# Workshop 5 Simulating Robots Using ROS and Gazebo

## Aims and Objectives

Understanding the idea about the working of robots in a virtual environment.

## Things to learn

- Understanding robotic simulation and Gazebo
- Simulation a model of a robotic arm for Gazebo
- Simulating the robotic arm with an *rgb-d* sensor
- Moving robot joints using ROS controllers in Gazebo

Gazebo is a multi-robot simulator for complex indoor and outdoor robotic simulation. We can simulate complex robots, robot sensors, and a variety of 3D objects. Gazebo already has simulation models of popular robots, sensors, and a variety of 3D objects in their repository (https://bitbucket.org/osrf/gazebo_models/). We can directly use these models without having to create new ones.

Gazebo has a good interface in ROS, which exposes the whole control of Gazebo in ROS. We can install Gazebo without ROS, and we should install the ROS-Gazebo interface to communicate from ROS to Gazebo. Go through the following three exercises; screen shot the observations and comments on them.

## Exercise

1. **Simulating the robotic arm using Gazebo and ROS**

   Before starting with Gazebo and ROS, we should install the following packages to work with Gazebo and ROS:

   **$ sudo apt-get install ros-melodic-gazebo-ros-pkgs ros-melodic-gazebo-msgs ros-melodic-gazebo-plugins ros-melodic-gazebo-ros-control**

   After installation, check whether Gazebo is properly installed using the following commands:

   **$ roscore & rosrun gazebo_ros gazebo**

   to open the Gazebo GUI

2. **Creating the robotic arm simulation model for Gazebo**

   You can create the package needed to simulate the robotic arm using the following command:

   **$ catkin_create_pkg seven_dof_arm_gazebo gazebo_msgs gazebo_plugins gazebo_ros gazebo_ros_control MyStudentID_robot_description_pkg**

   Please remember to change the MyStudentID in the last command into your own initial+yourID. You can also download a package from Teams (https://teams.microsoft.com/_#/school/files/Week%2006%20%20(28.10)%20Gazebbo%20simulation?threadId=19%3A3a9750fc021c4ce68f2558b5059b13c8%40thread.tacv2&ctx=cha

nnel&context=seven_dof_arm_gazebo&rootfolder=%252Fsites%252FEng_7_Rob_Workshop%252FShared%2520Documents%252FWeek%252007%2520%2520(0-4.11)%2520Gazebbo%2520simulation%252Fseven_dof_arm_gazebo)

You can see the complete simulation model of the robot in the seven_dof_arm.xacro file, placed in the MyStudentID_robot_description_pkg/urdf/ folder.
The file is filled with URDF tags, which are necessary for the simulation. We will define the sections of collision, inertial, transmission, joints, links, and Gazebo

To launch the existing simulation model, we can use the seven_dof_arm_gazebo package, which has a launch file called seven_dof_arm_world.launch. The file definition is as follows:

```
<launch>

    <!-- these are the arguments you can pass this launch file, for example paused:=true -->

    <arg name="paused" default="false"/>
    <arg name="use_sim_time" default="true"/>
    <arg name="gui" default="true"/>
    <arg name="headless" default="false"/>
    <arg name="debug" default="false"/>
    <!-- We resume the logic in empty_world.launch -->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">

        <arg name="debug" value="$(arg debug)" />
        <arg name="gui" value="$(arg gui)" />
        <arg name="paused" value="$(arg paused)"/>
        <arg name="use_sim_time" value="$(arg use_sim_time)"/>
        <arg name="headless" value="$(arg headless)"/>

    </include>
    <!-- Load the URDF into the ROS Parameter Server -->
    <param name="robot_description" command="$(find xacro)/xacro --inorder
    '$(find MyStudentID_robot_description_pkg)/urdf/seven_dof_arm.xacro'" />
    <!-- Run a python script to the send a service call to gazebo_ros to
    spawn a URDF robot -->
    <node name="urdf_spawner" pkg="gazebo_ros" type="spawn_model"
    respawn="false" output="screen"
    args="-urdf -model seven_dof_arm -param robot_description"/>

</launch>
```

Launch the following command and check what you get:
**$ roslaunch seven_dof_arm_gazebo seven_dof_arm_world.launch**

I shall discuss how to add to robot model the colurs and textures, transmission tags to actuate the model, gazebo_ros_control plugin, and 3D vision sensor to Gazebo in the class.

3. Moving robot joints using ROS controllers in Gazebo
In this exercise, we are going to discuss how to move each joint of the robot in Gazebo. To move each joint, we need to assign an ROS controller. In particular, for each joint we need to attach a controller that is compatible with the hardware interface mentioned inside the transmission tags.

An ROS controller mainly consists of a feedback mechanism that can receive a set point and control the output using the feedback from the actuators.

a) **Understanding the ros_control packages**

The ros_control packages have the implementation of robot controllers, controller managers, hardware interfaces, different transmission interfaces, and control toolboxes. The ros_controls packages are composed of the following individual packages:

control_toolbox: This package contains common modules (PID and Sine) that can be used by all controllers
controller_interface: This package contains the interface base class for controllers
controller_manager: This package provides the infrastructure to load, unload, start, and stop controllers
controller_manager_msgs: This package provides the message and service definition for the controller manager
hardware_interface: This contains the base class for the hardware interfaces
transmission_interface: This package contains the interface classes for the transmission interface (differential, four bar linkage, joint state, position, and velocity)

b) **Different types of ROS controllers and hardware interfaces**

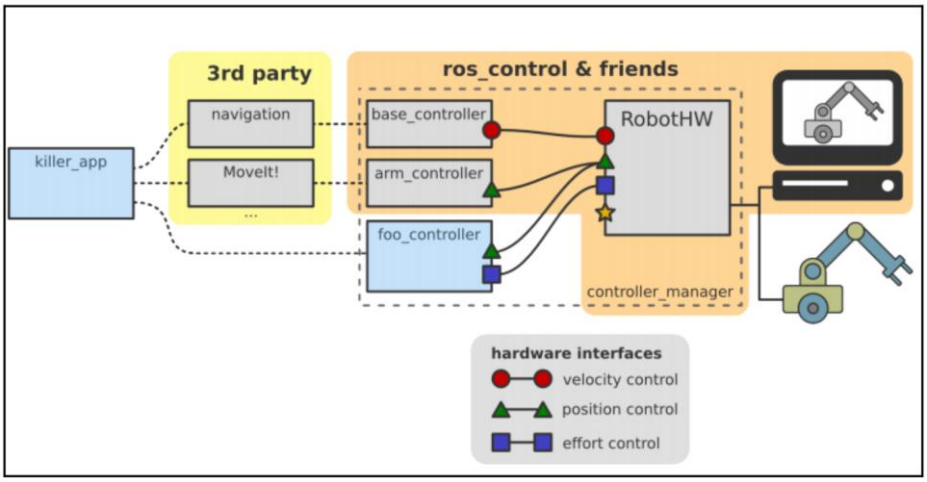
Let's see the list of ROS packages that contain the standard ROS controllers:

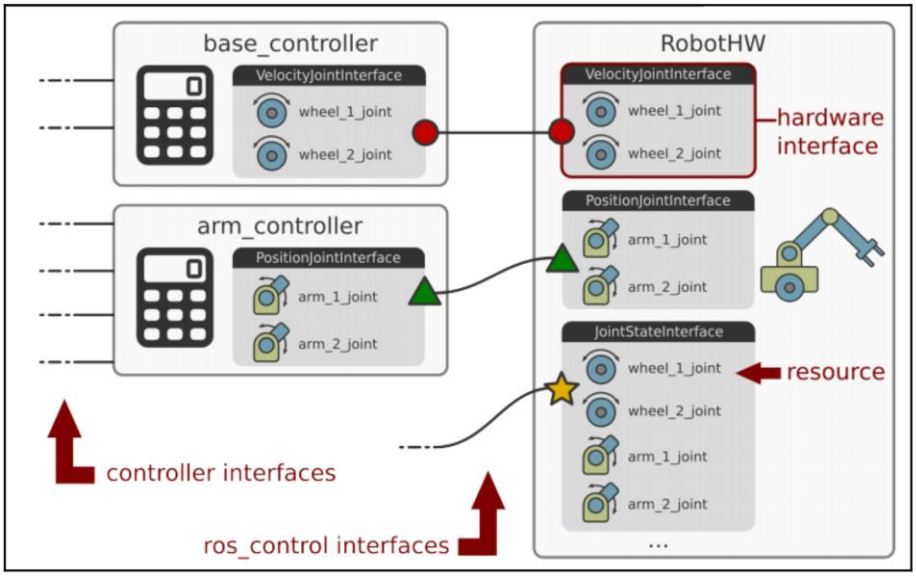
joint_position_controller: This is a simple implementation of the joint position controller
joint_state_controller: This is a controller to publish joint states
joint_effort_controller: This is an implementation of the joint effort (force) controller

The following are some of the commonly used hardware interfaces in ROS:

Joint Command Interfaces: This will send the commands to the hardware
Effort Joint Interface: This will send the effort command
Velocity Joint Interface: This will send the velocity command
Position Joint Interface: This will send the position command
Joint State Interfaces: This will retrieve the joint states from the actuators encode

c) **How the ROS controller interacts with Gazebo**




The hardware interface is decoupled from actual hardware and simulation. The values from the hardware interface can be fed to Gazebo for simulation or to the actual hardware itself.

d) **Interfacing joint state controllers and joint position controllers to the arm**

Interfacing robot controllers to each joint is a simple task. The first task is to write a configuration file for two controllers.

The joint state controllers will publish the joint states of the arm and the joint position controllers can receive a goal position for each joint and can move each joint.

We will find the configuration file for the controller at seven_dof_arm_gazebo_control.yaml in the seven_dof_arm_gazebo/config folder.

Here is the configuration file definition:

```
seven_dof_arm:
# Publish all joint states ----------------------------------
joint_state_controller:
type: joint_state_controller/JointStateController
publish_rate: 50
# Position Controllers ---------------------------------------
joint1_position_controller:
```

```
  type: position_controllers/JointPositionController
  joint: shoulder_pan_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}
joint2_position_controller:
  type: position_controllers/JointPositionController
  joint: shoulder_pitch_joint
  pid: {p: 100.0, i: 0.01, d: 10.0}

……..
```

We can see that all the controllers are inside the namespace seven_dof_arm, and the first line represents the joint state controllers, which will publish the joint state of the robot at the rate of 50 Hz.
The remaining controllers are joint position controllers, which are assigned to the first seven joints, and they also define the PID gains.

e) **Launching the ROS controllers with Gazebo**

If the controller configuration is ready, we can build a launch file that starts all the controllers along with the Gazebo simulation. Navigate to the seven_dof_arm_gazebo/launch directory and you can open the seven_dof_arm_gazebo_control.launch file. This launch files start the Gazebo simulation of the arm, load the controller configuration, load the joint state controller and joint position controllers, and, finally, run the robot state publisher, which publishes the joint states and TF.

```
$ roslaunch seven_dof_arm_gazebo seven_dof_arm_gazebo_control.launch
```

f) **Moving the robot joints**

After finishing the preceding topics, we can start commanding positions to each joint.
To move a robot joint in Gazebo, we should publish a desired joint value with a message type std_msgs/Float64 to the joint position controller command topics. Here is an example of moving the fourth joint to 1.0 radians:

```
$ rostopic pub /seven_dof_arm/joint4_position_controller/command
std_msgs/Float64 1.0
```

We can also view the joint states of the robot by using the following command:

```
$ rostopic echo /seven_dof_arm rostopic pub
/seven_dof_arm/joint4_position_controller/command std_msgs/Float64 1.0
/joint_states
```