지능 로봇 설계 공학 실험(RE510)

Experiment 4:

Manipulator Teleoperation

Prof. Jee-Hwan Ryu

T.A. Kwang-Hyun Lee, kwanghyun90@kaist.ac.kr

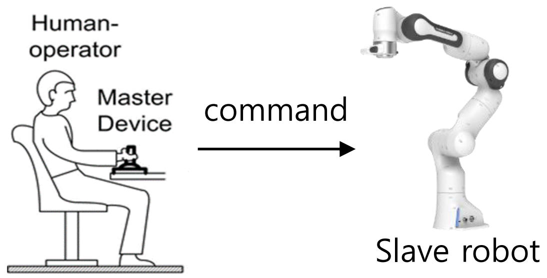
IRiS Lab, KAIST

Goal

- Learn the basic concept of the manipulator teleoperation.
- Implementing a simple teleoperation system for the 7-DoF manipulator.

Teleoperation

- Human-operator controls the remote robot, called a ***slave robot***.
- To generate motion commands, human operator handles a special device, called a ***master device***.



- Direct-Teleoperation
  - Human operator gives motion commands directly to the slave robot.

- Bilateral Teleoperation
  - Allows human operator to feel the interaction force between the slave robot and the environment while controlling the slave robot.
  - Special haptic master device is required to provide the feedback force.

- Shared Teleoperation
  - Control the slave robot together with other agents, such as other human operator or autonomy agents.

Direct Teleoperation of slave manipulator

In this class, we will learn the simple teleoperation schemes to control the slave manipulator.

Position to Position Control

- Updating the desired position of slave's end-effector with the increments of master device.
- Basically, $\dot{x}_m = k\dot{x}_s$

Forward Kinematics

$$q^{current} = \begin{bmatrix} \theta_0^{current} \\ \theta_1^{current} \\ \cdot \\ \cdot \\ \cdot \\ \theta_n^{current} \end{bmatrix}$$

$\Delta X_m(t)$

$$\Delta X_m(t) = X_m(t) - X_m(t-1)$$

or

$$\Delta X_m(t) = \dot{X}_m(t)\,\Delta T$$

<Master Controller>

$$X_s^{desired}(t) = X_s^{current}(t) + k(\Delta X_m(t))$$

or

$$= X_s^{desired}(t-1) + k(\Delta X_m(t))$$

<Slave Controller>

Inverse Kinematics

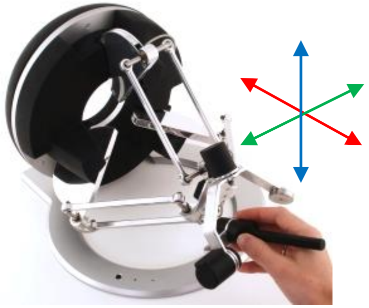
$$q^{desired} = \begin{bmatrix} \theta_0^{desired} \\ \theta_1^{desired} \\ \cdot \\ \cdot \\ \cdot \\ \theta_n^{desired} \end{bmatrix}$$

<Slave Joint Controller>

X is state in the Cartesian space, q is state in the joint space.

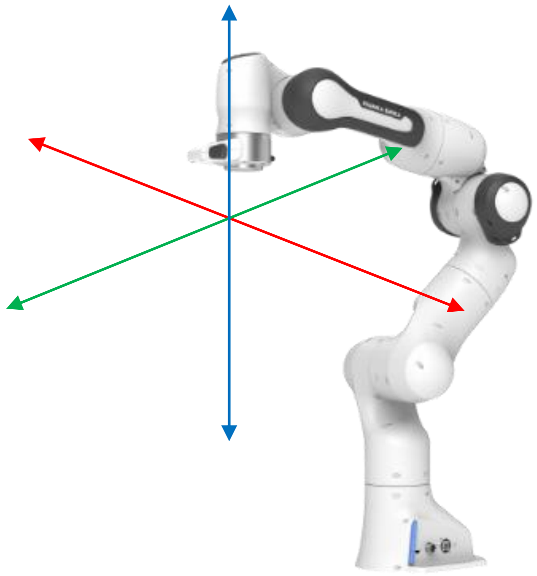
Covering Large workspace

- Usually master device has smaller workspace than slave robot.
- **Scaling:** One simple way to cover the slave robot's workspace with a small master device is amplifying motion commands with the constant motion gain. However, a large scaling gain can make the movement unstable and make it difficult for the operator to perform precise tasks.



$$\dot{x}_m = \underline{k}\dot{x}_s$$

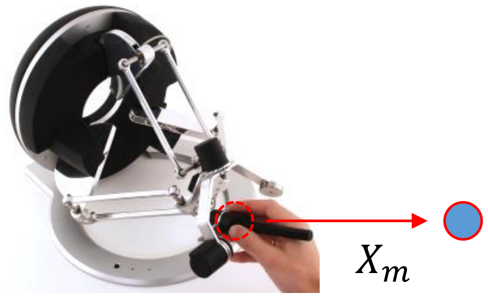
Using the high gain



Covering Large workspace

- **Indexing**: Enable and Disable the master command depends on situation.

Step 0



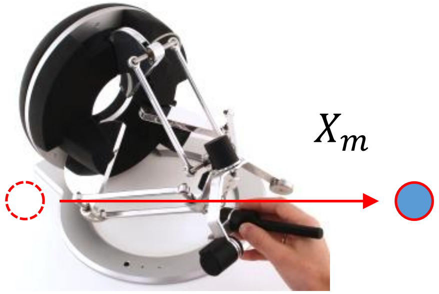
Step 1

Disable signals and move back.

It doesn't generate motion command

Step 2

Enable signals and move



Repeat

Rate Control (Position to Velocity control)

- Position displacement of the master device generates the velocity of slave robot's end-effector.
- End-effector moves in proportion to master displacement.
- Usually used for the mobile robot teleoperation. (imagine joystick)
- Needs some kind of "*return to origin*" for master device.

$X_m(t)$

<Master Controller>

$$X_s^{desired}(t) = X_s^{desired}(t-1) + k\,X_m(t)$$

$$\dot{X}_s^{desired}(t) = k\,X_m(t)$$

<Slave Controller>

Inverse Kinematics

$q^{desired}$ (for joint position control)

$\dot{q}^{desired}$ (for joint velocity control)

<Slave Joint Controller>

Experiment

Implementing Direct Teleoperation for 7 DoF Manipulator (Franka Panda robot, https://www.franka.de/technology) with given simulation system.

Experiment details and scoring

- **Implementing Teleoperation system with provided simulation setup (40%)**
  - Master-Slave coordinate mapping
    - Since coordinate of slave robot is different from master's, slave's end-effector will move to the different direction with your master commands.
    - Make the slave robot move in the same direction as the master device movement.
  - Position-to-Position control
    - Increasing the workspace of slave robot by using scaling and indexing methods.
  - Position-to-Velocity control

- **Performing the teleoperation task with your implemented system (20%)**
  - Task: Drop cans from the table by pushing it with slave robot.
  - Perform this task with each control methods(P-to-P, P-to-V) and record videos
    - Use screen recorder program, such as 'Kazam', 'SimpleScreenRecorder', and etc..

- **Report (40%)**
- *Submit source codes, videos and report.*

Environment

- **Ubuntu**
  - Tested in 16.04 and 18.04
- **ROS**
  - **Kinetic** (tested, preferred)
  - **Melodic** (tested, but not preferred. Provided system has minor issues with Gazebo simulator in Melodic version. If your Gazebo works fine with given source codes, then doesn’t matter)

Install additional required packages

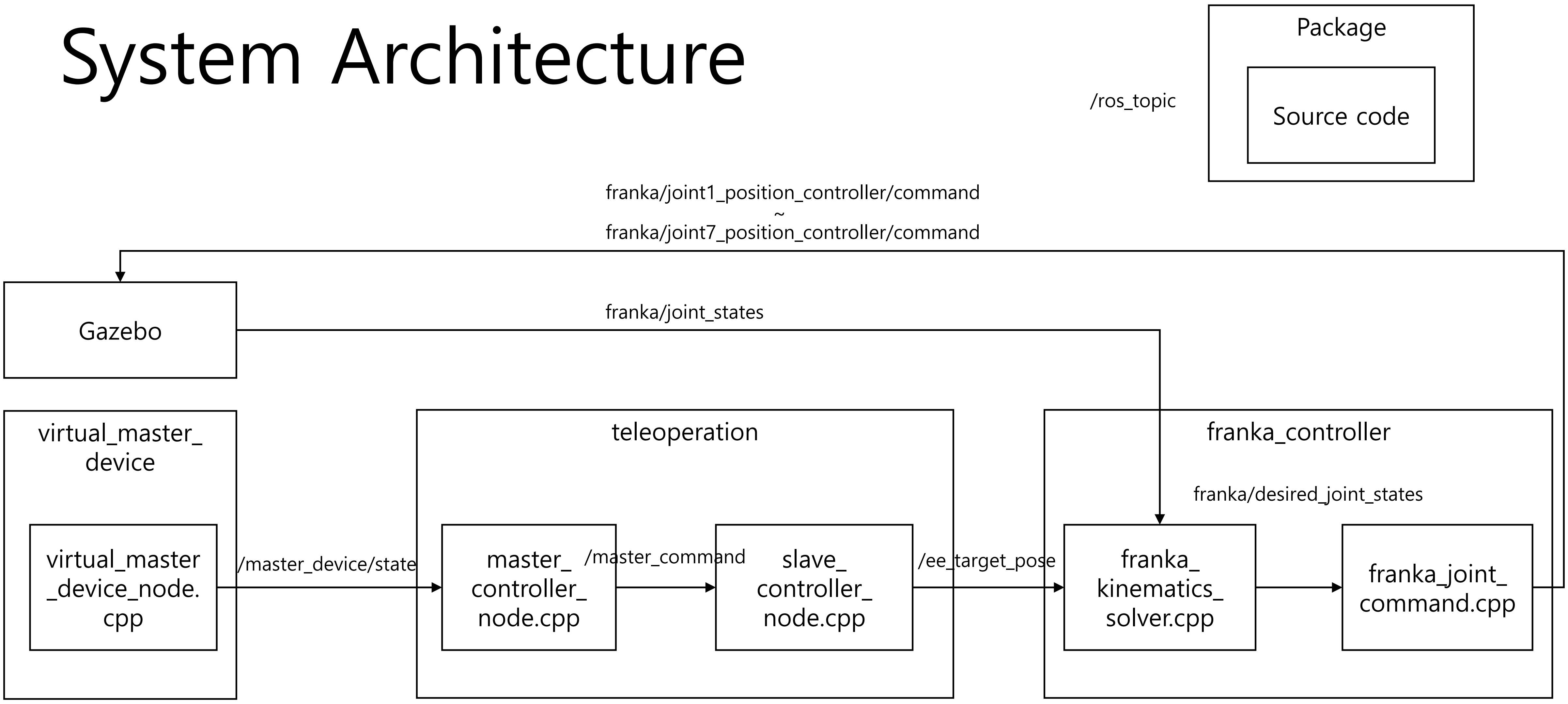
- Please make sure that your system has below packages
  - sudo apt-get install ros-kinetic-gazebo-ros-control
  - sudo apt-get install ros-kinetic-ros-controllers
  - sudo apt-get install ros-kinetic-ros-control

(replace ‘kinetic’ to ‘melodic’ for ROS Melodic)

Build ROS workspace

- Make a directory for ROS workspace
- Extract given '**src.tar.gz**' file to your ROS workspace folder
- Compile and Build
  - Open the terminal (short cut: ctrl+alt+T)
  - Go to your ROS workspace directory and command 'catkin_make'
  - Whenever you modify C++ source codes, you have to build with 'catkin_make' before run your program.

System Architecture



Source codes

- You are only allowed to modify
  - master_controller_node.cpp
  - slave_controller_node.cpp
  - franka_kinematics_solver.cpp

- However, you'd better to check all the other included *.cpp files and *.launch files as well to understand how system works.

teleoperation.launch

- teleoperation_mode
  - Position-to-Position control: type '1'
  - Position-to-Velocity control: type '2'

```xml
<launch>

    <param name="teleoperation_mode" type ="int" value="1"/>
    <!-- 1=increments command, 2=rate control -->

    <node name="master_controller_node" pkg="teleoperation" type="master_controller_node" output="screen"/>
    <node name="slave_controller_node" pkg="teleoperation" type="slave_controller_node" output="screen"/>
    <node type="rviz" name="rviz" pkg="rviz" args="-d $(find teleoperation)/config/franka_teleoperation.rviz" />


</launch>
```

master_controller_node.cpp

- Implement your controller to the ‘MasterDevStateCallback’ function

```cpp
        // The value of 'teleoperation_mode_' varaible is defined by the 'teleoperation_mode' parameter in the 'teleoperation.launch' file
        // 1.Position to Position : publish the increments command
        if(teleoperation_mode_ == 1){

            // Make your increments command

           master_command.pose.position.x = 0.0; // replace '0.0' to your command value
           master_command.pose.position.y = 0.0; // replace '0.0' to your command value
           master_command.pose.position.z = 0.0; // replace '0.0' to your command value


           double roll,pitch,yaw;

           roll = 0.0;  // replace '0.0' to your command value
           pitch = 0.0;  // replace '0.0' to your command value
           yaw = 0.0;  // replace '0.0' to your command value


           // Euler angle to Quaternion
           tf::Quaternion q;
           q.setRPY(roll,pitch,yaw);
           master_command.pose.orientation.x = q.x();
           master_command.pose.orientation.y = q.y();
           master_command.pose.orientation.z = q.z();
           master_command.pose.orientation.w = q.w();

        }

        // 2.Position to Velocity : publish the position command
        else if(teleoperation_mode_ == 2){
            // Same as 'teleoperation_mode_ == 1' case.

        }


        // Publish Master Command
        master_command.header = msg->pos.header;
        master_cmd_pub_.publish(master_command);
```

Implement your controller here.

slave_controller_node.cpp

- Implement your controller to the 'MasterCommandCallback' function
- You may need to transform master command data
  - Implement this transform in the 'MasterCommandCallback' function, but outside of the if statement.

```
        // The value of 'teleoperation_mode_' varaible is defined by the 'teleoperation_mode' parameter in the 'teleoperation.launch' file
        // 1.Position to Position : publish the increments command
        if(teleoperation_mode_ == 1){

            // Implement your controller

            // Update Desired End-effector Pose to the 'target_pose_' variable.

        }

        // 2.Position to Velocity : publish the position command
        else if(teleoperation_mode_ == 2){

            // Implement your controller

            // Update Desired End-effector Pose to the 'target_pose_' variable.

        }
```

Implement your controller here.

franka_kinematics_solver.cpp

- Convert the Cartesian end-effector pose to the joint state.

```cpp
    ///////////////////////////////////////////////////////////////////////////////////////////////////
    sensor_msgs::JointState desired_joint_state; // Put your desired joint values to this variable.



    //--- Implement your code here ---//  implement inverse kinematics by using included functions in this c++ class





    ///////////////////////////////////////////////////////////////////////////////////////////////////
```

- You can utilize included functions

```cpp
int KDLForwardKinematics(const KDL::Chain robot_chain, const KDL::JntArray joint_in, KDL::Frame& cart_pos_out){
```

```cpp
int KDLInverseKinematics(const KDL::Chain robot_chain, const KDL::Frame target_frame,
        const KDL::JntArray q_current, KDL::JntArray& q_out){
```

```cpp
void KDLFrameToPoseStampedMsg(const KDL::Frame kdl_frame_in, geometry_msgs::PoseStamped& pose_stamped_out){
```

```cpp
void PoseStampedMsgToKDLFrame(const geometry_msgs::PoseStamped pose_stamped_in, KDL::Frame& kdl_frame_out){
```

```cpp
void KDLJntArrayToJointStateMsg(const KDL::JntArray jnt_array_in, sensor_msgs::JointState& jointstate_out){
```

```cpp
void JointStateMsgToKDLJntArray(const sensor_msgs::JointState jointstate_in, KDL::JntArray& jnt_array_out){
```

Fundamental Knowledge

- Most of basic parts are already implemented and you can focus on the implementing teleoperation system, but you'd better to know fundamental knowledge to understand how system works (at least below things)

- **STL vector in C++**

- **ROS msgs**
  - geometry_msgs (in particular, PoseStamped and TwistStamped)
    - http://wiki.ros.org/geometry_msgs
  - sensor_msgs/JointState.h
    - http://docs.ros.org/melodic/api/sensor_msgs/html/msg/JointState.html
- **ROS topic publishing and subscribing**
  - Tutorials
    - http://wiki.ros.org/ROS/Tutorials/WritingPublisherSubscriber%28c%2B%2B%29
- **KDL library**
  - https://www.orocos.org/kdl_old/
  - https://www.orocos.org/kdl/examples

Running simulation systems

1. **Launch Gazebo Simulator**
   roslaunch gazebo_launch franka_gazebo.launch

2. **Launch Teleoperation system (It includes Rviz)**
   roslaunch teleoperation teleoperation.launch

3. **Launch Virtual master controller**
   roslaunch virtual_master_device virtual_master_device.launch

4. **Launch Slave franka manipulator controller**
   roslaunch franka_controller franka_controller.launch

Gazebo launch

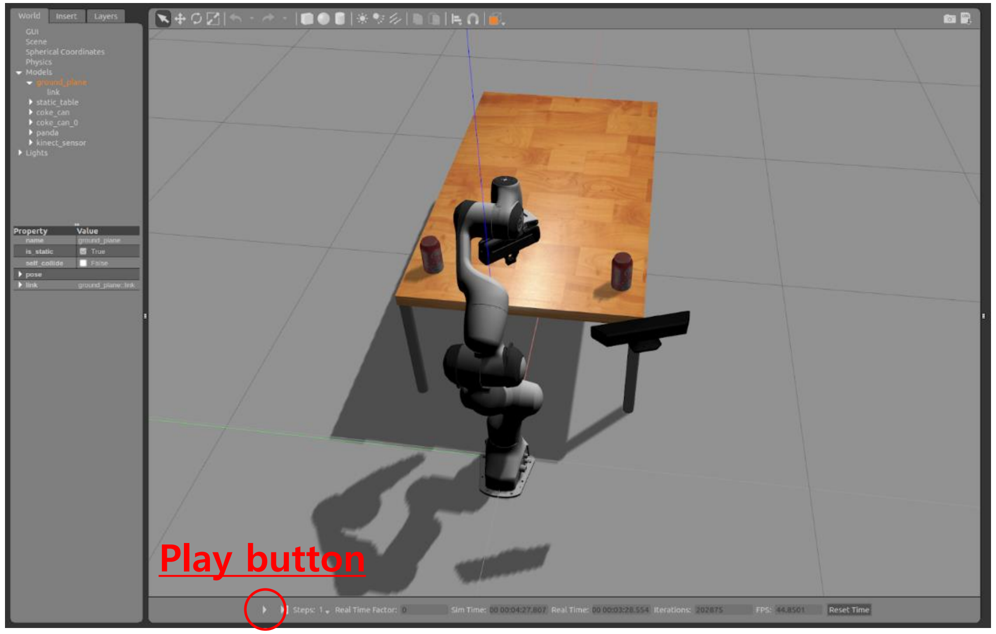
- roslaunch gazebo_launch franka_gazebo.launch
- You can see those errors but it doesn't matter. Don't be afraid.

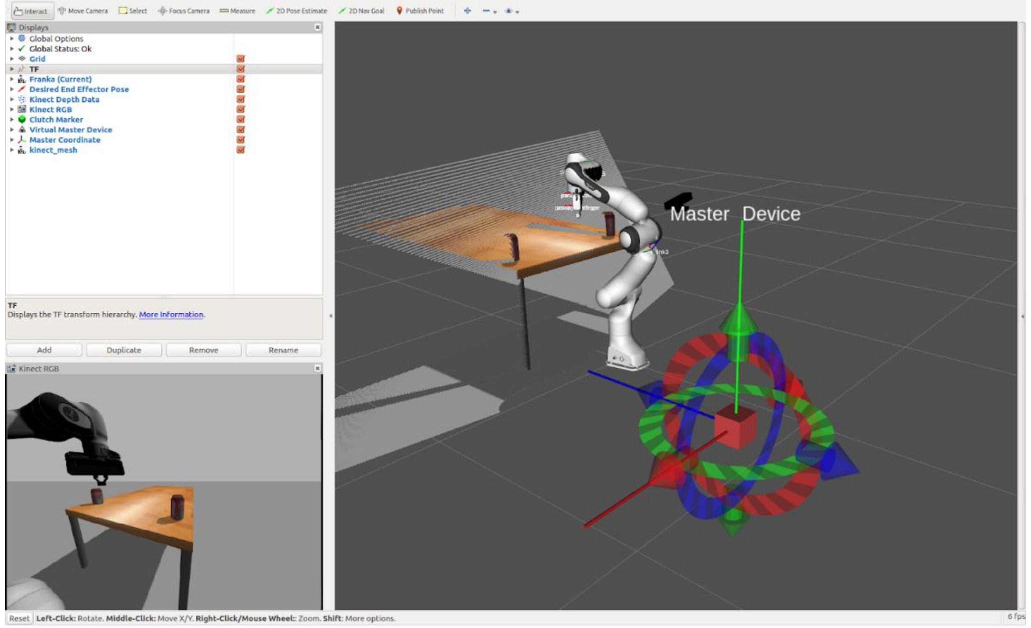
```
kwanghyun@kwanghyun-desktop: ~
kwanghyun@kwanghyun-desktop: ~ 115x34
[ INFO] [1590328325.728581597]: waitForService: Service [/gazebo/set_physics_properties] has not been advertised, w
aiting...
[ INFO] [1590328325.902073490]: Finished loading Gazebo ROS API Plugin.
[ INFO] [1590328325.902484924]: waitForService: Service [/gazebo/set_physics_properties] has not been advertised, w
aiting...
Warning [parser_urdf.cc:1232] multiple inconsistent <gravity> exists due to fixed joint reduction overwriting previ
ous value [false] with [true].
[ INFO] [1590328327.225640249]: Loading gazebo_ros_control plugin
[ERROR] [1590328327.225712998]: GazeboRosControlPlugin missing <legacyModeNS> while using DefaultRobotHWSim, defaul
ts to true.
This setting assumes you have an old package with an old implementation of DefaultRobotHWSim, where the robotNamesp
ace is disregarded and absolute paths are used instead.
If you do not want to fix this issue in an old package just set <legacyModeNS> to true.

[ INFO] [1590328327.225766606]: Starting gazebo_ros_control plugin in namespace: franka
[ INFO] [1590328327.226350543]: gazebo_ros_control plugin is waiting for model URDF in parameter [/robot_descriptio
n] on the ROS param server.
[ERROR] [1590328327.335933087]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint1
[ERROR] [1590328327.336688044]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint2
[ERROR] [1590328327.337466984]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint3
[ERROR] [1590328327.338280749]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint4
[ERROR] [1590328327.339024761]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint5
[ERROR] [1590328327.339771877]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint6
[ERROR] [1590328327.340568347]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_joint7
[ERROR] [1590328327.341357584]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_finger
_joint1
[ERROR] [1590328327.346211550]: No p gain specified for pid.  Namespace: /gazebo_ros_control/pid_gains/panda_finger
_joint2
[ INFO] [1590328327.350523930]: Loaded gazebo_ros_control.
[ INFO] [1590328327.724477754]: Camera Plugin: Using the 'robotNamespace' param: '/'
[ INFO] [1590328327.727195790]: Camera Plugin (ns = /)  <tf_prefix_>, set to ""
[spawn_kinect_urdf-7] process has finished cleanly
log file: /home/kwanghyun/.ros/log/c012007e-9dc5-11ea-be96-7085c239d78b/spawn_kinect_urdf-7*.log
```

Simulation Interface

- After launching four launch files in the previous slide, you can see two windows. One for Gazebo, the other for Rviz.
- Don't forget to click the play button in Gazebo after Gazebo launched.



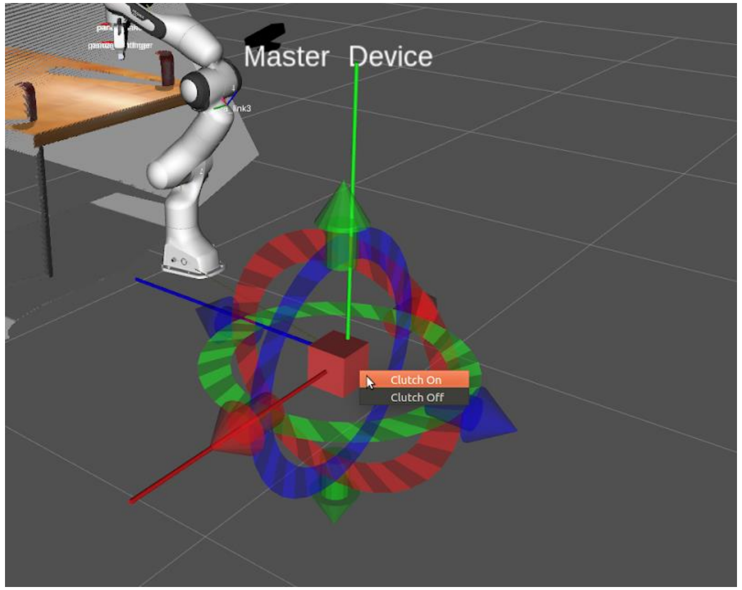


Perform the task while looking at Rviz, not Gazebo as much as possible. You only can observe the remote environment through the sensor data in real teleoperation scenarios.

Virtual Master Device

- You can get the ‘master_device/state’ topic by controlling this virtual master device.
  - 6 DoF
  - Click arrows and circles and Drag.
- If you click the center box, then clutch on/off menu will be popped up.
  - It changes the value of ‘button’ in ‘master_device/state’ topic.
  - It can be used to implement ‘indexing’.
- This controller is implemented in ‘virtual_master_device/src/virtual_master_device_node.cpp’

Warning: Don’t click the center box first just after launching ‘virtual_master_device.launch’. Sometimes Rviz is stopped. Make movements first.



Enjoy the teleoperation!