Making a robot ROS 2 powered
a case study using the UR manipulators

ROS World 2021 - Day 2: Oct, 21st 2021

Denis Štojl, Nathan Brooks, Lovro Ivanov, Andy Zelenak - PickNik Robotics
[ denis.stogl, nathan, lovro.ivanov, zelenak ] @picknik.ai
Rune Søe-Knudsen - Universal Robots
rsk@universal-robots.com
Outline

- Requirements on control software
- Support libraries in ROS/ROS2
- Hardware abstraction
- Planning and collision-avoidance with a manipulator
- What should I do if my robot has multiple control-modes?
- Handling of “generic” interfaces
- Using custom controllers

Requirements from Control Software

- **Robot movement**
  - Time-synchronized joint movements
  - Executing trajectories with time and spatial constraints
  - Support for different control modes, e.g., position, velocity

- **Feedback from integrated sensors**
  - Joint States
  - E.g., Force Torque Sensor (FTS)

- **Digital and Analog Inputs and Output**
  - Reading and controlling

- **Status feedback and general operation:**
  - Robot and Safety mode
  - Status: brakes, power
  - Program execution control
Universal Robots - Manipulators

- Movement control
  - Commands: position, velocity
  - States: position, velocity, effort
  - Cartesian: TCP position/velocities
- Sensors:
  - TCP Force Torque Sensor (FTS)
- I/O control
  - Analog IOs
  - Digital IOs
- Tool:
  - Output voltage and current
  - Analog Inputs
Universal Robots - Manipulators

- Status and General Operation:
  - Robot mode
  - Safety mode
  - General Operation: State
  - Teach pendant: “speed scaling”
  - Control:
    - Unlock protective stop
    - Restart Safety
    - Power on
    - Power off
    - Break release
    - Stop program
    - Play program
What to do in ROS/ROS2?

- **ros(2)_control**
  - control framework for controlling physical robots
  - set of standard controllers
  - hardware-agnostic

- **MoveIt(2)**
  - motion and manipulation planning library
  - environment modelling and collision avoidance
  - controlling robot with a joystick — Servoing
  - hardware-agnostic
What to do in ROS2?
Enabling a robot for ros2_control

URDF-description for ros2_control / Implementing hardware interface / Attaching standard controllers
URDF-description for ros2_control

```xml
<ros2_control name="ur_robot" type="system">
  <hardware>
    <plugin>ur_robot_driver/URPositionHardwareInterface</plugin>
    <param name="robot_ip">@[robot_ip]</param>
    <param name="robot_ip">@[robot_ip]</param>
    <param name="script_filename">@[script_filename]</param>
    <param name="output_recipe_filename">@[output_recipe_filename]</param>
    <param name="input_recipe_filename">@[input_recipe_filename]</param>
    <param name="headless_mode">false</param>
    <param name="reverse_port">5000</param>
    <param name="script_sender_port">5000</param>
    <param name="tf_prefix">@[tf_prefix]</param>
    <param name="num_blocking_read">5</param>
    <param name="servlet_gain">42000</param>
    <param name="servo_lkthead_time">5.93</param>
    <param name="use_tool_communication">false</param>
    <param name="kinematics_hash">@[kinematics_hash]</param>
    <param name="tool_voltage">@[tool_voltage]</param>
    <param name="tool_parity">true</param>
    <param name="tool_baud_rate">115200</param>
    <param name="tool_stop_bits">1</param>
    <param name="tool_rx_idle_chars">1</param>
    <param name="tool_tx_idle_chars">1</param>
    <param name="tool_device_name">@[tool_device_name]</param>
    <param name="tool_top_port">54321</param>
  </hardware>
  <command_interface name="@[prefix]shoulder_pan_joint">
    <param name="min">-2.0</param>
    <param name="max">2.0</param>
  </command_interface>
  <command_interface name="@[prefix]wrist_3_joint">
    <param name="min">-2.0</param>
    <param name="max">2.0</param>
  </command_interface>
</ros2_control>
```
## Implementing hardware interface (driver)

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>export_state_interfaces()</code></td>
<td>- Which states are available from HW?</td>
</tr>
<tr>
<td><code>export_command_interfaces()</code></td>
<td>- What can be commanded on HW?</td>
</tr>
<tr>
<td><code>on_init()</code></td>
<td>- read and process URDF parameters - initialize all variables and containers</td>
</tr>
<tr>
<td><code>on_activate (previous_state)</code></td>
<td>- activate power of HW to enable movement</td>
</tr>
<tr>
<td><code>on_deactivate (previous_state)</code></td>
<td>- disable HW movement</td>
</tr>
<tr>
<td><code>on_cleanup (previous_state)</code></td>
<td>- disable communication</td>
</tr>
<tr>
<td><code>on_configure (previous_state)</code></td>
<td>- initiate communication with the HW - be sure HW states can be read</td>
</tr>
<tr>
<td><code>on_error (previous_state)</code></td>
<td>- process and mitigate any errors - it can happen in any state - catching errors during read/write</td>
</tr>
<tr>
<td><code>on_shutdown (previous_state)</code></td>
<td>- initiate HW shutdown sequence - can be called from any state</td>
</tr>
</tbody>
</table>

https://design.ros2.org/articles/node_lifecycle.html
Configuring standard controllers

```yaml
controller_manager:
  update_rate: 500 # Hz

joint_state_broadcaster:
  type: joint_state_broadcaster/JointStateBroadcaster

force_torque_sensor_broadcaster:
  type: force_torque_sensor_broadcaster/ForceTorqueStateBroadcaster

joint_trajectory_controller:
  type: joint_trajectory_controller/JointTrajectoryController

forward_position_controller:
  type: position_controllers/JointGroupPositionController

force_torque_sensor_broadcaster:
  sensor_name: tcp_fts_sensor
  frame_id: tool0
  topic_name: ft_data

joint_trajectory_controller:
  joints:
    - shoulder_pan_joint
    - ...
    - wrist_3_joint
  command_interfaces:
    - position
  state_interfaces:
    - position
    - velocity

forward_position_controller:
  joints:
    - shoulder_pan_joint
    - ...
    - wrist_3_joint
```
Planning and collision avoidance with MoveIt 2
Creating configuration files for MoveIt 2 - details

- Beside URDF file of the robot, MoveIt2 additional configuration files
- Those files are usually placed in a separate package, e.g., "<robot>_moveit_config"

- "<robot>.srdf" — semantic robot description format
  - Planning groups, links and joints
  - End effector, virtual-joints
  - Pre-defined states (positions)

- "kinematics.yaml" — definition/configuration of kinematics plugin (IK and FK)

```
ur_manipulator:
  kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
  kinematics_solver_attempts: 3
```
Creating configuration files for MoveIt 2 - details

- "ompl_planning.yaml" — parameters for motion planning
- "servo.yaml" — configuration for MoveIt2-Servo
- "controllers.yaml" — controller definition used by MoveIt2

```yaml
controller_names:
  - joint_trajectory_controller

joint_trajectory_controller:
  action_ns: follow_joint_trajectory
  type: FollowJointTrajectory
  default: true
  joints:
    - shoulder_pan_joint
    - shoulder_lift_joint
    - elbow_joint
    - wrist_1_joint
    - wrist_2_joint
    - wrist_3_joint
```
Creating configuration files for MoveIt 2 - details

- Start “move_group” node with:
  - “kinematics.yaml”
  - “ompl_planing.yaml” → request adapters
  - configuration for “moveit_controller_manager” and “controllers.yaml”
  - configuration for trajectory execution and planning scene monitor

- Example resources:
  - moveit_resources:
    https://github.com/ros-planning/moveit_resources/tree/ros2
  - UR ROS2 driver:
  - ur_moveit.launch.py:
What should I do if my robot has multiple control-modes?
# Using different controllers for control modes

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>export_state_interfaces()</code></td>
<td>- Which states are available from HW?</td>
</tr>
<tr>
<td><code>export_command_interfaces()</code></td>
<td>- What can be commanded on HW?</td>
</tr>
<tr>
<td><code>on_init()</code></td>
<td>- read and process URDF parameters</td>
</tr>
<tr>
<td></td>
<td>- initialize all variables and containers</td>
</tr>
<tr>
<td><code>on_activate (previous_state)</code></td>
<td>- activate power of HW to enable movement</td>
</tr>
<tr>
<td><code>read()</code></td>
<td>- Fill states from HW readings</td>
</tr>
<tr>
<td><code>write()</code></td>
<td>- Write commands to HW</td>
</tr>
<tr>
<td><code>on_configure (previous_state)</code></td>
<td>- initiate communication with the HW</td>
</tr>
<tr>
<td><code>prepare_command_mode_switch (stop_interfaces, start_interfaces)</code></td>
<td>- Check if mode switch is possible w.r.t. given interfaces</td>
</tr>
<tr>
<td></td>
<td>- Only command interfaces are relevant</td>
</tr>
<tr>
<td></td>
<td>- Prepare robot for switching (initialize additional variables, etc.)</td>
</tr>
<tr>
<td><code>perform_command_mode_switch (stop_interfaces, start_interfaces)</code></td>
<td>- perform switching of the hardware</td>
</tr>
<tr>
<td></td>
<td>- set/reset internal variables for new/old control mode</td>
</tr>
<tr>
<td><code>on_shutdown (previous_state)</code></td>
<td>- initiate HW shutdown sequence</td>
</tr>
<tr>
<td></td>
<td>- can be called from any state</td>
</tr>
</tbody>
</table>
Add controllers for other control-mode

- Forwarding controller
- Joint Trajectory controller with different set of command interfaces
I want to control digital and analog IOs of my robot. Is this possible?
Handling of “generic” interfaces

- Using `<gpio>` tag for non-movement interfaces

- (Optional: using semantic components to simplify their use) — check the talk on ros2_control

```xml
<sensor name="tcp_fts_sensor">
  <state_interface name="force_x"/>
  <state_interface name="force_y"/>
  <state_interface name="force_z"/>
  <state_interface name="torque_x"/>
  <state_interface name="torque_y"/>
  <state_interface name="torque_z"/>
</sensor>

<gpio name="speed_scaling">
  <state_interface name="speed_scaling_factor"/>
  <param name="initial_speed_scaling_factor"/>
  <command_interface name="target_speed_fraction_cmd"/>
  <param name="async_handshake">async_success</param>
  <command_interface name="async_success"/>
</gpio>

<gpio name="flange_ids">
  <param name="async_handshake">async_success</param>
  <command_interface name="async_success"/>
  <command_interface name="digital_output" data_type="bool" size="18"/>
  <state_interface name="digital_output" data_type="bool" size="18"/>
  <state_interface name="analog_output" data_type="double" size="2"/>
  <state_interface name="analog_output" data_type="double" size="2"/>
  <state_interface name="analog_output" data_type="int" size="4"/>
</gpio>

<gpio name="tool">
  <state_interface name="mode"/>
  <state_interface name="output_voltage"/>
  <state_interface name="output_current"/>
  <state_interface name="temperature"/>
  <state_interface name="analog_input" data_type="double" size="2"/>
  <state_interface name="analog_input" data_type="double" size="2"/>
  <state_interface name="analog_input" data_type="int" size="2"/>
</gpio>

<gpio name="robot_status">
  <state_interface name="mode" data_type="int"/>
  <state_interface name="bit" data_type="bool" size="4"/>
</gpio>

<gpio name="safety_mode">
  <state_interface name="mode" data_type="int"/>
  <state_interface name="bit" data_type="bool" size="12"/>
</joint>
</ros2_control>
```
Creating a custom controller for my robot
Implementing a controller for ros2_control

- Implement “Controller Interface”-Class

<table>
<thead>
<tr>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>command_interface_configuration()</td>
<td>- Which command interfaces needs controller?</td>
</tr>
<tr>
<td>state_interface_configuration()</td>
<td>- Which state interfaces needs controller?</td>
</tr>
<tr>
<td>on_init()</td>
<td>- initialize all variables and containers</td>
</tr>
<tr>
<td></td>
<td>- declare parameters to default values</td>
</tr>
<tr>
<td>on_configure (previous_state)</td>
<td>- read parameters from parameter server</td>
</tr>
<tr>
<td></td>
<td>- setup controller according to parameters</td>
</tr>
<tr>
<td></td>
<td>- prepare controller for activation</td>
</tr>
<tr>
<td>on_activate (previous_state)</td>
<td>- set/reset commands to default values</td>
</tr>
<tr>
<td></td>
<td>- activate ROS2 interfaces (pubs, subs, srvs, actions)</td>
</tr>
<tr>
<td></td>
<td>- order assigned interfaces for simple access</td>
</tr>
<tr>
<td>on_deactivate (previous_state)</td>
<td>- clear variables</td>
</tr>
<tr>
<td></td>
<td>- deactivate ROS2 interfaces (pubs, subs, srvs, actions)</td>
</tr>
<tr>
<td>update (time, period)</td>
<td>- controller's &quot;update&quot; loop</td>
</tr>
<tr>
<td></td>
<td>- write commands based on states and/or inputs</td>
</tr>
</tbody>
</table>
Custom Controller for IOs and Status

- **Publishers:**
  - IO states
  - Tool data
  - Robot Mode
  - Safety Mode
  - Speed Scaling
  - Robot Program Status

- **Services:**
  - Set IO
  - Set Speed Slider (speed scaling)
  - Set Payload
  - Tare FTS Sensor
Velocity-scaling controller

- Extending standard Joint Trajectory Controller to support speed scaling
- Adapting the commanded Joint Trajectory with speed scale
- Speed slider can be controlled from teach pendant and from ROS2 side
And now the conclusion...
UR ROS2 Driver Capabilities

- Multi-command interface support
  - Switching between control modes
- Force-Torque Sensor access
- Digital and Analog IO control
- Access to robot’s status flags
- Readout and apply factory-kinematic calibration
- MoveIt2 and MoveIt2-Servo integration

Simple testing using “URSim” — Docker container → Execution testing in CI!
- URSim container: https://hub.docker.com/u/universalrobots
Contributions to ros2_control

- **Joint Trajectory Controller Extensions**
  - Velocity command support
  - Constraint propagation

- **Speed Scale Joint Trajectory Controller**
  - With hardware-feedback integration

- **Development of concepts**
  - IO control
  - Robot Status

- **Future influence of ros2_control-framework**
  - TCP — Pose Broadcaster
  - Cartesian space controllers
  - Generic Robot Status Broadcaster
Thank you for your attention!

User feedback and suggestion for improvements are very welcome!

ros@universal-robots.com / or open an issue at GitHub
Bonus: Setting up CI for a robot driver

- Doing proper testing, especially execution on simulated hardware is nontrivial :)

- 3-stage build CI (does not run tests/tests with hardware)
  - Enables different levels of “failure-anticipation” from upstream packages
  - Binary: all dependencies from binaries (exec not-yet-released) — industrial_ci
    - on: PR and merge
  - Semi-Binary: the main dependencies are built from source — industrial_ci
    - on: PR and merge
  - Source: also core ROS2 packages are built from source — ros-tooling/action-ros-ci@v0.2
    - scheduled (because it takes long time)

- Execution Tests
  - Enables check of driver and controllers execution
  - Run tests with simulated robot URSim
    - at least 2 workers — ros2_control and URSim
    - scheduled (because it needs “free” workers to get proper results)

- Format + ROS2 Lint: on PR
  - Keeps your code well formatted and clean