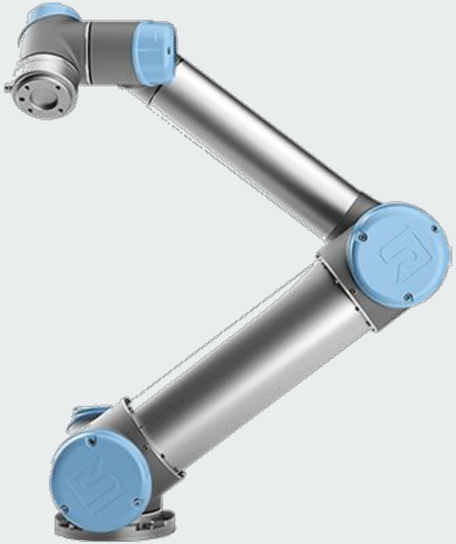
Making a robot ROS 2 powered

a case study using the UR manipulators



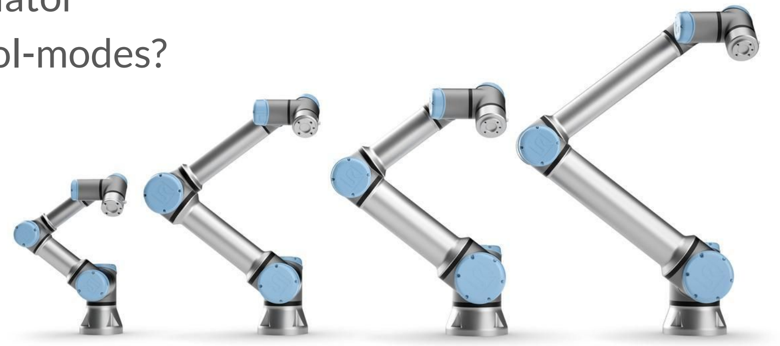
ROS World 2021 - Day 2: Oct, 21st 2021

Denis Štogl, 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

PICKNIK

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



Repository: https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver

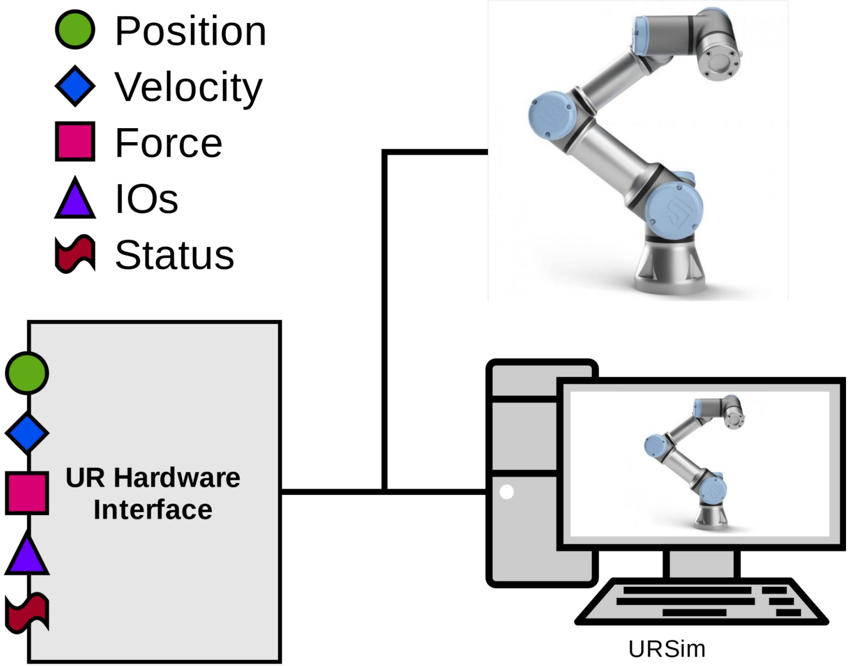
PICKNIK

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

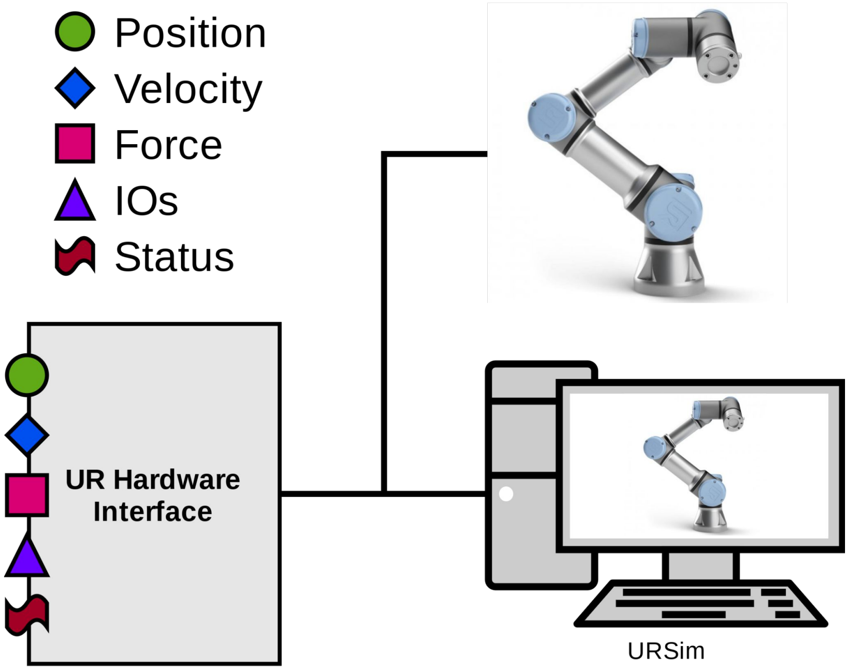
- Movement control
    - Commands: position, velocity
    - States: position, velocity, effort
    - Cartesian: TCP position/velocities
- Sensors:
    - TCP Force Torque Sensor (FTS)
- I/O control
    - Analog IOs
    - Digital IOs
- Tool:
    - Output voltage and current
    - Analog Inputs



PICKNIK

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



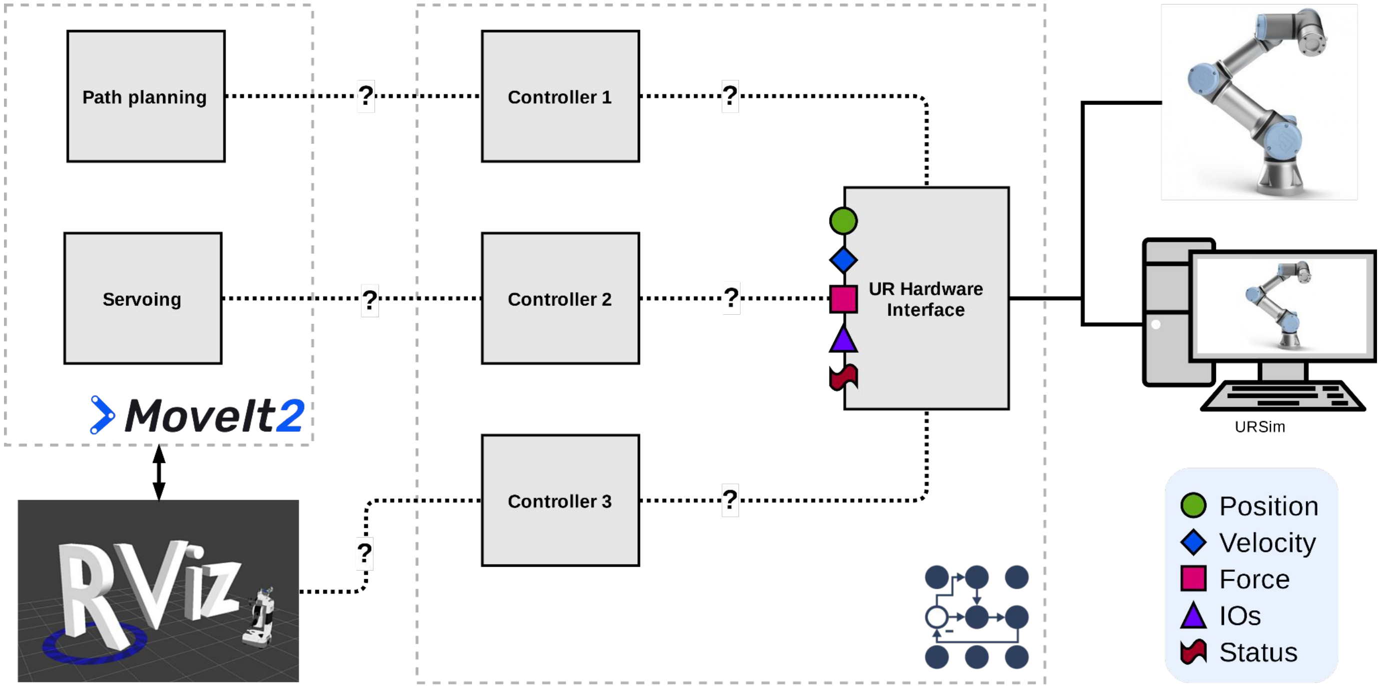
PICKNIK

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?



PICKNIK

Enabling a robot for ros2_control

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

PICKNIK

URDF-description for ros2_control

```
<ros2_control name="ur_robot" type="system">
  <hardware>
    <plugin>ur_robot_driver/URPositionHardwareInterface</plugin>
    <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">0</param>
    <param name="reverse_port">50001</param>
    <param name="script_sender_port">50002</param>
    <param name="tf_prefix">"${tf_prefix}"</param>
    <param name="non_blocking_read">0</param>
    <param name="servoj_gain">2000</param>
    <param name="servoj_lookahead_time">0.03</param>
    <param name="use_tool_communication">0</param>
    <param name="kinematics/hash">"${hash_kinematics}"</param>
    <param name="tool_voltage">0</param>
    <param name="tool_parity">0</param>
    <param name="tool_baud_rate">115200</param>
    <param name="tool_stop_bits">1</param>
    <param name="tool_rx_idle_chars">1.5</param>
    <param name="tool_tx_idle_chars">3.5</param>
    <param name="tool_device_name">/tmp/ttyUR</param>
    <param name="tool_tcp_port">54321</param>
  </hardware>
  <joint name="${prefix}shoulder_pan_joint">
    <command_interface name="position">
      <param name="min">{-2*pi}</param>
      <param name="max">{2*pi}</param>
    </command_interface>
    <command_interface name="velocity">
      <param name="min">-3.15</param>
      <param name="max">3.15</param>
    </command_interface>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="effort"/>
  </joint>
  . . .
  <joint name="${prefix}wrist_3_joint">
    <command_interface name="position">
      <param name="min">{-2*pi}</param>
      <param name="max">{2*pi}</param>
    </command_interface>
    <command_interface name="velocity">
      <param name="min">-3.2</param>
      <param name="max">3.2</param>
    </command_interface>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="effort"/>
  </joint>

  <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>

</ros2_control>
```

Implementing hardware interface (driver)

export_state_interfaces()
- Which states are available from HW?

export_command_interfaces()
- What can be commanded on HW?

on_init()
- read and process URDF parameters
- initialize all variables and containers

on_activate (previous_state)
- activate power of HW to enable movement

read()
- Fill states from HW readings

write()
- Write commands to HW

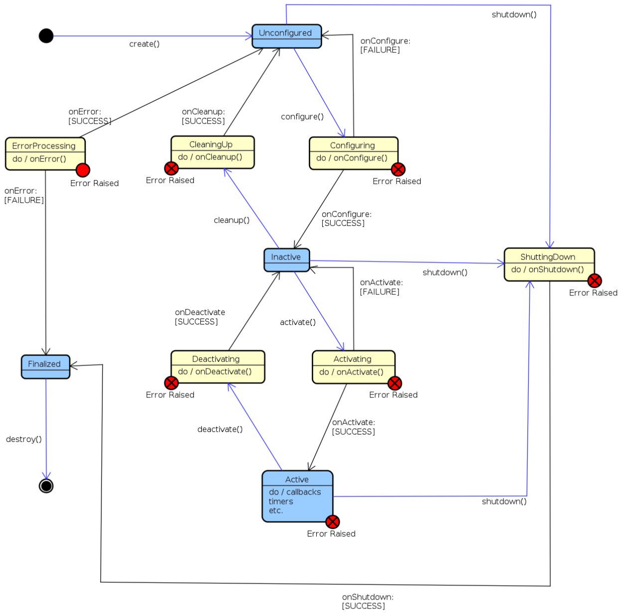
on_configure (previous_state)
- initiate communication with the HW
- be sure HW states can be read

on_deactivate (previous_state)
- disable HW movement

on_cleanup (previous_state)
- disable communication

on_error (previous_state)
- process and mitigate any errors
- it can happen in any state
- catching errors during read/write

on_shutdown (previous_state)
- initiate HW shutdown sequence
- can be called from any state

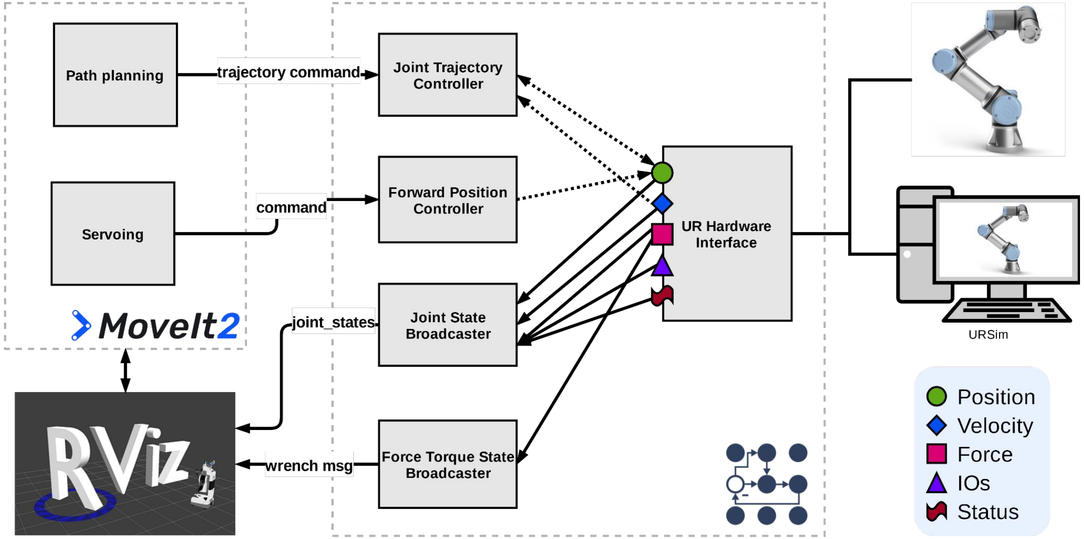


https://design.ros2.org/articles/node_lifecycle.html

PICKNIK

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
```



PICKNIK

Planning and collision avoidance with MoveIt 2

PICKNIK

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)

```yaml
ur_manipulator:
  kinematics_solver: kdl_kinematics_plugin/KDLKinematicsPlugin
  kinematics_solver_search_resolution: 0.005
  kinematics_solver_timeout: 0.005
  kinematics_solver_attempts: 3
```

PICKNIK

Creating configuration files for MoveIt 2 - details

- “ompl_planing.yaml” — parameters for motion planning
- “servo.yaml” — configuration for MoveIt2-Servo

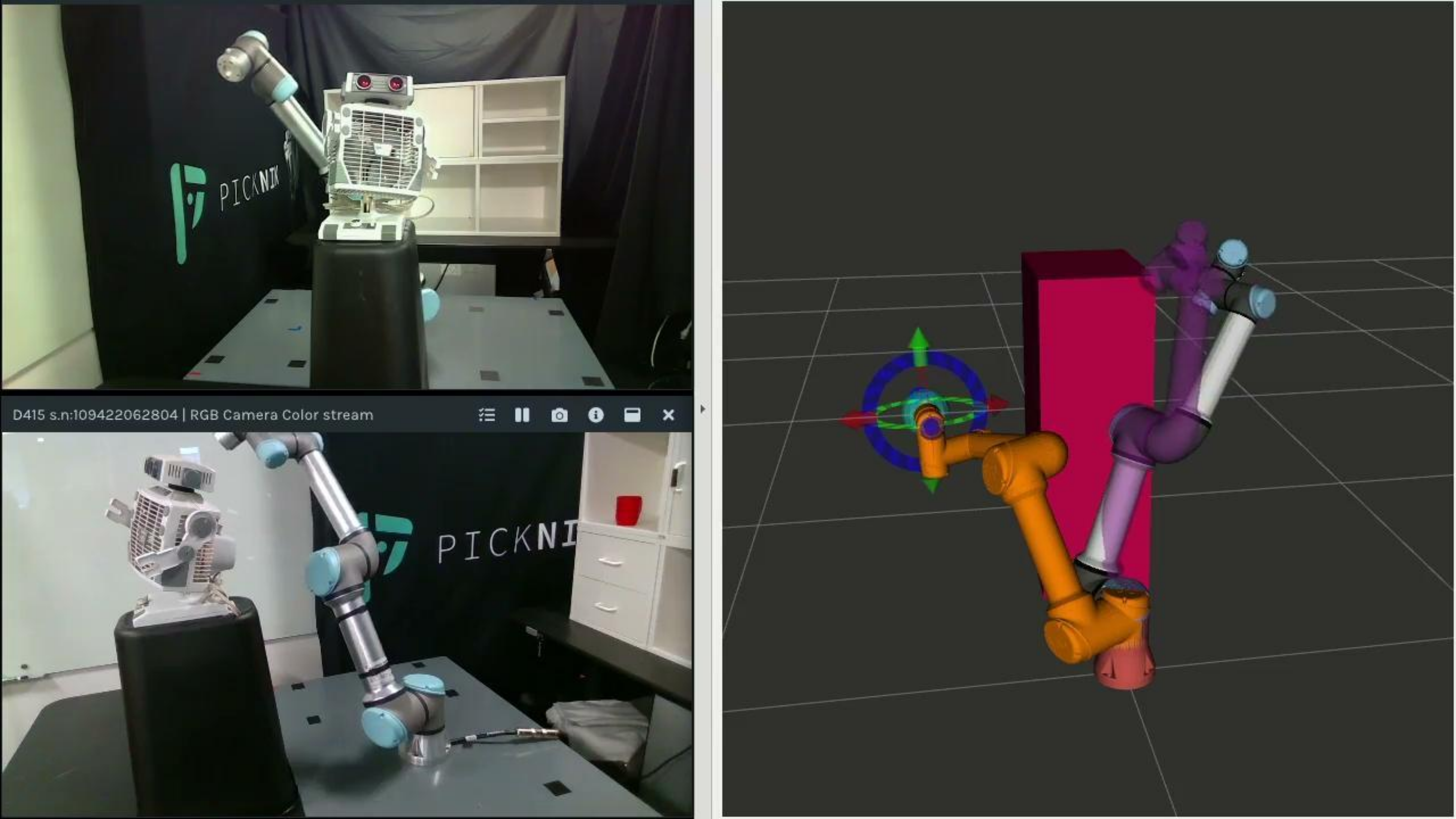
- “controllers.yaml” — controller definition used by MoveIt2

```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:
    https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver/tree/main/ur_moveit_config
  - ur_moveit.launch.py:
    https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver/blob/main/ur_bringup/launch/ur_moveit.launch.py


D415 s.n:109422062804 | RGB Camera Color stream

What should I do if my robot has multiple control-modes?

Using different controllers for control modes

export_state_interfaces()
- Which states are available from HW?

export_command_interfaces()
- What can be commanded on HW?

on_init()
- read and process URDF parameters
- initiallize all variables and containers

on_activate (previous_state)
- activate power of HW to enable movement

read()
- Fill states from HW readings

write()
- Write commands to HW

on_configure (previous_state)
- initiate communication with the HW

prepare_command_mode_switch (stop_interfaces, start_interfaces)
- Check if mode switch is possible w.r.t. given interfaces
- Only command interfaces are relevant
- Prepare robot for switching (initialize additional variables, etc.)

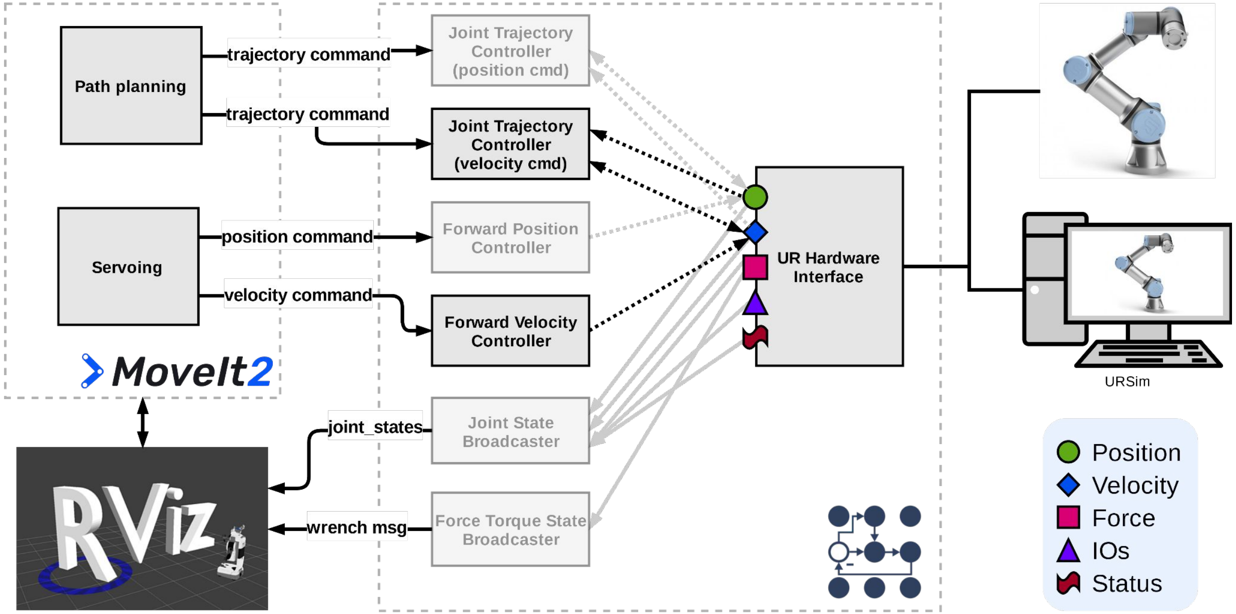
perform_command_mode_switch (stop_interfaces, start_interfaces)
- perform switching of the hardware
- set/reset internal variables for new/old control mode

on_shutdown (previous_state)
- initiate HW shoutdown sequence
- can be called from any state

PICKNIK

Add controllers for other control-mode

- Forwarding controller
- Joint Trajectory controller with different set of command interfaces



PICKNIK

I want to control digital and analog IOs of my robot. Is this possible?

PICKNIK

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 😀

```
<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">1</param>
  <command_interface name="target_speed_fraction_cmd"/>
  <param name="async_handshake">async_success</param>
  <command_interface name="async_success"/>
</gpio>

<gpio name="flange_IOs">
  <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" />
  <command_interface name="analog_output" data_type="double" size="2" />
  <state_interface name="analog_output" data_type="double" size="2" />

  <state_interface name="digital_input" data_type="bool" size="18" />
  <state_interface name="analog_input" data_type="double" size="2" />

  <state_interface name="analog_io_type" 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_type" 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="11"/>
</joint>

</ros2_control>
```

PICKNIK

Creating a custom controller for my robot

PICKNIK

Implementing a controller for ros2_control

- Implement “Controller Interface”-Class

command_interface_configuration()
- Which command interfaces needs controller?

state_interface_configuration()
- Which state interfaces needs controller?

on_init()
- initiallize all variables and containers
- declare parameters to default values

on_configure (previous_state)
- read parameters from parameter server
- setup controller according to parameters
- prepare controller for activation

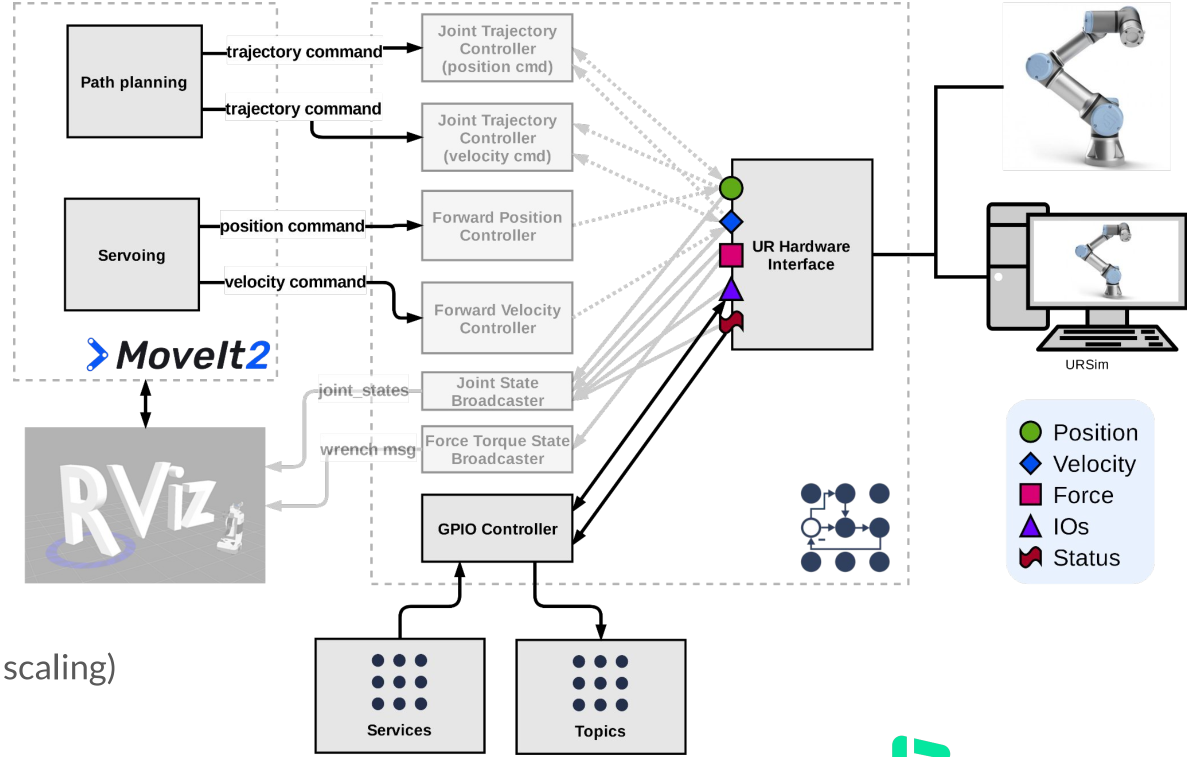
on_activate (previous_state)
- set/reset commands to default values
- activate ROS2 interfaces (pubs, subs, srvs, actions)
- order assigned interfaces for simple access

on_deactivate (previous_state)
- clear variables
- deactivate ROS2 interfaces (pubs, subs, srvs, actions)

update (time, period)
- controller's "update" loop
- write commands based on states and/or inputs

Custom Controller for IOs and Status

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

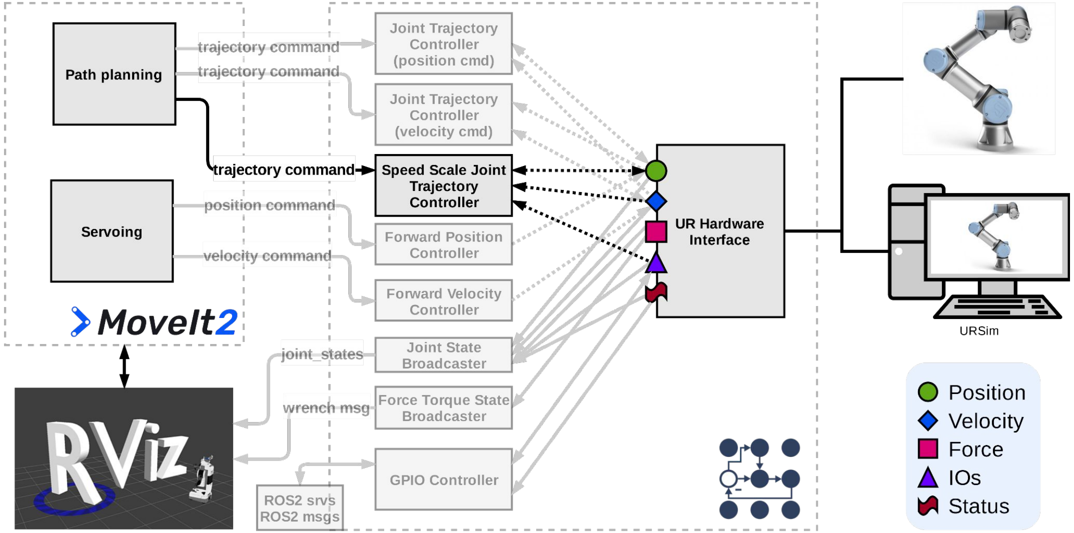


- Services:
  - Set IO
  - Set Speed Slider (speed scaling)
  - Set Payload
  - Tare FTS Sensor

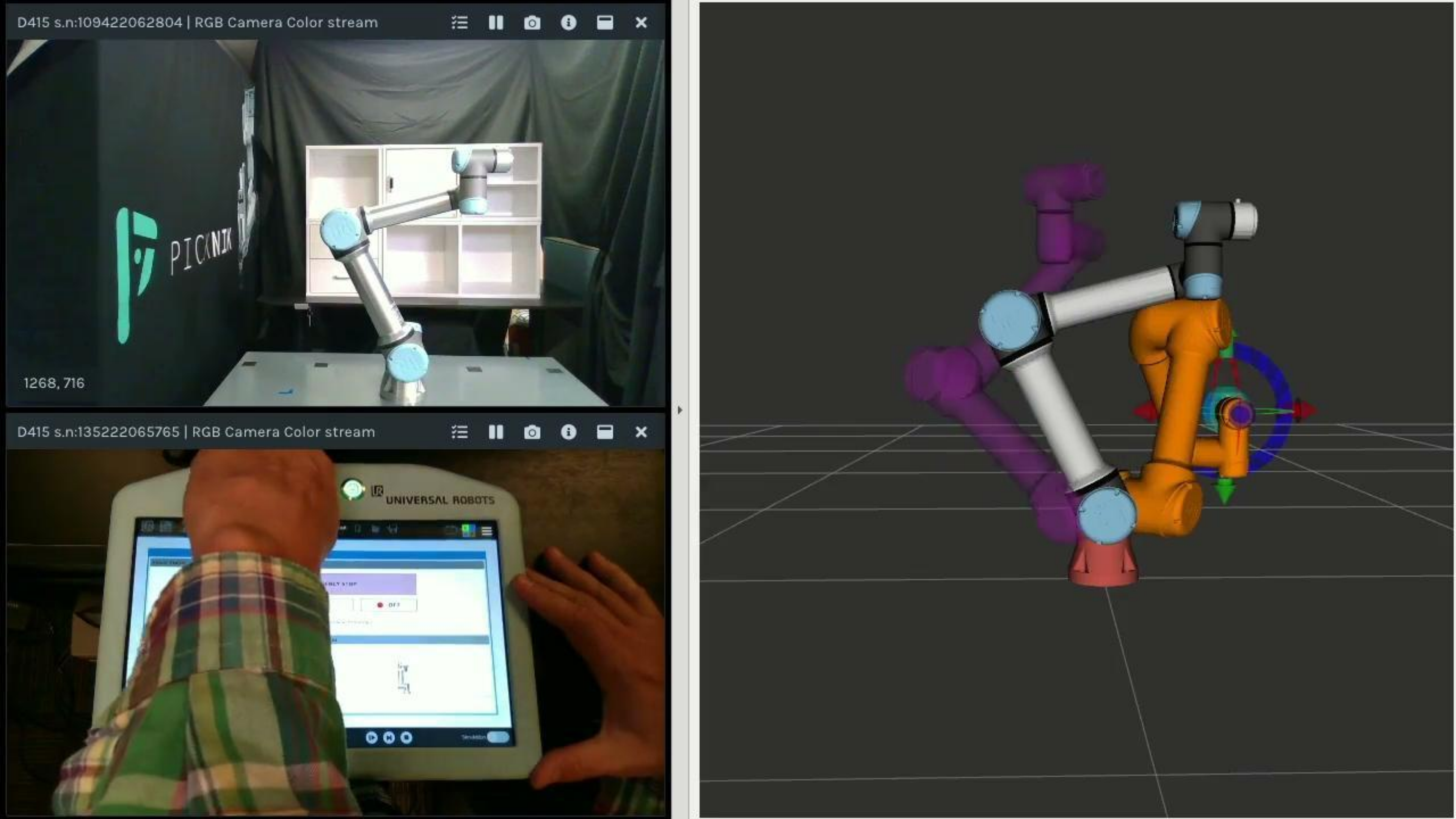
PICKNIK

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



PICKNIK


D415 s.n:109422062804 | RGB Camera Color stream
PICKNIK
1268, 716
D415 s.n:135222065765 | RGB Camera Color stream
UNIVERSAL ROBOTS

And now the conclusion…

PICKNIK

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

→ through "update" loop

- Readout and apply factory-kinematic calibration
- MoveIt2 and MoveIt2-Servo integration

- Simple testing using "URSim" — Docker container → Execution testing in CI!
  - Check repository: https://github.com/UniversalRobots/Universal_Robots_ROS2_Driver
  - URSim container: https://hub.docker.com/u/universalrobots

PICKNIK

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

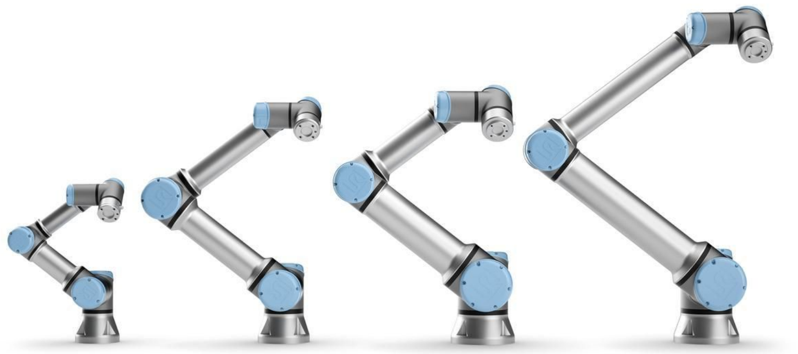
- Future influence of ros2_control-framework
  - TCP – Pose Broadcaster
  - Cartesian space controllers
  - Generic Robot Status Broadcaster

PICKNIK

Thank you for your attention!

User feedback and suggestion for improvements are very welcome!

ros@universal-robots.com / or open an issue at GitHub



PICKNIK

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 (exect 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

PICKNIK