Writing ROS Controllers and Visulization Plugins

⇒ We will continue with pluginlib based concepts such as ROS controllers and RViz plugins.

⇒ The main set of packages used to develop a controller generic to all robot is contained in the ros_control stack.

→ Rewritten version of Ros2_mechanism.

⇒ Useful packages that help us to write robot Controllers:

- ros_control

→ This package takes as input the joint state data directly from the robot's actuators and a desired set point, generating the output to send to send to its motors.

→ Output is usually represented by the joint position, velocity or effort.

- **Controller_manager**
  - Control manager can load and manage multiple controllers and can work them in a real-time compatible loop.
- **Controller_interface**
  - This is the controller base class package from which all custom controllers should inherit the controller base class.
  - The controller manager will only load the controller if it inherits from this package.
- **hardware_interface**
  - This package represents the interface between the implemented controller and hardware of the robot.
- **joint_limits_interface**
  - This package allows us to set joint limits to safely work with our robot.

- realtime_tools

  ⇒ Contains set of tools that can be used from a hard real-time thread.

★ Understanding ros_control package

@ The controller_interface package

⇒ The basic ROS low level controller that we want to implement must inherit a base class called

controller_interface :: Controller

↓

This is a base class containing four fundamental functions

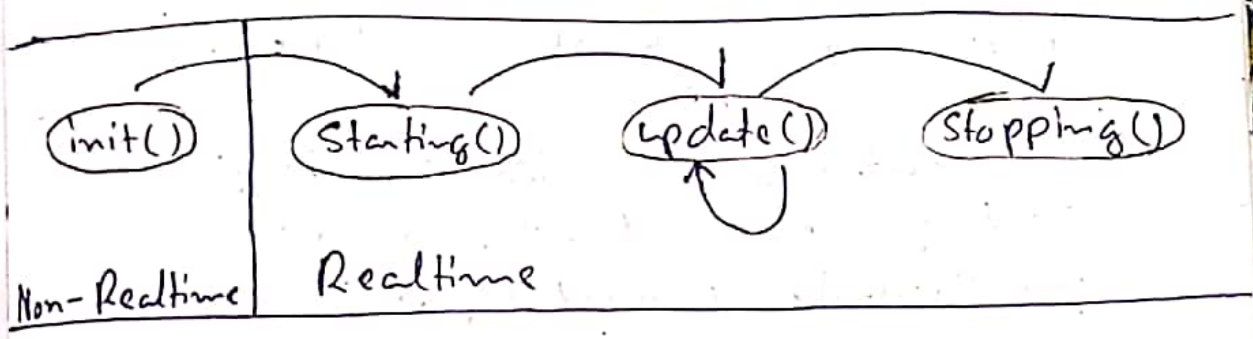
{ init(), start(), update() and stop() }

⇒ The basic structure of the controller class is given as follows:

```
namespace controller_interface
{
  class_Controller
    {
    Public:
      virtual bool init (hardware_Interface
                         *robotHW, ros:: NodeHandle &nh);
```

```
    virtual void starting();
    virtual void update();
    virtual void stopping();
};
}
```

⇒ The workflow of the controller class is shown as follows:



Initializing the controller

⇒ The first function executing when a controller is loaded is init().

↓

{It just initializes the controller}

hardware_interface ⇒ This variable represents the specific hardware interface used by the controller to develop.

↳ ROS contains a list of already-implemented hardware interfaces, such as:

- → Joint Command Interface (effort, velocity and position)
- → Joint State Interface
- → Actuator State Interface

↳ We can even create our own hardware interface.

ros::NodeHandle ⇒ The controller can read the robot configuration and even advertise topics using this NodeHandle.

Starting the ROS Controller

⇒ Starting() method executes once just before running the controller.

⇒ The controller can also call the starting() method when it restarts the controller without uploading it.

Updating the ROS controller

update() method, by default, executes the code inside it at a rate of 1000 Hz.

Stopping the controller

Stopping() method will be called when a controller is stopped.

@ The controller_manager

⇒ The controller_manager package can load and unload the desired controller.

⇒ The controller_manager also ensures that the controller will not set a goal value that is less than or greater than the safety limits of the joints.

⇒ The controller_manager also publishes the states of the joint in the /joint_state topic at a default rate of 100 Hz.

★ Writing a basic joint controller in ROS

Step 1: Creating the controller package with necessary dependencies

```
catkin_create_pkg my_controller roscpp
                  pluginlib controller_interface
```

(2) Write Controller code in C++

(3) Register or export the C++ class as plugin.

(4) Define plugin definition in a XML file

(5) Edit the CMakeList.txt and package.xml files for exporting the plugin

(6) Write the configuration for our controller

(7) Load the controller using the controller manager.

* Understanding the ROS visualization tool (RViz) and its plugins

(The standard method to build on ROS plugin is applicable for this plugin too)

⇒ RViz is written using a GUI framework called Qt.
