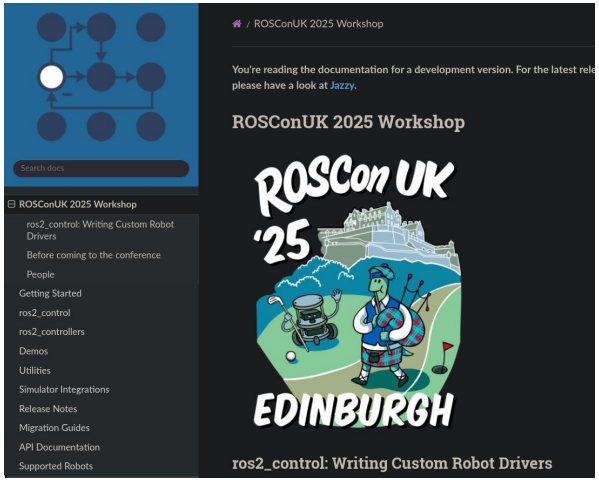
ros2_control:
Fun with Robot Drivers

Grab a USB drive or follow...

Open https://control.ros.org/rolling/doc/resources/roscon2025_workshop.html

Or https://control.ros.org



wget https://tinyurl.com/roscontrol2025 -O docker-compose.yaml

docker compose pull # if you are on ARM, run docker compose build

docker compose up -d

Attach from another terminal:

docker exec -it ros2_control_roscon25 bash

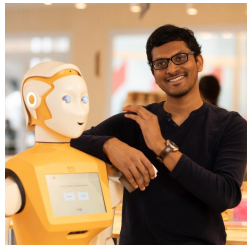
alias rc="docker exec -it ros2_control_roscon25 bash"

$whoami

Bence Magyar – [Bent’seh]

- PhD in Robotics
- Principal Software Engineer at Locus Robotics
- ros_control and ros2_control project lead

Marq Rasmussen
Locus Robotics



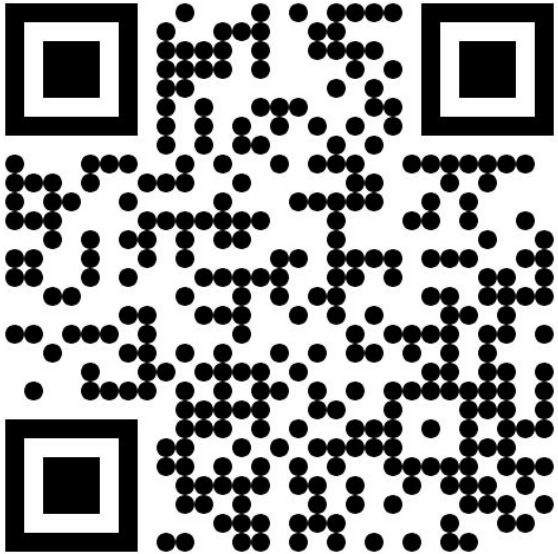
Sai Kishor Kothakota
PAL Robotics

Denis Štogl
Stogl Robotics Consulting

References

- https://control.ros.org/
- ros_control paper in

  the Journal of Open Source Software
- ros2_control presentations
  - https://control.ros.org/master/doc/resources/resources.html
- Github repositories at https://github.com/ros-controls



robotizer

PAL

LOCUS ROBOTICS

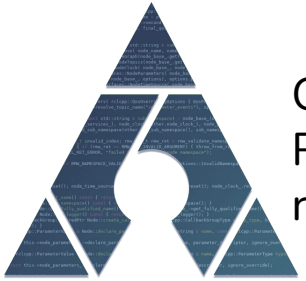
AIT AUSTRIAN INSTITUTE OF TECHNOLOGY
TOMORROW TODAY

Thank you for being here!!

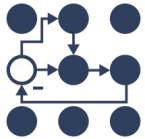
Special thanks to Gašper Jeršin & Ubiquity Robotics!

Bence Magyar, Denis Štogl, Christoph Froehlich, Sai Kishor Kothakota, Alejandro Hernández Cordero, Karsten Knese, Jordan Palacios, Shane Loretz, Dave Coleman, Jaron Lundwall, Jonathan Bohren, Felix Exner, Victor Lopez, Paul Gesel, Tyler Weaver, Manuel Muth, Julia Jia, Olivier Stasse, Soham Patil, Marq Rasmussen, Noel Jiménez García, Reza Kermani, Silvio Traversaro, Wiktor Bajor, Márk Szitanics, Andy Zelenak ..... and many more!

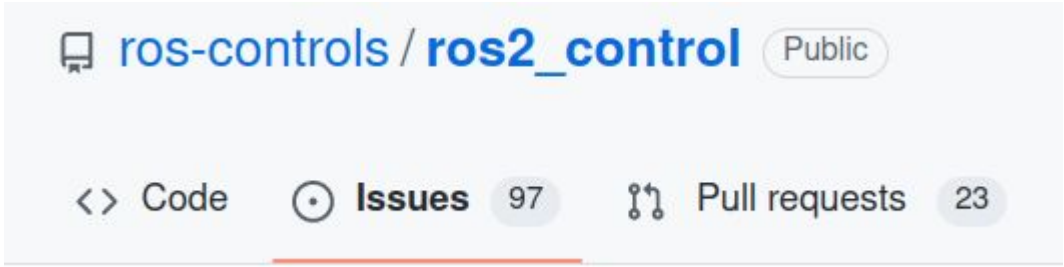
Contributing



Open Source
Robotics Alliance
ros-controls PMC



https://github.com/ros-controls



Online Meetings every second Wednesday!

Next one is 24th September!

Add additional return value to the hardware_interface::return_type good first issue good second issue help wanted

#815 opened 27 days ago by destogl

ros2_control reviewers

31 members

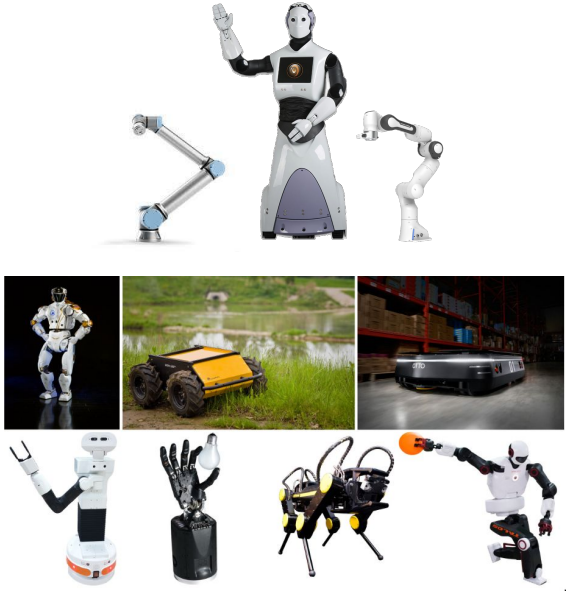
History

pr2_controller_manager

(pr2_mechanism)

ros_control

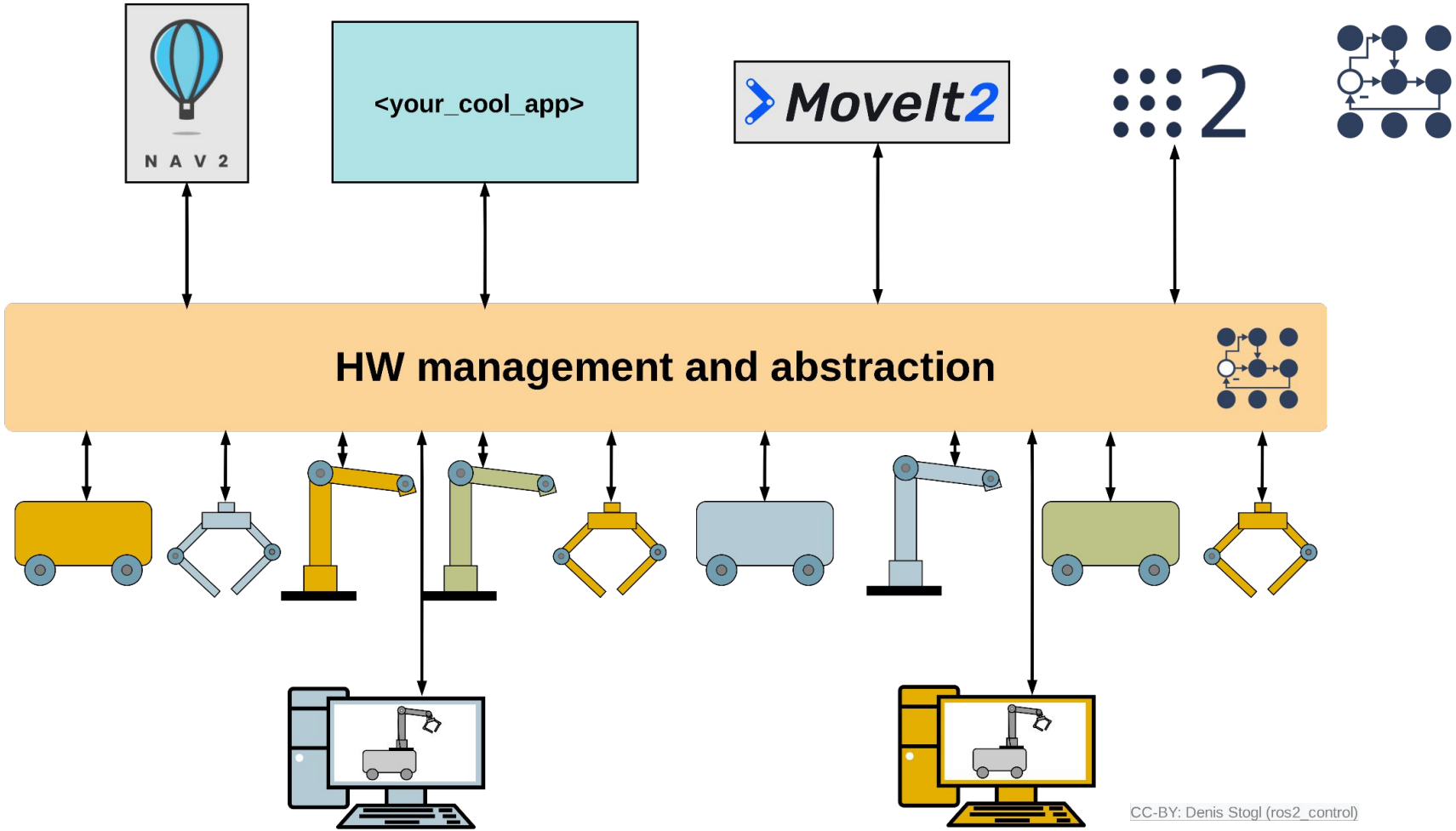
2012/2017

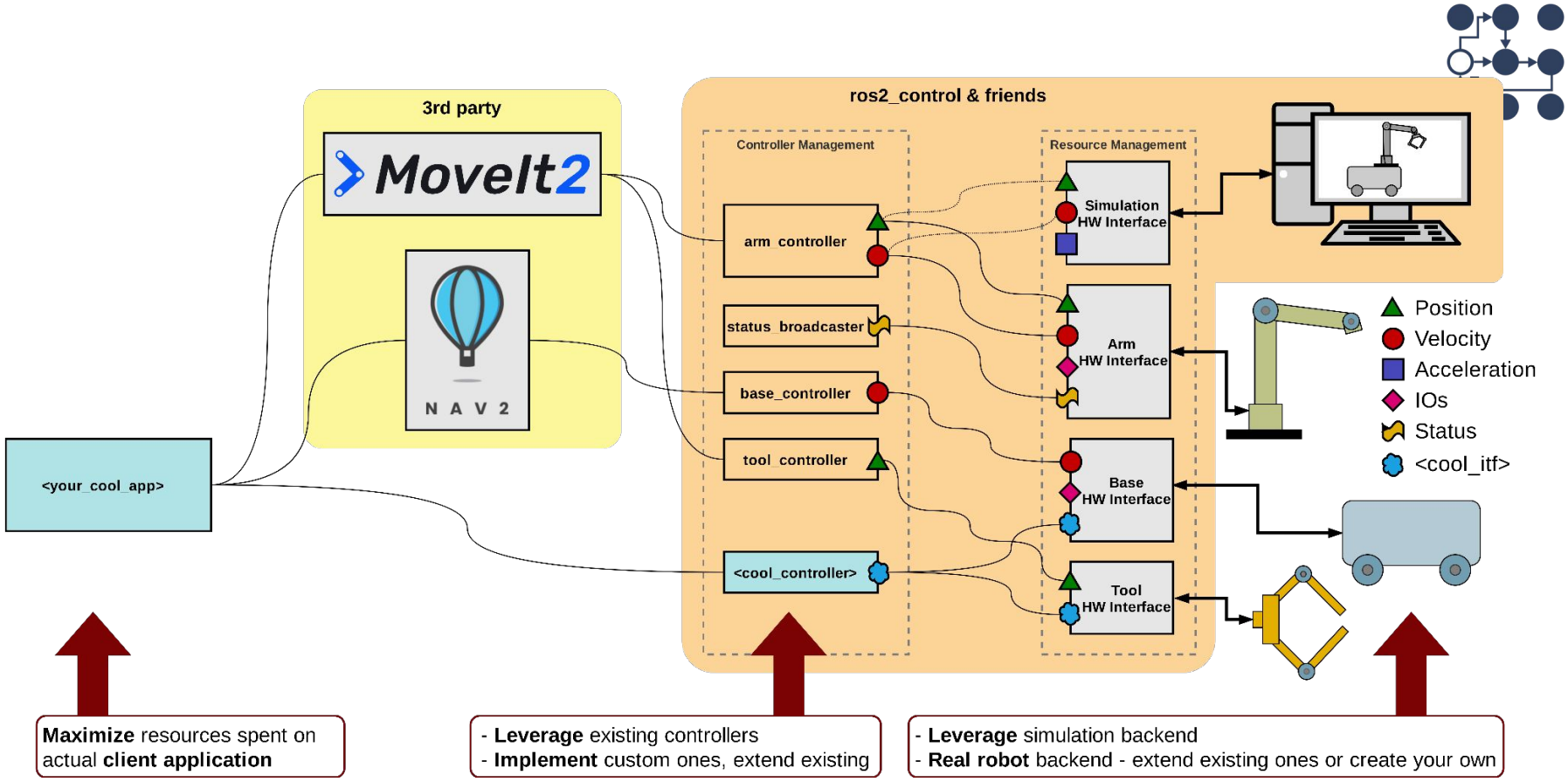


ros2_control

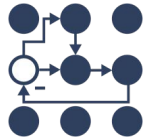
2017/today

https://control.ros.org/master/doc/supported_robots/supported_robots.html



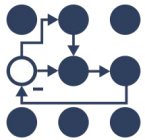

3rd party
MoveIt2
NAV 2
ros2_control & friends
Controller Management
Resource Management
arm_controller
status_broadcaster
base_controller
tool_controller
<cool_controller>
Simulation HW Interface
Arm HW Interface
Base HW Interface
Tool HW Interface
<your_cool_app>
Position
Velocity
Acceleration
IOs
Status
<cool_itf>
Maximize resources spent on actual **client application**
- **Leverage** existing controllers
- **Implement** custom ones, extend existing
- **Leverage** simulation backend
- **Real robot** backend - extend existing ones or create your own

Overview of the Main “Components”



- ros2_control_node—executable that runs Controller Manager » *when using the real or mock hardware*
  - Gazebo - the node is embedded in the simulator
  - Isaac Sim - uses ROS topic based interface
  - MuJoCo - uses a separate node, connects via MuJoCo API
- Hardware Components—hardware drivers
  - Defined & configured in URDF (xml - “ros2_control”-tag)
- Controllers
  - Configured through parameters (*yaml*) on Controller Manager
- ROS interfaces
  - ControllerManager services & diagnostics topics
  - Controllers define their own ROS interfaces, typically actions and/or topics
- Standard ROS CLI verb

Overview of Standard Controllers



Generics:

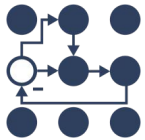
- PID
- GPIO Controller
- Chained Filter Controller – ❤️ filters library
- Forwarding Controller
  - Forward Command – multiple joints, one interface type
  - Multi Interfaces Fwd. Cmd. – one joint, multiple interface types
  - Position Controllers – same as Fwd Controller but only for *position* interface
  - Velocity Controllers – same as Fwd Controller but only for *velocity* interface
  - Effort Controllers – same as Fwd Controller but only for *effort* interface

Mobile Robots

- Steering Controllers
  - Bicycle – 1 drive joints, 1 steering joint
  - Tricycle – 2 drive joints, 1 steering joint (Tricycle Controller is an older version)
  - Ackerman – 2 drive joints, 2 steering joints
- Differential Drive (Diff drive) / Skid-steer
- Omni-wheel
- Mecanum

https://control.ros.org/rolling/doc/ros2_controllers/doc/controllers_index.html

Overview of Standard Controllers



Industrial Robotics (Arms)

- Joint Trajectory Controller (JTC)
  - The most used one – interface for MoveIt2 and similar frameworks
- Admittance Controller – force-position control in Cartesian space (using IK library from KDL)
- Motion Primitives (currently limited to UR and KUKA)

Grippers:

- Parallel Gripper Controller – 1 DoF gripp nally max vel and max effort interfaces

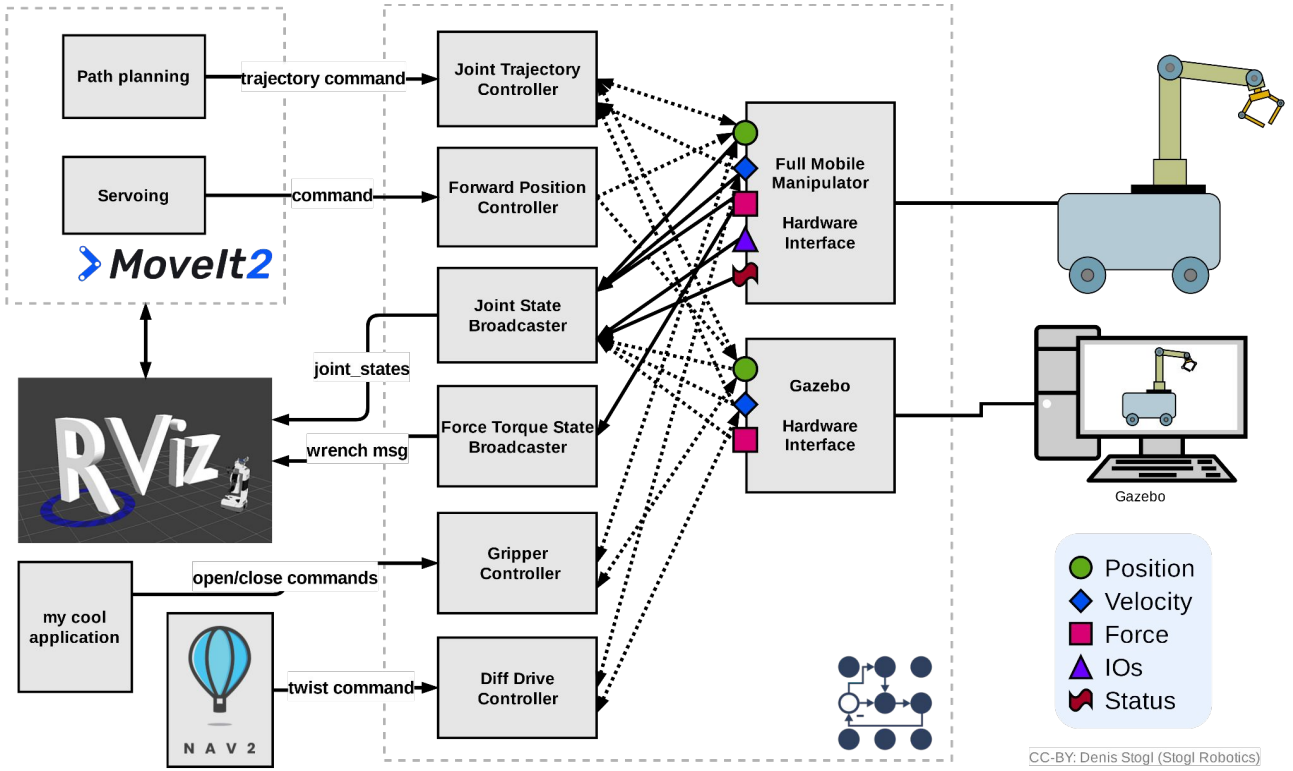
Non Controllers → Broadcasters

- Joint State Broadcaster – nothing works wit out it!!!
- Force Torque Sensor Broadcaster
  - Has funky stuff in it, like filtering – cool for using in chain ahead of Admittance Controller
- IMU Sensor Broadcaster
- Range Sensor Broadcaster
- GPS Sensor Broadcaster

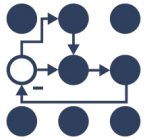
https://control.ros.org/rolling/doc/ros2_controllers/doc/controllers_index.html

Configuring standard controllers

```yaml
controller_manager:
  update_rate: 500  # Hz

  joint_trajectory_controller:
    type: joint_trajectory_controller/JointTrajectoryController

  forward_position_controller:
    type: position_controllers/JointGroupPositionController

  joint_state_broadcaster:
    type: joint_state_broadcaster/JointStateBroadcaster

  force_torque_sensor_broadcaster:
    type: force_torque_sensor_broadcaster/ForceTorqueStateBroadcaster

  gripper_controller:
    type: position_controllers/GripperActionController

  diff_drive_controller:
    type: diff_drive_controller/DiffDriveController

joint_trajectory_controller:
  joints:
    - joint1
    - ...
  command_interfaces:
    - position
  state_interfaces:
    - position
    - velocity

forward_position_controller:
  joints:
    - joint1
    - ...

force_torque_sensor_broadcaster:
  sensor_name: tcp_fts_sensor
  frame_id: tool0
  topic_name: ft_data


gripper_controller:
  joints:
    - gripper_joint
  command_interface: position

diff_drive_controller:
  left_weel_names:
    - left_wheel_1
    ...
```



Overview of Hardware Components



Base Classes:

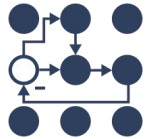
- System Component - state and command interfaces **all you want**
- Sensor Component - **only** state interfaces
- Actuator Component - interfaces only relevant to **one joint**

Technology stack

- Hardware Components are subclassed from the above
- Written in C++
- Exported as pluginlib plugins
- Configured via <ros2_control> XML section in URDF
- Can use internally anything they want

https://control.ros.org/rolling/doc/supported_robots/supported_robots.html

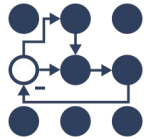
Example Hardware Definition *“the <ros2_control> - Tag”*



```
<ros2_control name="robot" type="system">

  <hardware>
    <plugin>robot_package/Robot</plugin>
    <param name="hardware_parameter">some_value</param>
  </hardware>

  <joint name="joint_first">
    <command_interface name="position"/>
    <state_interface name="acceleration"/>
  </joint>

  . . .

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

</ros2_control>


<ros2_control name="tool" type="actuator">

  <hardware>
    <plugin>tool_package/Tool</plugin>
    <param name="hardware_parameter">some_value</param>
  </hardware>

  <joint name="tool">
    <command_interface name="command"/>
  </joint>

</ros2_control>
```

```
<ros2_control name="robot" type="system">

  <hardware>
      <plugin>robot_package/Robot</plugin>
      <param name="hardware_parameter">some_value</param>
  </hardware>

  <joint name="joint_first">
    <command_interface name="position"/>
    <state_interface name="acceleration"/>
  </joint>
  . . .
  <joint name="joint_last">
    <command_interface name="velocity">
      <param name="min">-1</param>
      <param name="max">1</param>
    </command_interface>
    <state_interface name="temperature"/>
  </joint>

  <sensor name="tcp_sensor">
    <state_interface name="sensing_inteface"/>
    <param name="sensor_parameter">another_value</param>
  </sensor>

  <gpio name="flange_IOs">
    <command_interface name="digital_output" data_type="bool" size="8" />
    <state_interface name="digital_output" data_type="bool" size="8" />
    <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="4" />
    <state_interface name="analog_input" data_type="double" size="4" />
  </gpio>

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

  <joint name="tool">
    <command_interface name="command"/>
  </joint>

</ros2_control>
```

CLI - *ros2controlcli package*



- ros2 control TAB TAB (check autocompletion)

- Management of controllers and hardware components
- Introspection of the system
  - What is running, what is loaded, what is available?
  - Which interfaces are offered by HW component?
  - Which controller claims which interfaces?

Hot tip:

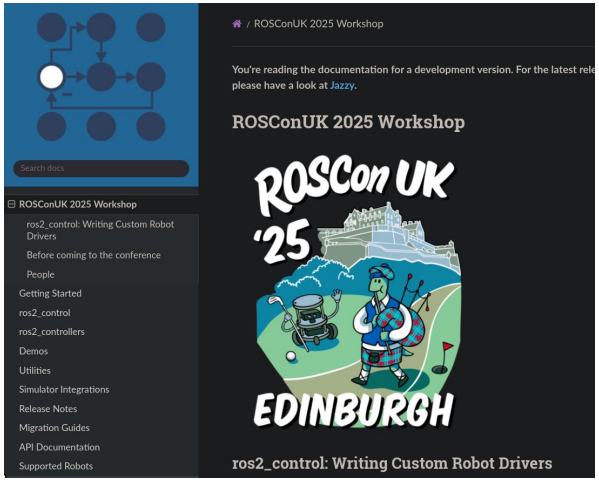
ros2 has a modular CLI system.
You can easily write your own CLI verbs too!

https://control.ros.org/rolling/doc/ros2_control/ros2controlcli/doc/userdoc.html

ros2_control:
Fun with Robot Drivers

Open https://control.ros.org/rolling/doc/resources/roscon2025_workshop.html

Or https://control.ros.org



wget https://tinyurl.com/roscontrol2025 -O docker-compose.yaml

docker compose pull

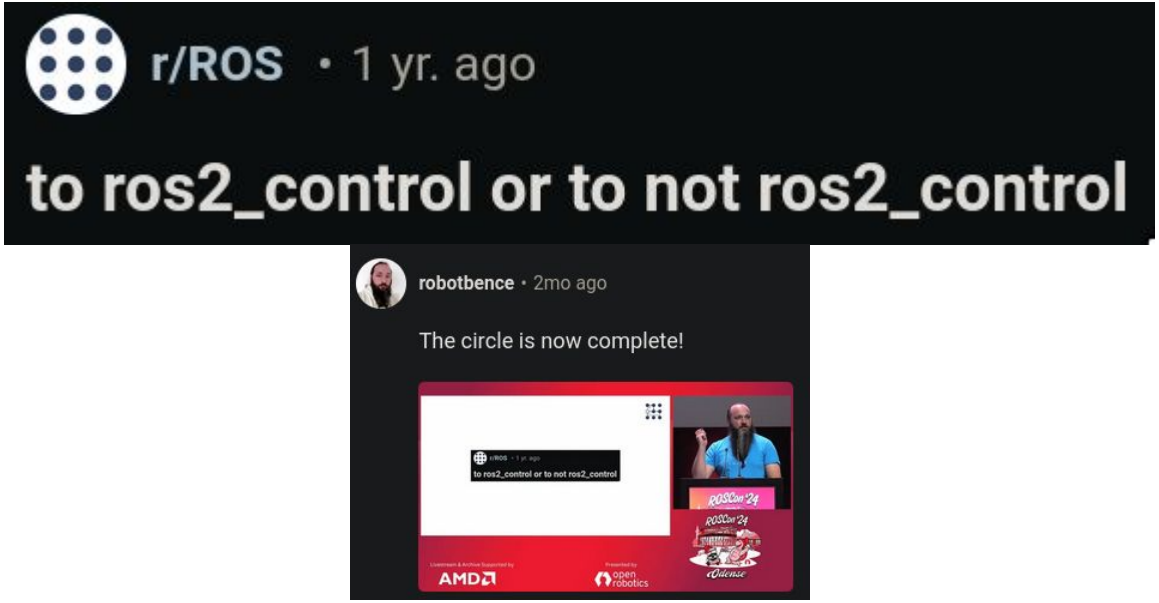
docker compose run ros2_control_roscon25

Attach from another terminal:

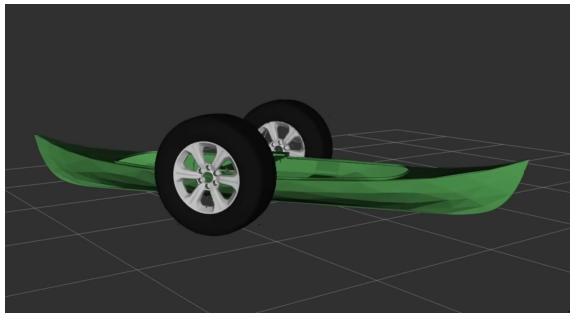
docker exec -it ros2_control_roscon25 bash

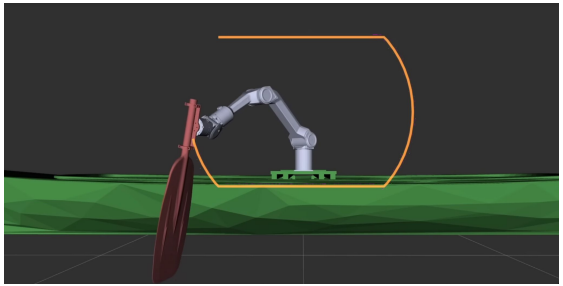
alias rc="docker exec -it ros2_control_roscon25 bash"

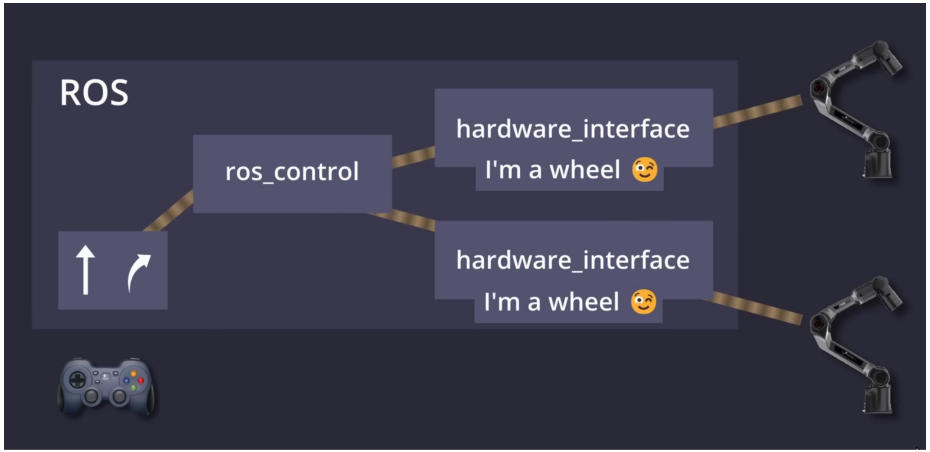
https://tinyurl.com/reddit-ros2-control



Rowing boat with 2 robot arms & ros2_control





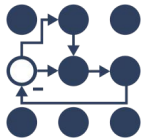


diff_drive_controller robot

By Dave's Armoury

https://youtu.be/XQX0SXHnbyk https://github.com/davesarmoury/rowboboat

Cheatsheet



Run this please:

- echo 'alias rc="docker exec -it ros2_control_roscon25 bash"' >> ~/.bashrc

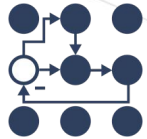
Cheat codes:

- rc = open shell in the container
- z = start up the zenoh daemon
- cb = colcon build ...
- s = source install/setup.bash

Everything also available in https://tinyurl.com/ros2control-workshop

RUN “docker compose up -d”

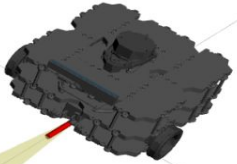
Task1: Diff Drive Controller with MockHW

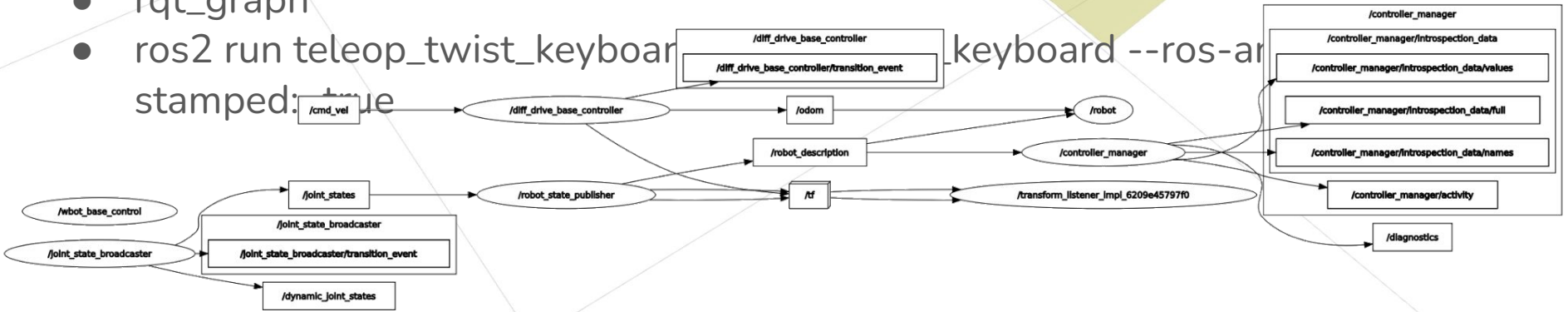


Run all commands in a new terminal inside the container:

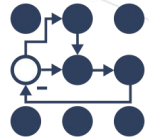
`rc` if tere are x11 errors: run “xhost +”

- z
- ros2 launch wbot_bringup wbot.launch.xml
- rqt_graph
- ros2 run teleop_twist_keyboar _keyboard --ros-ar
  stamped: true

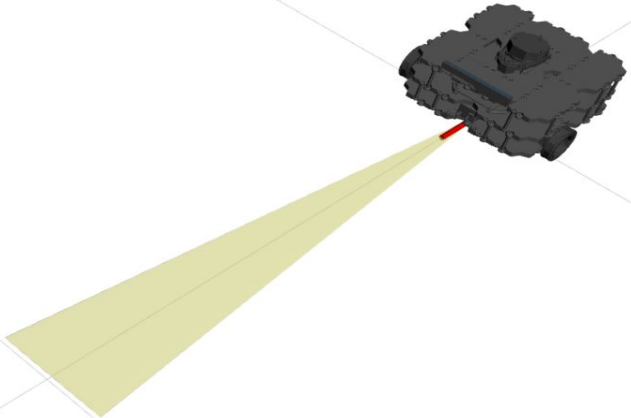




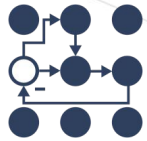
Task1: Diff Drive Controller with MockHW

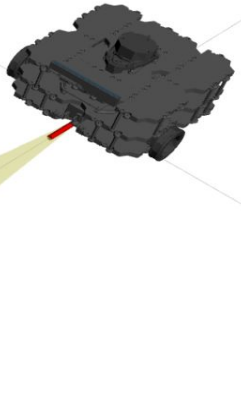


- ros2 control list_controllers
- ros2 control list_controller_types
- ros2 control list_hardware_components -v
- ros2 control list_hardware_interfaces
- ros2 topic list
- ros2 topic echo ...



Task1: Diff Drive Controller with MockHW

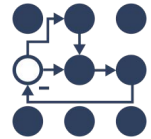




```
<?xml version="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:macro name="wbot_ros2_control" params="
    name
    mock_hardware:=false
    joint_command_topic:=/joint_command_topic
    joint_states_topic:=/joint_states_topic">

    <ros2_control name="${name}" type="system">
      <hardware>
        <xacro:if value="${mock_hardware}">
          <plugin>mock_components/GenericSystem</plugin>
          <param name="fake_sensor_commands">false</param>
          <param name="state_following_offset">0.0</param>
          <param name="calculate_dynamics">true</param>
        </xacro:if>
...
      </hardware>
...
```

```
...
      <joint name="wbot_wheel_left_joint">
        <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="wbot_wheel_right_joint">
        <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>
    </ros2_control>
  </xacro:macro>
</robot>
```

ros2_control & Embedded Systems

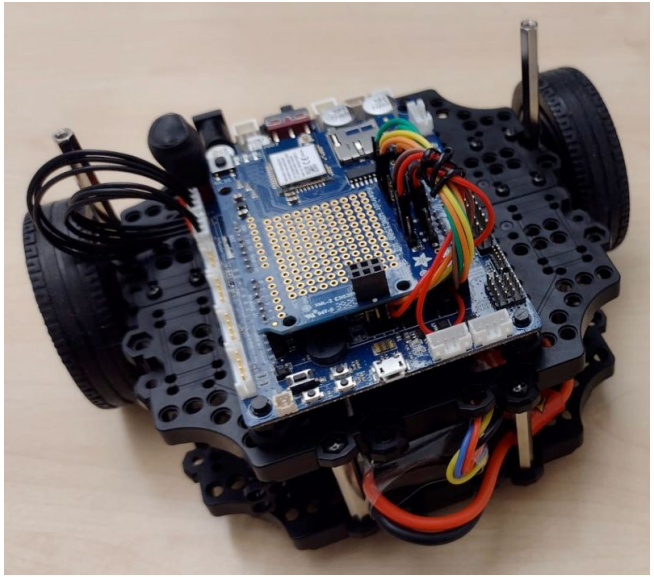


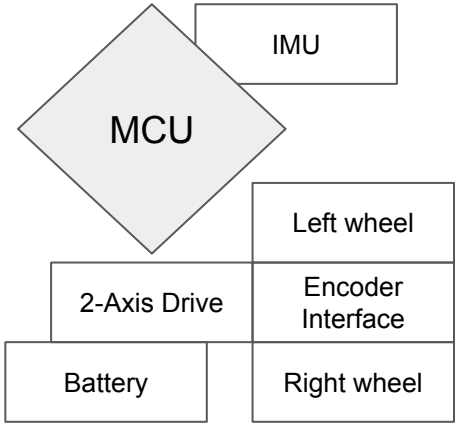
“Simple” single purpose program

Good at RT loops

GPIO and peripherals

Watchdog



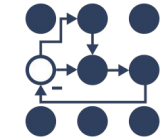


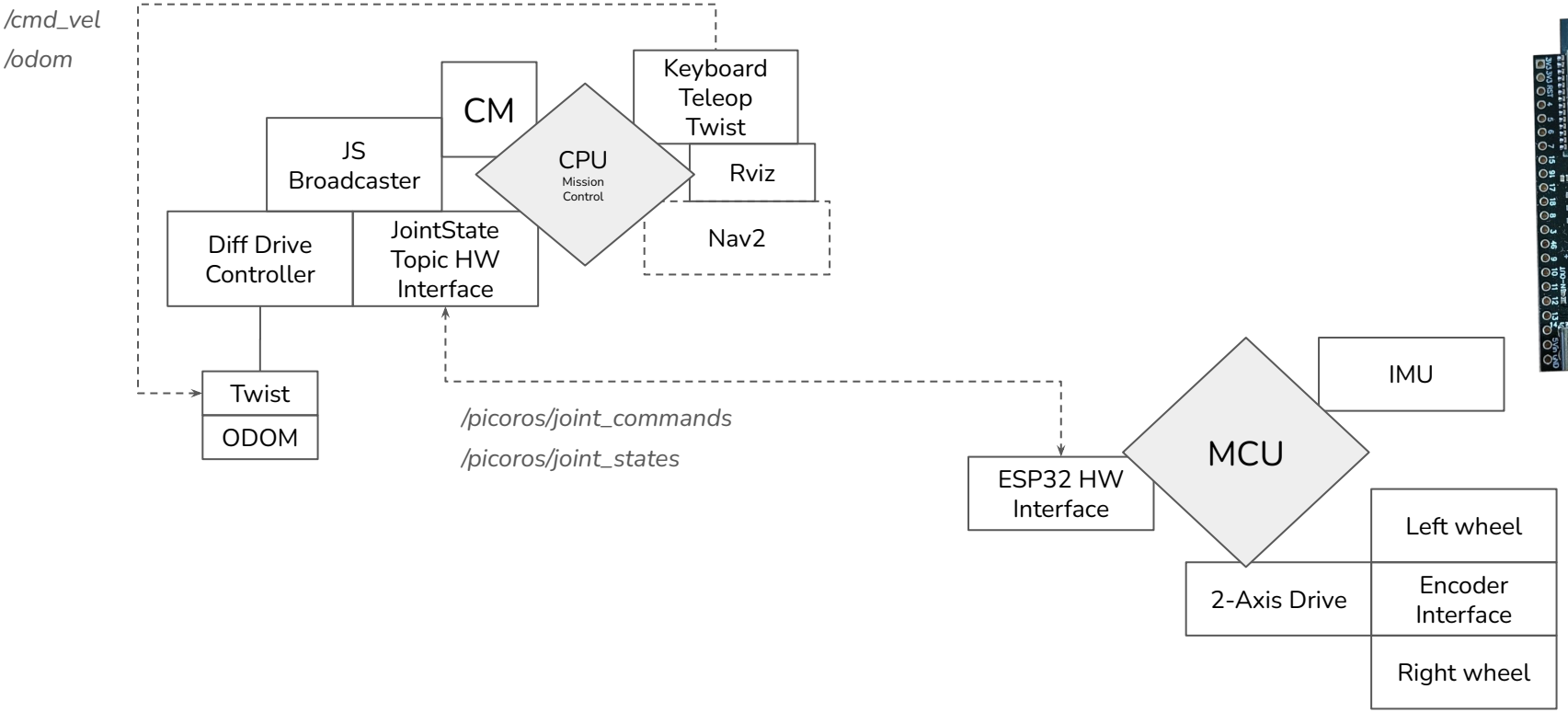
Task 2: ESP32 via zenoh hands-on

- Plug in ESP32 as shown
- Verify device name: `ls /dev/ttyACM*`. We want `/dev/ttyACM0`
- New terminal: `rc & z` – should get past the red light on the ESP32
- Inspect: ros2 topic tools, plotjuggler? + “ros2 topic publish {json jointstate}”
- Open the README.md file from the repo and copy&paste from there

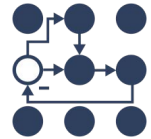
- ros2 topic list
- ros2 topic echo /picoros/joint_states
- ros2 topic hz /picoros/joint_states
- ros2 topic pub /picoros/joint_commands sensor_msgs/msg/JointState '{name: ["wbot_wheel_left_joint", "wbot_wheel_right_joint", "dummy"], velocity: [1.0, 0.0, 0.0]}'

Task 3: ros2_control + hardware (ESP32)





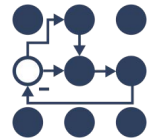
Task 3: ros2_control + hardware (ESP32)



```
<?xml version="1.0"?>

<robot xmlns:xacro="http://www.ros.org/wiki/xacro">
  <xacro:macro name="wbot_ros2_control" params="
    name
    mock_hardware:=false
    joint_command_topic:=/joint_command_topic
    joint_states_topic:=/joint_states_topic">

    <ros2_control name="${name}" type="system">
      <hardware>
...
        <plugin>joint_state_topic_hardware_interface/JointStateTopicSystem</plugin>
        <param name="joint_commands_topic">${joint_command_topic}</param>
        <param name="joint_states_topic">${joint_states_topic}</param>
        <param name="trigger_joint_command_threshold">-1</param>
...
      </hardware>
...
```

```
...
    <joint name="wbot_wheel_left_joint">
      <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="wbot_wheel_right_joint">
      <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>
  </ros2_control>
 </xacro:macro>
</robot>
```

Task 3: ros2_control + hardware (ESP32)

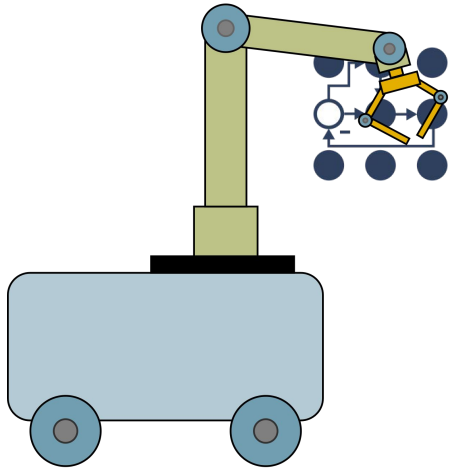


Each new terminal should be started with `rc`.

- z
- ros2 launch wbot_bringup wbot.launch.xml mock_hardware:=false

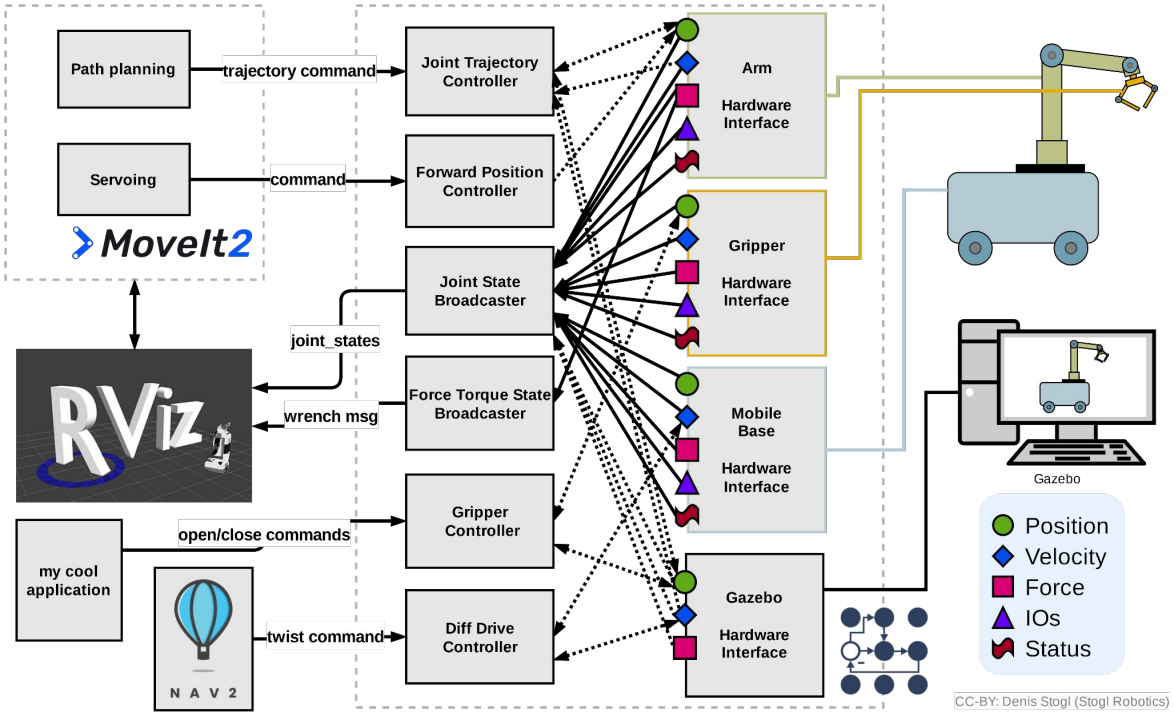
- Rqt_graph
- ros2 topic hz /picoros/joint_commands
- ros2 topic hz /picoros/joint_states
- ros2 topic list
- ros2 topic hz /joint_states
- ros2 topic echo /joint_states
- ros2 topic echo /picoros/joint_states
- ros2 control list_controllers
- ros2 control list_controller_types
- ros2 control list_hardware_components -v
- ros2 control list_hardware_interfaces

About hardware modelling



- Choose hardware interface architecture to your needs
  - *Guideline*: one communication path—one hardware interface
- Check the ros2_control_demos repository for different architecture examples.
- Profit from modularity of hardware interfaces – “implement only one time”

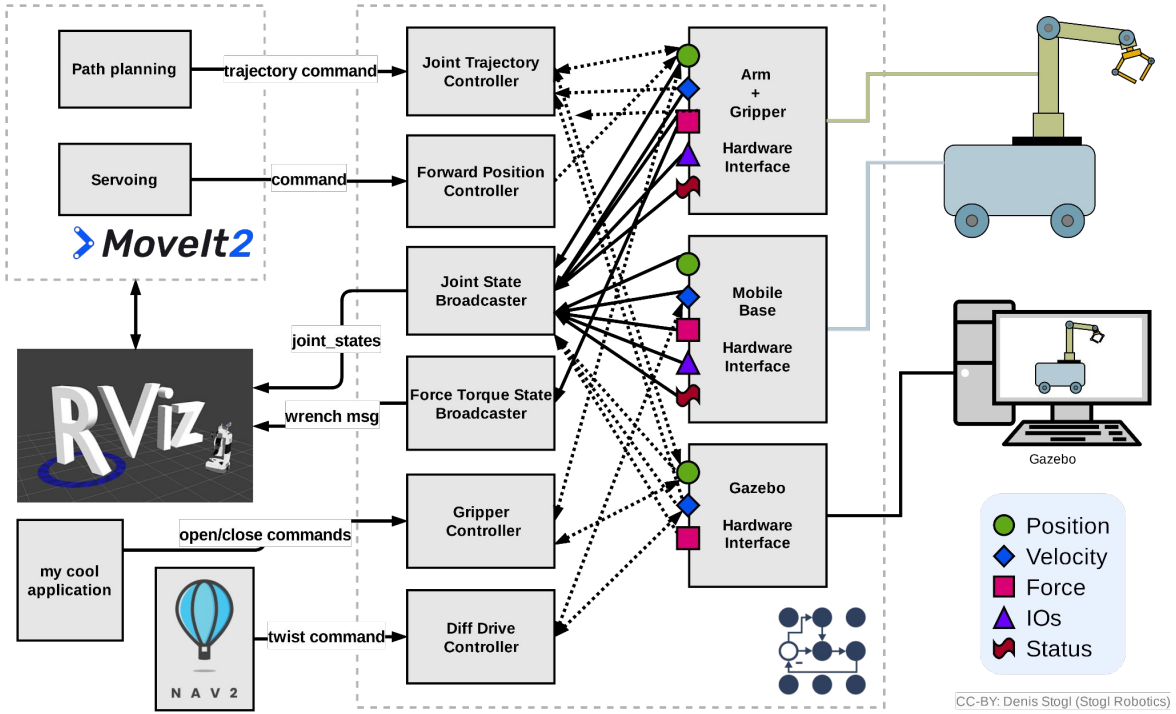
Modelling complex hardware – individual components




```xml
<ros2_control name="MobileBase" type="system">
  <hardware>
    <plugin>ros2_contro_robot/Mobile_Base</plugin>
    ...
  </hardware>
  <joint name="wheel1_joint">
    <command_interface name="velocity"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
</ros2_control>

<ros2_control name="Arm" type="system">
  <hardware>
    <plugin>ros2_contro_robot/Arm</plugin>
    ...
  </hardware>
  <joint name="joint1">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
</ros2_control>

<ros2_control name="Gripper" type="actuator">
  <hardware>
    <plugin>ros2_contro_robot/Gripper</plugin>
    ...
  </hardware>
  <joint name="gripper_joint">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
</ros2_control>
```

ros2_control_demos Example: "Modular Robots with separate communication to each actuator"

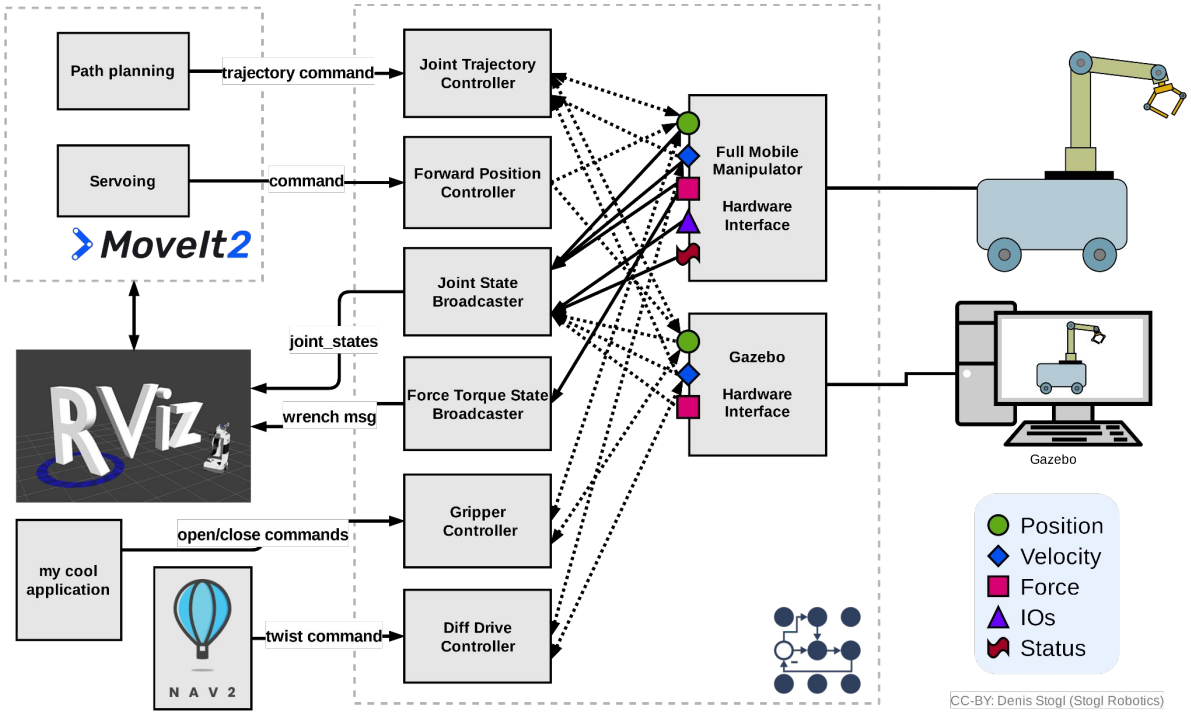
Modelling complex hardware – “bus through arm” + base




```xml
<ros2_control name="MobileBase" type="system">
  <hardware>
    <plugin>ros2_contro_robot/Mobile_Base</plugin>
    ...
  </hardware>
  <joint name="wheel1_joint">
    <command_interface name="velocity"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
</ros2_control>

<ros2_control name="ArmWithGripper" type="system">
  <hardware>
    <plugin>ros2_contro_robot/Arm_With_Gripper</plugin>
    ...
  </hardware>
  <joint name="joint1">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
  <joint name="gripper_joint">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
</ros2_control>
```

ros2_control_demos Example: "Industrial robot with integrated sensor"

Modelling complex hardware – monolithic components




```
<ros2_control name="MobileManipulator" type="system">
  <hardware>
    <plugin>ros2_contro_robot/Full_Mobile_Manipulator</plugin>
    ...
  </hardware>
  <joint name="wheel1_joint">
    <command_interface name="velocity"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
  <joint name="joint1">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
  <joint name="gripper_joint">
    <command_interface name="position"/>
    <state_interface name="position"/>
    <state_interface name="velocity"/>
    <state_interface name="status"/>
  </joint>
  ...
</ros2_control>
```

ros2_control_demos Example: "Robots with multiple interfaces"

Implementing a 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

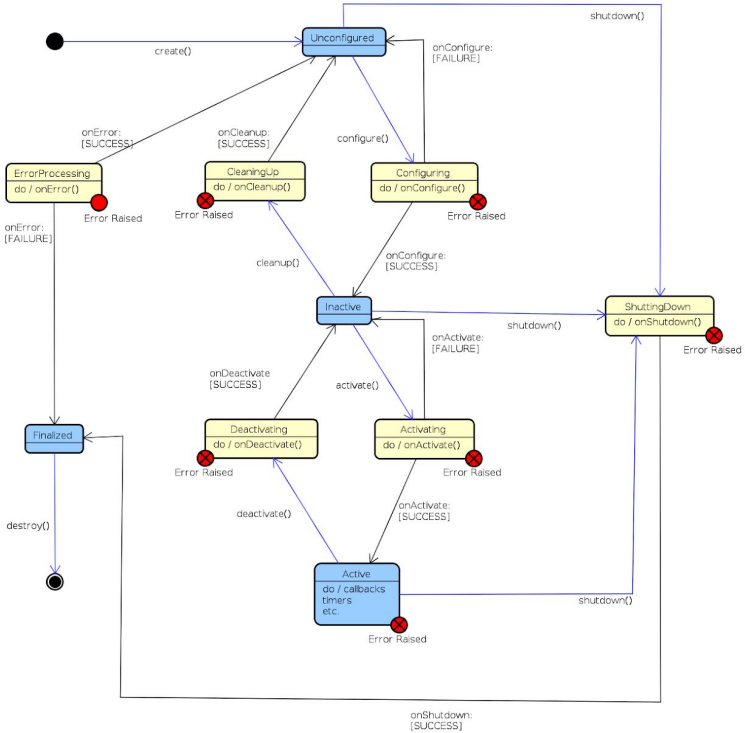
on_configure (previous_state)
- initiate communication with the HW
- be sure HW states can be read

on_deactivate (previous_state)
- disable HW movement

on_cleanup (previous_state)
- disable communication

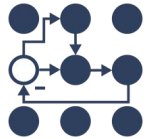
on_error (previous_state)
- process and mitigate any errors
- it can happen in any state
- catching errors during read/write

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



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

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

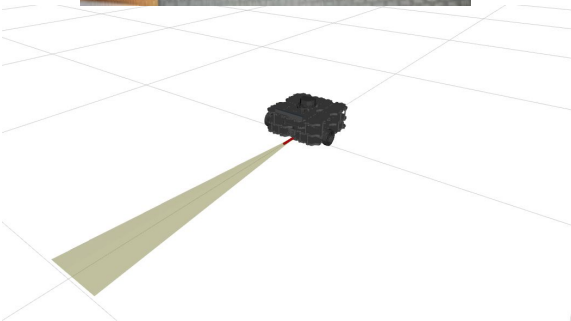
on_shutdown (previous_state)
- initiate HW shoutdown sequence
- can be called from any state

Task 4: Add limiting to topic based hardware interface

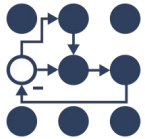
- Let’s review a PR!
  https://github.com/ros-controls/topic_based_hardware_interfaces/pull/30

`rc`

- cat src/wbot_description/urdf/wbot.ros2_control.xacro
- ros2 launch wbot_bringup wbot.launch.xml mock_hardware:=false **enable_command_limiting:=true**
- ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -p stamped:=true
- ros2 topic echo /picoros/joint_commands



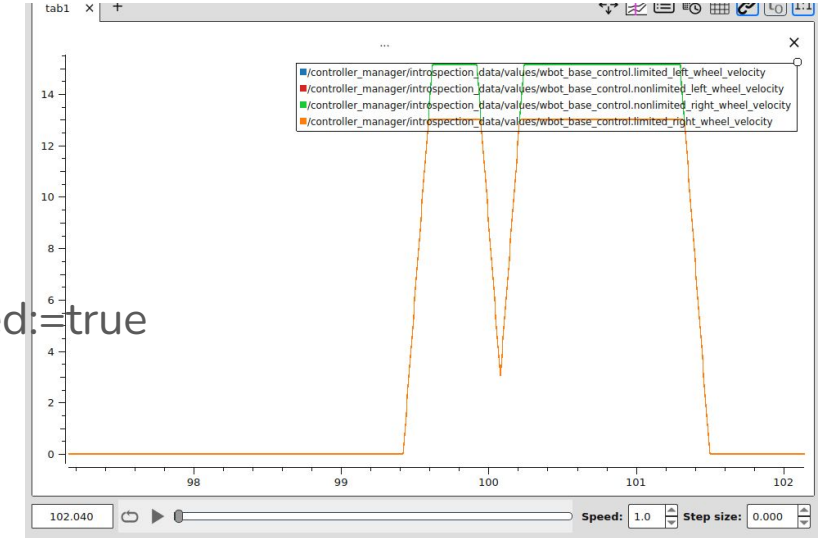
Task 5: Demo pal_statistics for hardware introspection



- Let’s review another PR!
  https://github.com/ros-controls/topic_based_hardware_interfaces/pull/31

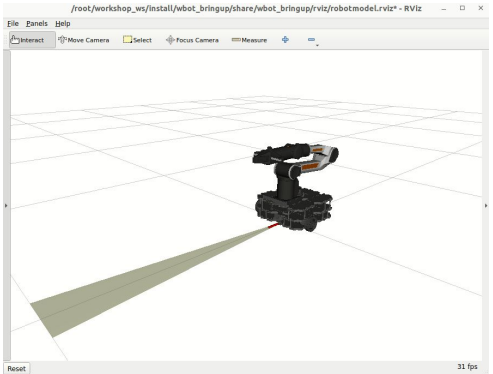
`rc`

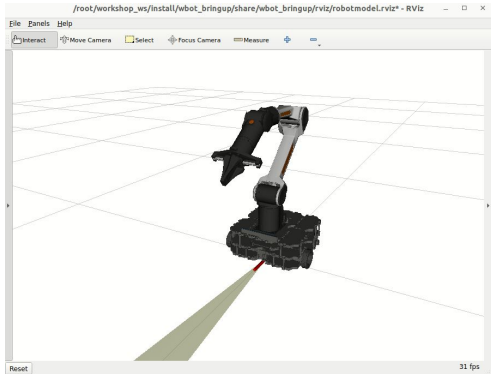
- ros2 launch wbot_bringup wbot.launch.xml mock_hardware:=false **enable_command_limiting:=true**
- ros2 run teleop_twist_keyboard teleop_twist_keyboard --ros-args -p stamped:=true
- ros2 topic echo /controller_manager/introspection_data/full

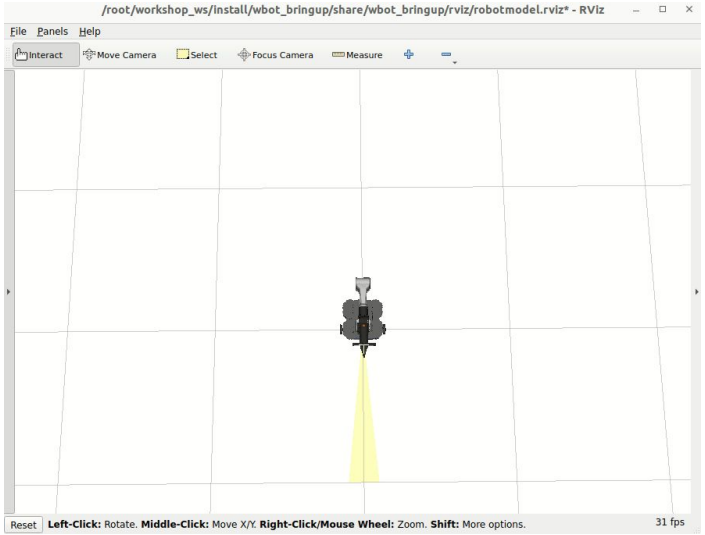


Task 6: Mock and Real Hardware together

- wbot with piper
- ros2_control xml with MockHardware for Piper, topic-based for esp32
- Exercise: send the arm a command, send the gripper a command, send the base a command
- Inspect: ros2 control cli, rviz
- Do on your own: repo README.md

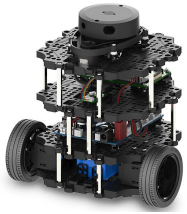






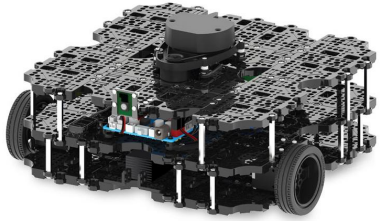
Robots

Turtlebot3 Burger



https://robotis.co.uk/turtle-burg.html

Turtlebot3 Waffle



https://robotis.co.uk/turtle-burg.html

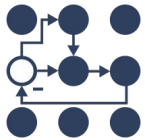
Agilex Piper

https://autodiscovery.co.uk/equipment/piper.html
https://global.agilex.ai/products/piper

Please provide us feedback!

tinyurl.com/control-feedback

Recap



- MockHardware for simple simulations
- Talking to an embedded board using zenoh-pico and native ROS topics
- Set up a basic driver using ROS topics for the embedded board
- Implemented a basic velocity command limiter
- Looked at adding parameters to hardware components
- Studied <ros2_control> tag for different hardware components
- Added introspection to the velocity command limiter setup
- Demoed a mixed setup with 2 hardware components, one mock, one topic-based + 3 controllers
- Inspect, inspect, inspect

- Technology stack used:
  - ros2_control
  - diff_drive_controller
  - ESP32 board
  - ros2 CLI
  - zenoh, rmw_zenoh, zenoh-pico, pico-ros
  - docker
  - teleop_twist_keyboard
  - rqt_graph
  - rviz2
  - URDF, xacro
  - joint_trajectory_controller, parallel_gripper_controller

robotizer

PAL

LOCUS ROBOTICS

AIT AUSTRIAN INSTITUTE OF TECHNOLOGY
TOMORROW TODAY

Thank you for being here!!

Special thanks to Gašper Jeršin & Ubiquity Robotics!

Bence Magyar, Denis Štogl, Christoph Froehlich, Sai Kishor Kothakota, Alejandro Hernández Cordero, Karsten Knese, Jordan Palacios, Shane Loretz, Dave Coleman, Jaron Lundwall, Jonathan Bohren, Felix Exner, Victor Lopez, Paul Gesel, Tyler Weaver, Manuel Muth, Julia Jia, Olivier Stasse, Soham Patil, Marq Rasmussen, Noel Jiménez García, Reza Kermani, Silvio Traversaro, Wiktor Bajor, Márk Szitanics, Andy Zelenak ..... and many more!