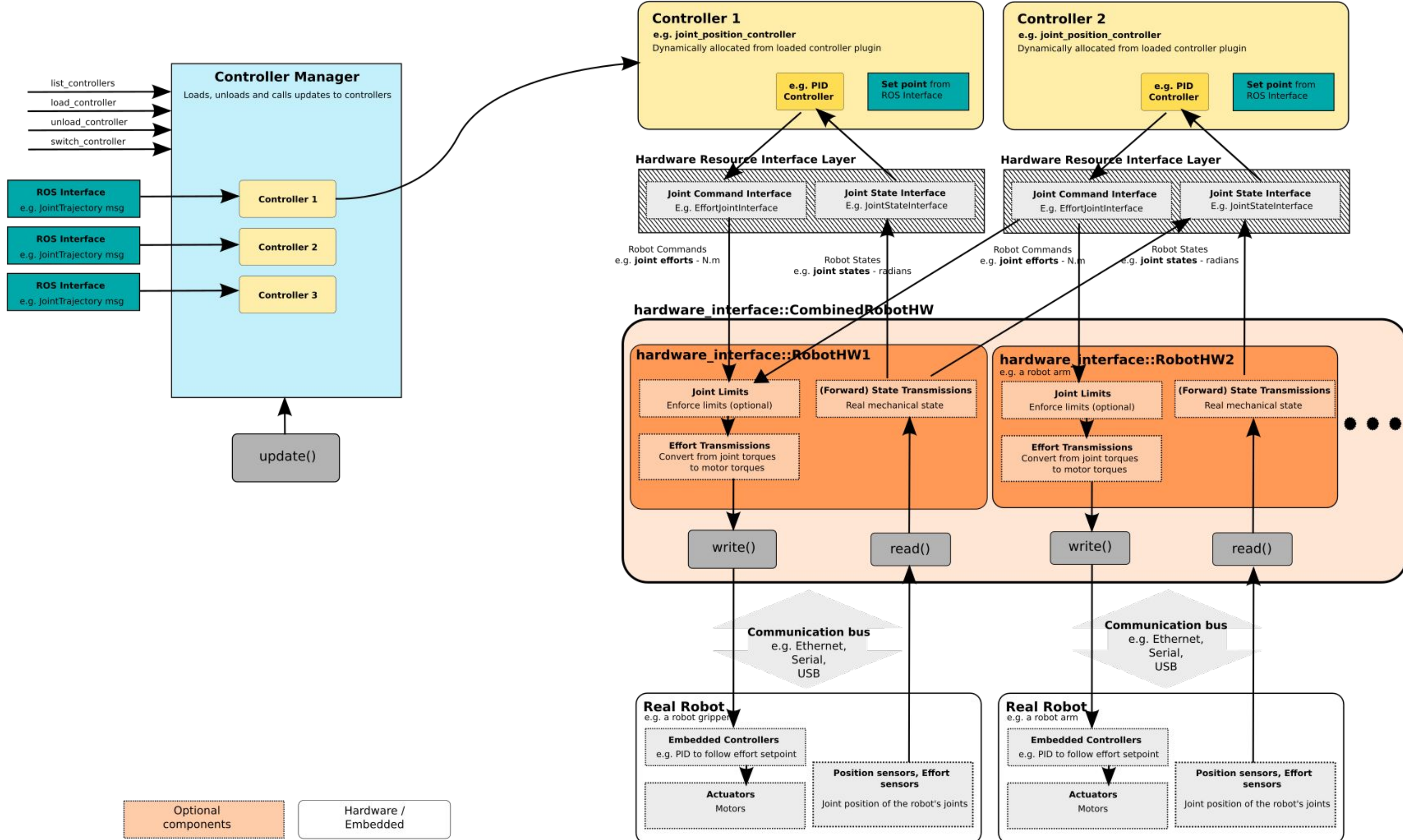# ROS packages

qb robotics®
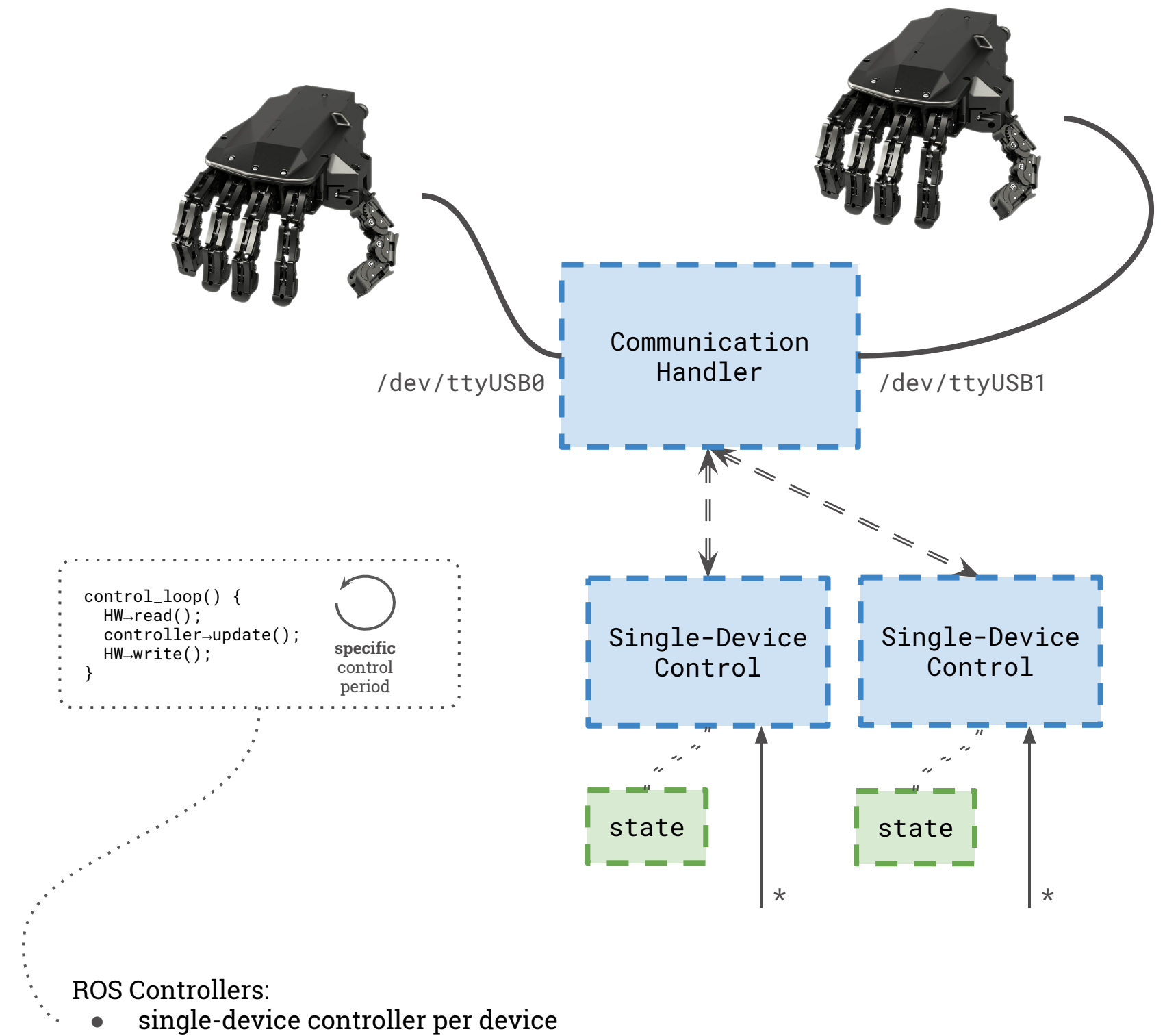
recap: ros_control
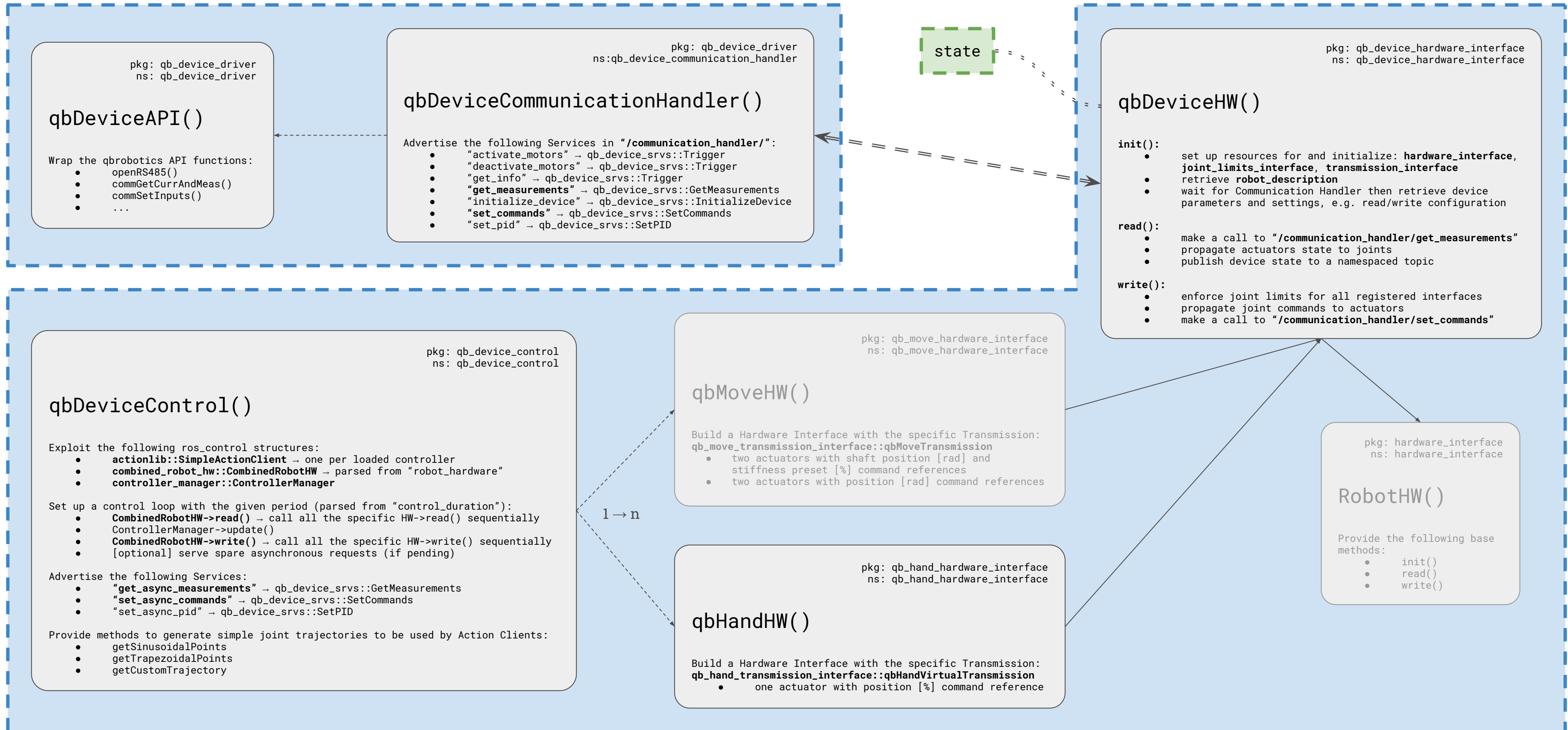
one synchronous control node

Communication Handler

/dev/ttyUSB0   /dev/ttyUSB1

```
control_loop() {
  HW2→read();
  HW1→read();

  controllers→update();

  HW2→write();
  HW1→write();

  // async requests
}
```
global control period

2 DoF Multi-Device Control

state

state

ROS Controllers:
- single-device controller per device
- multi-device controller (for all)
- mixed configuration

*
- slider GUI → basic testing;
- preset waypoints → testing/demo;
- ROS Services → normal application;
- ROS Actions → normal application;
- ROS API → advanced application;

asynchronous control nodes

Communication Handler

/dev/ttyUSB0   /dev/ttyUSB1

```
control_loop() {
  HW→read();
  controller→update();
  HW→write();
}
```
specific control period

Single-Device Control

Single-Device Control

state

state

*   *

ROS Controllers:
- single-device controller per device

ROS Topic	ROS Node	—— wiring	→ user inputs	←-→ ROS Services	= = = ROS Messages

qbrobotics®

3

packages overview

pkg: qb_device_driver
ns: qb_device_driver

qbDeviceAPI()

Wrap the qbrobotics API functions:
- openRS485()
- commGetCurrAndMeas()
- commSetInputs()
- ...

pkg: qb_device_driver
ns:qb_device_communication_handler

qbDeviceCommunicationHandler()

Advertise the following Services in **"/communication_handler/"**:
- "activate_motors" → qb_device_srvs::Trigger
- "deactivate_motors" → qb_device_srvs::Trigger
- "get_info" → qb_device_srvs::Trigger
- **"get_measurements"** → qb_device_srvs::GetMeasurements
- "initialize_device" → qb_device_srvs::InitializeDevice
- **"set_commands"** → qb_device_srvs::SetCommands
- "set_pid" → qb_device_srvs::SetPID

state

pkg: qb_device_hardware_interface
ns: qb_device_hardware_interface

qbDeviceHW()

init():
- set up resources for and initialize: **hardware_interface**, **joint_limits_interface**, **transmission_interface**
- retrieve **robot_description**
- wait for Communication Handler then retrieve device parameters and settings, e.g. read/write configuration

read():
- make a call to **"/communication_handler/get_measurements"**
- propagate actuators state to joints
- publish device state to a namespaced topic

write():
- enforce joint limits for all registered interfaces
- propagate joint commands to actuators
- make a call to **"/communication_handler/set_commands"**

pkg: qb_device_control
ns: qb_device_control

qbDeviceControl()

Exploit the following ros_control structures:
- **actionlib::SimpleActionClient** → one per loaded controller
- **combined_robot_hw::CombinedRobotHW** → parsed from "robot_hardware"
- **controller_manager::ControllerManager**

Set up a control loop with the given period (parsed from "control_duration"):
- **CombinedRobotHW->read()** → call all the specific HW->read() sequentially
- ControllerManager->update()
- **CombinedRobotHW->write()** → call all the specific HW->write() sequentially
- [optional] serve spare asynchronous requests (if pending)

Advertise the following Services:
- **"get_async_measurements"** → qb_device_srvs::GetMeasurements
- **"set_async_commands"** → qb_device_srvs::SetCommands
- "set_async_pid" → qb_device_srvs::SetPID

Provide methods to generate simple joint trajectories to be used by Action Clients:
- getSinusoidalPoints
- getTrapezoidalPoints
- getCustomTrajectory

1→n

pkg: qb_move_hardware_interface
ns: qb_move_hardware_interface

qbMoveHW()

Build a Hardware Interface with the specific Transmission:
qb_move_transmission_interface::qbMoveTransmission
- two actuators with shaft position [rad] and stiffness preset [%] command references
- two actuators with position [rad] command references

pkg: hardware_interface
ns: hardware_interface

RobotHW()

Provide the following base methods:
- init()
- read()
- write()

pkg: qb_hand_hardware_interface
ns: qb_hand_hardware_interface

qbHandHW()

Build a Hardware Interface with the specific Transmission:
qb_hand_transmission_interface::qbHandVirtualTransmission
- one actuator with position [%] command reference

C++ Class   ROS Topic   ROS Node   ----> exploit   — extend   <--> ROS Services   = = ROS Messages

multi-device synchronous setup:

```xml
<launch>
<!-- robot settings -->
<arg name="control_duration" default="0.01" doc="The duration of the control loop [s]."/>
<arg name="robot_hardware" default="[hand1, hand2, hand3]" doc="The robot hardware interface namespaces, e.g. [device1, device2, ...]."/>  ⚠
<arg name="robot_name" default="3dof_chain" doc="The unique robot namespace."/>
<arg name="robot_namespace" default="$(arg robot_name)" doc="The unique robot namespace."/>
<arg name="robot_package" default="qb_chain" doc="The base package name prefix for the robot configurations [urdf, rviz, ...]."/>
<arg name="source_list" default="[control/joint_states]" doc="The joint_states source list for the joint_state_publisher."/>
<!-- read/write settings -->
<arg name="get_currents" default="true" doc="Choose whether or not to retrieve current measurements from the device."/>
<arg name="get_positions" default="true" doc="Choose whether or not to retrieve position measurements from the device."/>
<arg name="get_distinct_packages" default="false" doc="Choose whether or not to retrieve current and position measurements from the device in
two distinct packages."/>
<arg name="max_repeats" default="3" doc="The maximum number of consecutive repetitions to mark retrieved data as corrupted."/>
<arg name="set_commands" default="true" doc="Choose whether or not to send command positions to the device."/>
<arg name="set_commands_async" default="true" doc="Choose whether or not to send commands without waiting for ack."/>
<!-- initialization settings -->
<arg name="activate_on_initialization" default="false" doc="Choose whether or not to activate the motors on node startup."/>
<arg name="rescan_on_initialization" default="false" doc="Choose whether or not to rescan the serial ports on node startup."/>
<!-- launch settings -->
<arg name="standalone" default="false" doc="Choose whether or not to start the Communication Handler."/>
<arg name="use_controller_gui" default="false" doc="Choose whether or not to use the controller GUI."/>
<arg name="use_rviz" default="true" doc="Choose whether or not to use rviz."/>
<arg name="use_waypoints" default="false" doc="Choose whether or not to use the waypoint references."/>


<include file="$(find qb_device_driver)/launch/communication_handler.launch" if="$(arg standalone)"/>

<include file="$(find qb_device_bringup)/launch/device_bringup.launch" pass_all_args="true">
  <arg name="device_id" value="1"/>
  <arg name="device_name" value="hand1"/>  ⚠
  <arg name="device_type" value="qbhand"/>
</include>
<include file="$(find qb_device_bringup)/launch/device_bringup.launch" pass_all_args="true">
  <arg name="device_id" value="2"/>
  <arg name="device_name" value="hand2"/>  ⚠
  <arg name="device_type" value="qbhand"/>
</include>
<include file="$(find qb_device_bringup)/launch/device_bringup.launch" pass_all_args="true">
  <arg name="device_id" value="3"/>
  <arg name="device_name" value="hand3"/>  ⚠
  <arg name="device_type" value="qbhand"/>
  <!-- override global settings -->
  <arg name="get_currents" value="false"/>
  <arg name="get_positions" value="true"/>
  <arg name="max_repeats" value="1"/>
  <arg name="set_commands" value="false"/>
</include>


<include file="$(find qb_device_bringup)/launch/robot_bringup.launch" pass_all_args="true"/>
</launch>
```

single-device setup:

```xml
<launch>
<!-- device info -->
<arg name="device_id" default="1" doc="The ID of the device [1, 128]."/>
<arg name="device_type" value="qbhand" doc="The type of the device [qbhand, qbmove, ...]."/>
<arg name="device_name" default="$(arg device_type)$(arg device_id)" doc="The unique device name used in the yaml controller  ⚠
configurations (also in the urdf if not already specified there)."/>
<!-- robot settings -->
<arg name="control_duration" default="0.01" doc="The duration of the control loop [s]."/>
<arg name="robot_hardware" default="[$(arg device_name)]" doc="The robot hardware interface namespaces, e.g. [device1, device2, ...]."/>  ⚠
<arg name="robot_name" default="$(arg device_type)" doc="The unique robot namespace."/>
<arg name="robot_namespace" default="$(arg device_name)" doc="The unique robot namespace."/>
<arg name="robot_package" default="qb_hand" doc="The base package name prefix for the robot configurations [urdf, rviz, ...]."/>
<arg name="source_list" default="[control/joint_states]" doc="The joint_states source list for the joint_state_publisher."/>
<!-- read/write settings -->
<arg name="get_currents" default="true" doc="Choose whether or not to retrieve current measurements from the device."/>
<arg name="get_positions" default="true" doc="Choose whether or not to retrieve position measurements from the device."/>
<arg name="get_distinct_packages" default="true" doc="Choose whether or not to retrieve current and position measurements from the device in
two distinct packages."/>
<arg name="max_repeats" default="3" doc="The maximum number of consecutive repetitions to mark retrieved data as corrupted."/>
<arg name="set_commands" default="true" doc="Choose whether or not to send command positions to the device."/>
<arg name="set_commands_async" default="true" doc="Choose whether or not to send commands without waiting for ack."/>
<!-- initialization settings -->
<arg name="activate_on_initialization" default="false" doc="Choose whether or not to activate the motors on node startup."/>
<arg name="rescan_on_initialization" default="false" doc="Choose whether or not to rescan the serial ports on node startup."/>
<!-- launch settings -->
<arg name="standalone" default="false" doc="Choose whether or not to start the Communication Handler."/>
<arg name="use_controller_gui" default="false" doc="Choose whether or not to use the controller GUI."/>
<arg name="use_rviz" default="true" doc="Choose whether or not to use rviz."/>
<arg name="use_waypoints" default="false" doc="Choose whether or not to use the waypoint references."/>


<include file="$(find qb_device_driver)/launch/communication_handler.launch" if="$(arg standalone)"/>

<include file="$(find qb_device_bringup)/launch/device_bringup.launch" pass_all_args="true"/>

<include file="$(find qb_device_bringup)/launch/robot_bringup.launch" pass_all_args="true"/>
</launch>
```
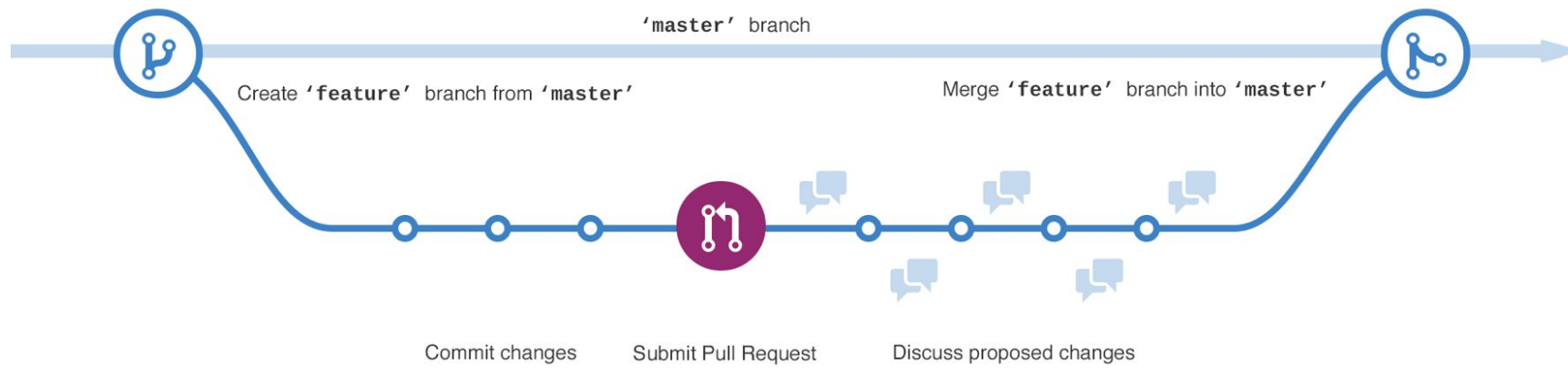
⚠ The use of `combined_robot_hw::CombinedRobotHW` requires the ROS Parameter **robot_hardware** to be set and filled with all the device name list.

qbrobotics®

feature request

'master' branch

Create 'feature' branch from 'master'

Merge 'feature' branch into 'master'

Commit changes   Submit Pull Request   Discuss proposed changes

Useful links:
- [Making a Pull Request](#)
- [5 elements of a perfect pull request](#)
- [A successful Git branching model](#)

bug report / need help?

support@qbrobotics.com ×

bug.reporter@qbusers.com

ROS : bug report <brief>

<details go here>

INVIA