지능 로봇 설계 공학 실험 (RE510)

Experiment 1:

Manipulator Teleoperation

Prof. Jee-Hwan Ryu

T.A. SeongSu Park, simon.park@kaist.ac.kr

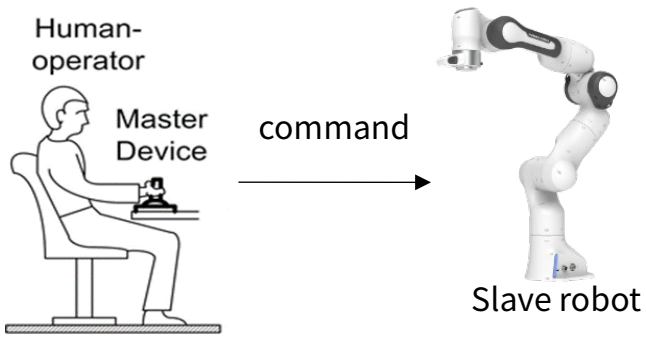
IRiS Lab, KAIST

Goal

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

Teleoperation

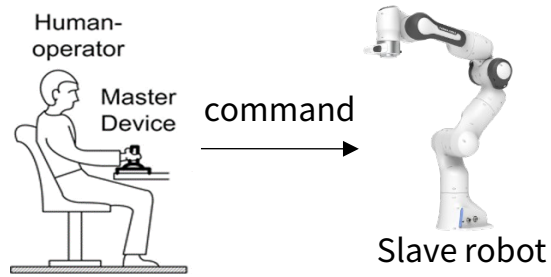
- Human-operator controls the remote robot, called a ***slave robot.***
- To generate motion commands, human operator handles a special device, called a ***master device***.



https://www.youtube.com/watch?v=bCcy2SFDAuY

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.

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>

Position to Position Control

- Updating the desired position of slave's end-effector with the increments of master device.
- Basically,

$\Delta X_m(t)$

or

<Master Controller>

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

or

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

<Slave Controller>

Forward Kinematics

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

Inverse Kinematics

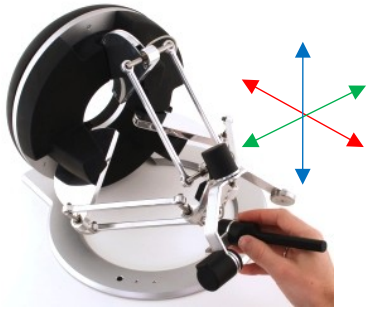
$$q^{desired} = \begin{bmatrix} \theta_0^{desired} \\ \theta_1^{desired} \\ \vdots \\ \theta_n^{desired} \end{bmatrix}$$

<Slave Joint Controller>

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

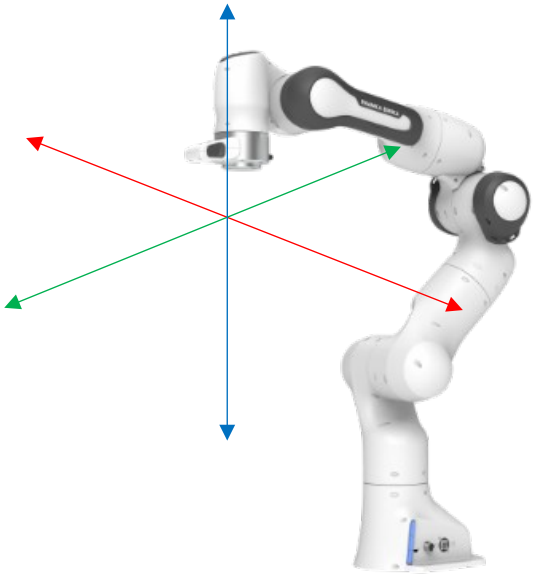
Covering Large workspace

- 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.



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

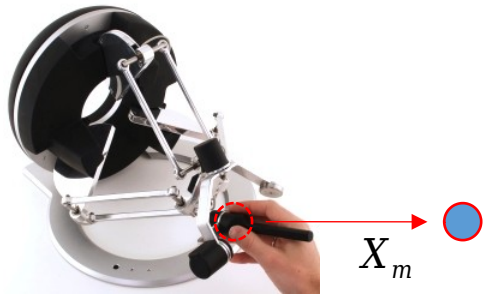
Using the high gain



Covering Large workspace

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

Step 0



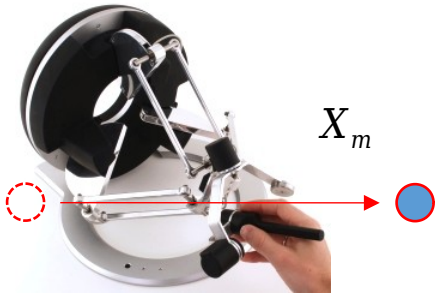
Step 1

Disable signals and move back.

It doesn't generate motion command

Step 2

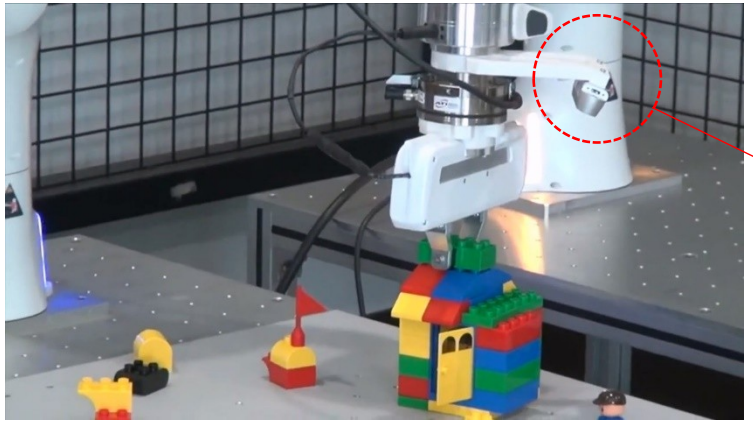
Enable signals again and move



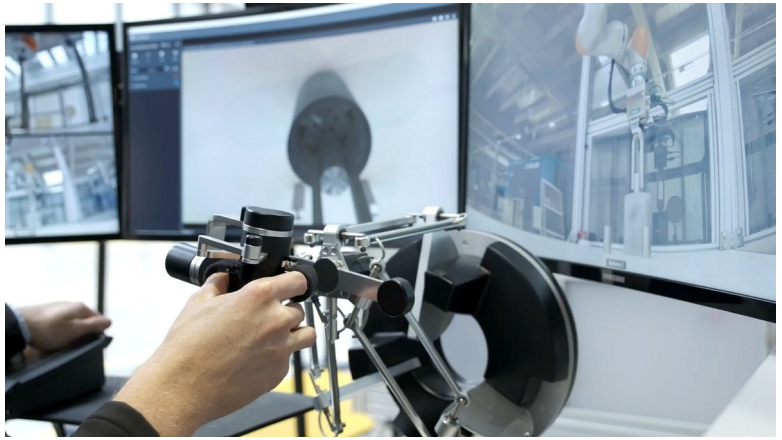
Repeat

Master Device-Cam. Mapping

- EIH (Eye-In-Hand)



- ETH (Eye-To-Hand)



Operators command
based on what they observe.

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 in default setting.
    - Human operator will control the master, with camera image information.
    - Therefore, the correct mapping between master and slave is required. (Check sample video in p.16)
  - 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) in each configuration (EIH, ETH) 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 18.04 and 20.04

- **ROS**
  - **Melodic** for Ubuntu 18.04
  - **Noetic** for Ubuntu 20.04
  - **Installation Guide** at http://wiki.ros.org/ROS/Installation

Install additional required packages

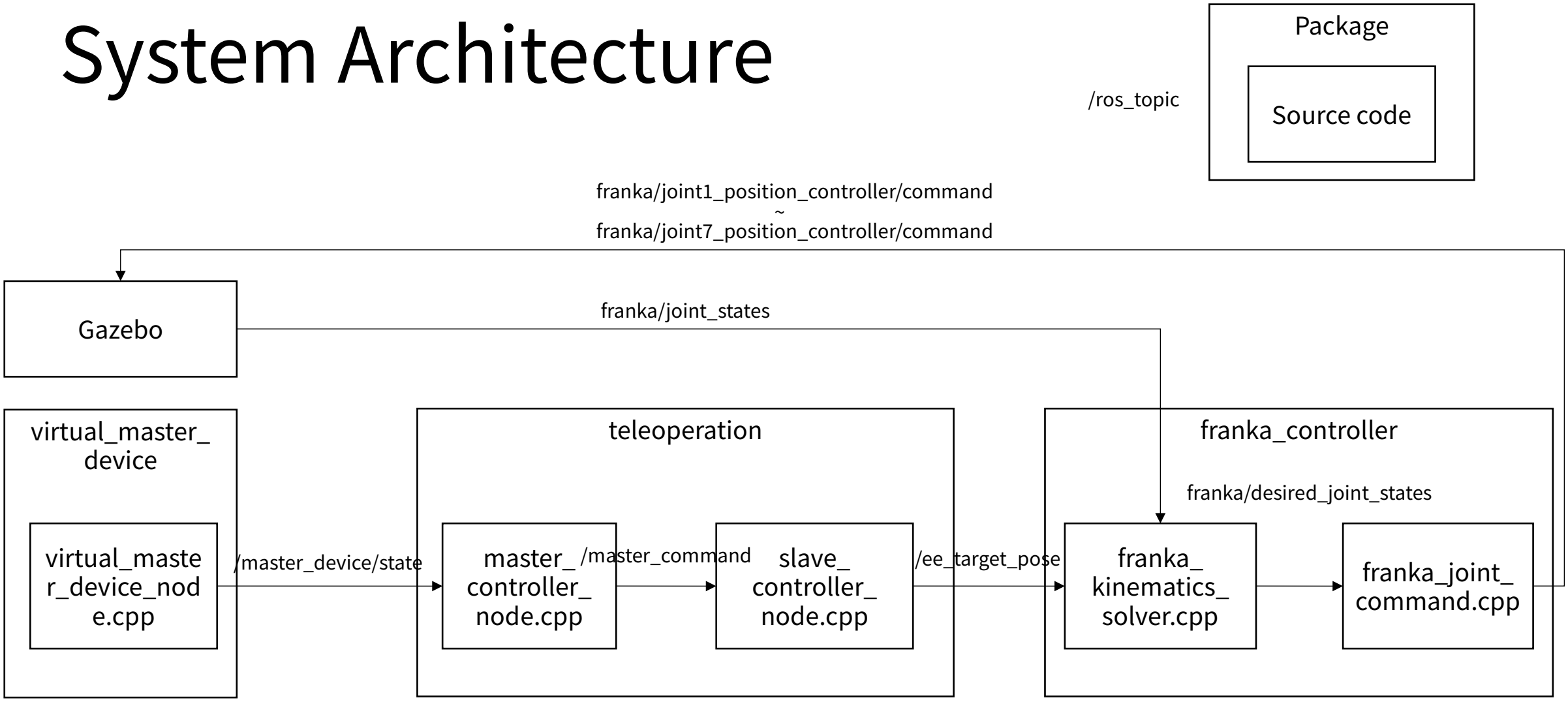
- Please make sure that your system has packages below
  - sudo apt-get install ros-noetic-gazebo-ros-control
  - sudo apt-get install ros-noetic-gazebo-plugins
  - sudo apt-get install ros-noetic-ros-controllers
  - sudo apt-get install ros-noetic-ros-control

  (replace ‘noetic’ to ‘melodic’ for installing in ROS Melodic)

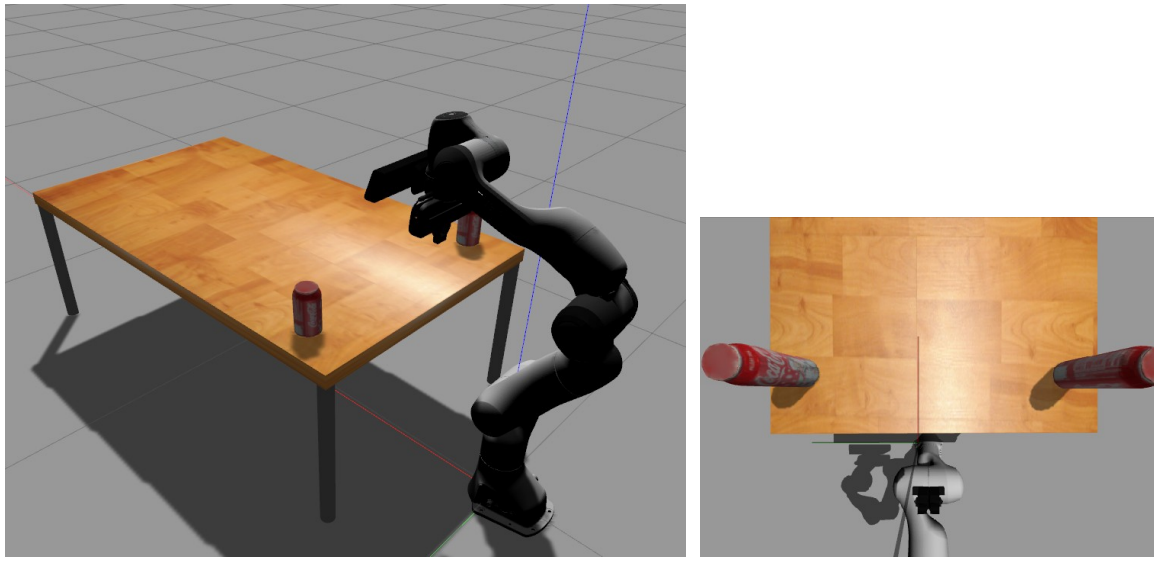
Build ROS workspace

- Make a directory for ROS workspace
- Extract given '**src.zip**' file to your ROS workspace folder
- Compile and Build
  - Open the terminal (shortcut: 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

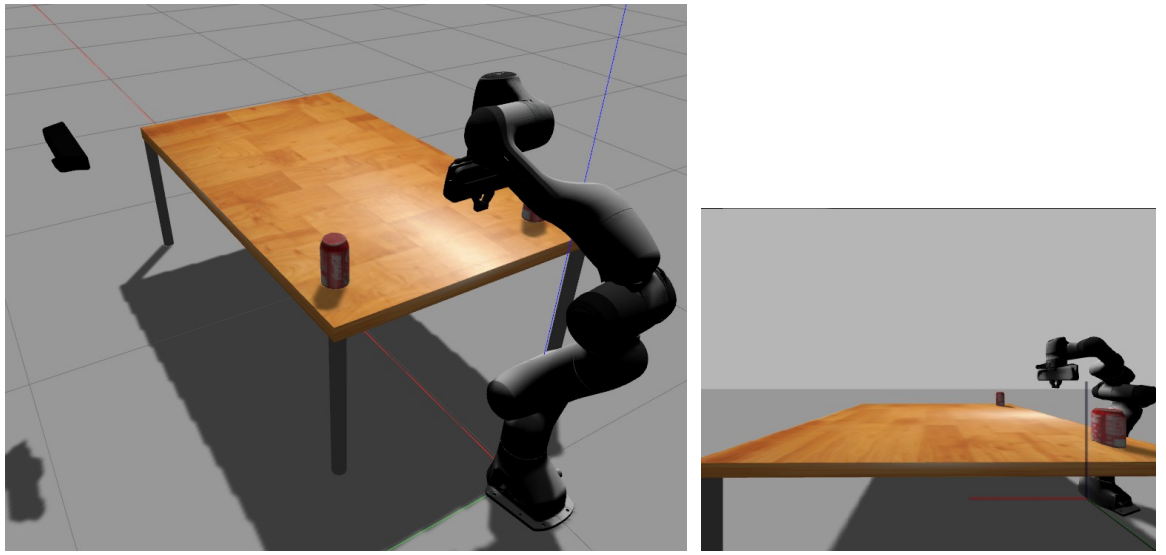


2 different Configurations in Simulation



Eye-in-hand (EIH)

Camera is attached at end-effector of the robot arm.

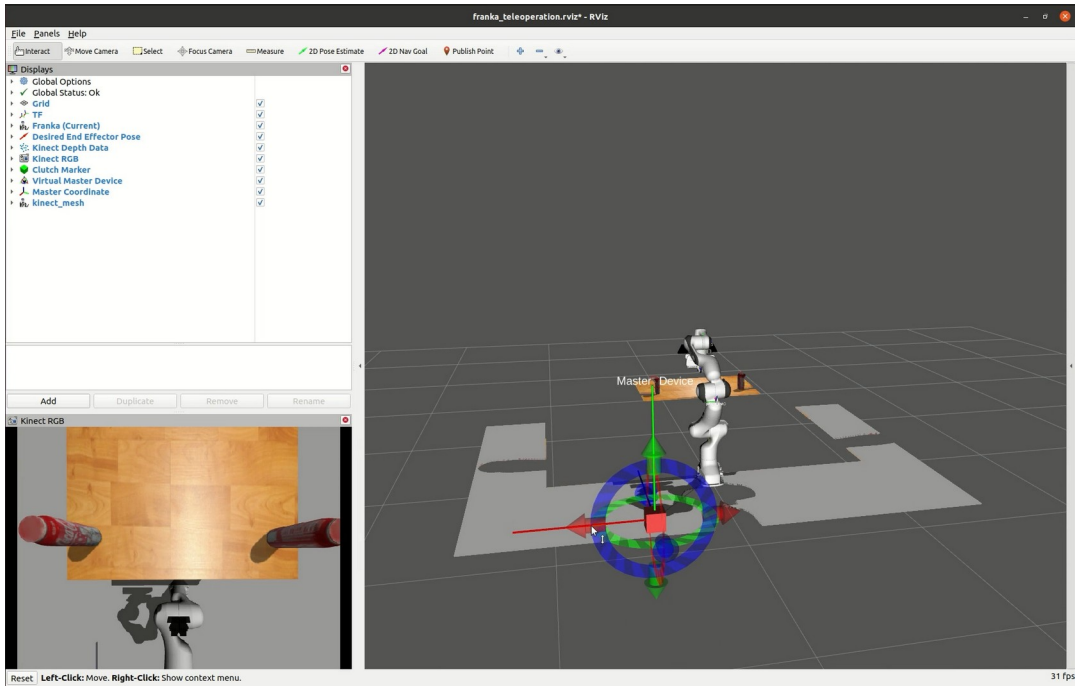


Eye-to-hand (ETH)

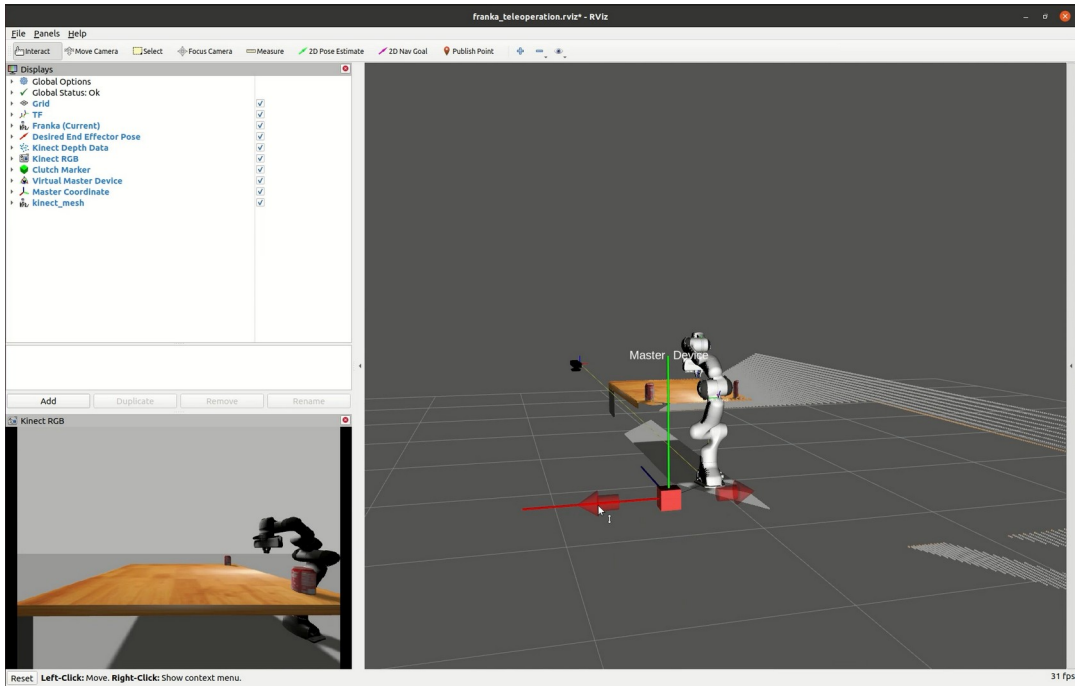
Camera is placed to show the overall environment near the robot.

2 different Configurations in Simulation

- Correct mapping between slave and master is required. **(Check videos below)**



Eye-in-hand (EIH)



Eye-to-hand (ETH)

Source codes

- You are only allowed to modify
  - master_controller_node.cpp (EIH, ETH)
  - slave_controller_node.cpp (EIH, ETH)
  - franka_kinematics_solver.cpp

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

teleoperation.launch (EIH, ETH)

- teleoperation_mode
  - Position-to-Position control: type ‘1’
  - Position-to-Velocity control: type ‘2’

```
<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 (EIH, ETH)

- Implement your controller to the ‘MasterDevStateCallback’ function

```
// 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

    // Translation
    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

    // Orientation
    master_command.pose.orientation.x = 0.0; // replace value to your command value
    master_command.pose.orientation.y = 0.0; // replace value to your command value
    master_command.pose.orientation.z = 0.0; // replace value to your command value
    master_command.pose.orientation.w = 1.0; // replace value to your command value
}
```

Implement your con-troller here.

slave_controller_node.cpp (EIH, ETH)

- Implement your controller to the ‘MasterCommandCallback’ function
- You may need to transform master command data

```
// 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.

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

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

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

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

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

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

203     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
- **Eigen library**
  - https://eigen.tuxfamily.org/dox/GettingStarted.html

Running simulation systems (EIH Config.)

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

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

Running simulation systems (ETH Config.)

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

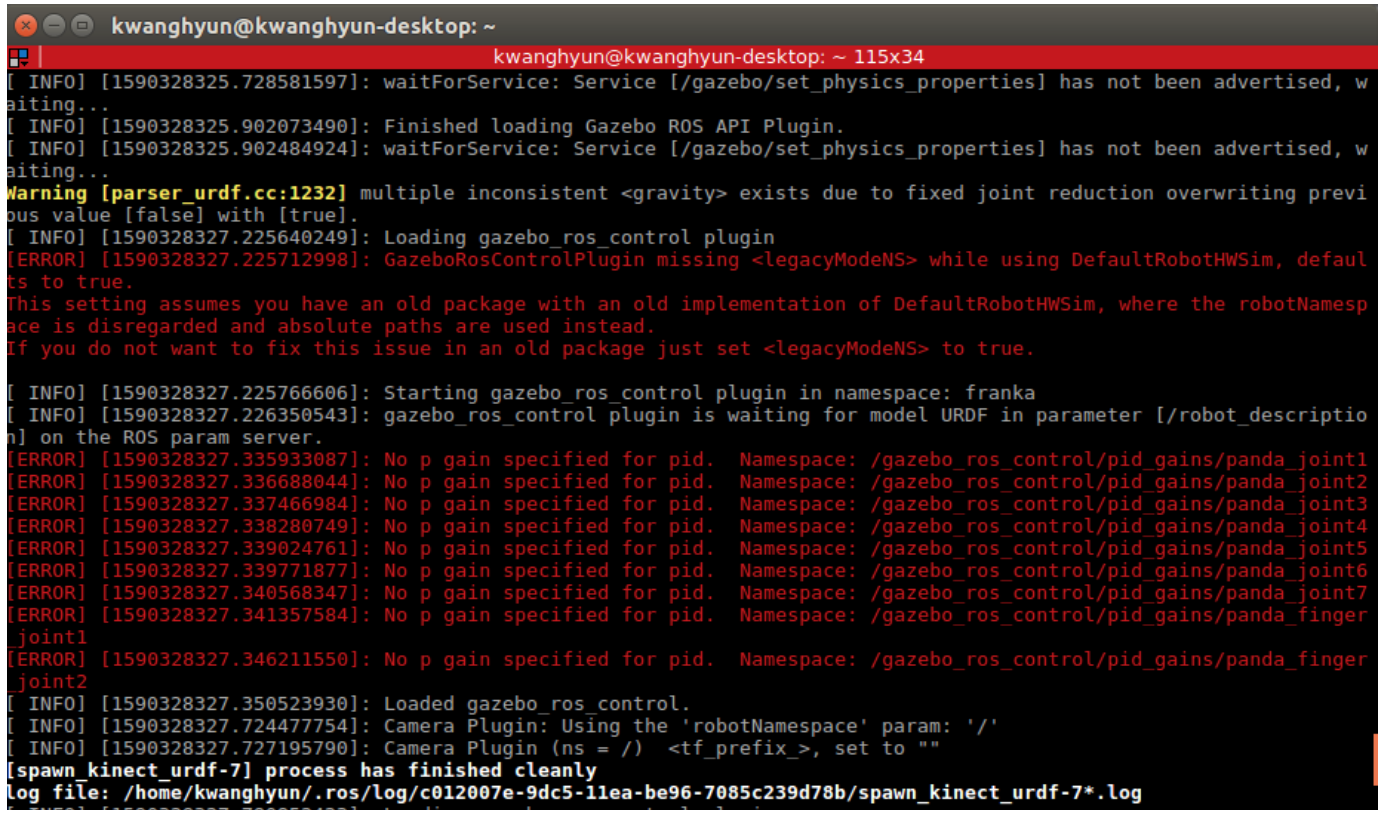
2. **Launch Teleoperation system (It includes Rviz)**
   roslaunch teleoperation teleoperation_ETH.launch

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

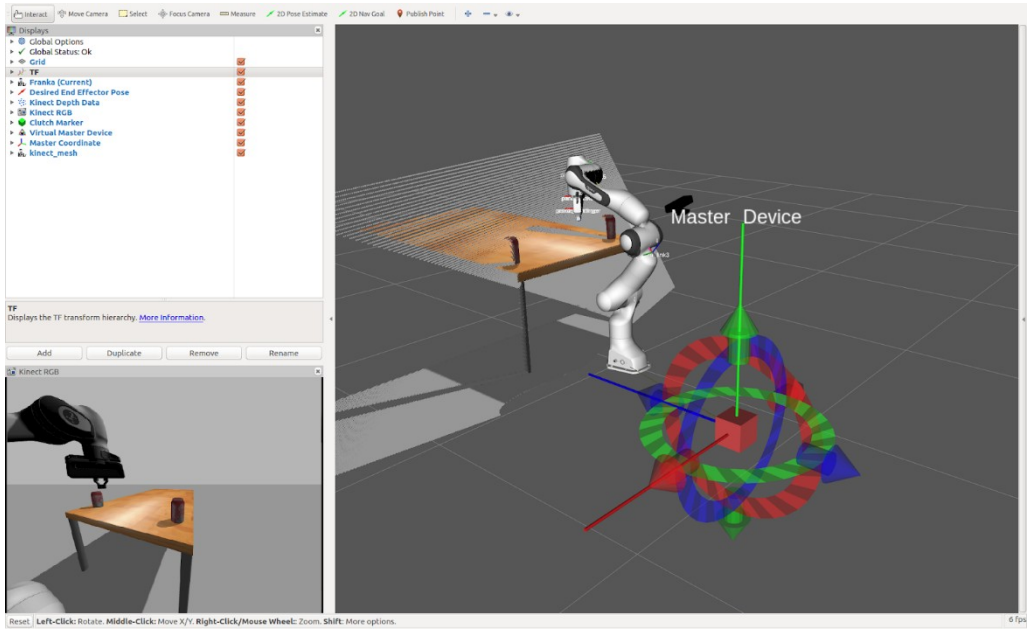
Gazebo launch

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



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.

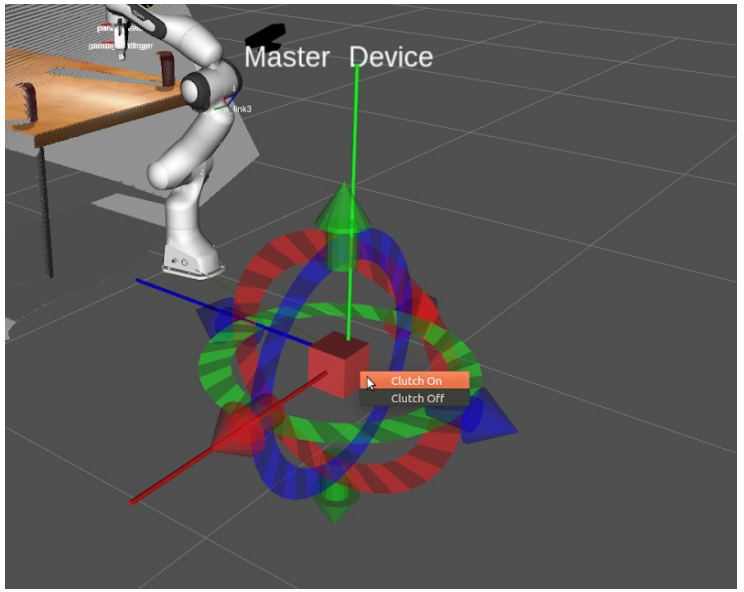


Perform the task while looking at Rviz, not Gazebo as much as possible. You only can observe the remote environment through the sensor data in real teleoperation scenarios.

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 stops.
Make movements first.



Enjoy the teleoperation!

If you have any problem or question, please email TA
simon.park@kaist.ac.kr