Moving the Robot

Estimated time to completion: **7 minutes**

5.7 Move a Joint programmatically

In this Section you will learn how to move a joint (in this case the laser's joint) programmatically using Python. Let's start by creating a new Python script:

▶ Execute in Terminal 3

In [ ]:
```
cd ~/ros2_ws/src
```

In [ ]:
```
mkdir -p my_box_bot_gazebo/scripts
```

In [ ]:
```
touch my_box_bot_gazebo/scripts/move_laser.py
```

In [ ]:
```
chmod +x my_box_bot_gazebo/scripts/move_laser.py
```

Here is the Python script that implements a code that will publish the joint trajectories:

move_laser.py

In [ ]:

```python
#! /usr/bin/env python3

import sys
import rclpy
from rclpy.duration import Duration
from rclpy.action import ActionClient
from rclpy.node import Node
from control_msgs.action import FollowJointTrajectory
from trajectory_msgs.msg import JointTrajectoryPoint
# ros2 action list -t
# ros2 action info /joint_trajectory_controller/follow_joint_trajectory -t
# ros2 interface show control_msgs/action/FollowJointTrajectory

class SteeringActionClient(Node):

    def __init__(self):
        super().__init__('move_laser_actionclient')
        self._action_client = ActionClient(self, FollowJointTrajectory, '/joint_trajectory_controller/follow_joint_trajectory')

    def send_goal(self, position_value):
        goal_msg = FollowJointTrajectory.Goal()

        # Fill in data for trajectory
        joint_names = ["laser_scan_link_joint"]

        points = []

        point_to_move_to = JointTrajectoryPoint()
        point_to_move_to.time_from_start = Duration(seconds=1, nanoseconds=0).to_msg()
        point_to_move_to.positions = [position_value]

        points.append(point_to_move_to)

        goal_msg.goal_time_tolerance = Duration(seconds=1, nanoseconds=0).to_msg()
        goal_msg.trajectory.joint_names = joint_names
        goal_msg.trajectory.points = points

        self._action_client.wait_for_server()
        self._send_goal_future = self._action_client.send_goal_async(goal_msg, feedback_callback=self.feedback_callback)
```

```
        self._send_goal_future.add_done_callback(self.goal_response_callback)

    def goal_response_callback(self, future):
        goal_handle = future.result()
        if not goal_handle.accepted:
            self.get_logger().info('Goal rejected :(')
            return

        self.get_logger().info('Goal accepted :)')

        self._get_result_future = goal_handle.get_result_async()
        self._get_result_future.add_done_callback(self.get_result_callback)

    def get_result_callback(self, future):
        result = future.result().result
        self.get_logger().info('Result: '+str(result))
        rclpy.shutdown()

    def feedback_callback(self, feedback_msg):
        feedback = feedback_msg.feedback



def main(args=None):

    rclpy.init()

    action_client = SteeringActionClient()

    position_value = float(sys.argv[1])
    future = action_client.send_goal(position_value)

    rclpy.spin(action_client)


if __name__ == '__main__':
    main()
```

Review some elements:

The **joint_trajectory_controller** you loaded gives you an action named **/joint_trajectory_controller/follow_joint_trajectory**.

You will use this action to send commands to the prismatic joints:

In [ ]:
```
position_value = float(sys.argv[1])
future = action_client.send_goal(position_value)
```

Using this, you will get the argument passed through the command line to send the joint to the position given within the limits described in the URDF.

In [ ]:
```
# Fill in data for trajectory
joint_names = ["laser_scan_link_joint"]

points = []

point_to_move_to = JointTrajectoryPoint()
point_to_move_to.time_from_start = Duration(seconds=1, nanoseconds=0).to_msg()
point_to_move_to.positions = [position_value]

points.append(point_to_move_to)
```

Here set the following:

- Joint names. In this case, you only have one joint controlled by this controller (laser_scan_link_joint).
- You can set many different points. In this case, you only send one point to be achieved by the joint, but you could send an array of them to each joint.

Update CMakeLists.txt files

To use this **move_laser.py**, modify the **CMakeLists.txt**. You will also add the dependencies for the **Gazebo ROS2 Control**. These dependencies are not needed to make this work, but it is **good practice** so other developers can use your package more easily.

Copy the following piece of code and overwrite the existing **my_box_bot_gazebo/CMakeLists.txt** with it:

CMakeLists.txt

In [ ]:

```
cmake_minimum_required(VERSION 3.8)
project(my_box_bot_gazebo)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(rclpy REQUIRED)
find_package(gazebo_ros REQUIRED)
find_package(my_box_bot_description REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()

install(
  DIRECTORY
    launch
    worlds
    models
  DESTINATION
    share/${PROJECT_NAME}/
)

install(PROGRAMS
  scripts/move_laser.py
  DESTINATION lib/${PROJECT_NAME}
)

ament_package()
```

Open **my_box_bot_description/CMakeLists.txt** and replace the full content with this:

CMakeLists.txt

In [ ]:

```
cmake_minimum_required(VERSION 3.8)
project(my_box_bot_description)

if(CMAKE_COMPILER_IS_GNUCXX OR CMAKE_CXX_COMPILER_ID MATCHES "Clang")
  add_compile_options(-Wall -Wextra -Wpedantic)
endif()

# find dependencies
find_package(ament_cmake REQUIRED)
find_package(urdf REQUIRED)
find_package(xacro REQUIRED)
# For control
find_package(ros2_control REQUIRED)
find_package(gazebo_ros2_control REQUIRED)
find_package(joint_state_broadcaster REQUIRED)
find_package(joint_trajectory_controller REQUIRED)
find_package(velocity_controllers REQUIRED)

if(BUILD_TESTING)
  find_package(ament_lint_auto REQUIRED)
  ament_lint_auto_find_test_dependencies()
endif()

install(
  DIRECTORY
    urdf
    rviz
    launch
    meshes
    config
  DESTINATION
    share/${PROJECT_NAME}/
)


ament_package()
```

You have also added the dependencies for the **velocity control** that you will do in the next section of this unit.

Update the `<depend>` tags inside the **my_box_bot_description/package.xml** file accordingly:

package.xml

In [ ]:
```
<?xml version="1.0"?>
<?xml-model href="http://download.ros.org/schema/package_format3.xsd" schematypens="http://www.w3.org/2001/XMLSchema"?>
<package format="3">
  <name>my_box_bot_description</name>
  <version>0.0.0</version>
  <description>TODO: Package description</description>
  <maintainer email="duckfrost@gmail.com">tgrip</maintainer>
  <license>TODO: License declaration</license>

  <buildtool_depend>ament_cmake</buildtool_depend>

  <depend>urdf</depend>
  <depend>xacro</depend>
  <depend>ros2_control</depend>
  <depend>gazebo_ros2_control</depend>
  <depend>joint_state_broadcaster</depend>
  <depend>joint_trajectory_controller</depend>
  <depend>velocity_controllers</depend>

  <test_depend>ament_lint_auto</test_depend>
  <test_depend>ament_lint_common</test_depend>

  <export>
    <build_type>ament_cmake</build_type>
  </export>
</package>
```

With the changes in place, apply them by recompiling the workspace:

▶ Execute in Terminal 3

In [ ]:
```
cd ~/ros2_ws
```

In [ ]:
```
colcon build
```

In [ ]:
```
source install/setup.bash
```

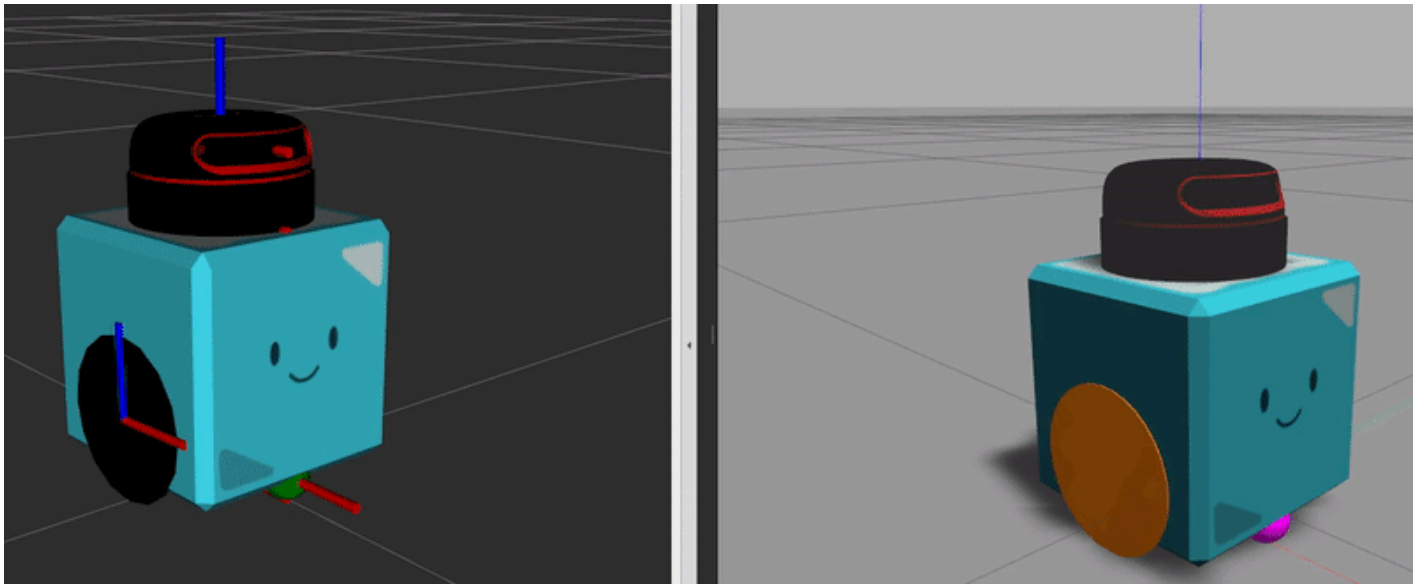
Now execute the code and see how the **laser moves up and down based on the values you send**:

Raise joint to highest position:

In [ ]:
```
ros2 run my_box_bot_gazebo move_laser.py 0.0
```

Lower to minimum position:

In [ ]:
```
ros2 run my_box_bot_gazebo move_laser.py -0.05
```



16/11/2023