ur_documentation Documentation

Release 0.1

Felix Exner

Feb 17, 2022

CONTENTS:

CHAPTER

ONE

EXAMPLES

This chapter documents the examples created in https://github.com/UniversalRobots/Universal_Robots_ROS_Tutorials.

Those examples explain certain use cases in which the ur_robot_driver could be used.

1.1 Installation

The following sections will assume that you've created a catkin_workspace and you've cloned and built the driver and examples as explained in the following.

First, create a catkin_workspace, clone the repositories and build them:

```
# create the workspace
source /opt/ros/<your_ros_version>/setup.bash
mkdir -p catkin_ws/src && cd catkin_ws

# clone all necessary repositories
git clone https://github.com/UniversalRobots/Universal_Robots_ROS_Driver.git
git clone -b calibration_devel https://github.com/fmauch/universal_robot.git
git clone https://github.com/UniversalRobots/Universal_Robots_ROS_Tutorials.git

# install dependencies
sudo apt update -qq
rosdep update
rosdep install --from-paths src --ignore-src -y

# build
catkin_make
source devel/setup.bash
```

Remember to always source your workspace (as in the last line of the code snippet above) in every new shell.

1.1.1 Docker setup

For most examples to work, you'll need **docker** and **docker-compose** installed and your user needs to be able to run docker containers. Those tools will be installed by the `rosdep` command above, but you will have to setup your user to execute docker containers accordingly. See the Docker documentation for details.

The short version:

```
sudo groupadd docker
sudo usermod -aG docker $USER
# log out and log back in
```

After that you should be able to run docker containers using your user account:

```
docker run --rm -it universalrobots/ursim_e-series
```

1.2 Dual robot workcell

Example about integrating two UR arms into one workcell. We will build a workspace description and start two drivers for the two robots.

1.2.1 ur_example_dual_robot

This demo is about integrating two robots into one URDF and starting a driver for both robots.

1.2.1.1 Requirements & Build

You'll have to have the `ur_robot_driver` setup and installed as explained in its documentation.

To build and use this package, copy it to your catkin_workspace containing the driver, install its dependencies using `rosdep install --ignore-src --from-paths . -r -y` and build your workspace as usual.

For this demo to work, you'll need **docker** and **docker-compose** installed and your user needs to be able to run docker containers. Those tools will be installed by the `rosdep` command above, but you will have to setup your user to execute docker containers accordingly. See the Docker documentation for details

1.2.1.2 Startup

You'll need two shells: One for starting two simulated robots using docker + ursim and one for the ROS components.

In the first shell execute

```
rosrun ur_example_dual_robot docker_alice_bob.sh
```

Wait, until the robots are started up. You can connect to the robots using their web interface:

- Alice: http://10.5.0.5:6080/vnc.html
- Bob: http://10.5.0.6:6080/vnc.html

When the robots have booted, start the driver instances as follows

```
roslaunch ur_example_dual_robot dual_robot_startup.launch
```

This should startup the drivers, an RViz instance and an rqt_joint_trajectory_controller window.

To steer the robots, you'll first have to start the external_control program on both using the web interface (the programs should be loaded by default, simply start the robots and press the play button). In the shell running the drivers, you should now see `Robot connected to reverse interface. Ready to receive control commands.` twice.

Using the rqt_joint_trajectory_controller window you can select one of the robots, click on the big red button and then use the sliders to move the robots around.

1.2.2 Step by step explanation of a dual-robot setup

This chapter explains all the steps necessary to create a dual-arm setup using the `ur_robot_driver`.

In order to create a multi-robot workcell we basically have to consider three aspects:

- When integrating multiple robots into one *robot_description* we face one challenge: We can't have multiple *tf* frames with the same name, e.g. `base_link` in the same *robot_description*. Therefore, we'll have to use prefixes.
- Once the robot description uses tf prefixes, the joint names will also use those prefixes. Therefore, we'll need to adapt our robot controllers that they are defined on those "modified" joint names.
- As we will be starting multiple instances of the same ROS nodes, we'll have to take care that their names (nodes, topics, services) do not collide. One easy way to achieve this is using ROS namespaces

1.2.2.1 Assembling the URDF

The ur_description package provides macro files to generate an instance of a Universal Robots arm. We'll use this to assemble a description containing a Box with a UR3e and a UR10e ontop:

Listing 1: ur_example_dual_robot/urdf/dual_robot.xacro

```
<?xml version="1.0"?>
<robot xmlns:xacro="http://wiki.ros.org/xacro" name="my_work_cell">
  <!--Load the macro for creating a robot-->
  <xacro:include filename="$(find ur_description)/urdf/inc/ur10e_macro.xacro"/>
  <xacro:include filename="$(find ur_description)/urdf/inc/ur3e_macro.xacro"/>

  <!--Instanciate the robots-->
  <xacro:ur3e_robot prefix="bob_" kinematics_parameters_file="$(arg bob_kinematics)"/>
  <xacro:ur10e_robot prefix="alice_" kinematics_parameters_file="$(arg alice_kinematics)
↪" />

  <!--common link where the tf tree originates from-->
  <link name="world"/>

  <!--Define the robot poses in the world-->
  <joint name="world_to_bob" type="fixed">
    <parent link="world" />
    <child link = "bob_base_link" />
    <origin xyz="-0.5 0 0" rpy="0 0 0" />
  </joint>
  <joint name="world_to_alice" type="fixed">
    <parent link="world" />
    <child link = "alice_base_link" />
    <origin xyz="0.5 0 0" rpy="0 0 0" />
```

(continues on next page)

(continued from previous page)

```
  </joint>
</robot>
```

Let's break it down:

First, we'll have to **include** the macros to generate the two arms:

Listing 2: ur_example_dual_robot/urdf/dual_robot.xacro

```
  <xacro:include filename="$(find ur_description)/urdf/inc/ur10e_macro.xacro"/>
  <xacro:include filename="$(find ur_description)/urdf/inc/ur3e_macro.xacro"/>
```

The two include lines only loaded the macro for generating robots. Next, we can call the macros to actually **create the arms**.

Listing 3: ur_example_dual_robot/urdf/dual_robot.xacro

```
  <xacro:ur3e_robot prefix="bob_" kinematics_parameters_file="$(arg bob_kinematics)"/>
  <xacro:ur10e_robot prefix="alice_" kinematics_parameters_file="$(arg alice_kinematics)
↪" />
```

This creates the two robots `alice` and `bob`. We choose their names as a tf-prefix in order to make the link and joint names unique. Note the trailing underscore in the prefixes, as the prefixes will be added in front of the link and joint names without adding a further underscore.

The last thing to do is **connecting** the two robots to our common `world` link by adding two fixed joints.

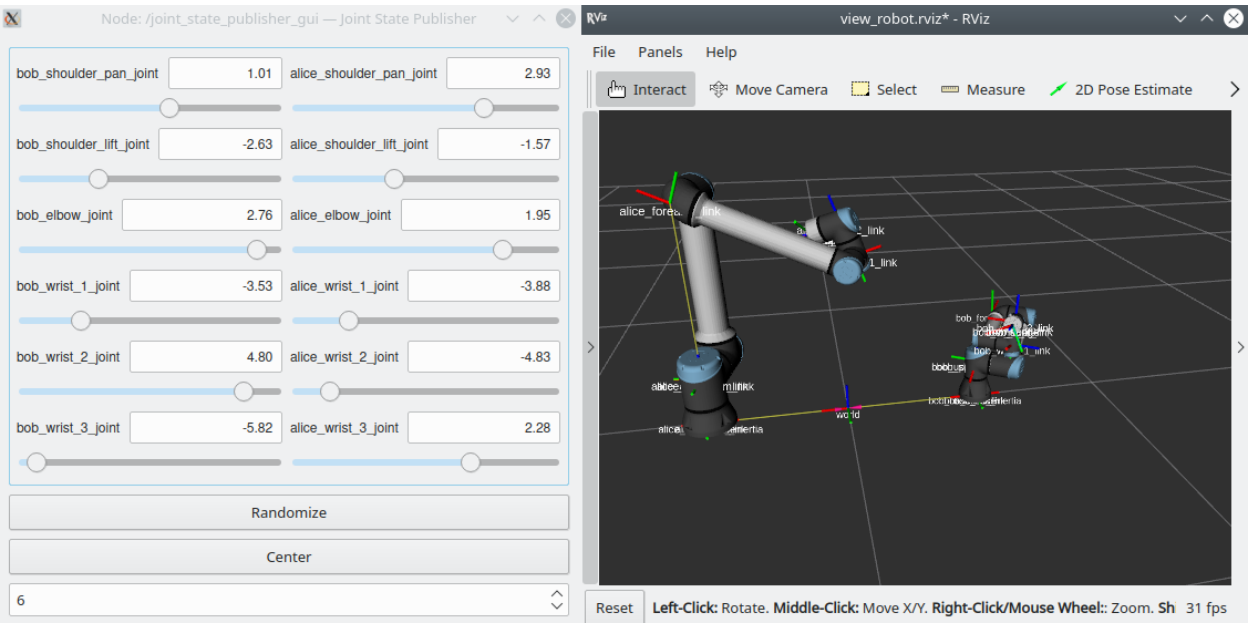
Listing 4: ur_example_dual_robot/urdf/dual_robot.xacro

```
  <joint name="world_to_bob" type="fixed">
    <parent link="world" />
    <child link = "bob_base_link" />
    <origin xyz="-0.5 0 0" rpy="0 0 0" />
  </joint>
  <joint name="world_to_alice" type="fixed">
    <parent link="world" />
    <child link = "alice_base_link" />
    <origin xyz="0.5 0 0" rpy="0 0 0" />
  </joint>
```

We can view our custom workcell by running

```
roslaunch ur_example_dual_robot view_dual_robot.launch
```

Use the sliders of the joint_state_publisher_gui to move the virtual robots around. It should look something like this:



1.2.2.2 Define controller configuration files for the two robots

In the last paragraph we created a description containing both robots. To avoid name clashes, both robots got a prefix `alice_` and `bob_` respectively.

For using the `ur_robot_driver` with this description we'll have to make sure that we load controllers containing the joint names **with** the prefixes.

To keep things simple, we'll copy the controller configuration from the `ur_robot_driver` and only rename the joints inside. The following listing shows how that could be achieved, if you cloned this repo you will already have the corresponding files.

```
cd <src-location-of-ur_example_dual_robot>
mkdir -p etc
roscp ur_robot_driver ur10e_controllers.yaml etc/alice_controllers.yaml
roscp ur_robot_driver ur3e_controllers.yaml etc/bob_controllers.yaml
```

With a text editor, we open the files and change the joint names that they contain the prefixes

Listing 5: ur_example_dual_robot/etc/alice_controllers.yaml

```
# Settings for ros_control hardware interface
ur_hardware_interface:
   joints: &robot_joints
     - alice_shoulder_pan_joint
     - alice_shoulder_lift_joint
     - alice_elbow_joint
     - alice_wrist_1_joint
     - alice_wrist_2_joint
     - alice_wrist_3_joint
```

Listing 6: ur_example_dual_robot/etc/bob_controllers.yaml

```
# Settings for ros_control hardware interface
ur_hardware_interface:
   joints: &robot_joints
     - bob_shoulder_pan_joint
     - bob_shoulder_lift_joint
     - bob_elbow_joint
     - bob_wrist_1_joint
     - bob_wrist_2_joint
     - bob_wrist_3_joint
```

1.2.2.3 Create a launchfile and start drivers for both robots

Now we have everything in place to startup two driver instances for *alice* and *bob*. The `ur_robot_driver` offers different levels of abstractions inside its launchfiles. Basically, we need the following components:

1. load the description
2. Driver for *alice*
3. Driver for *bob*
4. robot_state_publisher(s?)

First of all, loading the description can be done straight-forward:

Listing 7: ur_example_dual_robot/launch/dual_robot_startup

```
  <include file="$(find ur_example_dual_robot)/launch/load_dual_description.launch">
    <arg name="bob_kinematics" value="$(arg bob_kinematics)"/>
    <arg name="alice_kinematics" value="$(arg alice_kinematics)"/>
  </include>
```

If you want, ignore the two arguments passed to this launchfile for now. See *Bonus: Use correct robot calibration with dual_robot setup* for details.

To start the drivers we again need to watch out for name clashes. Both drivers start the same controllers which are using the same topics to communicate. To avoid clashes, we will start each driver in a separate namespace. We use the `ur_control.launch` launchfile from `ur_robot_driver` for that:

Listing 8: ur_example_dual_robot/launch/dual_robot_startup

```
  <group ns="alice">
    <node name="robot_state_publisher" pkg="robot_state_publisher" type="robot_state_
↪publisher"/>

    <include file="$(find ur_robot_driver)/launch/ur_control.launch">
      <arg name="use_tool_communication" value="$(arg use_tool_communication)"/>
      <arg name="controller_config_file" value="$(arg alice_controller_config_file)"/>
      <arg name="robot_ip" value="$(arg alice_ip)"/>
      <arg name="reverse_port" value="$(arg alice_reverse_port)"/>
      <arg name="script_sender_port" value="$(arg alice_script_sender_port)"/>
      <arg name="trajectory_port" value="$(arg alice_trajectory_port)"/>
      <arg name="kinematics_config" value="$(arg alice_kinematics)"/>
```

(continues on next page)

(continued from previous page)

```
      <arg name="tf_prefix" value="alice_"/>
      <arg name="controllers" value="$(arg controllers)"/>
      <arg name="stopped_controllers" value="$(arg stopped_controllers)"/>
    </include>
  </group>
```

With this, each robot will have its `joint_state_controller` running inside its namespace, meaning that the `joint_states` topic will be inside the respecive namespaces, namely `/alice/joint_states` and `/bob/joint_states`. Therefore, we start a `robot_state_publisher` (that will convert `joint_states` messages into TF messages to produce up-to-date poses of each link). This could also be done differently, e.g. by having one `robot_state_publisher` in the top-level namespace and adding a joint_state_publisher that collects the two topics from their namespaces into `/joint_states`. If you wish to have a `/joint_states` topic, you might want to take that route.

1.2.2.4 Run the demo

Now, we've got everyting together to actually run the full demo. For running the demo you will need a working Docker setup as explained in the *Installation*.

For startup you'll need two shells: One for starting two simulated robots using docker + ursim and one for the ROS components.

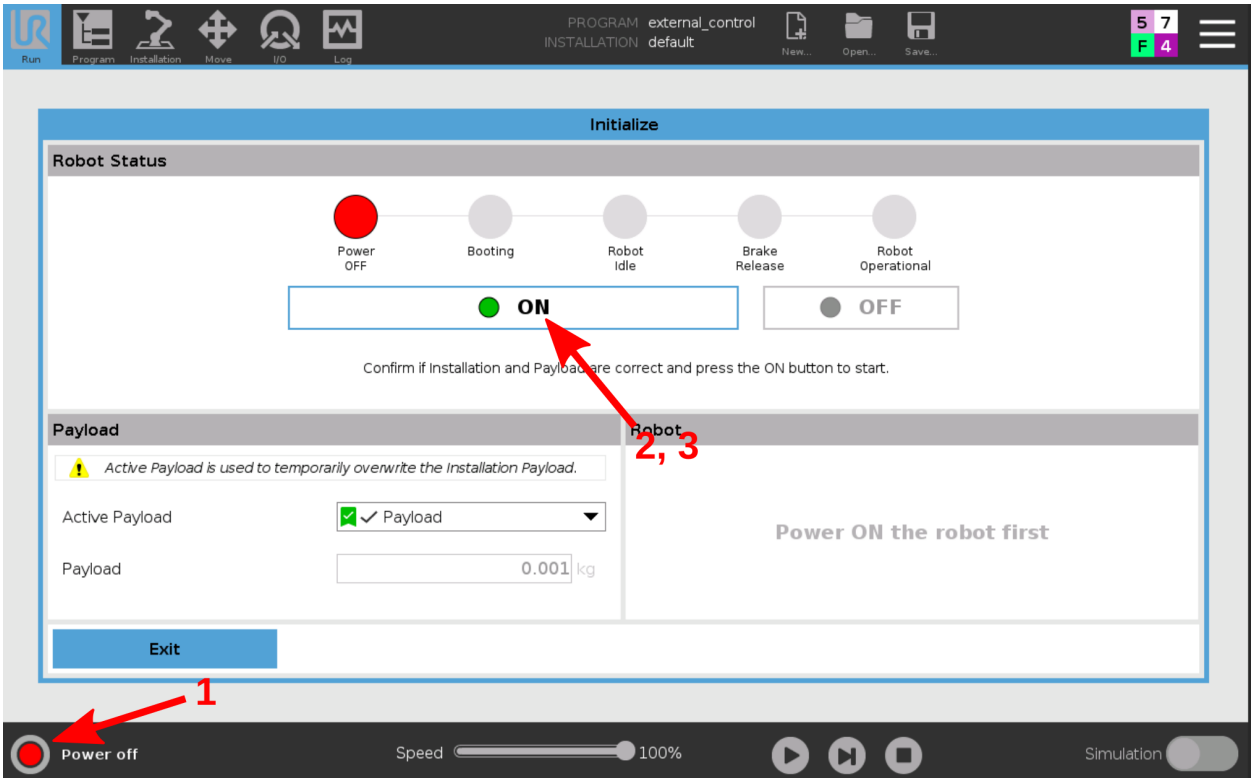
In the first shell execute

```
rosrun ur_example_dual_robot docker_alice_bob.sh
```

This will use docker-compose to start two docker containers running a simulated robot. Wait, until the robots are started up. You can connect to the robots using their web interface:

- Alice: http://10.5.0.5:6080/vnc.html
- Bob: http://10.5.0.6:6080/vnc.html

Once the robots have booted, start them as you would with a normal robot using the red button in the lower left corner:



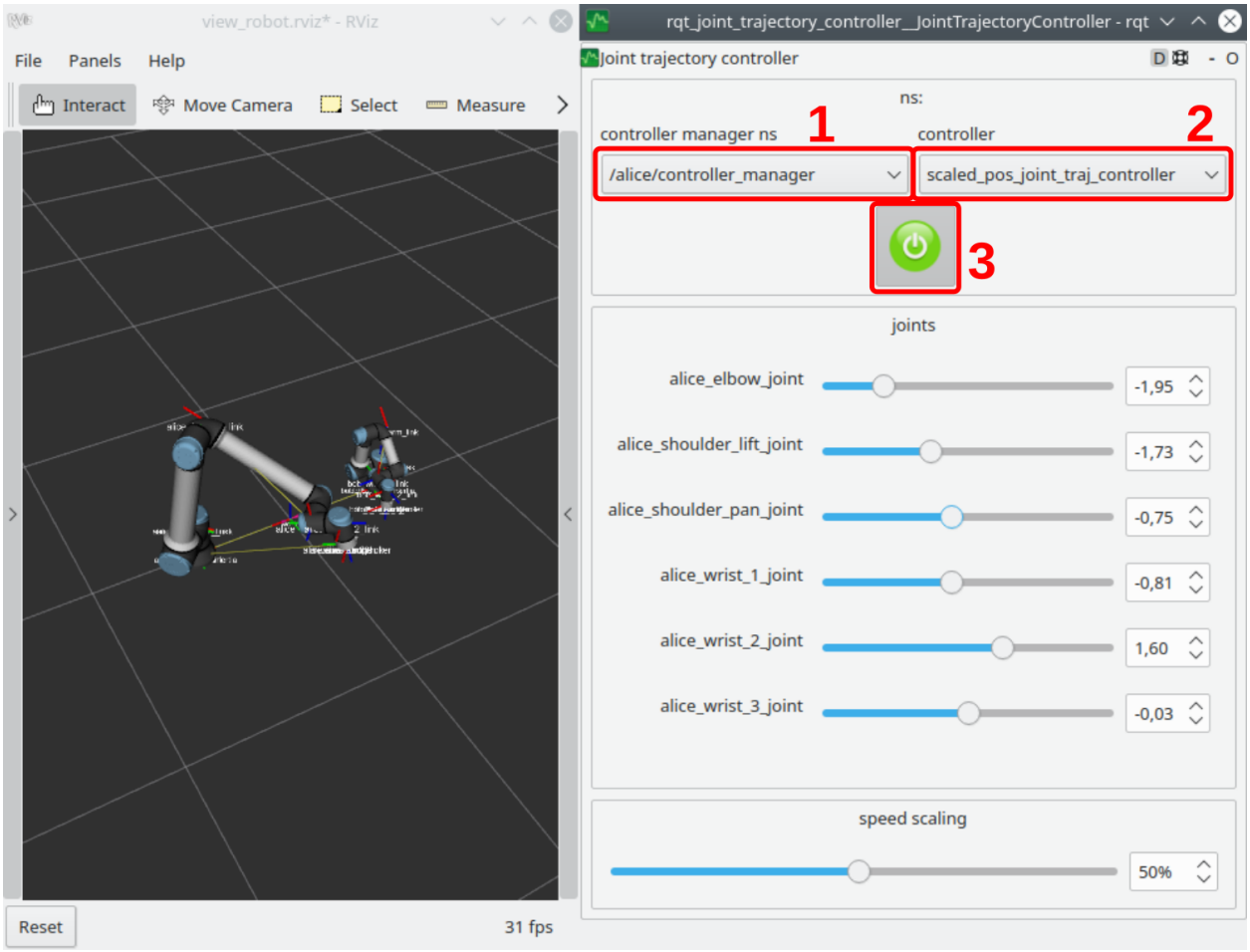
When the robots have booted, start the driver instances as follows

```
roslaunch ur_example_dual_robot dual_robot_startup.launch
```

This should startup the drivers, an RViz instance and an rqt_joint_trajectory_controller window.

The last thing we need to do is to start the `external_control` program on the robots. They have the respective URCap already installed and a program created and loaded. Simply press the play button on each robot to start external control from ROS. In the shell running the drivers, you should now see `Robot connected to reverse interface. Ready to receive control commands.` twice.

Using the rqt_joint_trajectory_controller window you can select one of the robots (1), select the controller to use (2, it should just be one) click on the big red button (3, it will turn green as shown in the image) and then use the sliders to move the robots around.



1.2.2.5 Bonus: Use correct robot calibration with dual_robot setup