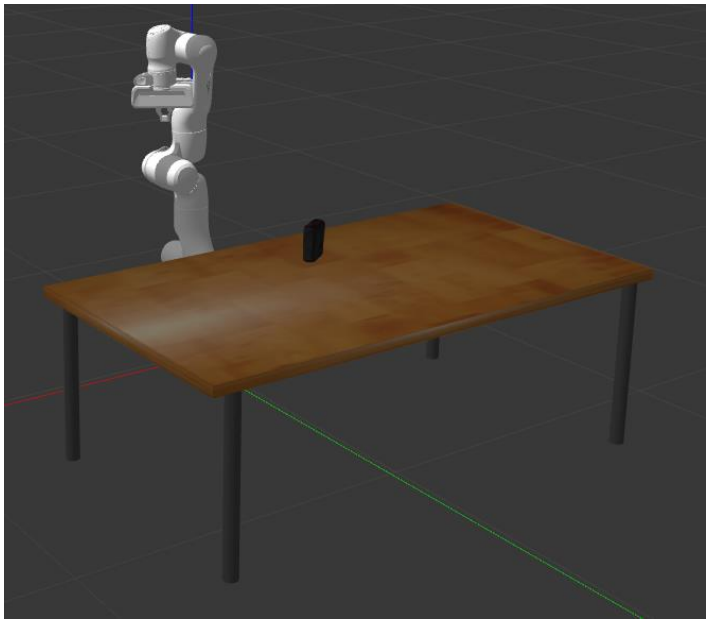
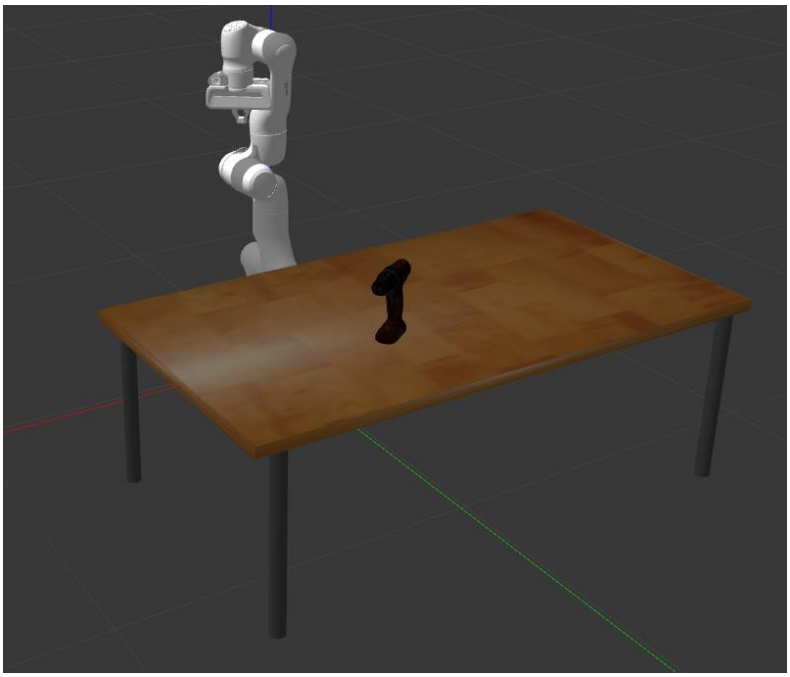
RBE 450X- Group Assignment

Team members: Chinmay Todankar, Prathamesh Bhamare

Step 1:



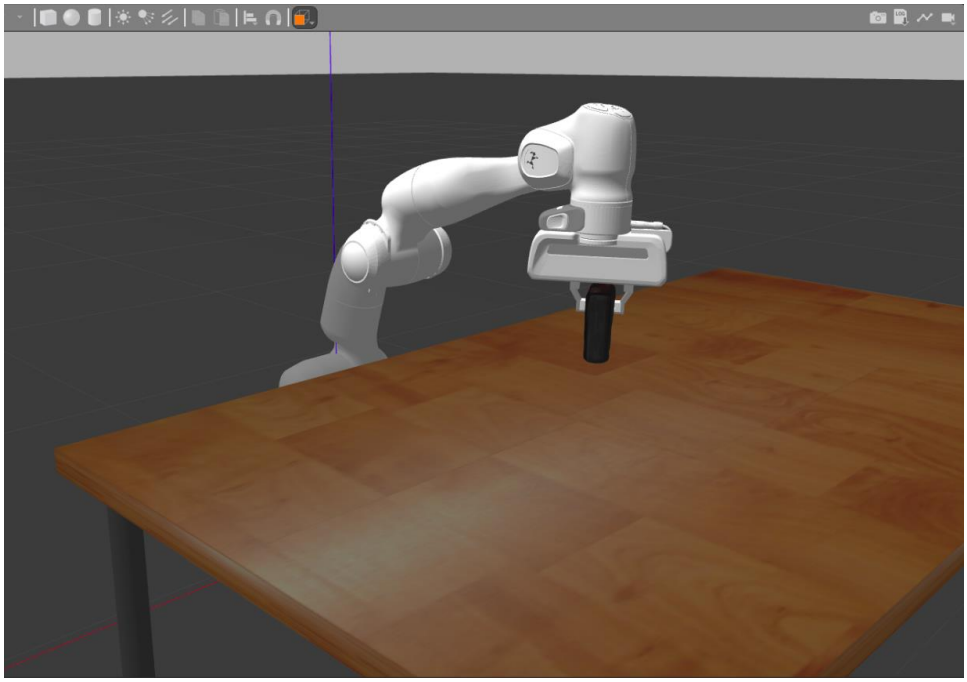




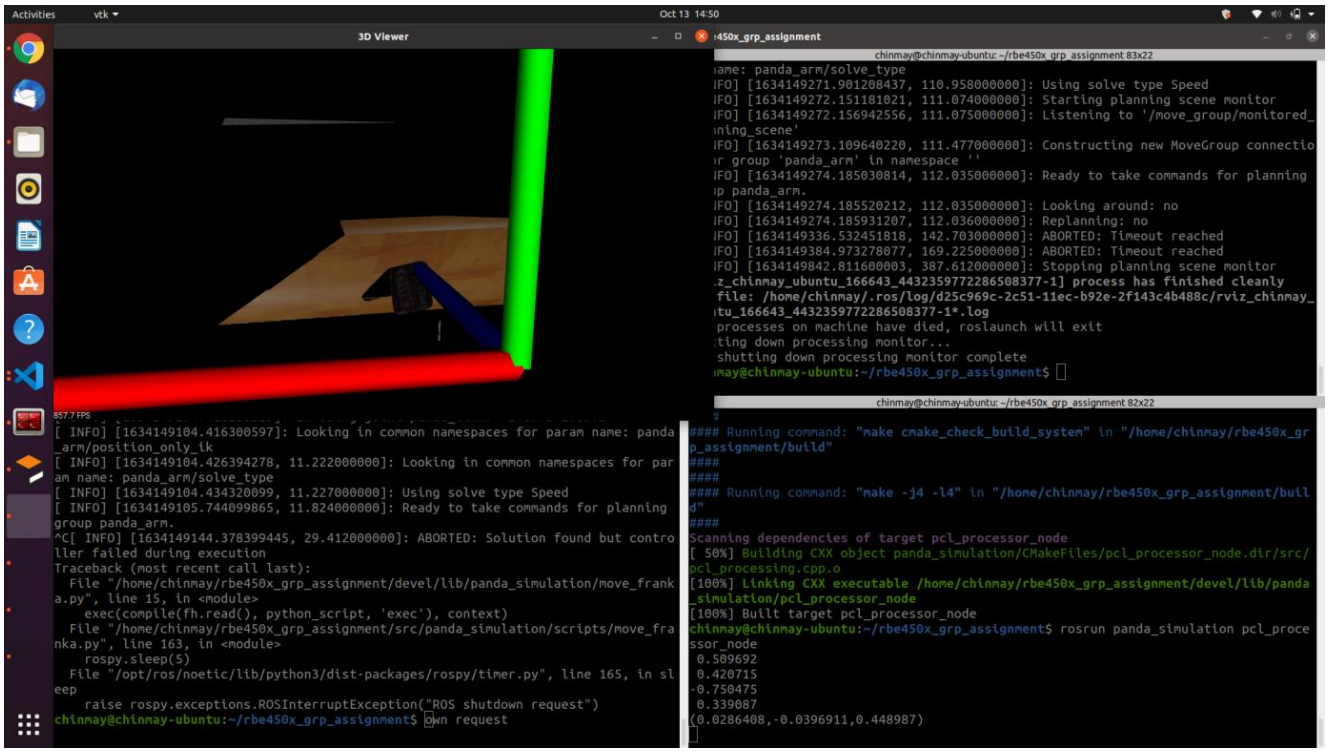
We created ".dae" file from the obj file and texture files from the database using meshlab software, so that the texture was retained in the gazebo models.

Since, the default table height was larger, so we scaled down the table height as per our need and the sdf files are present in the model folders of the panda simulation.

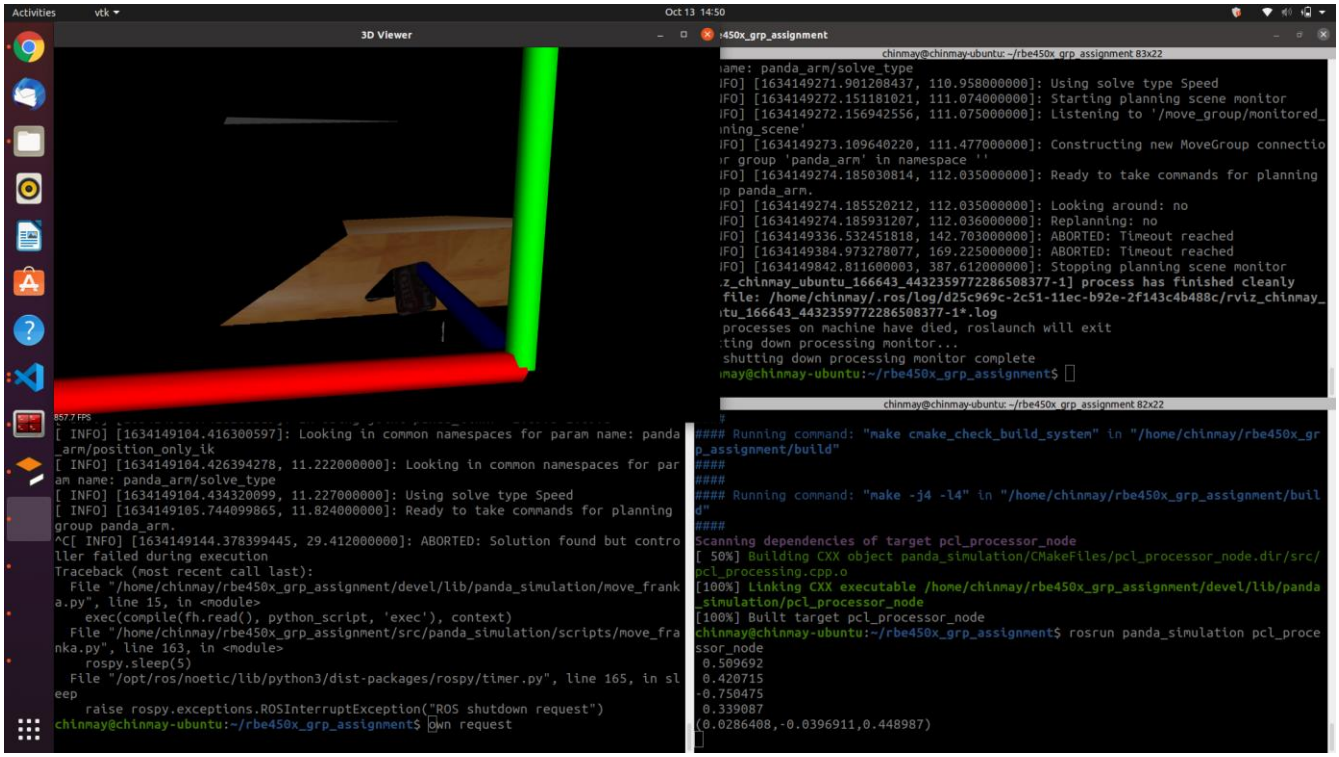
Step 2:



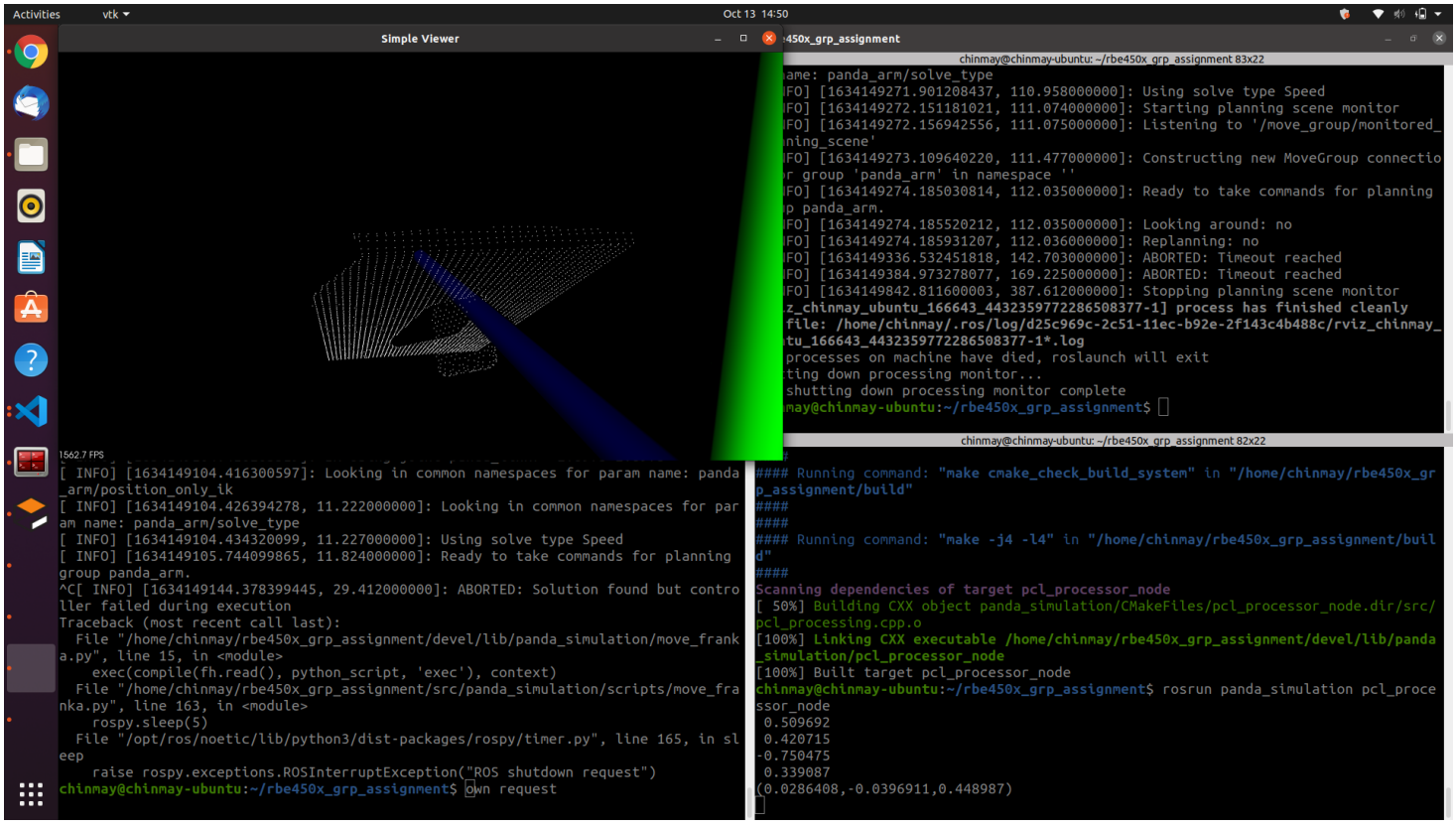
Step 3:



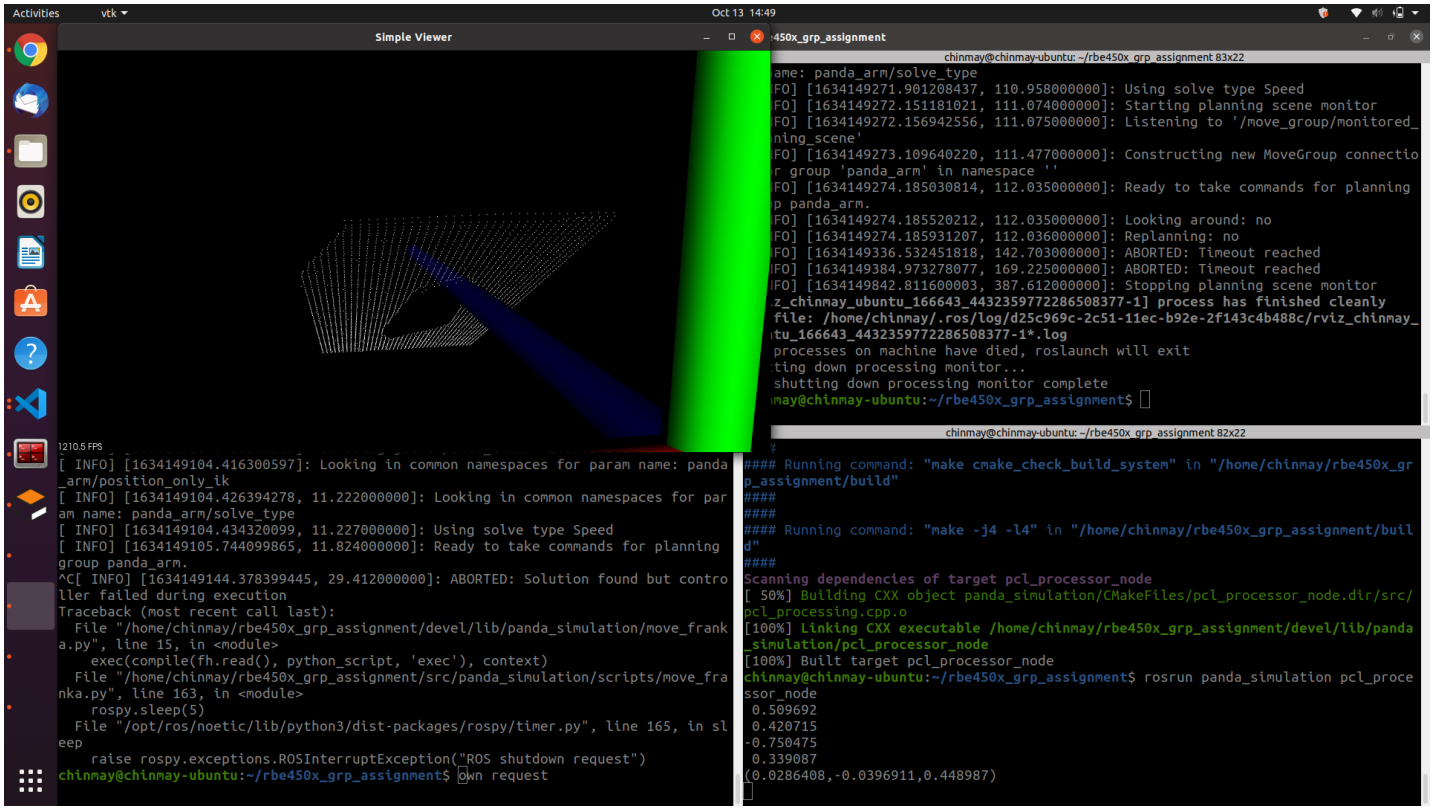
Raw point cloud



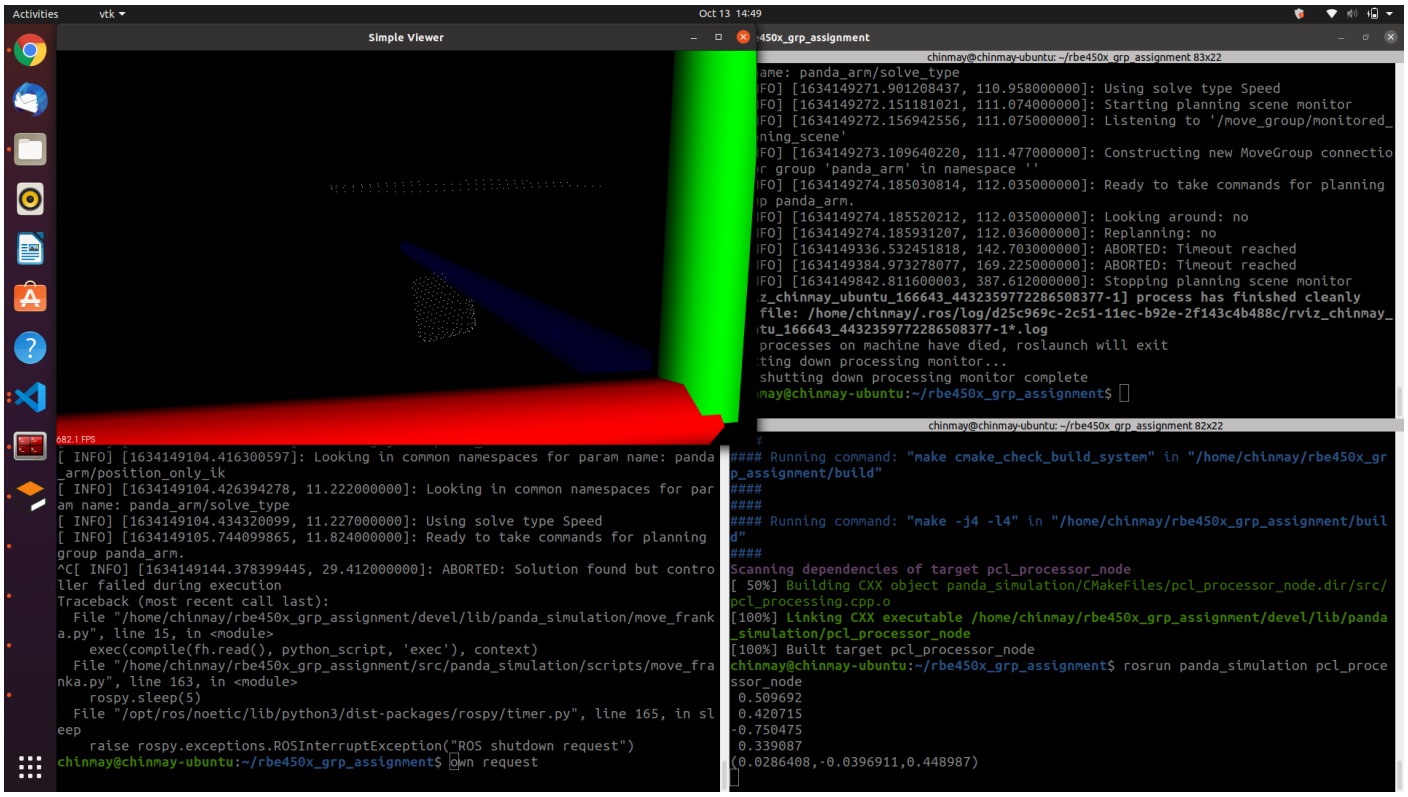
Down sampled point cloud



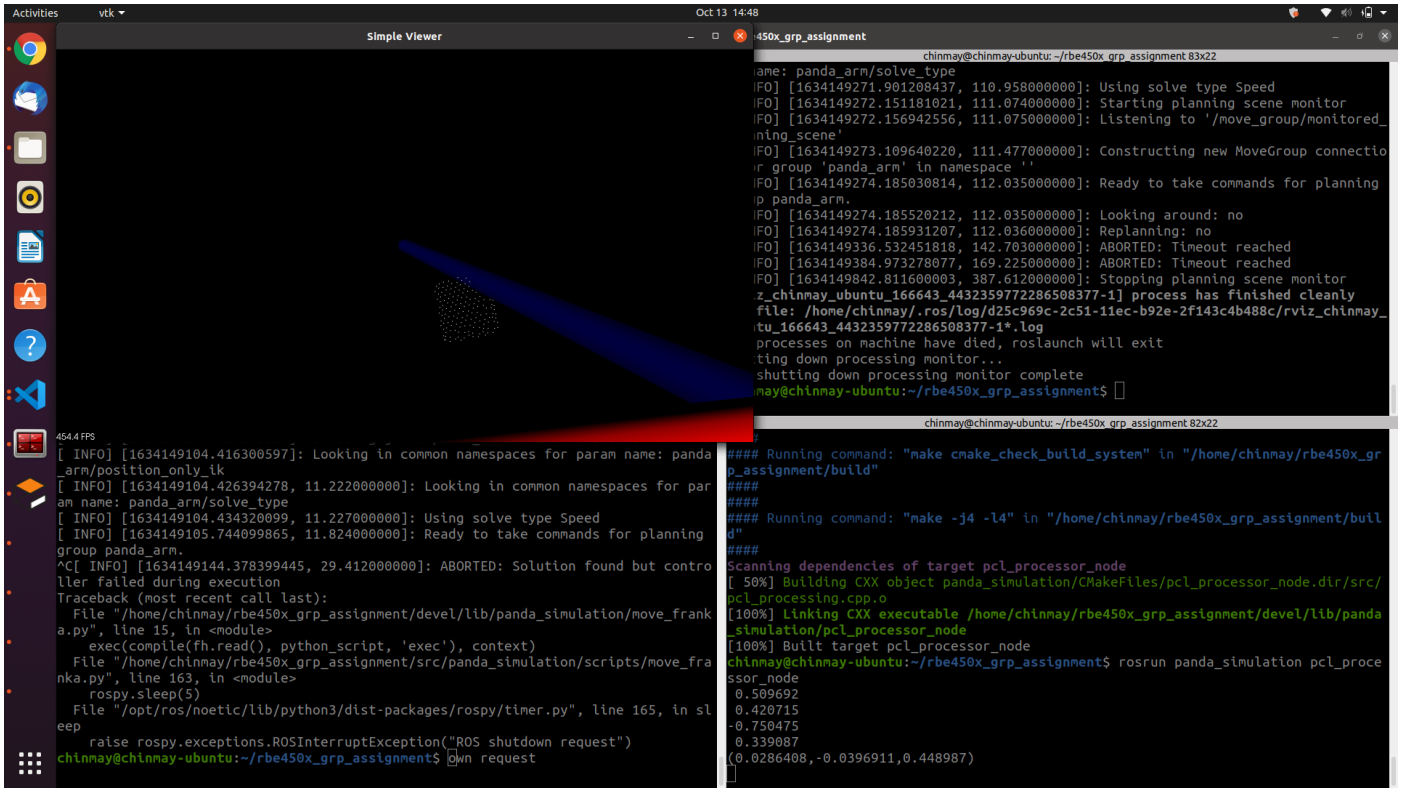
Pass through filtered point cloud



Segmented major plane

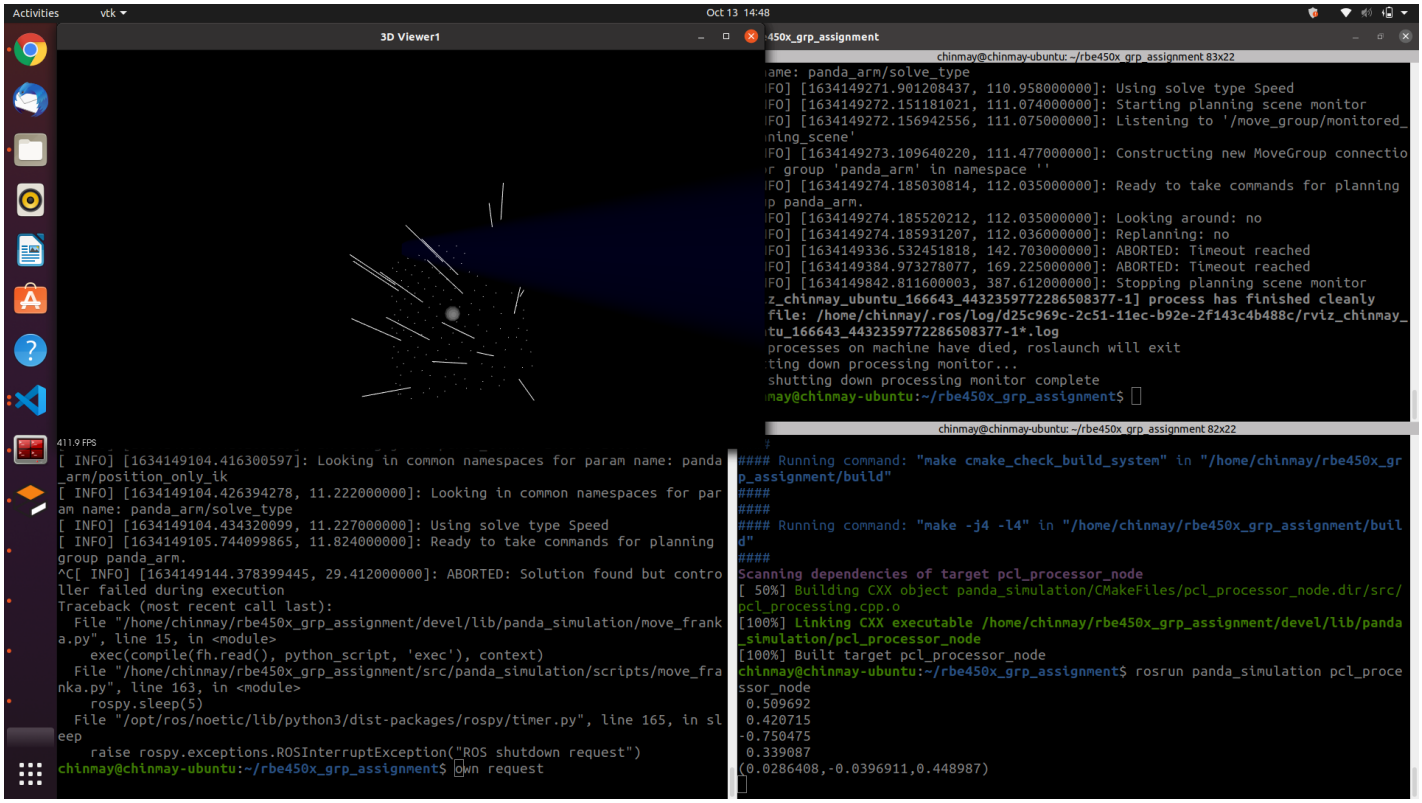


Point cloud without major plane



Object plane

Step 4:



Normals and centroid

Progress: We were able to detect centroid and normals successfully. Later, we were stuck at how to use this centroid coordinate information to estimate the pose of the gripper. We were able to find the translation of gripper pose in world frame using tf library, but we were not able to find the required rotation of the frame. For finding out the rotation axis, we tried taking the direction of the normals, but we were not getting the other 2 orthogonal axes.

The code is attached in the following pages.

```cpp
#include "ros/ros.h"
#include "pcl_ros/point_cloud.h"
#include <pcl/console/parse.h>
#include <pcl/filters/extract_indices.h>
#include <pcl/io/pcd_io.h>
#include <pcl/point_types.h>
#include "pcl/point_types.h"
#include "pcl_conversions/pcl_conversions.h"
#include <pcl/sample_consensus/ransac.h>
#include <pcl/sample_consensus/sac_model_plane.h>
#include <pcl/sample_consensus/sac_model_sphere.h>
#include <pcl/visualization/cloud_viewer.h>
#include <pcl/point_types.h>
#include <pcl/features/normal_3d.h>
#include <thread>
#include <pcl/filters/voxel_grid.h>
#include <pcl/filters/passthrough.h>
#include <geometry_msgs/PoseStamped.h>

using namespace std;
using namespace ros;
using namespace pcl;

typedef pcl::PointCloud<PointXYZ> PointCloud;


pcl::visualization::PCLVisualizer::Ptr simpleVis
(pcl::PointCloud<pcl::PointXYZ>::ConstPtr cloud)
{
  // --------------------------------------------
  // -----Open 3D viewer and add point cloud-----
  // --------------------------------------------
  pcl::visualization::PCLVisualizer::Ptr viewer (new
  pcl::visualization::PCLVisualizer ("Simple Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  viewer->addPointCloud<pcl::PointXYZ> (cloud, "sample cloud");

//   viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 1, "sample cloud2");
  viewer->addCoordinateSystem (1.0);
  viewer->initCameraParameters ();
  return (viewer);
}

pcl::visualization::PCLVisualizer::Ptr rgbVis
(pcl::PointCloud<pcl::PointXYZRGB>::ConstPtr cloud)
{
  // --------------------------------------------
  // -----Open 3D viewer and add point cloud-----
  // --------------------------------------------
  pcl::visualization::PCLVisualizer::Ptr viewer (new
  pcl::visualization::PCLVisualizer ("3D Viewer"));
  viewer->setBackgroundColor (0, 0, 0);
  pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZRGB> rgb(cloud);
  viewer->addPointCloud<pcl::PointXYZRGB> (cloud, rgb, "sample cloud");
//   viewer->setPointCloudRenderingProperties
(pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud");
  viewer->addCoordinateSystem (1.0);
  viewer->initCameraParameters ();
  return (viewer);
}

pcl::visualization::PCLVisualizer::Ptr normalsVis (
    pcl::PointCloud<pcl::PointXYZ>::ConstPtr cloud,
    pcl::PointCloud<pcl::Normal>::ConstPtr normals)
{
  // --------------------------------------------------------
  // -----Open 3D viewer and add point cloud and normals-----
  // --------------------------------------------------------
  pcl::visualization::PCLVisualizer::Ptr viewer (new
  pcl::visualization::PCLVisualizer ("3D Viewer1"));
  viewer->setBackgroundColor (0, 0, 0);
//   pcl::visualization::PointCloudColorHandlerRGBField<pcl::PointXYZ> rgb(cloud);
```

```
66  //   viewer->addPointCloud<pcl::PointXYZ> (cloud, rgb, "sample cloud");
67      viewer->addPointCloud<pcl::PointXYZ> (cloud, "sample cloud");
68  //   viewer->setPointCloudRenderingProperties
    (pcl::visualization::PCL_VISUALIZER_POINT_SIZE, 3, "sample cloud1");
69    viewer->addPointCloudNormals<pcl::PointXYZ, pcl::Normal> (cloud, normals, 10,
      0.05, "normals");
70    viewer->addCoordinateSystem (1.0);
71    viewer->initCameraParameters ();
72    return (viewer);
73  }
74
75  std::mutex viewerMutex;
76  pcl::visualization::PCLVisualizer::Ptr viewerN;
77  pcl::visualization::PCLVisualizer::Ptr viewerN1, n2, n3, n4, n5, n6, n7, n8;
78
79  pcl::PointCloud<PointXYZRGB>::iterator b1;
80  pcl::PointCloud<PointXYZ>::iterator b2;
81  pcl::PointCloud<PointXYZRGB> cloudRGB;
82  pcl::PointCloud<PointXYZ> cloud;
83
84  pcl::PointCloud<PointXYZRGB> majorPlaneRGB;
85  pcl::PointCloud<PointXYZ> majorPlane;
86
87  pcl::PointCloud<PointXYZ> object;
88  pcl::PointCloud<PointXYZ> objectPlane;
89  pcl::PointCloud<PointXYZRGB> objectPlaneRGB;
90  vector<int> inliers;
91  Publisher pub;
92
93  void cloudCB(const sensor_msgs::PointCloud2 &input)
94  {
95
96      // fromROSMsg (input, cloud);
97      fromROSMsg (input, cloudRGB);
98      // n2 = rgbVis(cloudRGB.makeShared());
99      pcl::PointCloud<PointXYZ>::Ptr cloud_filtered (new pcl::PointCloud<PointXYZ>);
100     pcl::PointCloud<PointXYZRGB>::Ptr cloud_filteredRGB (new
        pcl::PointCloud<PointXYZRGB>);
101     pcl::VoxelGrid<PointXYZRGB> sor;
102     sor.setInputCloud (cloudRGB.makeShared());
103     sor.setLeafSize (0.01f, 0.01f, 0.01f);
104     sor.filter (cloudRGB);
105
106     pcl::copyPointCloud(cloudRGB, cloud);
107
108     // n3 = simpleVis(cloud.makeShared());
109     // pcl::copyPointCloud(*cloud_filteredRGB, *cloud_filtered);
110
111
112     // n3 = simpleVis(cloud.makeShared());
113
114     pcl::PassThrough<pcl::PointXYZ> passZ;
115     passZ.setInputCloud (cloud.makeShared());
116     passZ.setFilterFieldName ("z");
117     passZ.setFilterLimits (0.25, 1.0);
118     passZ.filter (cloud);
119
120     pcl::PassThrough<pcl::PointXYZ> passX;
121     passX.setInputCloud (cloud.makeShared());
122     passX.setFilterFieldName ("x");
123     passX.setFilterLimits (-0.15, 0.25);
124     passX.filter (cloud);
125
126     pcl::PassThrough<pcl::PointXYZ> passY;
127     passY.setInputCloud (cloud.makeShared());
128     passY.setFilterFieldName ("y");
129     passY.setFilterLimits (-0.25, 0.25);
130     passY.filter (cloud);
131     // viewerN1 = simpleVis(cloud.makeShared());
132
133     // n4 = simpleVis(cloud.makeShared());
134
135     for(b1 = cloudRGB.points.begin(), b2 = cloud.points.begin(); b1 <
```

```
        cloudRGB.points.end(); b1++, b2++) {
136         b1->y = -b1->y;
137         b1->z = -b1->z;
138 
139         b2->y = -b2->y;
140         b2->z = -b2->z;
141     }
142 
143 
144 
145 
146     pcl::SampleConsensusModelPlane<pcl::PointXYZ>::Ptr model_p (new
        pcl::SampleConsensusModelPlane<pcl::PointXYZ>(cloud.makeShared()));
147 
148     pcl::RandomSampleConsensus<pcl::PointXYZ> ransac (model_p);
149     ransac.setDistanceThreshold (.012);
150     ransac.computeModel();
151     ransac.getInliers(inliers);
152 
153     pcl::copyPointCloud (cloudRGB, inliers, majorPlaneRGB);
154     pcl::copyPointCloud (cloud, inliers, majorPlane);
155 
156     // n5 = simpleVis(majorPlane.makeShared());
157 
158     pcl::PointIndices::Ptr inliers1 (new pcl::PointIndices ());
159     inliers1->indices = inliers;
160 
161     pcl::ExtractIndices<pcl::PointXYZ> extract;
162     extract.setInputCloud(cloud.makeShared());
163     extract.setIndices(inliers1);
164     extract.setNegative(true);
165     extract.filter(object);
166 
167     // n6 = simpleVis(object.makeShared());
168 
169     pcl::SampleConsensusModelPlane<pcl::PointXYZ>::Ptr model_o (new
        pcl::SampleConsensusModelPlane<pcl::PointXYZ>(object.makeShared()));
170 
171     pcl::RandomSampleConsensus<pcl::PointXYZ> ransac1 (model_o);
172     ransac1.setDistanceThreshold (0.01);
173     ransac1.computeModel();
174     ransac1.getInliers(inliers);
175     pcl::copyPointCloud (object, inliers, objectPlane);
176     Eigen::VectorXf coeff;
177     ransac1.getModelCoefficients(coeff);
178     cout << coeff << endl;
179 
180     // n7 = simpleVis(objectPlane.makeShared());
181 
182 
183     pcl::CentroidPoint<pcl::PointXYZ> centroid;
184 
185     for(b2 = objectPlane.points.begin(); b2 < objectPlane.points.end(); b2++) {
186         pcl::PointXYZ p(b2->x, b2->y, b2->z);
187         centroid.add(p);
188     }
189     pcl::PointXYZ c;
190     centroid.get(c);
191 
192     cout << c << endl;
193 
194 
195     pcl::NormalEstimation<pcl::PointXYZ, pcl::Normal> ne;
196     ne.setInputCloud (objectPlane.makeShared());
197     pcl::search::KdTree<pcl::PointXYZ>::Ptr tree (new
        pcl::search::KdTree<pcl::PointXYZ> ());
198     ne.setSearchMethod (tree);
199     pcl::PointCloud<pcl::Normal>::Ptr cloud_normals (new
        pcl::PointCloud<pcl::Normal>);
200     ne.setRadiusSearch (0.03);
201     ne.compute (*cloud_normals);
202     pcl::PointCloud<PointXYZRGB> objectPlaneRGB;
203     pcl::copyPointCloud (objectPlane, objectPlaneRGB);
```

```
204
205         n8 = normalsVis(objectPlane.makeShared(), cloud_normals);
206
207         // viewerN.addPointCloudNormals<pcl::PointXYZ,pcl::Normal> (objectPlane,
            cloud_normals);
208
209         pcl::PointXYZ p(coeff[0], coeff[1], coeff[2]);
210         pcl::ModelCoefficients line_coeff;
211         line_coeff.values.resize (6);    // We need 6 values
212         line_coeff.values[0] = c.x;
213         line_coeff.values[1] = c.y;
214         line_coeff.values[2] = c.z;
215         cout << coeff.x() << ", " << coeff.y() << ", " << coeff.z() << endl;
216         line_coeff.values[3] = coeff.x();
217         line_coeff.values[4] = coeff.y();
218         line_coeff.values[5] = coeff.z();
219         // viewerN->addSphere(c, 0.005, "sphere");
220         n8->addSphere(c, 0.005, "sphere1");
221         n8->addLine(line_coeff, "line");
222         n8->addLine(c, pcl::PointXYZ(c.x, c.y+1, c.z), "line1");
223
224         geometry_msgs::PoseStamped pose;
225         pose.pose.position.x = c.x;
226         pose.pose.position.y = c.y;
227         pose.pose.position.z = c.z;
228
229         pose.pose.orientation.x = coeff.x();
230         pose.pose.orientation.y = coeff.y();
231         pose.pose.orientation.z = coeff.z();
232         pose.pose.orientation.w = coeff.w();
233
234         pub.publish(pose);
235
236         while (!(n8->wasStopped ()))
237         {
238             // viewerN1->spinOnce (100);
239             // viewerN->spinOnce (100);
240             // n2->spinOnce(100);
241             // n3->spinOnce(100);
242             // n4->spinOnce(100);
243             // n5->spinOnce(100);
244             // n6->spinOnce(100);
245             // n7->spinOnce(100);
246             n8->spinOnce(100);
247             std::this_thread::sleep_for(100ms);
248         }
249
250     }
251     int main(int argc, char** argv) {
252
253         init(argc, argv, "pcl_node");
254         NodeHandle nh;
255         Subscriber sub = nh.subscribe("/panda_camera/depth/points", 1, cloudCB);
256         pub = nh.advertise<geometry_msgs::PoseStamped>("/pcl_node/pose",1);
257         sleep(1);
258         spinOnce();
259         return 0;
260     }
```

```python
#!/usr/bin/env python3

from __future__ import print_function
from os import wait

import sys
import copy
import rospy
import moveit_commander
import moveit_msgs.msg
import geometry_msgs.msg
import tf
from math import pi, tau, dist, fabs, cos, sin
from std_msgs.msg import Float64
from tf2_msgs.msg import TFMessage
from sensor_msgs.msg import JointState
from geometry_msgs.msg import PoseStamped
from std_msgs.msg import Header
import math
import numpy


transformer = tf.TransformerROS(True, rospy.Duration(10))

jointNameToIndices = {'map': 0,
                      'world': 1,
                      'panda_link0': 2,
                      'panda_link1': 3,
                      'panda_link2': 4,
                      'panda_link3': 5,
                      'panda_link4': 6,
                      'panda_link5': 7,
                      'panda_link6': 8,
                      'panda_link7': 9,
                      'panda_hand' : 10,
                      'panda_rightfinger' : 11,
                      'panda_leftfinger' : 12}
jointPositions = []
transforms = {}

def getTransformation(a, d, theta, alpha):
    A = [[0 for i in range(4)] for j in range(4)]
    A[0][0] = cos(theta)
    A[0][1] = -sin(theta)*cos(alpha)
    A[0][2] = sin(theta)*sin(alpha)
    A[0][3] = a*cos(theta)
    A[1][0] = sin(theta)
    A[1][1] = cos(theta)*cos(alpha)
    A[1][2] = -cos(theta)*sin(alpha)
    A[1][3] = a*sin(theta)
    A[2][0] = 0
    A[2][1] = sin(alpha)
    A[2][2] = cos(alpha)
    A[2][3] = d
    A[3][0] = 0
    A[3][1] = 0
    A[3][2] = 0
    A[3][3] = 1


'''
header:
  seq: 0
  stamp:
    secs: 1911
    nsecs: 832000000
  frame_id: "world"
child_frame_id: "panda_link0"
transform:
  translation:
    x: 0.0
    y: 0.0
```

```
      z: 0.0
    rotation:
      x: 0.0
      y: 0.0
      z: 0.0
      w: 1.0
'''
statics = []
def transformsStaticListener(data):
    global statics
    msg = data.transforms
    for i in msg:
        transformer.setTransform(i)
    statics = msg
    # print(msg)
    # while(True):
    #     for i in msg:
    #         i.header.stamp = rospy.Time.now()
    #         transformer.setTransform(i)
    #     # print(msg)
    #     rospy.sleep(0.08)

def transformsListener(data):

    msg = data.transforms
    for i in msg:
        transformer.setTransform(i)
    for j in statics:
        # print(j)
        j.header.stamp = rospy.Time.now()
        # print("=================")
        # print(j)
        transformer.setTransform(j)
        # key = (jointNameToIndices[i.header.frame_id],
        jointNameToIndices[i.child_frame_id])
        # transforms[key] = i.transform
    # print("======================================")
    # print(transforms)

    # print(transformer.allFramesAsDot())
    # print(transformer.canTransform("panda_camera_optical_link","world",
    rospy.Time.now()))
    # transformer.waitForTransform("panda_link7","world", rospy.Time(0),
    rospy.Duration(10))
    # while(not transformer.canTransform("panda_link7","world", rospy.Time(0))):
    #     pass
    # if(transformer.canTransform("panda_camera_optical_link","world",
    rospy.Time.now())):
    #     print("====================================================")
        # header = Header()
        # header.frame_id = "panda_link7"
        # header.stamp = rospy.Time.now()
        # mat = transformer.asMatrix("world",header)
        # print(mat)


        # pose_target.pose.position.x = 0.0259903
        # pose_target.pose.position.y = -0.0419027
        # pose_target.pose.position.z = 0.450229
        # quaternion = tf.transformations.quaternion_from_euler(1.825868, -0.772411,
        2.923791)
        # pose_target.pose.orientation.x = quaternion[0]
        # pose_target.pose.orientation.y = quaternion[1]
        # pose_target.pose.orientation.z = quaternion[2]
        # pose_target.pose.orientation.w = quaternion[3]
        # pose_target.header.frame_id = "panda_camera_optical_link"

        # p = transformer.transformPose("world",pose_target)
        # print(p)
        # move_group.set_pose_target(pose_target)
        # move_group.go(wait=True)
        # rospy.sleep(10)
```

```
140         # while(1):
141         #     pass)
142 
143 # def normalize(v):
144 #     norm = numpy.linalg.norm(v)
145 #     if norm == 0:
146 #        return v
147 #     return v / norm
148 
149 # def poseListener(data):
150 #     pose_rcvd = data.pose
151 #     pose_target = geometry_msgs.msg.PoseStamped()
152 #     pose_target.header.stamp = rospy.Time.now()
153 
154 #     while(not transformer.canTransform("panda_camera_optical_link","world",
    rospy.Time.now())):
155 #         continue
156 #     pose_target.pose.position.x = pose_rcvd.position.x
157 #     pose_target.pose.position.y = pose_rcvd.position.y
158 #     pose_target.pose.position.z = pose_rcvd.position.z
159 
160 #     a = pose_rcvd.orientation.x
161 #     b = pose_rcvd.orientation.y
162 #     c = pose_rcvd.orientation.z
163 
164 #     axisY = numpy.array([a, b, c])
165 #     axisX = numpy.array([0, 0, 1])
166 #     axisZ = numpy.cross(axisX,axisY)
167 
168 #     axisX = normalize(axisX)
169 #     axisY = normalize(axisY)
170 #     axisZ = normalize(axisZ)
171 
172 #     rot = numpy.array([axisX, axisY, axisZ])
173 #     print(rot)
174 #     quat = tf.transformations.quaternion_from_matrix(rot)
175 #     pose_target.pose.orientation = quat
176 #     print(pose_target)
177 #     # quaternion = tf.transformations.quaternion_from_euler(1.825868, -0.772411,
    2.923791)
178 #     # pose_target.pose.orientation.x = quaternion[0]
179 #     # pose_target.pose.orientation.y = quaternion[1]
180 #     # pose_target.pose.orientation.z = quaternion[2]
181 #     # pose_target.pose.orientation.w = quaternion[3]
182 #     pose_target.header.frame_id = "panda_camera_optical_link"
183 
184 #     p = transformer.transformPose("world",pose_target)
185 #     print(p)
186 
187 # def jointsListener(data):
188 #     jointPositions = data.position
189 #     # print(jointPositions)
190 #     rospy.sleep(0.01)
191 
192 rospy.init_node("move_franka", anonymous=True)
193 moveit_commander.roscpp_initialize(sys.argv)
194 
195 robot = moveit_commander.RobotCommander()
196 scene = moveit_commander.PlanningSceneInterface()
197 group_name = "panda_arm"
198 move_group = moveit_commander.MoveGroupCommander(group_name)
199 rospy.Subscriber("/tf", TFMessage, transformsListener)
200 rospy.Subscriber("/tf_static", TFMessage, transformsStaticListener)
201 # rospy.Subscriber("/pcl_node/pose", PoseStamped, poseListener)
202 # rospy.Subscriber("/panda/joint_states", JointState, jointsListener)
203 rospy.spin()
204 
205 
206 if __name__ == '__main__':
207 
208     pose_target = geometry_msgs.msg.Pose()
209     quaternion = tf.transformations.quaternion_from_euler(1.977813, -0.791686,
```

```
        2.910482)
210     # quaternion = tf.transformations.quaternion_from_euler(-3.1412, 0.0735, 0.002658)
211     pose_target.position.x = 0.011974
212     pose_target.position.y = 0.134315
213     pose_target.position.z = 0.766702
214
215     # pose_target.position.x = 0.0355959
216     # pose_target.position.y = -0.0628825
217     # pose_target.position.z = -0.423587
218
219     # pose_target.position.x = 0.115024
220     # pose_target.position.y = 0.000172
221     # pose_target.position.z = 1.031304
222     pose_target.orientation.x = quaternion[0]
223     pose_target.orientation.y = quaternion[1]
224     pose_target.orientation.z = quaternion[2]
225     pose_target.orientation.w = quaternion[3]
226     move_group.set_pose_target(pose_target)
227     move_group.go(wait=True)
228     # move_group.stop()
229     # move_group.clear_pose_targets()
230
231     # leftfinger =
        rospy.Publisher('/panda/panda_finger1_controller/command',Float64,queue_size=1)
232     # rightfinger =
        rospy.Publisher('/panda/panda_finger2_controller/command',Float64,queue_size=1)
233     # obj = Float64()
234     # obj.data = 0.04
235     # leftfinger.publish(obj)
236     # rightfinger.publish(obj)
237     # rospy.spin()
238     # print("checkflag")
239
240     move_group.stop()
241     move_group.clear_pose_targets()
242
243     rospy.sleep(5)
244     moveit_commander.roscpp_shutdown()
245
246
247
```