Step 4: Adding a real robot into the world

Download the `Universal_Robots_ROS2_Description` Package

If you are using eceprog, the `Universal_Robots_ROS2_Description` package has already been installed, and you do not need to download it. Instead, just visit the package on github and look around the files there.

If you are not using eceprog, clone the `Universal_Robots_ROS2_Description` into your ws2/src folder and build it. This depends on which version of ROS you are using. (If you forgot, just run `ls /opt/ros/` and see if humble or jazzy is in that folder.) If you are using ROS2 Humble on Ubuntu 22.04, run the following commands to install the package:

```
rosds
git clone
https://github.com/UniversalRobots/Universal_Robots_ROS2_Description.git
git checkout origin/humble
cb
```

If you are using ROS2 Jazzy on Ubuntu 24.04, run the following command to install the package:

```
rosds
git clone
https://github.com/UniversalRobots/Universal_Robots_ROS2_Description.git
git checkout origin/jazzy
cb
```

For all students, verify that you can see the robot arm in RViz and manipulate the joints.

```
ros2 launch ur_description view_ur.launch.py ur_type:=ur3e
```

Browse through the `Universal_Robots_ROS2_Description/urdf/ur.urdf.xacro` file to see how the ur_macro.xacro is used. We will copy and paste a lot of this code in order to put the ur3e onto our table!

Task 1: Create ur3e_on_table package

```
ros2 pkg create --build-type ament_python ur3e_on_table
```

Add in the `view_ur3e_on_table.launch.py` and corresponding `ur3e_on_table.urdf.xacro` file to appropriate locations in your package, just like you did in Step 2. Build the package, then run the launch script. Take a screenshot of the robot on top of the table, with random joint positions.

Publishing to the Joint State Topic

Until now, we have used the `joint_state_publisher_gui` node to publish the joint data for our robot. Now, we will learn to create our own ROS node which publishes our own joint data.

Run the command

```
ros2 launch ur3e_on_table view_ur3e_on_table.launch.py
```

and then run

```
ros2 topic list -t
```

Observe that `/joint_states` is of type `sensor_msgs/msg/JointState`. Now run

```
ros2 topic echo --once /joint_states
```

to understand what type of message we will need to send. You should see something like this:

```
header:
  stamp:
    sec: 1720633538
    nanosec: 252970928
  frame_id: ''
name:
- shoulder_pan_joint
- shoulder_lift_joint
- elbow_joint
- wrist_1_joint
- wrist_2_joint
- wrist_3_joint
position:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
velocity: []
effort: []
---
```

Now let's add some ros2 nodes filled with Python code to publish `sensor_msgs/msg/JointState` messages to the `/joint_states` topic to move the robot. Move the `joint_publisher_test.py` and `joint_publisher_lissajous.py` files inside your `ws2/src/ur3e_on_table/ur3e_on_table` folder (just like Lab 1, Step 3). Your `package.xml` file should contain the following dependencies (simply the import statements in our python files):

```
    <exec_depend>rclpy</exec_depend>
    <exec_depend>sensor_msgs</exec_depend>
    <exec_depend>numpy</exec_depend>
```

Also copy and paste the `lissajous.csv` file into the `resource` folder. Next, modify your `setup.py` file as follows:

```
import os
from glob import glob
from setuptools import find_packages, setup

package_name = 'ur3e_on_table'
# ...

    data_files=[
        # ...
        (os.path.join('share', package_name, 'launch'), glob('launch/*')),
        (os.path.join('share', package_name, 'rviz'), glob('rviz/*')),
        (os.path.join('share', package_name, 'urdf'), glob('urdf/*')),
    ],
    # ...
    entry_points={
        'console_scripts': [
            'test = ur3e_on_table.joint_publisher_test:main',
            'lissajous = ur3e_on_table.joint_publisher_lissajous:main',
        ],
    },
```

Build your ros2 packages with `cb`. In a terminal run the test code executable:

```
ros2 run ur3e_on_table test
```

This example uses time to calculate the robot joint position:

```
msg.position = [2*np.pi*t/5, -1+0.5*np.cos(2*np.pi*t/10),
np.sin(2*np.pi*t/15), 0.0, 0.0, 0.0]
```

You should see lots of position data being printed to the terminal. In another terminal, do a `ros2 node list` and a `ros2 topic list -t` to do a sanity check and confirm your `joint_publisher_test` node is publisher to `/joint_states`. Do the same thing for the `lissajous` executable to confirm it can read data from the csv file correctly.

Task 2: Tracing the Test Curve

Write a launch file called `test_ur3e_on_table.launch.py` which visualizes the ur3e robot on a table, and uses the `test` executable from the `ur3e_on_table` package to publish the joint states. This script should be very similar to `view_ur3e_on_table.launch.py`, but you will need to modify the `joint_state_publisher_node` within the launch script so that the `test` executable runs.

Run your launch script. RViz should open, and the robot should be moving. In RViz select `RobotModel > Links > tool0 > Show Trail`.

Take a screenshot of your RViz window, with at least part of the blue curve visible.

Task 3: Tracing the Lissajous Curve

Create another launch script called `lissajous_ur3e_on_table.launch.py` which shows the ur3e arm on the table, and publishes data from the lissajous executable.

Enable the trail for `tool0`. You should see a blue lissajous curve (figure of 8) appear. Take a screenshot of your Rviz window for the lab report.

Task 4: Linear Interpolation through Joint Space

Congrats on getting this far in Lab 2! Since the start of the semester, you have learned a lot about ROS nodes/topics, URDF/XACRO files, and ROS packages. Now let's combine these all together, with some basic motion planning!

Your task is to create a new executable called `pick-and-place` within the `ur3e_on_table` package which can move between two positions in space. We will define two joint position vectors $p_1$ and $p_2$, which represent the beginning and end of the trajectory. These vectors live in $\mathbb{R}^6$ because the robot arm has 6 joints. We can interpolate between the two joint positions linearly with a parameter $0 \le a \le 1$ as follows:

$$ p = (1 - a) p_1 + a p_2 $$

where $p$ represents the current joint position. Note that when $a=0$, $p=p_1$; when $a=1$, $p=p_2$.

We want to start at $p_1$, wait 5 seconds, move to $p_2$, wait 5 seconds, and then move back to $p_1$, before repeating the process. Here is some starter code which implements this behavior:

```python
# other imports...
import numpy as np

def linear_interpolate(p1, p2, a):
    return (1-a)*p1 + a*p2
```

```python
class JointPublisherPickAndPlace(Node):

    def __init__(self):
        # other ros code ...

        # define your start/end points
        self.p1 = np.array([0.0,0.0,0.0,0.0,0.0,0.0])
        self.p2 = np.array([-1.0,-1.0,-1.0,-1.0,-1.0,-1.0])

    def timer_callback(self):

        # 0..5 seconds: stay at p1
        # 5..10 seconds: move towards p2
        # 10..15 seconds: stay at p2
        # 15..20 seconds:: move towards p1
        if self.t < 5:
            a = 0
        elif self.t < 10:
            a = (self.t - 5)/5
        elif self.t < 15:
            a = 1
        elif self.t < 20:
            a = 1 - (self.t - 15)/5

        p = linear_interpolate(self.p1, self.p2, a).tolist()

        # other code ...
        msg.position = p
        # other code ...

        # reset self.t after 20 seconds are up
        self.t += self.timer_period
        if self.t >= 20:
            self.t = 0
```

Create a new ROS node `JointPublisherPickAndPlace` and a corresponding executable called `pick-and-place`, which publishes joint positions to smoothly move between joint positions $p_1$ and $p_2$. Write a corresponding launch script which visualizes the robot arm doing this pick and place motion. Visualize the robot moving in RViz with the trail enabled for the `tool0` link. Take a screenshot for your lab report.

Free Response Questions

There are additional free-response questions for this problem. See the Overleaf template.

Push to GitHub

Push your work from Step 4 to GitHub.