Arunrach Julapak 6410110573

ROS2 Project – Gazebo Simulation of the Object Grasping with OpenMANIPULATOR-X Robotic Arm.

This Projects aims to create packages, which using the Gazebo to simulate the object grasping of OpenMANIPULATOR-X robotic arm controlled by keyboard. (In ROS2 Humble, Ubuntu 22.04)

1. Open Terminal Install these dependencies

```
sudo apt install \
  ros-humble-dynamixel-sdk \
  ros-humble-ros2-control \
  ros-humble-moveit* \
  ros-humble-gazebo-ros2-control \
  ros-humble-ros2-controllers \
  ros-humble-controller-manager \
  ros-humble-position-controllers \
  ros-humble-joint-state-broadcaster \
  ros-humble-joint-trajectory-controller \
  ros-humble-gripper-controllers \
  ros-humble-hardware-interface \
  ros-humble-xacro
```

2. Go to ros2_ws/src directory and clone the package repositories by copy these code.

```
cd ~/ros2_ws/src/
git clone https://github.com/IFRA-Cranfield/IFRA_LinkAttacher.git
git clone https://github.com/rorgot1/open_manipulator.git
```

3. Go back to ros2_ws directory and build all packages.

```
cd ~/ros2_ws
colcon build
```

If there is no ros2_linkattacher in the ros2_ws/build and ros2_ws/install you may use this code to build the specific packages.

```
colcon build --packages-select \
    open_manipulator_x_description \
    linkattacher_msgs \
    open_manipulator_x_gui \
    open_manipulator_x_playground \
    open_manipulator_x_bringup \
    open_manipulator_x_moveit_config \
    open_manipulator_x_teleop \
    ros2_linkattacher \
    open_manipulator
```

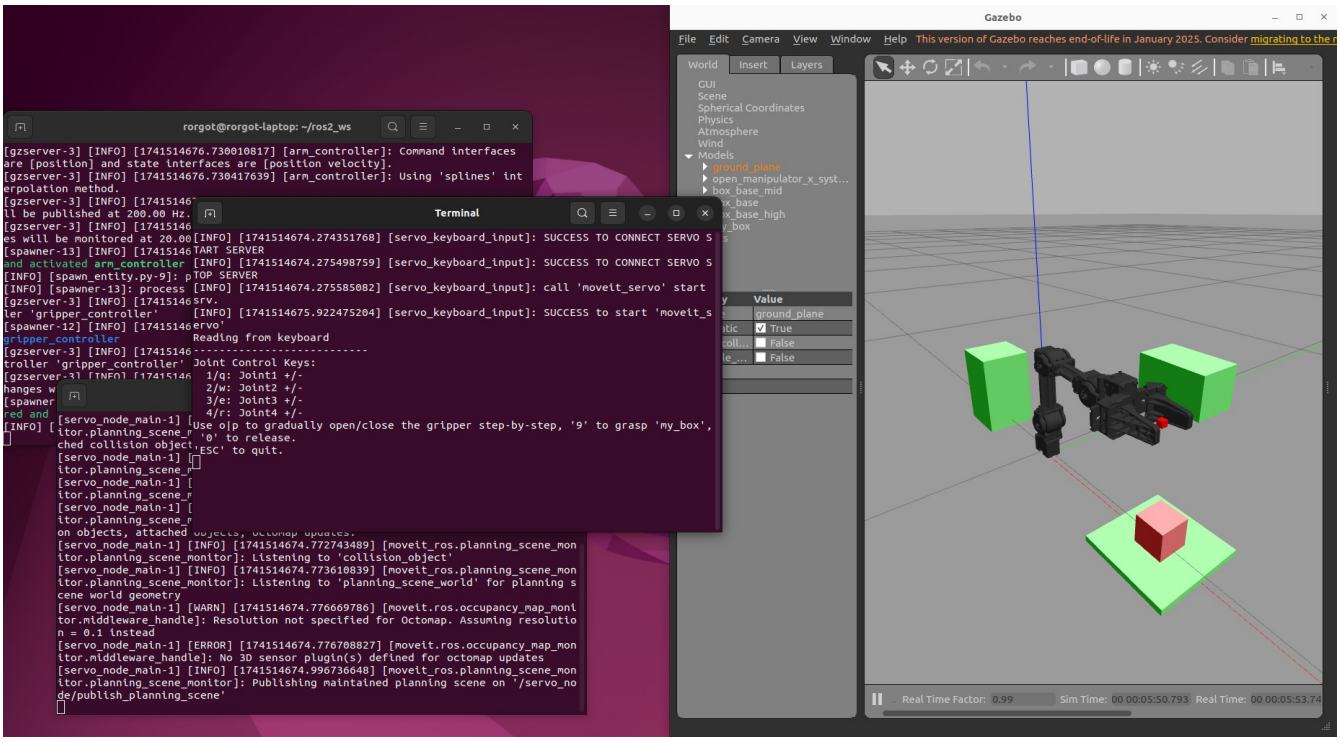
4. Source the setup file

```
cd ~/ros2_ws && source install/setup.bash
```

5. Run the launch file

```
ros2 launch open_manipulator_x_bringup gazebo.launch.py
```

6. Another 2 terminals will appear on the screen and then use the keyboard control terminal to control the robotic arm.



Arunrach Julapak 6410110573

Joint Control Keys:

Joint1 control (waist)	keys: **1** for CCW(+) / **q** for CW(-)
Joint2 control (shoulder)	keys: **2** for CCW(+) / **w** for CW(-)
Joint3 control (elbow)	keys: **3** for CCW(+) / **e** for CW(-)
Joint4 control (wrist)	keys: **4** for CCW(+) / **r** for CW(-)

Gripper Control Keys:

key **o**: gripper open animation
key **p**: gripper close animation
key **9**: object grasp (object attach to link)
key **0**: object release (object detach from link)

Your mission is pick up the red box and place it on the middle or tall green crates.

Detail about code (no need to copy these to your code):

The developer created this project by improve the original code from OpenMANIPULATOR-X, which added the gripper linear movement animation (prismatic joint), the red box object item, the middle and tall green crates, apply the linkattacher_msgs service (by IFRA) to pick the object with gripper. This section shows the modified code in red comment.

-open_manipulator_x_teleop.hpp in *~/ros2_ws/src/open_manipulator/open_manipulator_x_teleop/src*

header file for controlling the Open Manipulator X robotic arm via keyboard teleoperation.

```
#ifndef OPEN_MANIPULATOR_X_TELEOP__OPEN_MANIPULATOR_X_TELEOP_HPP_
#define OPEN_MANIPULATOR_X_TELEOP__OPEN_MANIPULATOR_X_TELEOP_HPP_

#include <rclcpp/rclcpp.hpp>
#include <std_srvs/srv/trigger.hpp>
#include <geometry_msgs/msg/twist_stamped.hpp>
#include <control_msgs/msg/joint_jog.hpp>
#include <control_msgs/action/gripper_command.hpp>
#include <rclcpp_action/rclcpp_action.hpp>
//header for use linkattacher
#include <linkattacher_msgs/srv/attach_link.hpp>
#include <linkattacher_msgs/srv/detach_link.hpp>

#include <signal.h>
#include <stdio.h>
```

```
#include <termios.h>
#include <unistd.h>
#include <chrono>
#include <string>
#include <memory>

#define KEYCODE_1 0x31
#define KEYCODE_2 0x32
#define KEYCODE_3 0x33
#define KEYCODE_4 0x34
#define KEYCODE_Q 0x71
#define KEYCODE_W 0x77
#define KEYCODE_E 0x65
#define KEYCODE_R 0x72
#define KEYCODE_O 0x6F
#define KEYCODE_P 0x70
#define KEYCODE_ESC 0x1B
//add keyboard code for 0 and 9
#define KEYCODE_9 0x39
#define KEYCODE_0 0x30

const char ARM_JOINT_TOPIC[] = "/servo_node/delta_joint_cmds";
const size_t ROS_QUEUE_SIZE = 10;
const char BASE_FRAME_ID[] = "link1";
const double ARM_JOINT_VEL = 3.0;

class KeyboardReader
{
public:
  KeyboardReader();
  void readOne(char * c);
  void shutdown();
private:
  int kfd;
  struct termios cooked;
};

class KeyboardServo
{
public:
  KeyboardServo();
  ~KeyboardServo();
  int keyLoop();
  void connect_moveit_servo();
  void start_moveit_servo();
  void stop_moveit_servo();
  void send_goal(float position);
private:
  void pub();
//animation for controlling gripper gradually
```

```
  void send_gradual_goal(float start_pos, float end_pos, int steps, int delay_ms);
  void goal_result_callback(const
rclcpp_action::ClientGoalHandle<control_msgs::action::GripperCommand>::WrappedResult &
result);

  rclcpp::Node::SharedPtr nh_;
  rclcpp::Client<std_srvs::srv::Trigger>::SharedPtr servo_start_client_;
  rclcpp::Client<std_srvs::srv::Trigger>::SharedPtr servo_stop_client_;
  rclcpp::Publisher<control_msgs::msg::JointJog>::SharedPtr joint_pub_;
  rclcpp_action::Client<control_msgs::action::GripperCommand>::SharedPtr client_;
//add client for gripper attach/detach
  rclcpp::Client<linkattacher_msgs::srv::AttachLink>::SharedPtr attach_client_;
  rclcpp::Client<linkattacher_msgs::srv::DetachLink>::SharedPtr detach_client_;
  rclcpp::executors::SingleThreadedExecutor executor_; // เพิ่ม executor

  control_msgs::msg::JointJog joint_msg_;
  bool publish_joint_;
  bool publish_gripper_;
  bool object_attached_;  //attach flag
};

KeyboardReader input;

void quit(int sig)
{
  (void)sig;
  input.shutdown();
  rclcpp::shutdown();
  exit(0);
}

int main(int argc, char ** argv)
{
  rclcpp::init(argc, argv);
  KeyboardServo keyboard_servo;
  signal(SIGINT, quit);
  int rc = keyboard_servo.keyLoop();
  input.shutdown();
  rclcpp::shutdown();
  return rc;
}

#endif  // OPEN_MANIPULATOR_X_TELEOP__OPEN_MANIPULATOR_X_TELEOP_HPP_
```

-open_manipulator_x_teleop.cpp in *~/ros2_ws/src/open_manipulator/open_manipulator_x_teleop/include/open_manipulator_x_teleop*

the implementation of the teleoperation system for the Open Manipulator X robotic arm.

```
#include <algorithm>
#include <memory>
#include <cmath>

#include "open_manipulator_x_teleop/open_manipulator_x_teleop.hpp"

// KeyboardReader
KeyboardReader::KeyboardReader()
: kfd(0)
{
  tcgetattr(kfd, &cooked);
  struct termios raw;
  memcpy(&raw, &cooked, sizeof(struct termios));
  raw.c_lflag &= ~(ICANON | ECHO);
  raw.c_cc[VEOL] = 1;
  raw.c_cc[VEOF] = 2;
  tcsetattr(kfd, TCSANOW, &raw);
}

void KeyboardReader::readOne(char * c)
{
  int rc = read(kfd, c, 1);
  if (rc < 0) {
    throw std::runtime_error("read failed");
  }
}

void KeyboardReader::shutdown()
{
  tcsetattr(kfd, TCSANOW, &cooked);
}

// KeyboardServo
KeyboardServo::KeyboardServo()
: publish_joint_(false), publish_gripper_(false), object_attached_(false) //add obj attached flag
{
  nh_ = rclcpp::Node::make_shared("servo_keyboard_input");

  servo_start_client_ = nh_->create_client<std_srvs::srv::Trigger>("/servo_node/start_servo");
  servo_stop_client_ = nh_->create_client<std_srvs::srv::Trigger>("/servo_node/stop_servo");

  joint_pub_ = nh_->create_publisher<control_msgs::msg::JointJog>(ARM_JOINT_TOPIC,
ROS_QUEUE_SIZE);
  client_ = rclcpp_action::create_client<control_msgs::action::GripperCommand>(nh_,
"gripper_controller/gripper_cmd");
//add Clients for attach/detach services
```

```
  attach_client_ = nh_->create_client<linkattacher_msgs::srv::AttachLink>("/ATTACHLINK");
  detach_client_ = nh_->create_client<linkattacher_msgs::srv::DetachLink>("/DETACHLINK");

  executor_.add_node(nh_);  //Add node to executor
}

KeyboardServo::~KeyboardServo()
{
  stop_moveit_servo();
}

int KeyboardServo::keyLoop()
{
  char c;

  connect_moveit_servo();
  start_moveit_servo();
//Updated instructions to include grasp/release
  puts("Reading from keyboard");
  puts("---------------------------");
  puts("Joint Control Keys:");
  puts("  1/q: Joint1 +/-");
  puts("  2/w: Joint2 +/-");
  puts("  3/e: Joint3 +/-");
  puts("  4/r: Joint4 +/-");
  puts("Use o|p to gradually open/close the gripper step-by-step, '9' to grasp 'my_box', '0' to
release.");
  puts("'ESC' to quit.");

  std::thread{std::bind(&KeyboardServo::pub, this)}.detach();

  bool servoing = true;
  float current_gripper_pos = -0.01;
  const float GRIPPER_STEP = 0.0029;  //Variables for gradual step gripper control

  while (servoing && rclcpp::ok()) {
    try {
      input.readOne(&c);
    } catch (const std::runtime_error &) {
      perror("read():");
      return -1;
     }

    RCLCPP_INFO(nh_->get_logger(), "value: 0x%02X", c);

    joint_msg_.joint_names.clear();
    joint_msg_.velocities.clear();

    switch (c) {
      case KEYCODE_1:
```

```
    joint_msg_.joint_names.push_back("joint1");
    joint_msg_.velocities.push_back(ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint1 +");
    break;
  case KEYCODE_2:
    joint_msg_.joint_names.push_back("joint2");
    joint_msg_.velocities.push_back(ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint2 +");
    break;
  case KEYCODE_3:
    joint_msg_.joint_names.push_back("joint3");
    joint_msg_.velocities.push_back(ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint3 +");
    break;
  case KEYCODE_4:
    joint_msg_.joint_names.push_back("joint4");
    joint_msg_.velocities.push_back(ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint4 +");
    break;
  case KEYCODE_Q:
    joint_msg_.joint_names.push_back("joint1");
    joint_msg_.velocities.push_back(-ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint1 -");
    break;
  case KEYCODE_W:
    joint_msg_.joint_names.push_back("joint2");
    joint_msg_.velocities.push_back(-ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint2 -");
    break;
  case KEYCODE_E:
    joint_msg_.joint_names.push_back("joint3");
    joint_msg_.velocities.push_back(-ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint3 -");
    break;
  case KEYCODE_R:
    joint_msg_.joint_names.push_back("joint4");
    joint_msg_.velocities.push_back(-ARM_JOINT_VEL);
    publish_joint_ = true;
    RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint4 -");
    break;
  case KEYCODE_O:  //Add Gradual opening
    if (current_gripper_pos < 0.019) {
      current_gripper_pos += GRIPPER_STEP;
```

```
        if (current_gripper_pos > 0.019) current_gripper_pos = 0.019;
        send_gradual_goal(current_gripper_pos, current_gripper_pos, 1, 0);
        RCLCPP_INFO_STREAM(nh_->get_logger(), "Gripper Opening Step: " <<
current_gripper_pos);
      } else {
        RCLCPP_INFO_STREAM(nh_->get_logger(), "Gripper Fully Open");
      }
      break;
    case KEYCODE_P: //Add Gradual closing
      if (current_gripper_pos > -0.01) {
        current_gripper_pos -= GRIPPER_STEP;
        if (current_gripper_pos < -0.01) current_gripper_pos = -0.01;
        send_gradual_goal(current_gripper_pos, current_gripper_pos, 1, 0);
        RCLCPP_INFO_STREAM(nh_->get_logger(), "Gripper Closing Step: " <<
current_gripper_pos);
      } else {
        RCLCPP_INFO_STREAM(nh_->get_logger(), "Gripper Fully Closed");
      }
      break;
    case KEYCODE_9: //Add Grasp object
      {
        //Create request to service to linkattacher_msgs to grasp and identify model name
        auto request = std::make_shared<linkattacher_msgs::srv::AttachLink::Request>();
        request->model1_name = "open_manipulator_x_system";
        request->link1_name = "gripper_right_link";
        request->model2_name = "my_box";
        request->link2_name = "box_link";

        if (attach_client_->wait_for_service(std::chrono::seconds(1))) {
          auto future = attach_client_->async_send_request(request);
          executor_.spin_until_future_complete(future);
          if (future.get()->success) {
            RCLCPP_INFO(nh_->get_logger(), "Grasped my_box successfully");
            object_attached_ = true;
          } else {
            RCLCPP_ERROR(nh_->get_logger(), "Failed to grasp my_box: %s", future.get()-
>message.c_str());
          }
        } else {
          RCLCPP_ERROR(nh_->get_logger(), "Attach service not available");
        }
      }
      break;
    case KEYCODE_0:  //Add Release object
      if (object_attached_) {
        //Create request to service to linkattacher_msgs to release and identify model name
        auto request = std::make_shared<linkattacher_msgs::srv::DetachLink::Request>();
        request->model1_name = "open_manipulator_x_system";
        request->link1_name = "gripper_right_link";
        request->model2_name = "my_box";
```

```
        request->link2_name = "box_link";

        if (detach_client_->wait_for_service(std::chrono::seconds(1))) {
          auto future = detach_client_->async_send_request(request);
          executor_.spin_until_future_complete(future); // ใช้ executor แทน
          if (future.get()->success) {
            RCLCPP_INFO(nh_->get_logger(), "Released my_box successfully");
            object_attached_ = false;
          } else {
            RCLCPP_ERROR(nh_->get_logger(), "Failed to release my_box: %s", future.get()-
>message.c_str());
          }
        } else {
          RCLCPP_ERROR(nh_->get_logger(), "Detach service not available");
        }
      } else {
        RCLCPP_WARN(nh_->get_logger(), "No object attached to release");
      }
      break;
    case KEYCODE_ESC:
      RCLCPP_INFO_STREAM(nh_->get_logger(), "quit");
      servoing = false;
      break;
    default:
      RCLCPP_WARN_STREAM(nh_->get_logger(), "Unassigned input : " << c);
      break;
   }

   executor_.spin_some(); // spin เพื่อประมวลผล callback อื่นๆ
  }

  return 0;
}

void KeyboardServo::send_goal(float position)
{
  auto goal_msg = control_msgs::action::GripperCommand::Goal();
  goal_msg.command.position = position;
  goal_msg.command.max_effort = 100.0;

  auto send_goal_options =
rclcpp_action::Client<control_msgs::action::GripperCommand>::SendGoalOptions();
  send_goal_options.result_callback = std::bind(&KeyboardServo::goal_result_callback, this,
std::placeholders::_1);

  RCLCPP_INFO(nh_->get_logger(), "Sending goal");
  client_->async_send_goal(goal_msg, send_goal_options);
}

void KeyboardServo::send_gradual_goal(float start_pos, float end_pos, int steps, int delay_ms)
```

```
{//function to send gripper goals incrementally
 for (int i = 0; i <= steps; i++) {
   float position = start_pos + (end_pos - start_pos) * i / steps;
   auto goal_msg = control_msgs::action::GripperCommand::Goal();
   goal_msg.command.position = position;
   goal_msg.command.max_effort = 100.0;

   auto send_goal_options =
rclcpp_action::Client<control_msgs::action::GripperCommand>::SendGoalOptions();
   send_goal_options.result_callback = std::bind(&KeyboardServo::goal_result_callback, this,
std::placeholders::_1);

   RCLCPP_INFO(nh_->get_logger(), "Sending gradual goal: position = %f", position);
   client_->async_send_goal(goal_msg, send_goal_options);

   rclcpp::sleep_for(std::chrono::milliseconds(delay_ms));
 }
 publish_gripper_ = true;
}

void KeyboardServo::connect_moveit_servo()
{
 for (int i = 0; i < 10; i++) {
   if (servo_start_client_->wait_for_service(std::chrono::seconds(1))) {
     RCLCPP_INFO_STREAM(nh_->get_logger(), "SUCCESS TO CONNECT SERVO START
SERVER");
     break;
   }
   RCLCPP_WARN_STREAM(nh_->get_logger(), "WAIT TO CONNECT SERVO START
SERVER");
   if (i == 9) {
     RCLCPP_ERROR_STREAM(
       nh_->get_logger(),
       "fail to connect moveit_servo." <<
         "please launch 'servo.launch' at 'open_manipulator_x_moveit_configs' pkg.");
   }
 }
 for (int i = 0; i < 10; i++) {
   if (servo_stop_client_->wait_for_service(std::chrono::seconds(1))) {
     RCLCPP_INFO_STREAM(nh_->get_logger(), "SUCCESS TO CONNECT SERVO STOP
SERVER");
     break;
   }
   RCLCPP_WARN_STREAM(nh_->get_logger(), "WAIT TO CONNECT SERVO STOP
SERVER");
   if (i == 9) {
     RCLCPP_ERROR_STREAM(
       nh_->get_logger(),
       "fail to connect moveit_servo." <<
         "please launch 'servo.launch' at 'open_manipulator_x_moveit_configs' pkg.");
```

```
    }
  }
}

void KeyboardServo::start_moveit_servo()
{
  RCLCPP_INFO_STREAM(nh_->get_logger(), "call 'moveit_servo' start srv.");
  auto future = servo_start_client_->async_send_request(
    std::make_shared<std_srvs::srv::Trigger::Request>());
  executor_.spin_until_future_complete(future);
  if (future.get()->success) {
    RCLCPP_INFO_STREAM(nh_->get_logger(), "SUCCESS to start 'moveit_servo'");
  } else {
    RCLCPP_ERROR_STREAM(nh_->get_logger(), "FAIL to start 'moveit_servo', execute without
'moveit_servo'");
  }
}

void KeyboardServo::stop_moveit_servo()
{
  RCLCPP_INFO_STREAM(nh_->get_logger(), "call 'moveit_servo' END srv.");
  auto future = servo_stop_client_->async_send_request(
    std::make_shared<std_srvs::srv::Trigger::Request>());
  executor_.spin_until_future_complete(future);
  if (future.get()->success) {
    RCLCPP_INFO_STREAM(nh_->get_logger(), "SUCCESS to stop 'moveit_servo'");
  }
}

void KeyboardServo::pub()
{
  while (rclcpp::ok()) {
    if (publish_joint_) {
      joint_msg_.header.stamp = nh_->now();
      joint_msg_.header.frame_id = BASE_FRAME_ID;
      joint_pub_->publish(joint_msg_);
      publish_joint_ = false;
      RCLCPP_INFO_STREAM(nh_->get_logger(), "Joint PUB");
    }
    if (publish_gripper_) {
      publish_gripper_ = false;
    }
    rclcpp::sleep_for(std::chrono::milliseconds(10));
  }
}

void KeyboardServo::goal_result_callback(const
rclcpp_action::ClientGoalHandle<control_msgs::action::GripperCommand>::WrappedResult &
result)
{//Added detailed logging for each action result
```

```
 switch (result.code) {
   case rclcpp_action::ResultCode::SUCCEEDED:
     RCLCPP_INFO(nh_->get_logger(), "Gripper action succeeded");
     break;
   case rclcpp_action::ResultCode::ABORTED:
     RCLCPP_ERROR(nh_->get_logger(), "Gripper action aborted");
     break;
   case rclcpp_action::ResultCode::CANCELED:
     RCLCPP_WARN(nh_->get_logger(), "Gripper action canceled");
     break;
   default:
     RCLCPP_ERROR(nh_->get_logger(), "Unknown result code");
     break;
 }
}
```

-gazebo.launch.py in *~/ros2_ws/src/open_manipulator/open_manipulator_x_bringup/launch*

sets up a simulation environment for the Open Manipulator X robotic arm in Gazebo.

```
import os
from launch import LaunchDescription
from launch.actions import DeclareLaunchArgument
from launch.actions import IncludeLaunchDescription, ExecuteProcess
from launch.launch_description_sources import PythonLaunchDescriptionSource
from launch.substitutions import LaunchConfiguration
from launch.substitutions import PathJoinSubstitution
from launch.substitutions import ThisLaunchFileDir

from launch_ros.actions import Node
from launch_ros.substitutions import FindPackageShare


def is_valid_to_launch():
    # Path includes model name of Raspberry Pi series
    path = '/sys/firmware/devicetree/base/model'
    if os.path.exists(path):
        return False
    else:
        return True

def generate_launch_description():
    if not is_valid_to_launch():
        print('Can not launch fake robot in Raspberry Pi')
        return LaunchDescription([])
```

```
start_rviz = LaunchConfiguration('start_rviz')
prefix = LaunchConfiguration('prefix')
use_sim = LaunchConfiguration('use_sim')

world = LaunchConfiguration(
    'world',
    default=PathJoinSubstitution(
        [
            FindPackageShare('open_manipulator_x_bringup'),
            'worlds',
            'empty_world.model'
        ]
    )
)

pose = {'x': LaunchConfiguration('x_pose', default='0.00'),
        'y': LaunchConfiguration('y_pose', default='0.00'),
        'z': LaunchConfiguration('z_pose', default='0.01'),
        'R': LaunchConfiguration('roll', default='0.00'),
        'P': LaunchConfiguration('pitch', default='0.00'),
        'Y': LaunchConfiguration('yaw', default='0.00')}

return LaunchDescription([
    DeclareLaunchArgument(
        'start_rviz',
        default_value='false',
        description='Whether execute rviz2'),

    DeclareLaunchArgument(
        'prefix',
        default_value='""',
        description='Prefix of the joint and link names'),

    DeclareLaunchArgument(
        'use_sim',
        default_value='true',
        description='Start robot in Gazebo simulation.'),

    DeclareLaunchArgument(
        'world',
        default_value=world,
        description='Directory of gazebo world file'),

    DeclareLaunchArgument(
        'x_pose',
        default_value=pose['x'],
        description='position of open_manipulator_x'),

    DeclareLaunchArgument(
        'y_pose',
```

```
        default_value=pose['y'],
        description='position of open_manipulator_x'),

    DeclareLaunchArgument(
        'z_pose',
        default_value=pose['z'],
        description='position of open_manipulator_x'),

    DeclareLaunchArgument(
        'roll',
        default_value=pose['R'],
        description='orientation of open_manipulator_x'),

    DeclareLaunchArgument(
        'pitch',
        default_value=pose['P'],
        description='orientation of open_manipulator_x'),

    DeclareLaunchArgument(
        'yaw',
        default_value=pose['Y'],
        description='orientation of open_manipulator_x'),

    IncludeLaunchDescription(
        PythonLaunchDescriptionSource([ThisLaunchFileDir(), '/base.launch.py']),
        launch_arguments={
            'start_rviz': start_rviz,
            'prefix': prefix,
            'use_sim': use_sim,
        }.items(),
    ),

    IncludeLaunchDescription(
        PythonLaunchDescriptionSource(
            [
                PathJoinSubstitution(
                    [
                        FindPackageShare('gazebo_ros'),
                        'launch',
                        'gazebo.launch.py'
                    ]
                )
            ]
        ),
        launch_arguments={
            'verbose': 'false',
            'world': world,
        }.items(),
    ),
```

```
        Node(
            package='gazebo_ros',
            executable='spawn_entity.py',
            arguments=[
                '-topic', 'robot_description',
                '-entity', 'open_manipulator_x_system',
                '-x', pose['x'], '-y', pose['y'], '-z', pose['z'],
                '-R', pose['R'], '-P', pose['P'], '-Y', pose['Y'],
                ],
            output='screen',
        ),

        Node( #Spawn base for red box
            package='gazebo_ros',
            executable='spawn_entity.py',
            namespace='box_base',
            arguments=[
                '-file', PathJoinSubstitution([FindPackageShare('open_manipulator_x_bringup'), 'worlds',
'base.sdf']),
                '-entity', 'box_base',
                '-x', '0.3', '-y', '0.0', '-z', '0.0',
            ],
            output='screen',
        ),
        Node( #Spawn middle crate
            package='gazebo_ros',
            executable='spawn_entity.py',
            namespace='mid_base',
            arguments=[
                '-file', PathJoinSubstitution([FindPackageShare('open_manipulator_x_bringup'), 'worlds',
'middle_base.sdf']),
                '-entity', 'box_base_mid',
                '-x', '0.0', '-y', '0.4', '-z', '0.05',
            ],
            output='screen',
        ),
        Node( #Spawn tall crate
            package='gazebo_ros',
            executable='spawn_entity.py',
            namespace='high_base',
            arguments=[
                '-file', PathJoinSubstitution([FindPackageShare('open_manipulator_x_bringup'), 'worlds',
'high_base.sdf']),
                '-entity', 'box_base_high',
                '-x', '-0.2', '-y', '0.0', '-z', '0.05',
            ],
            output='screen',
        ),
        Node( #Spawn red box
            package='gazebo_ros',
```

```
            executable='spawn_entity.py',
            namespace='boxie',
            arguments=[
                '-file', PathJoinSubstitution([FindPackageShare('open_manipulator_x_bringup'), 'worlds',
'box.sdf']),
                '-entity', 'my_box',
                '-x', '0.3', '-y', '0.0', '-z', '0.23',
            ],
            output='screen',
        ),
        #launch MoveIt Servo and Teleop
        ExecuteProcess(
            cmd=['gnome-terminal', '--', 'ros2', 'launch', 'open_manipulator_x_moveit_config',
'servo.launch.py'],
            output='screen'
        ),
        ExecuteProcess(
            cmd=['gnome-terminal', '--', 'ros2', 'run', 'open_manipulator_x_teleop',
'open_manipulator_x_teleop'],
            output='screen'
        ),
    ])
```

Develop By Arunrach Julapak – 6410110573 (23 MtE)

Mechatronics Engineering – Prince of Songkla University
Course: Introduction to Robot Operating System 219-431

Email: got.arunrach@gmail.com
Facebook: www.facebook.com/arunrach.julapak
Github: github.com/rorgot1

Reference Packages
OpenMANIPULATOR-X
IFRA_LinkAttacher