# Lab 8: Robot Arm 2

The objective of the tutorial is to introduce MoveIt!, a framework that incorporates motion planning, 3D perception, collision avoidance, and so on. MoveIt! fully supports with ROS, so you can find many MoveIt! packages of industrial robot arms such as ABB IRB series, Universal Robots series, Sawyer, etc.

In this tutorial, we will first use MoveIt! to simulate UR5 in RViz and then have a live demo for the real one.

## Hardware and Software Setup

**laptop $ sudo apt-get update**

**laptop $ sudo apt-get install -only-upgrade=ros-kinetic\***

**laptop $ sudo apt-get install ros-kinetic-moveit \**

**ros-kinetic-moveit-visual-tools**

**laptop $ cd ~/sis\_lab\_all\_2020 && git checkout devel-[your\_student\_ID]**

**laptop $ git pull origin master**

**laptop $ cd ~/sis\_lab\_all\_2020/08-Robot\_Arm\_2/catkin\_ws && catkin\_make**

**laptop $ source devel/setup.bash**

**If catkin\_make failed, please follow these step:**

**Ubuntu 18.04**

**Step 1:**

**laptop $ vim src/ur\_kinematics/include/ur\_kinematics/ur\_moveit\_plugin.h**

**In terminal**

**Comment line 97, 98.**

**Step 2:**

**laptop $ vim src/moveit\_tutorial/src/topic1.cpp**

**In terminal  
 Find two Eigen::Affine3d, and change them into Eigen::Isometry3d.**

**Step 3:**

**laptop $ catkin\_make**

**Note: this error is caused by the melodic version problems.**

**Ubuntu 16.04**

**laptop $ sudo apt-get update && sudo apt-get dist-upgrade**

**laptop $ catkin\_make**

[**Lab 8: Robot Arm 2**](#_gjdgxs) **1**

[Hardware and Software Setup](#_30j0zll) 1

[Overview](#_1fob9te) 1

[Topics and Activities](#_3znysh7) 2

[Topic/Activity 1 Simulate UR5 in RViz](#_2et92p0) 2

[Topic/Activity 2 Control of Real Yumi Robot](#_tyjcwt) 3

[Checkpoint](#_c75cm4g2n725) 5

[Reference](#_3dy6vkm) 5

## Overview

Estimated Time to Finish: 1 hours

After completing this tutorial you should

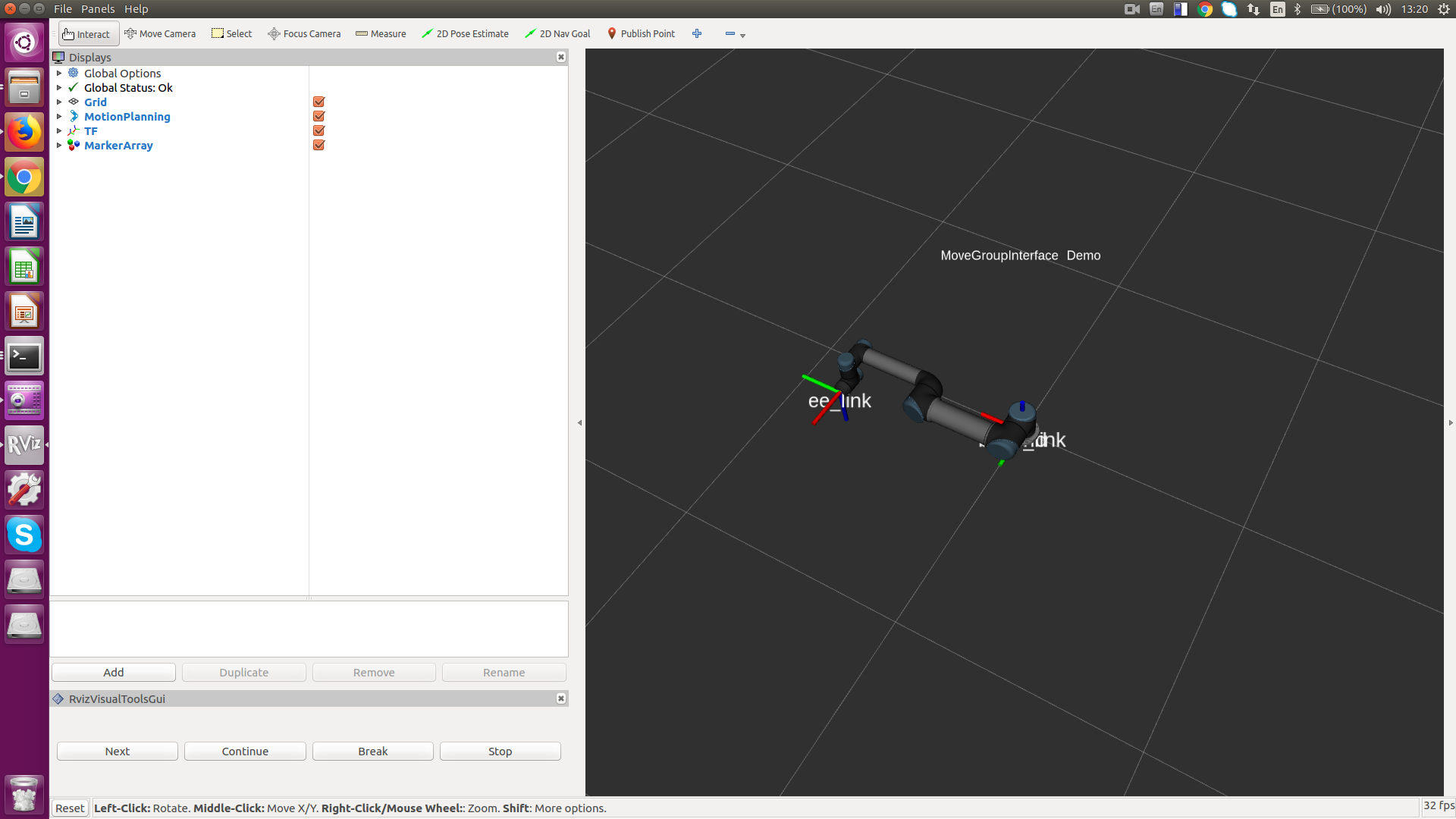
* be able to use MoveIt! to do motion planning, control for your robot arm
* understand how to control UR5

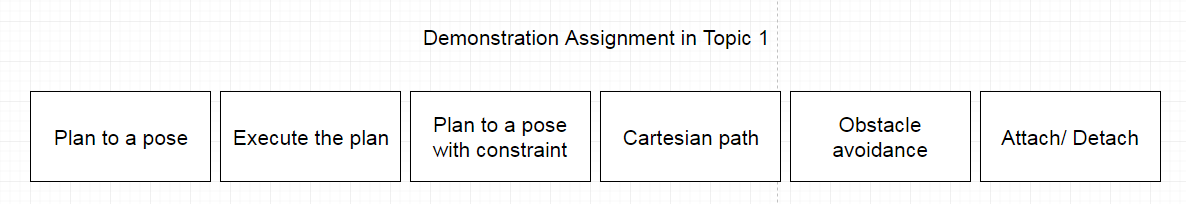
## Topics and Activities

### Topic/Activity 1 Simulate UR5 in RViz

**laptop $ roslaunch moveit\_tutorial moveit\_tutorial.launch**

You will see





Click the ‘Next’ button lower left, and you will see the robot model start to do some demo.

Stage 1. We plan the end effector of UR5 to a given pose, and Moveit! helps us calculate IK

and show the corresponding trajectory animation repeatedly.

Stage 2. The arm will start to execute the plan above.

Stage 3. We do the same job as the first one but this time we use joint space pose. That is, we assign the specified angle to each joint so UR5 can move to the pose.

Stage 4. We keep the orientation of end effector as base\_link frame and plan it to

another pose.

Stage 5. The end effector starts to traverse through a square with side length 0.2 meter.

Stage 6. There are two boxes in RVIZ. Click “Next” again and you will notice that the arm avoid from these boxes and goes to another pose in other side of the obstacle. Notice that we used the sampling based method, RRT connect, to avoid the obstacle so the result will not be the best or shortest path. If you don’t satisfy, you can press ‘r’ in terminal to replan your trajectory, or ‘c’ to jump to the next stage.

Stage 7. Click ‘Next’ again and the boxes disappeared, a cube in purple replaced the two. Click and the cube will start to move along end effector. Click again the cube will disappear and click again the program will shut down by itself.

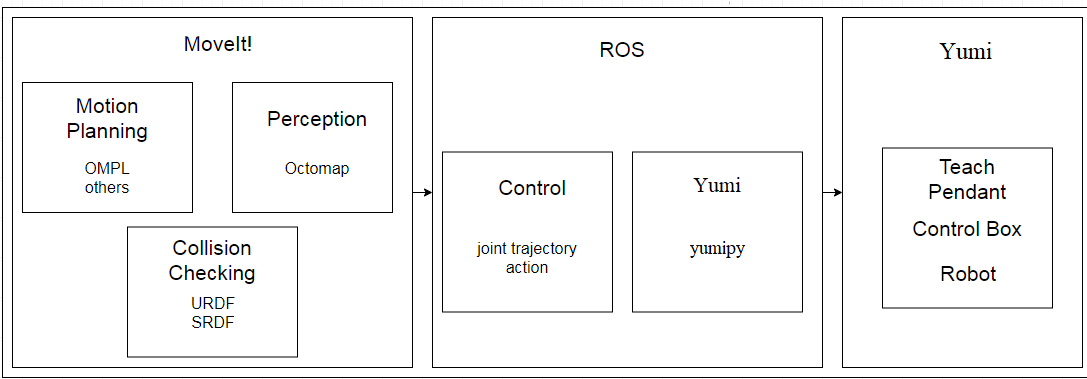
Though we only demonstrate the power of MoveIt! in RViz, it can truly transplant the system to the real one.

The node available at **~/sis\_lab\_all\_2020/08-Robot\_Arm\_2/catkin\_ws/src/moveit\_tutorial/src/topic1.cpp** is revised from the tutorial of [MoveIt!](https://ros-planning.github.io/moveit_tutorials/doc/move_group_interface/move_group_interface_tutorial.html), there is also a [Python API provided](https://ros-planning.github.io/moveit_tutorials/doc/move_group_python_interface/move_group_python_interface_tutorial.html), for simple application, MoveIt! also provides a [command line tool](https://ros-planning.github.io/moveit_tutorials/doc/moveit_commander_scripting/moveit_commander_scripting_tutorial.html).

**Discussion:**

1. **In MoveIt! MoveGroup class, what functions should be called to set a target pose and a joint space goal respectively?**
2. **What function is used to plan the motion? And what argument should be passed to the function? After we have the planned path, how to move the robot with this plan?**
3. **What is the meaning of the trajectories in RViz? (whose trajectory?)**

### Topic/Activity 2 Control of Real Yumi Robot



Currently there’s no official ROS package for yumi control through ROS, but we can achieve it by using these repo

[You don't have to do these steps below, just a note]

**workstation $ cd [path\_to\_your\_catkin\_ws]/src**

**workstation $ git clone** [**https://github.com/BerkeleyAutomation/autolab\_core.git**](https://github.com/BerkeleyAutomation/autolab_core.git)

**workstation $ cd autolab\_core && python setup.py install**

install the autolab\_core following installation guide

**workstation $ git clone** [**https://github.com/BerkeleyAutomation/yumipy.git**](https://github.com/BerkeleyAutomation/yumipy.git)

**workstation $ git clone** [**https://github.com/BerkeleyAutomation/optimal\_manipulation\_simulator.git**](https://github.com/BerkeleyAutomation/optimal_manipulation_simulator.git)

**workstation $ cd your [path\_to\_your\_catkin\_ws] && catkin\_make**

**workstation $ source devel/setup.bash**

Make sure your robot and workstation are in the same network segment and robot with IP 192.168.125.1 .

till here, we could connect and control Yumi through terminal like this

**workstation $ python**

**$ from yumipy import YuMiRobot**

**$ robot = YuMiRobot(arm\_type=’remote’)**

**$ pose = robot.left.get\_pose()**

**$ pose.translation[0] += 0.05**

**$ robot.left.goto\_pose(pose)**

**$ robot.left.reset\_home()**

above move the robot’s left arm 5cm forward and then reset left arm to home pose.

by looking into yumi\_arm.py in yumipy package, we can find out that currently there are up to 26 functions for controlling yumi or retrieving data from the robot like **goto\_pose**, **goto\_state**, **get\_pose**, **get\_state**,...etc

Since currently there are some objects in the robot’s operating range that we don’t want the robot to collide with, **MoveIt** is used here to avoid those objects when doing path planning.

To do so, first clone the package which provides yumi’s urdf file

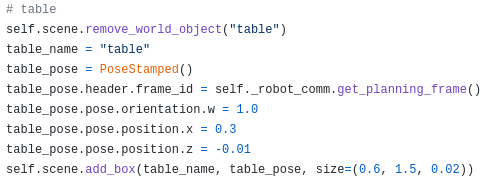
[You don't have to do these steps below, just a note]

**workstation $ cd [path\_to\_your\_catkin\_ws]/src**

**workstation $ git clone** [**https://github.com/BerkeleyAutomation/optimal\_manipulation\_simulator.git**](https://github.com/BerkeleyAutomation/optimal_manipulation_simulator.git)

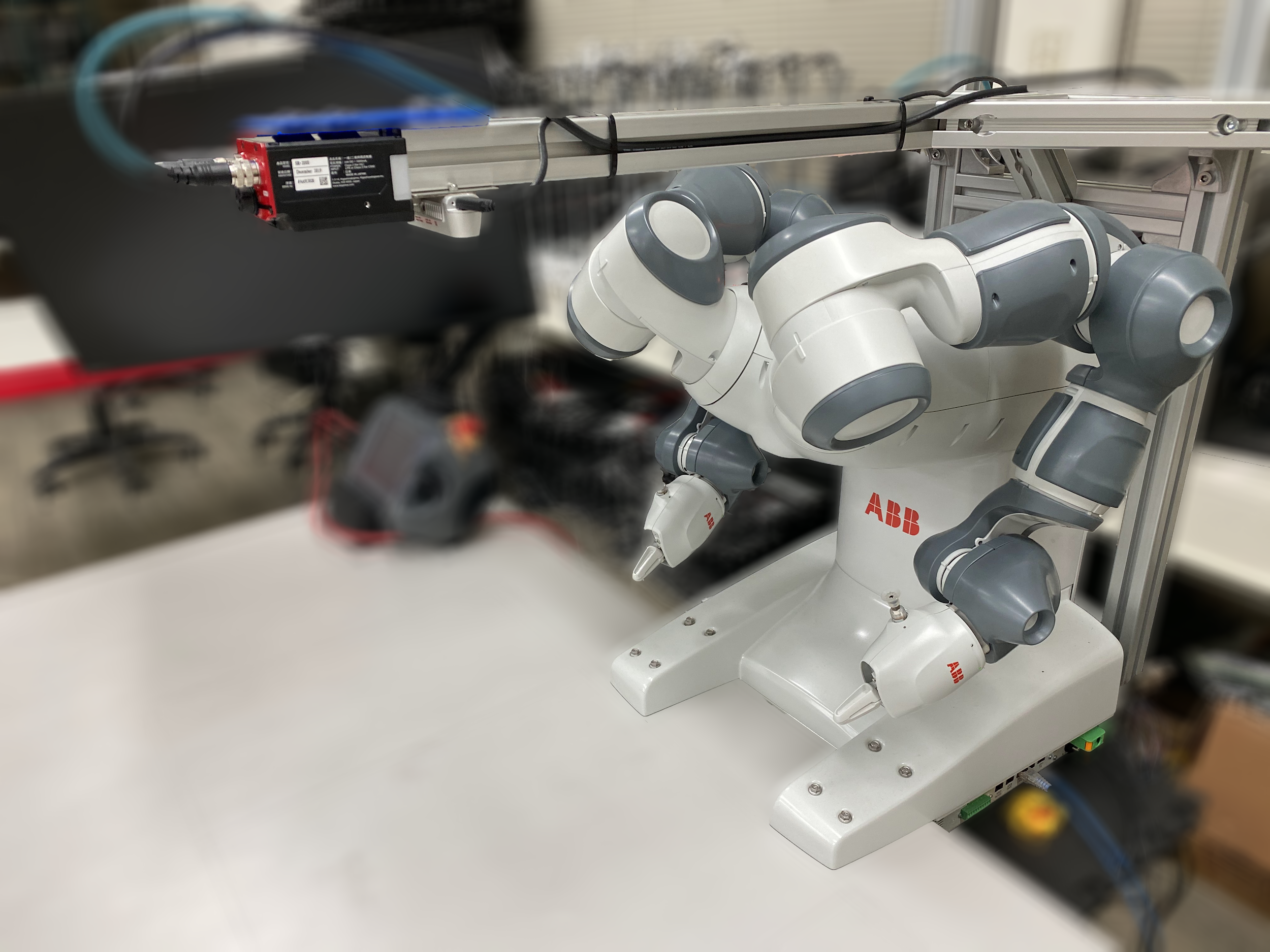
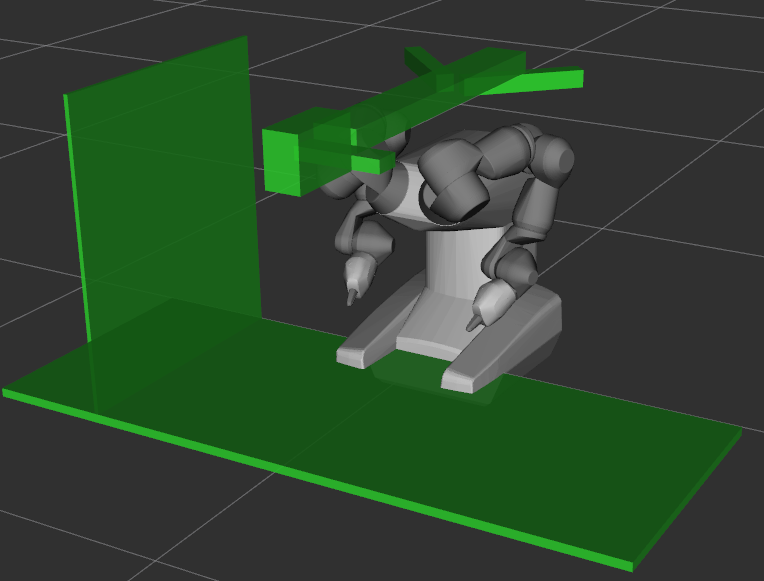
**workstation $ cd your [path\_to\_your\_catkin\_ws] && catkin\_make**

and then add the information objects to avoid in **yumi\_planner.py**. for example



above picture shows how to add a static obstacle named ‘table’ with 2 cm height, 1.5 meter width and 60 cm length. The distance between the center of the ‘table’ object and the planning frame, which is the robot frame currently, is 30 cm along the x-axis and -1 cm along the z-axis.

After setting all the static obstacles’ information, the virtual scene will be something like



### Checkpoint

Please finish the content of callback function in **sis\_lab\_all\_2020/08-Robot\_Arm\_2/catkin\_ws/src/tutorial/src/click\_point.py**

the callback function receives a msg with [**PointStamped**](http://docs.ros.org/melodic/api/geometry_msgs/html/msg/PointStamped.html) format, the content is a 3D position in meter unit, you need to transform the values into mm unit and fill the values into a self defined request called GotoPoseRequest(), also the ‘arm’ and ‘quat’ parameters. After filling those values, call the service to drive either the left arm or the right arm to the position.

**[This checkpoint will be tested on a real robot, please create a new branch and push your code onto your own branch, and then ask TA-Allen to help you test it]**

## Reference

[MoveIt! kinetic tutorial](https://ros-planning.github.io/moveit_tutorials/index.html)

[planning\_interface API](http://docs.ros.org/jade/api/moveit_ros_planning_interface/html/classmoveit_1_1planning__interface_1_1MoveGroup.html#details)