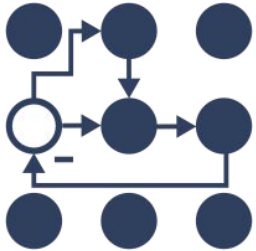
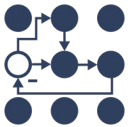
Getting started with ros2_control



Presentation outline

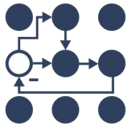
- Present outline ⟵ you are here!
- History & basic concepts
- What it takes to implementing a HAL
- Implementing a controller
- Modifying a controller
- Written exam



$whoami

Bence Magyar

- PhD in Robotics
- Lead Software Engineer at FiveAI
- ros_control and ros2_control maintainer



What & where

pr2_controller_manager

(pr2_mechanism)

2009

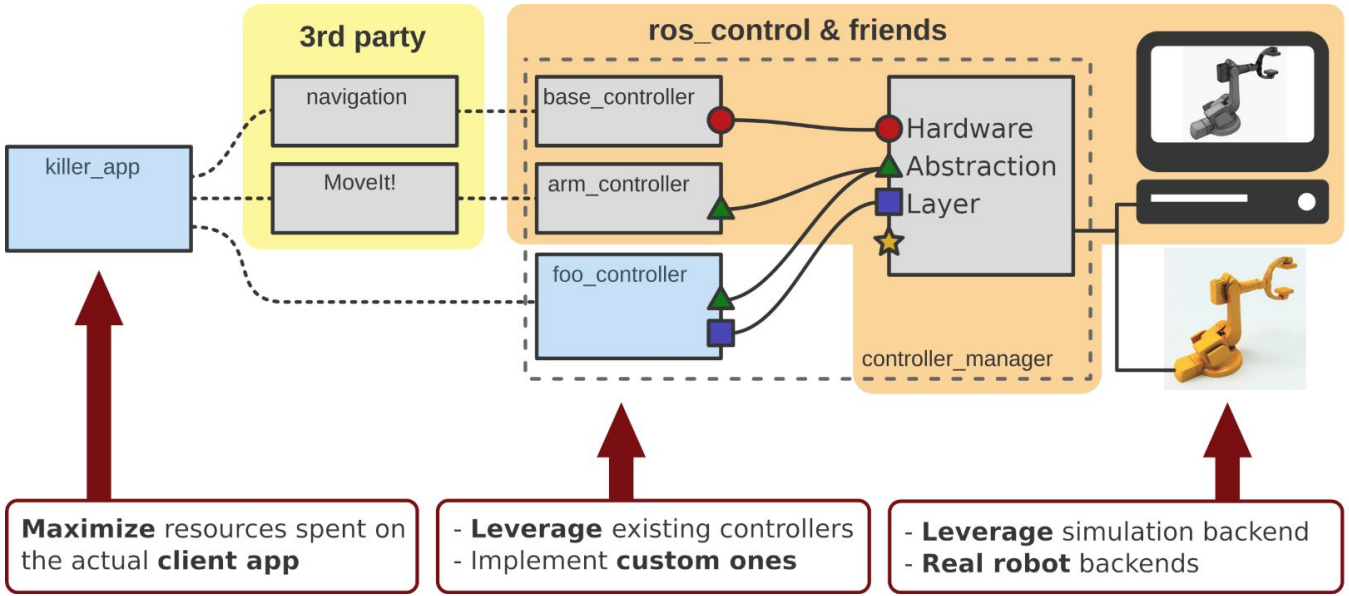
ros_control

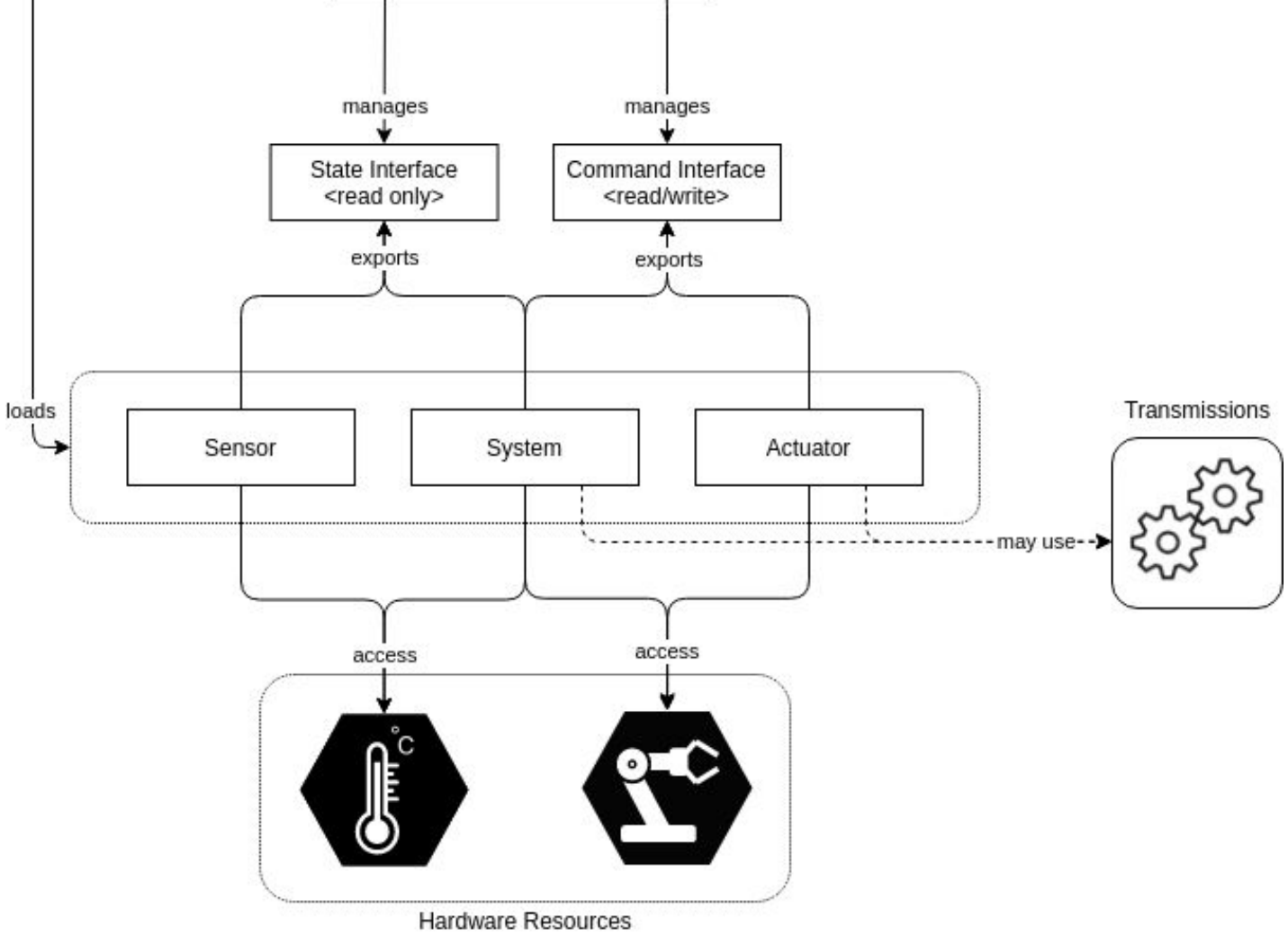
2012/2013

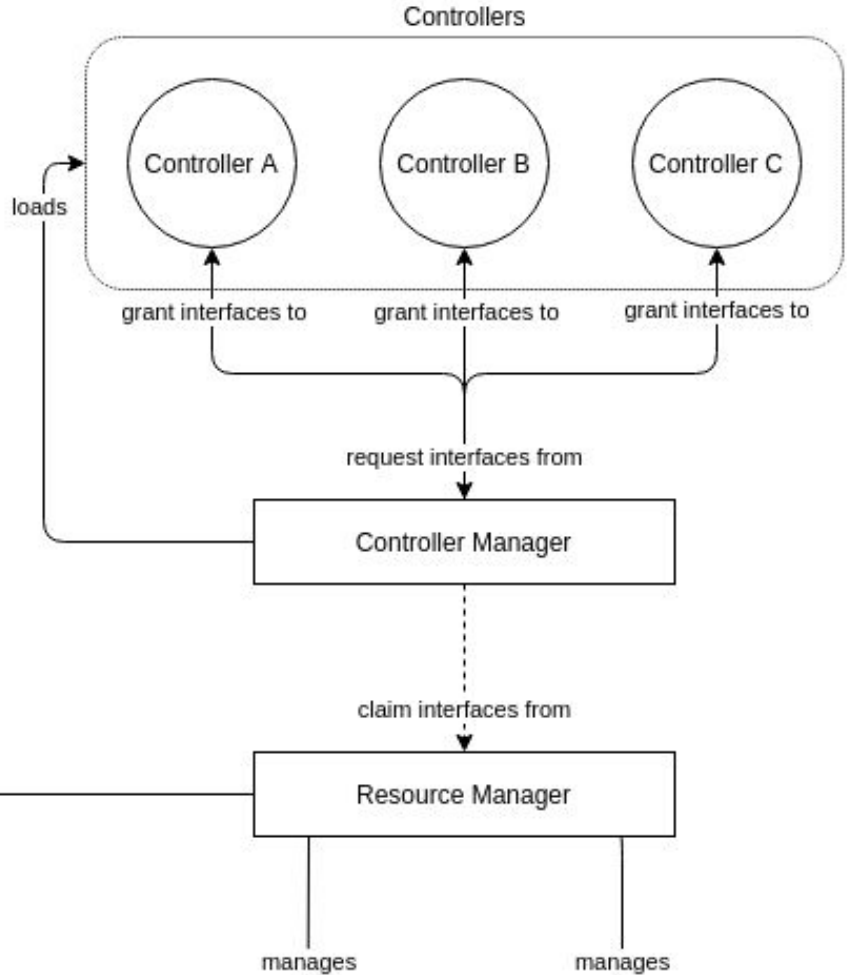
ros2_control

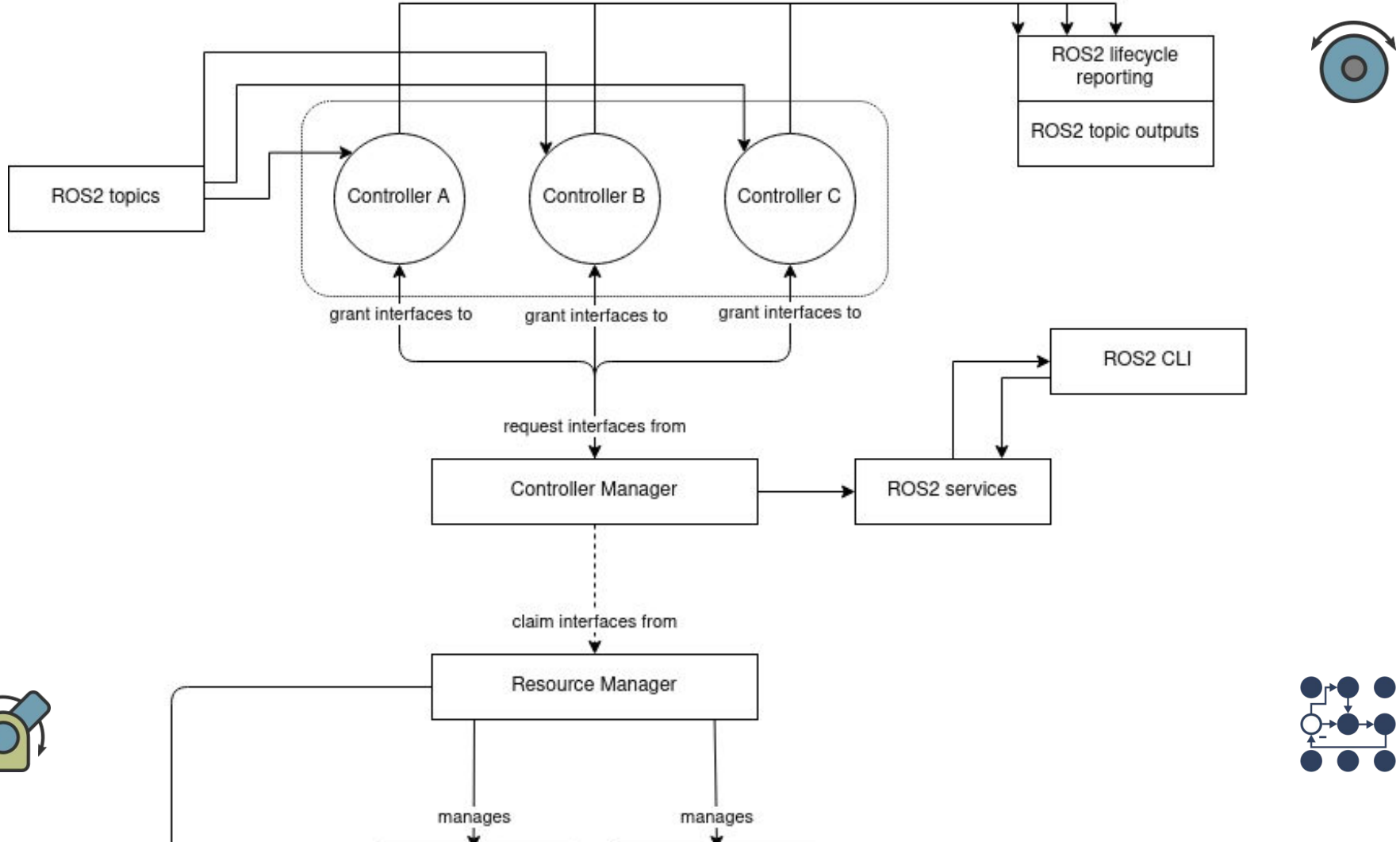
2017/2020

Image from ros_control [paper]


3rd party
navigation
MoveIt!
killer_app
ros_control & friends
base_controller
arm_controller
foo_controller
Hardware
Abstraction
Layer
controller_manager
resource interfaces
velocity_joint
position_joint
effort_joint
image credits:
visualpharm.com
turbosquid.com
Maximize resources spent on the actual client app
- Leverage existing controllers
- Implement custom ones
- Leverage simulation backend
- Real robot backends

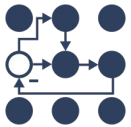

manages
manages
State Interface
<read only>
Command Interface
<read/write>
exports
exports
loads
Sensor
System
Actuator
Transmissions
may use
access
access
°C
Hardware Resources


Controllers
Controller A
Controller B
Controller C
loads
grant interfaces to
grant interfaces to
grant interfaces to
request interfaces from
Controller Manager
claim interfaces from
Resource Manager
manages
manages


ROS2 lifecycle reporting
ROS2 topic outputs
ROS2 topics
Controller A
Controller B
Controller C
grant interfaces to
grant interfaces to
grant interfaces to
ROS2 CLI
request interfaces from
Controller Manager
ROS2 services
claim interfaces from
Resource Manager
manages
manages

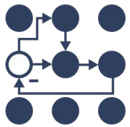
URDF and ros2_control

```
<ros2_control name="${name}" type="system">
  <hardware>
    <plugin>fake_components/GenericSystem</plugin>
  </hardware>
  <joint name="joint1">
    <command_interface name="position">
      <param name="min">-1</param>
      <param name="max">1</param>
    </command_interface>
    <state_interface name="position"/>
  </joint>
  <joint name="joint2">
    <command_interface name="position">
      <param name="min">-1</param>
      <param name="max">1</param>
    </command_interface>
    <state_interface name="position"/>
  </joint>
</ros2_control>
```



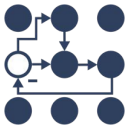
Implementing a system component

```
class RRBotHardwareInterface
  : public hardware_interface::BaseInterface<hardware_interface::SystemInterface>
{
public:
  hardware_interface::return_type configure(const hardware_interface::HardwareInfo & info) override;

  std::vector<hardware_interface::StateInterface> export_state_interfaces() override;

  std::vector<hardware_interface::CommandInterface> export_command_interfaces() override;

  hardware_interface::return_type start() override;

  hardware_interface::return_type stop() override;

  hardware_interface::return_type read() override;

  hardware_interface::return_type write() override;

private:
  std::vector<double> hw_commands_;
  std::vector<double> hw_states_;
};
```



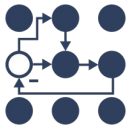
Implementing a system component

```
hardware_interface::return_type RRBotHardwareInterface::configure(
  const hardware_interface::HardwareInfo & info)
{
  if (configure_default(info) != hardware_interface::return_type::OK) {
    return hardware_interface::return_type::ERROR;
  }

  hw_states_.resize(info_.joints.size(), std::numeric_limits<double>::quiet_NaN());
  hw_commands_.resize(info_.joints.size(), std::numeric_limits<double>::quiet_NaN());

  status_ = hardware_interface::status::CONFIGURED;
  return hardware_interface::return_type::OK;
}
```



Implementing a system component

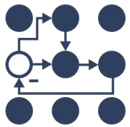
```
hardware_interface::return_type RRBotHardwareInterface::read()
{
  // read robot states from hardware, in this example print only
  RCLCPP_INFO(rclcpp::get_logger("RRBotHardwareInterface"), "Reading...");

  // write command to hardware, in this example do mirror command to states
  for (size_t i = 0; i < hw_states_.size(); ++i){
    RCLCPP_INFO(
      rclcpp::get_logger("RRBotHardwareInterface"),
      "Got state %.2f for joint %d!", hw_states_[i], i);
  }

  return hardware_interface::return_type::OK;
}

hardware_interface::return_type RRBotHardwareInterface::write()
{
  // write command to hardware, in this example do mirror command to states
  for (size_t i = 0; i < hw_commands_.size(); ++i){
    hw_states_[i] = hw_states_[i] + (hw_commands_[i] - hw_states_[i]) / 100.0;
  }

  return hardware_interface::return_type::OK;
}
```



Implementing a system component

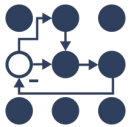
```
<ros2_control name="${name}" type="system">
  <hardware>
    <plugin>rrbot_hardware_interface/RRBotHardwareInterface</plugin>
  </hardware>
  <joint name="joint1">
    <command_interface name="position">
      <param name="min">-1</param>
      <param name="max">1</param>
    </command_interface>
    <state_interface name="position"/>
  </joint>
  <joint name="joint2">
    <command_interface name="position">
      <param name="min">-1</param>
      <param name="max">1</param>
    </command_interface>
    <state_interface name="position"/>
  </joint>
</ros2_control>
```

```
ros2 launch rrbot_bringup rrbot.launch.py
```



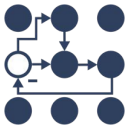
Implementing a forwarding controller

```
class RRBotControllerArray : public controller_interface::ControllerInterface
{
public:
  controller_interface::return_type init(const std::string & controller_name) override;

  controller_interface::InterfaceConfiguration command_interface_configuration() const override;

  controller_interface::InterfaceConfiguration state_interface_configuration() const override;

  CallbackReturn on_configure(const rclcpp_lifecycle::State & previous_state) override;

  CallbackReturn on_activate(const rclcpp_lifecycle::State & previous_state) override;

  CallbackReturn on_deactivate(const rclcpp_lifecycle::State & previous_state) override;

  controller_interface::return_type update() override;

...

};
```



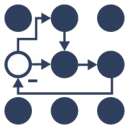
Implementing a forwarding controller

```
class RRBotControllerArray : public controller_interface::ControllerInterface
{
...

protected:
  std::vector<std::string> joint_names_;
  std::string interface_name_;

  using ControllerCommandMsg = example_interfaces::msg::Float64MultiArray;

  rclcpp::Subscription<ControllerCommandMsg>::SharedPtr command_subscriber_ = nullptr;
  realtime_tools::RealtimeBuffer<std::shared_ptr<ControllerCommandMsg>> input_command_;

  using ControllerStateMsg = control_msgs::msg::JointControllerState;
  using ControllerStatePublisher = realtime_tools::RealtimePublisher<ControllerStateMsg>;

  rclcpp::Publisher<ControllerStateMsg>::SharedPtr s_publisher_;
  std::unique_ptr<ControllerStatePublisher> state_publisher_;
};
```



Implementing a forwarding controller

```
controller_interface::return_type RRBotControllerArray::update()
{
  auto current_command = input_command_.readFromRT();

  for (size_t i = 0; i < command_interfaces_.size(); ++i) {
    if (!std::isnan((*current_command)->data[i])) {
      command_interfaces_[i].set_value((*current_command)->data[i]);
    }
  }

  if (state_publisher_ && state_publisher_->trylock()) {
    state_publisher_->msg_.header.stamp = get_node()->now();
    state_publisher_->msg_.set_point = command_interfaces_[0].get_value();

    state_publisher_->unlockAndPublish();
  }

  return controller_interface::return_type::OK;
}
```



Implementing a forwarding controller

In rrbot_controller.xml:

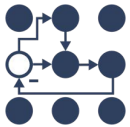
```
<library path="librrbot_controller_array">
  <class name="rrbot_controller/RRBotControllerArray"
         type="rrbot_controller::RRBotControllerArray"
 base_class_type="controller_interface::ControllerBase">
    <description>
      RRBotControllerArray ros_control controller.
    </description>
  </class>
</library>
```

In controller.cpp

```
#include "pluginlib/class_list_macros.hpp"

PLUGINLIB_EXPORT_CLASS(rrbot_controller::RRBotControllerArray, controller_interface::ControllerBase)
```

In CMakelists.txt:

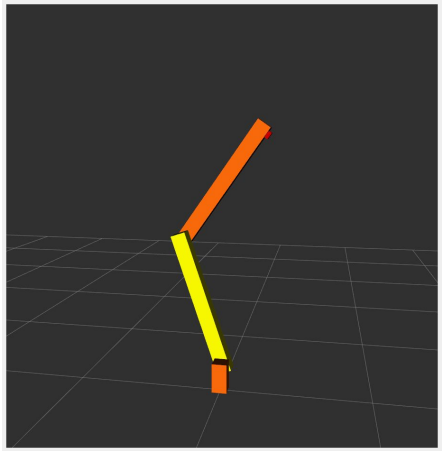
```
pluginlib_export_plugin_description_file(controller_interface rrbot_controller.xml)
```



Let’s test it all!

ros2 launch rrbot_bringup rrbot_with_rrbot_controller_array.launch.py

ros2 topic pub /rrbot_controller/commands
example_interfaces/msg/Float64MultiArray "data:
- 0.5
- 0.5"



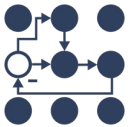
Messages and modifying a controller

example_msgs/Float64MultiArray

```
std_msgs/MultiArrayLayout layout
  std_msgs/MultiArrayDimension[] dim
    string label
    uint32 size
    uint32 stride
  uint32 data_offset
float64[] data
```

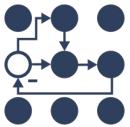
control_msgs/JointJog

```
std_msgs/Header header
string[] joint_names
float64[] displacements
float64[] velocities
float64 duration
```



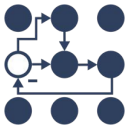
Messages and modifying a controller

```
class RRBotController : public controller_interface::ControllerInterface
{
public:
  ...
protected:
  ...

  using ControllerCommandMsg = control_msgs::msg::JointJog;

  ...
};
```



Messages and modifying a controller

```
controller_interface::return_type RRBotController::update()
{
  auto current_command = input_command_.readFromRT();

  for (size_t i = 0; i < command_interfaces_.size(); ++i) {
    if (!std::isnan((*current_command)->displacements[i])) {
      command_interfaces_[i].set_value((*current_command)->displacements[i]);
    }
  }

...

  return controller_interface::return_type::OK;
}
```



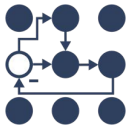
Messages and modifying a controller

ros2 launch rrbot_bringup rrbot_with_rrbot_controller.launch.py

ros2 control list_controllers
ros2 control list_hardware_interfaces

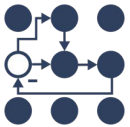
ros2 topic echo /rrbot_controller/state
ros2 topic echo /joint_states

```
ros2 topic pub /rrbot_controller/commands
control_msgs/msg/JointJog "joint_names:
- joint1
- joint2
displacements:
- 0.5
- 0.5"
```



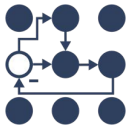
Standard controllers

- joint_state_broadcaster
- diff_drive_controller
- joint_trajectory_controller
- gripper_controllers
- Forwarding controllers for groups of joints
  - position_controllers
  - velocity_controllers
  - effort_controllers

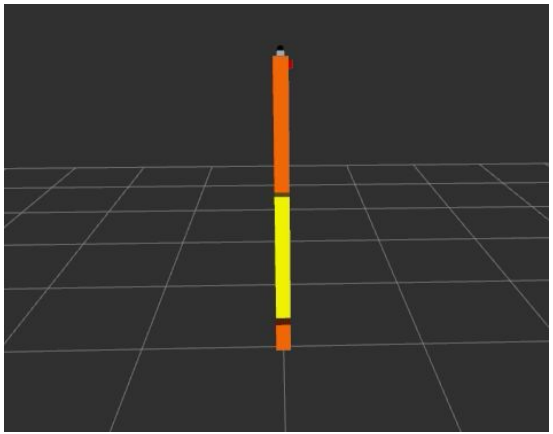


References

- ros_control paper in the Journal of Open Source Software
- ros2_control resources
  - https://ros-controls.github.io/control.ros.org/
  - https://github.com/ros-controls/ros2_control
  - https://github.com/ros-controls/ros2_controllers
  - https://github.com/ros-controls/ros2_control_demos
  - https://github.com/ros-controls/roadmap/blob/master/documentation_resources.md



Thank you!



Denis Štogl, Karsten Knese, Victor Lopez, Jordan Palacios, Olivier Stasse, Mathias Arbo, Colin MacKenzie, Matthew Reynolds, Andy Zelenak, Jafar Abdi, Anas Abou Allaban, Yutaka Kondo, Mateus Amarante, Auguste Bourgois and many more!

PAL ROBOTICS