# can i install cuda if i don't have nvidia driverLab 7: Robot Arm I

The objective of this tutorial is to introduce the concept of kinematic with the help of DH model. You will be given a simple 2-joint robot arm combined with three Dynamixel servo motors, one for gripper control, which reduce our burden of computation. And then we will learn how to visualize our robot in RViz with URDF.

Also we introduce PyRobot, an open-source robotics framework for research and benchmarking.

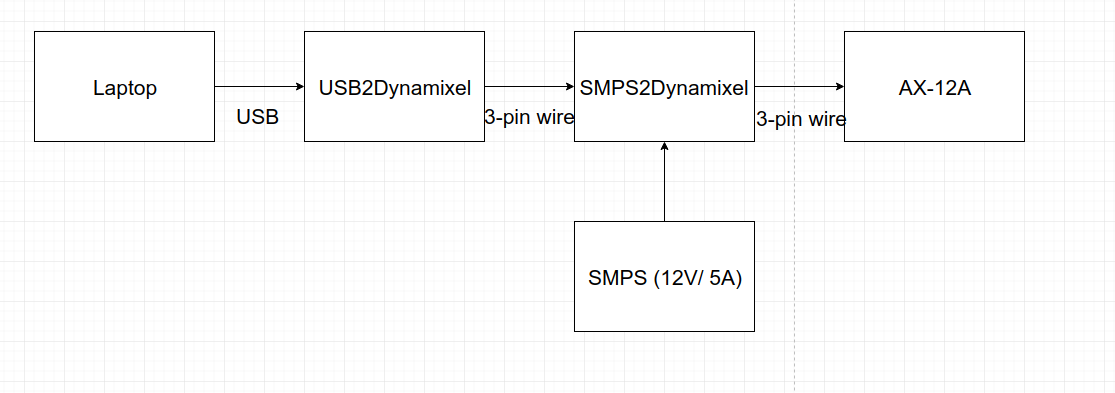
## Hardware and Software Setup

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

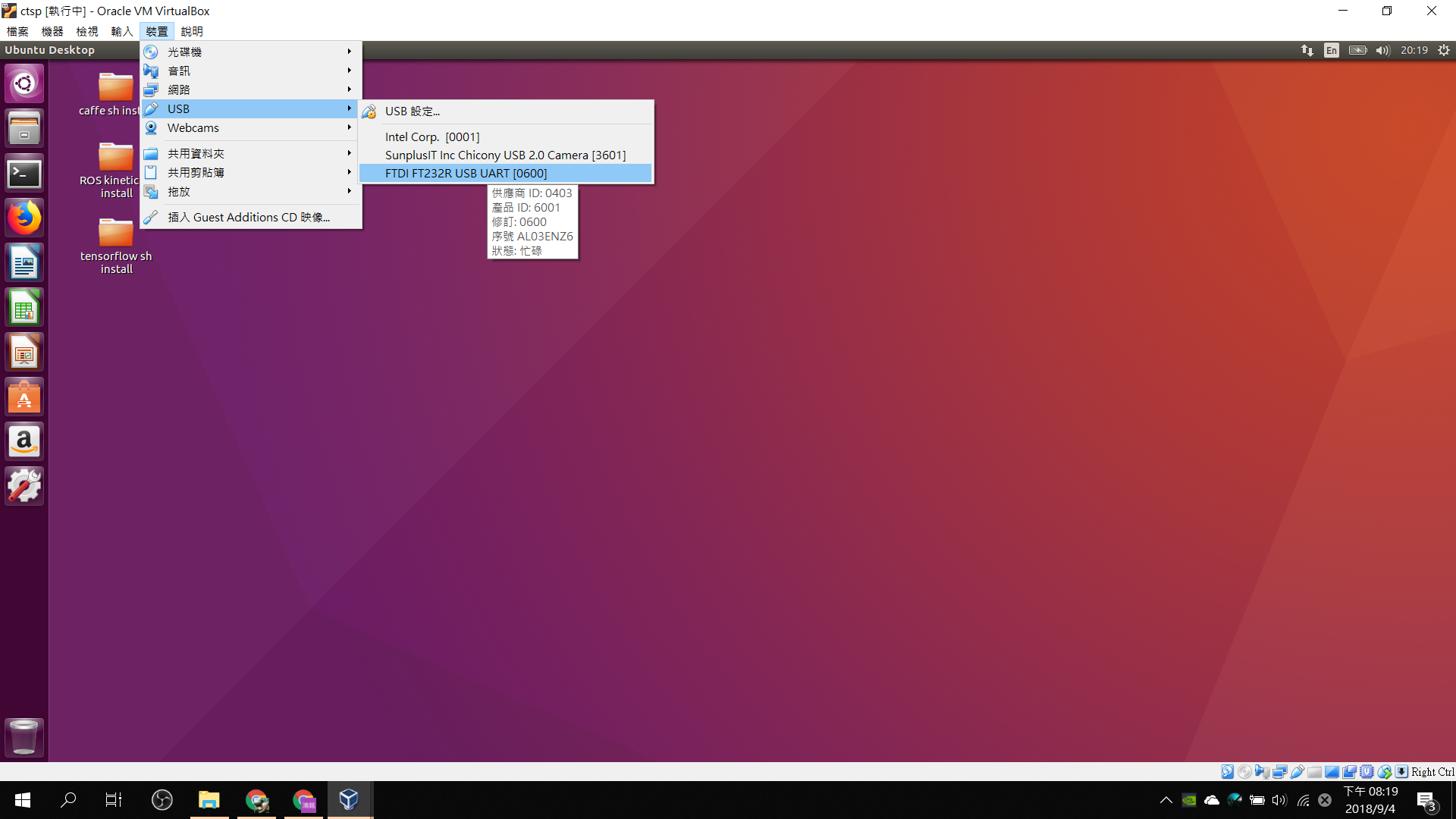
laptop $ git pull origin master

laptop $ cd ~/sis\_lab\_all\_2020/07-Robot\_Arm\_1/catkin\_ws && catkin\_make

Wiring Diagram



After you plug the USB2Dynamixel to your laptop, enable USB device in VirtualBox (Device->USB->FTDI FT232R)e script to control L



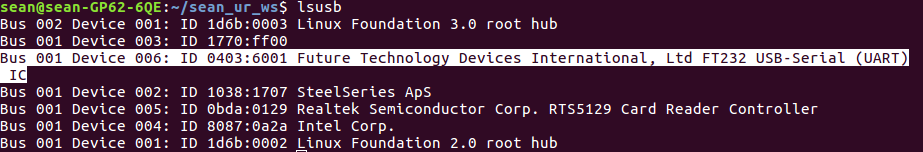
For those who cannot see the device, please go to [here](https://www.virtualbox.org/wiki/Downloads), download the **extension pack**, install it and enable USB3.0.



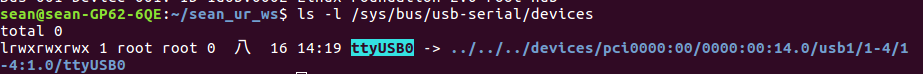
After installation, turn off your machine and in Setting/USB, check USB 3.0 controller

laptop $ lsusb

You will see a device named “Future Technology Devices International, Ltd FT232 USB-Serial (UART)”



laptop $ ls -l /sys/bus/usb-serial/devices



laptop $ sudo chmod 777 /dev/ttyUSB0 # Make the port readable and writable

[**Lab 7: Robot Arm I**](#_74gram21yqax) **1**

[Hardware and Software Setup](#_chc57ow4x0s1) 1

[Overview](#_x5puhc1p2jjo) 3

[Topics and Activities](#_g0fw0bgwxhdt) 3

[Topic/Activity 1 DH Model and Kinematic](#_aadm982kyolo) 3

[Topic/Activity 2 Visualize Your Robot With URDF](#_60rts91qlmd) 6

[Topic/Activity 3 PyRobot and LoCoBot](#_wbwkv3cgp2h6) 10

## Overview

Estimated Time to Finish: 1 hours

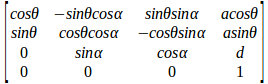
After completing this tutorial you should

* understand the concept of kinematic一起守護健康~
* be able to use servo motor to construct your simple arm and control it
* use an URDF to describe your robot model
* control [LoCoBot](http://locobot.org/) in Rviz by using PyRobot

## Topics and Activities

### Topic/Activity 1 DH Model and Kinematic

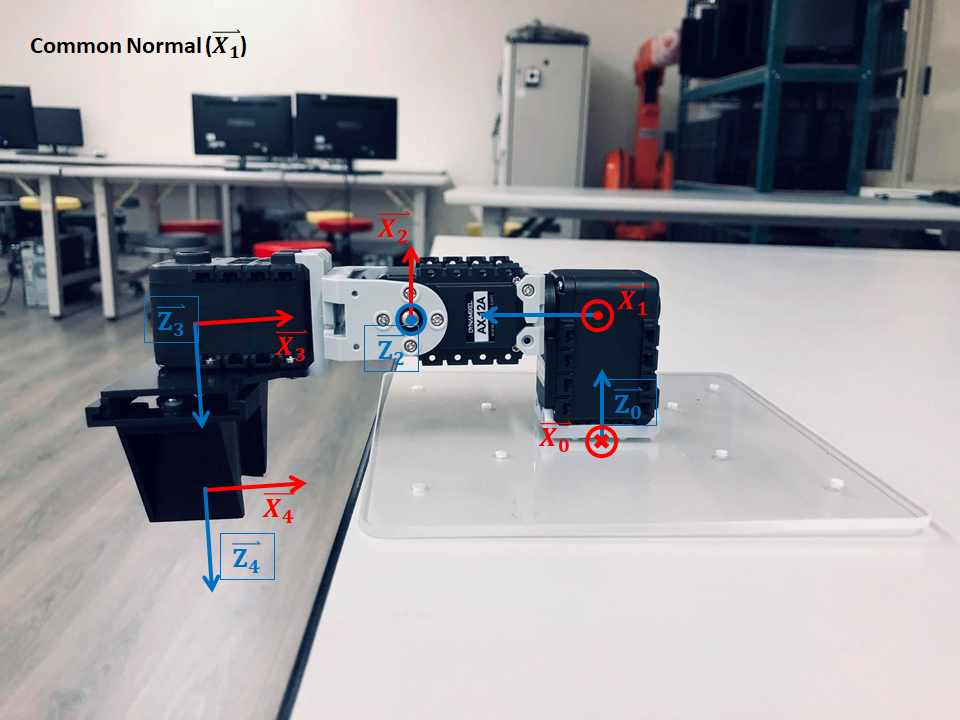
We know that in 3D space, we need at most 6 Dof(degree-of-freedom) to describe two different frames, while the fewer variables the better. So can we get less variables? Yes, we can! But we have to follow some rules, and finally we need only 4 variables to describe the difference! This is known as Denavit-Hartenberg Model, or DH Model. The rules to defines your coordinates are: 1) the z-axis is in the direction of the joint axis 2) the x-axis is parallel to the common normal of consecutive two z-axes, i.e., xn = zn-1zn. After constructed our frames, we need four steps to coincide the consecutive two: 1) traverse z-axis a distance d 2) rotate z-axis aboutto align two x-axes 3) traverse along new x-axis a distance a 4) rotate new x-axis about to align two z-axes. The [video](https://www.youtube.com/watch?v=rA9tm0gTln8&t=92s) is a good demonstration. After the procedure, you will get a transformation matrix, from previous frame to next frame, as



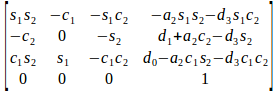
One DH parameters table of our simple arm is as below

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Frame | d | θ | a | α |
| 0 -> 1 | d0 = 0.045 | π | 0 | π/2 |
| 1 -> 2 | d1 = 0.06 | π/2+ | 0 | π/2 |
| 2 -> 3 | 0 | -π/2+ | -a2 = -0.065 | π/2 |
| 3 -> 4 | d3 = 0.07 | 0 | 0 | 0 |

Dimension approxiamated from measured



(Red: x-axis, Blue: z-axis)

After the calculation, the transformation matrix of end effector w.r.t base is

So, given the joints, we then know the pose of end effector, such procedure is called as forward kinematic. On the other hand, if we are given the pose of end effector or tool center point and we want to know the sets of joints such that satisfy the pose, the procedure is called as inverse kinematic. FK is easier than IK,while unlucky, in practice case, we meet IK problems more frequently.

Let’s take a look at the UR\_kinematic, the code is available at [here](https://github.com/ros-industrial/universal_robot/blob/kinetic-devel/ur_kinematics/src/ur_kin.cpp).



Input:

Output:

What a messy, isn’t it? But slide down the file, IK part takes more than one hundred lines…

So, let’s use our simple 2 DoF arm to understand the inverse kinematic concept.

Since there are only two joint variables, we cannot reach any position in workspace with arbitrary orientation, instead we can only assign position value, i.e, x, y and z. From the result above with simple trigonometric algebra, we can calculate the joint values from given position, the code is available at **~/sis\_lab\_all\_2020/07-Robot\_Arm\_1/catkin\_ws/src/sis\_arm/src/sis\_arm\_server.cpp**

laptop $ **roslaunch sis\_arm sis\_arm\_server.launch # Don’t turn off this launch until assignment**

In another terminal, type

laptop $ **cd ~/sis\_lab\_all\_2020/07-Robot\_Arm\_1/catkin\_ws/ && source devel/setup.bash**

laptop $ **rosservice call /sis\_arm\_control/goto\_pose\_srv "x: 0.0**

**y: 0.125**

**z: -0.025**

**finger\_percentage: 0"**

The request of the server is x, y, z and finger\_percentage, which indicates the percentage of gripper closure.

Try arbitrary position you want, you may see plan\_result as “Success”, “Joint out of range” or “Given point out of range”. If the values you give are in range, you will see the motors start to rotate sequentially.

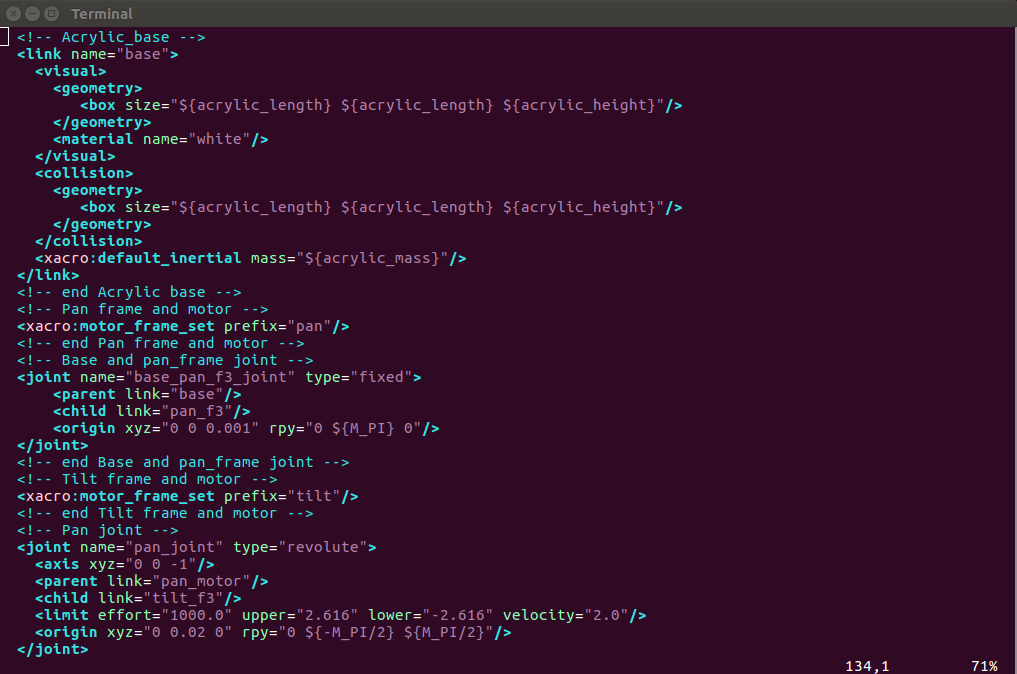
**Discussion:**

* **Derive the DH parameters table above to TAs.**
* **Where is the difference between FK and IK? Explain it respectively.**
* **In our configuration, is it possible to have multiple joint solutions with given position, without regarding it orientation? If yes, explain the condition it occurs.**

### Topic/Activity 2 Visualize Your Robot With URDF

So now we have a simple robot arm, can we visualize it in RViz? Sure we can, but we need the help of URDF, unified robot description format, to describe our robot. URDF is an XML-format file like launch for representing your robot model. The file for our robot is available at **~/sis\_lab\_all\_2020/07-Robot\_Arm\_1/catkin\_ws/src/robot\_description/urdf/sis\_arm.urdf.xacro**. Let’s take a look.

The snapshot of the file is as following.



A link is a part of your robot, which will also create an TF frame. Between two different links, we need a joint to describe the relation of them, for example, fixed, revolute and prismatic. The child link inherited from the parent link just like TF frames. Visual tag implements what we see in RViz as its name. Geometry describes the appearance of the part, we can use simple geometry shape such as box, cylinder, while we can also include an STL file or an DAE file. Different from visual, collision describes the physical interaction of the part with other object in simulation engines such as Gazebo and Unity. Inertia interprets the mass and rotation relation of the part. The origin of the part is described in origin tag. You can specify the origin of joint, visual and collision. Xacro clean up the file which make it looks simpler, for example, you can define a constant value that used several times, do some simple computation likes M\_PI/2 and also define a macro even with some parameters and blocks. Xacro is an useful tool for both code readable and reusable. The file **sis\_arm.urdf** is generated from **sis\_arm.urdf.xacro** with command

laptop $ **rosrun xacro xacro --inorder sis\_arm.urdf.xacro > sis\_arm.urdf** # Don’t run it, just a demonstration

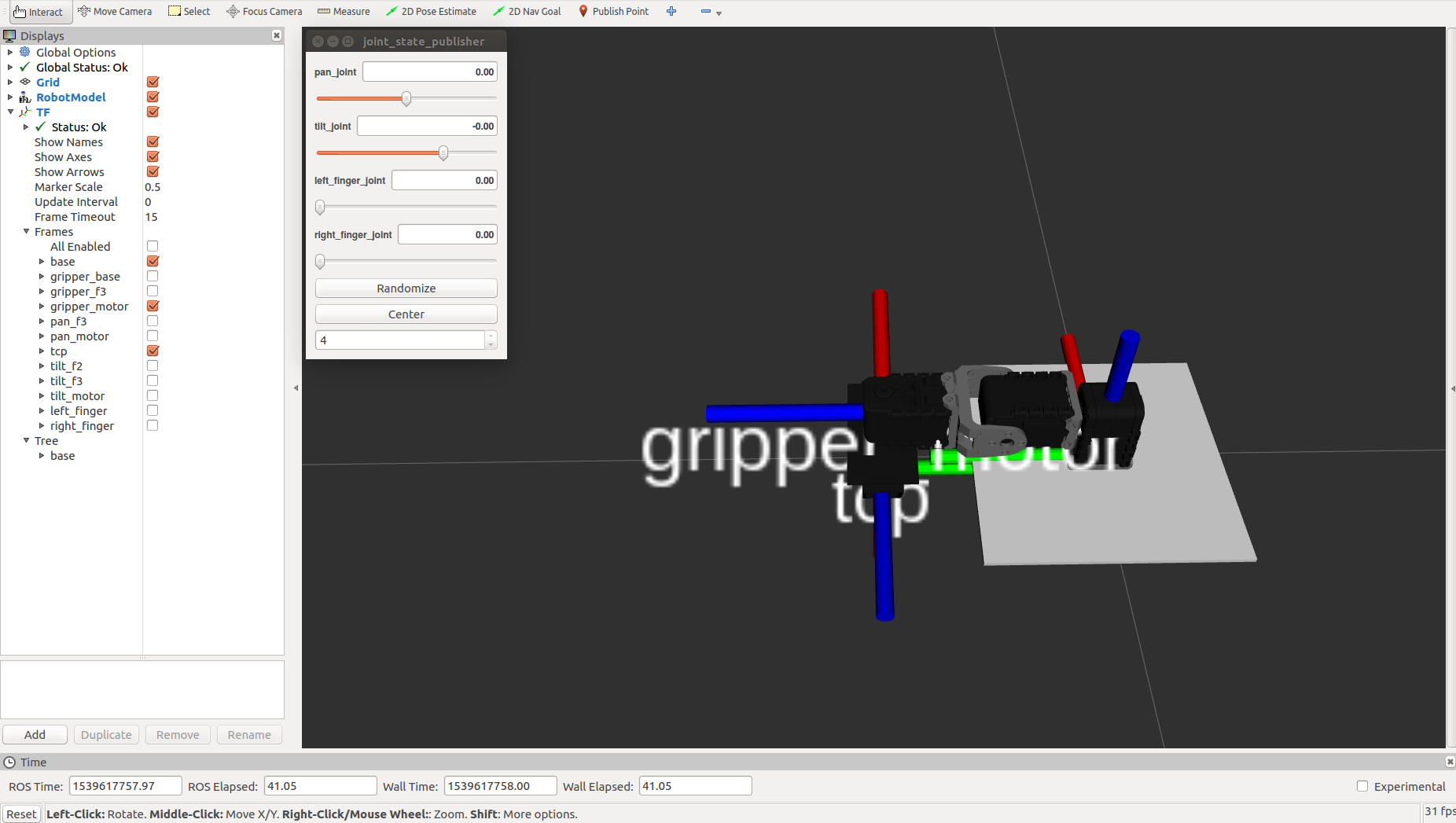
There are too many tags in URDF which beyond the scope of this tutorial. You can follow the official tutorial [here](http://wiki.ros.org/urdf/Tutorials) to draw an R2D2, it is also a good choice to check other people's URDF files and learn how to write it.

Let’s take a look what our model looks like. Open another terminal,

laptop $ **cd ~/sis\_lab\_all\_2020/07-Robot\_Arm\_1/catkin\_ws && source devel/setup.bash**

laptop $ **roslaunch robot\_description display.launch use\_sim:=true**

You will see the following scene



Besides RViz window, there is an additional window named joint\_state\_publisher, with four bars inside. Slide the bars and see what magic happen.

We can also visualize the workspace of our simple arm, open another terminal, and

laptop $ **cd ~/sis\_lab\_all\_2020/07-Robot\_Arm\_1/catkin\_ws && source devel/setup.bash**

laptop $ **rosrun sis\_arm vis\_workspace**

You may see



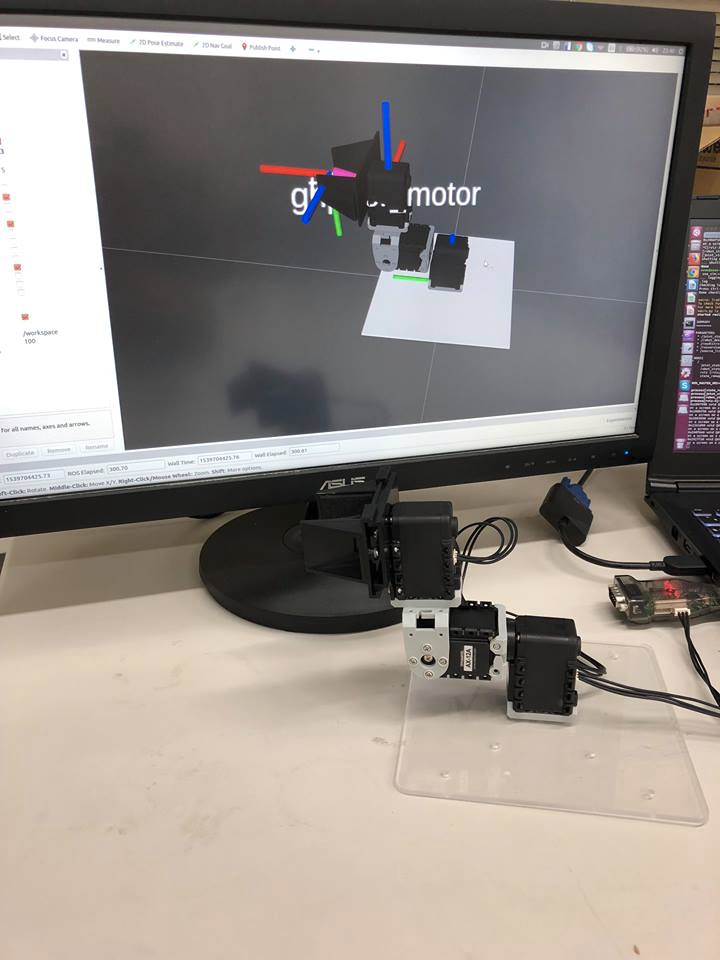
Every green point represents a reachable position, since the computation constraint we cannot draw a complete surface and notices also that there are not orientation presented.

Turn off the node by typing Ctrl + C.

Now we can simulate our simple arm in RViz, but what we want is to link from real-world robot to RViz, let’s change use\_sim parameter from true to false. Ctrl + C to stop display.launch and

laptop $ **roslaunch robot\_description display.launch use\_sim:=false**

Now, the joint\_state\_publisher window is disappeared and the pose of our model is the same as the real one.



**Note: It is suggested that you write your own URDF file at first time, then you will understand the meaning of each tag.**

### Topic/Activity 3 PyRobot and [LoCoBot](http://locobot.org/)

PyRobot is a Python package for benchmarking and running experiments in robot learning. The goal of this project is to abstract away the low-level controls for individual robots from the high-level motion generation and learning in an easy-to-use way. Using PyRobot will allow you to run robots without having to deal with the robot specific software along with enabling better comparisons.

LoCoBot, a low cost mobile manipulator robot.

We will run through the basic manipulation tools currently available on PyRobot.

laptop $ **cd ~/sis\_lab\_all\_2020/07-Robot\_Arm\_1**

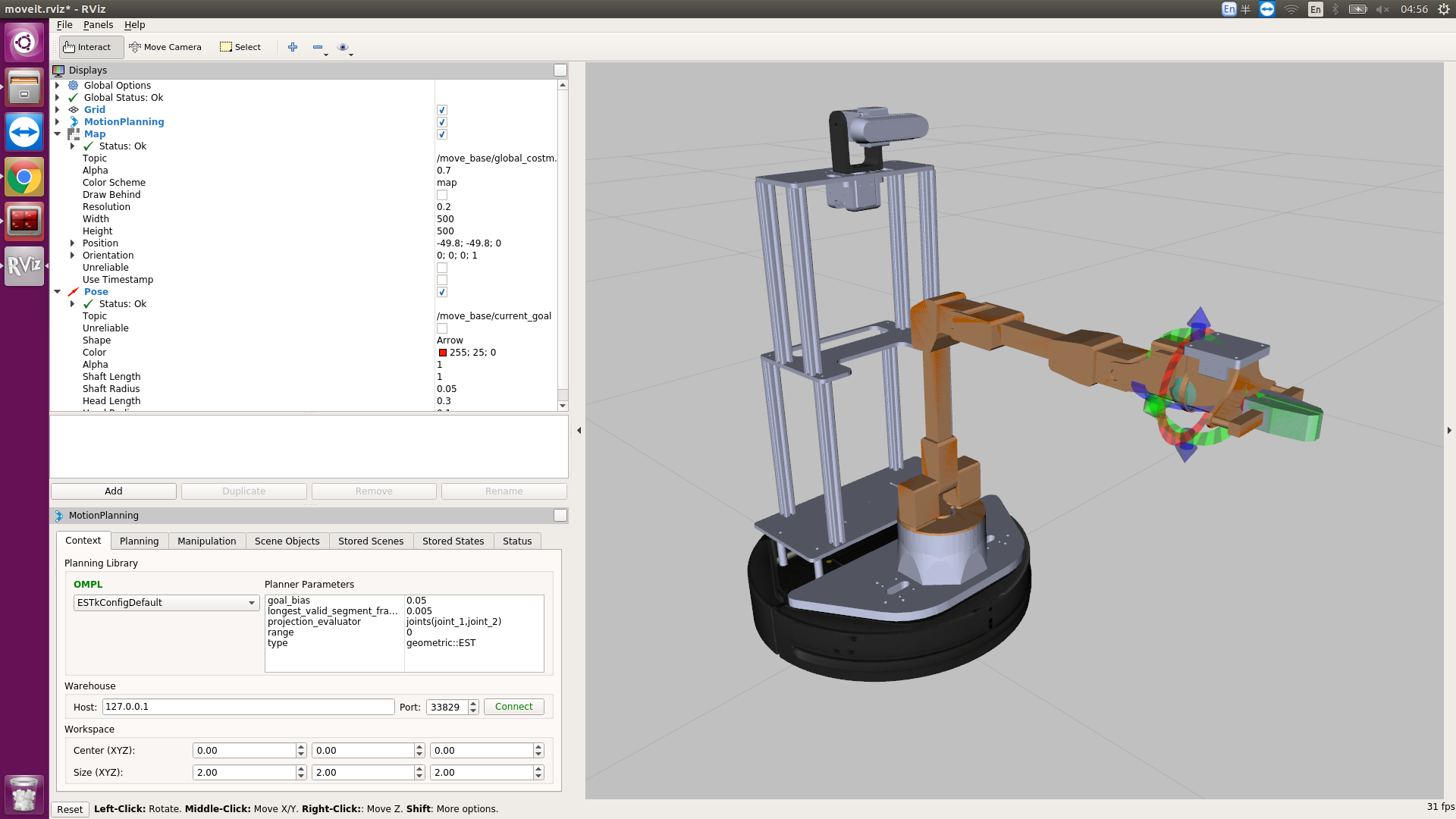
laptop $ **source docker\_run.sh cuda10**

docker $ **export PATH="/usr/lib/nvidia-384/bin":${PATH}**

docker $ **export LD\_LIBRARY\_PATH="/usr/lib/nvidia-384:/usr/lib32/nvidia-384":${LD\_LIBRARY\_PATH}**

docker $ **roslaunch locobot\_control main.launch use\_arm:=true use\_sim:=true**

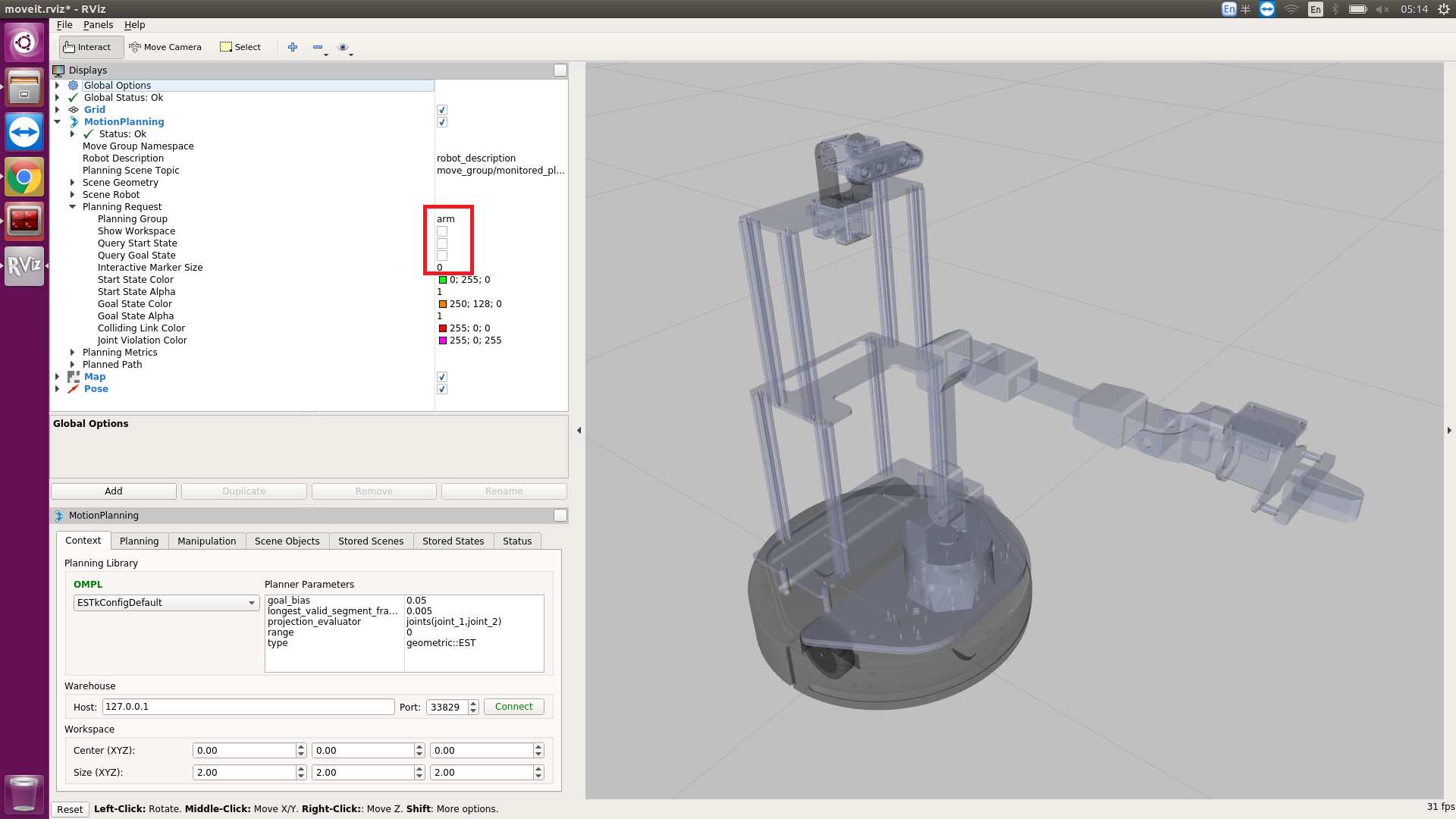
Now you can see LoCoBot in Rviz.



Please uncheck this options.



Like the following scene.



Next we will control LoCoBot’s arm by using PyRobot.

Open another terminal, type

laptop $ **source docker\_run.sh same**

docker $ **source /root/pyrobot\_catkin\_ws/devel/setup.bash**

docker $ **source /root/pyenv\_pyrobot\_python3/bin/activate**

docker $ **cd /home/developer/sis\_lab\_all\_2020/07-Robot\_Arm\_1/script**

docker $ **python joint\_control.py**

You can see LoCoBot’s arm move to another position.

Next we make LoCoBot’s arm go to the home pose.

docker $ **python go\_home.py**

**End-effector pose control**

docker $ **python end-effector\_pose\_control.py**

docker $ **python go\_home.py**

**End-effector Position and Pitch Roll Control**

docker $ **python end-effector\_position\_and\_Pitch\_Roll\_Control.py**

docker $ **python go\_home.py**

**Gripper control**

docker $ **gripper\_control.py**

**Discussion:**

* See **end-effector\_position\_and\_Pitch\_Roll\_Control.py.**

Why you can specify its pose with an end-effector position (x,y,z), a pitch angle and a roll angle and **no need to specify yaw angle ?**