3

ROS Control

Controller interface

Controller manager

transmissions

hardware_interface

→ ros_control is real-time ready.

goals

→ Provide reuse of Control code

→ Provide ready-to-use tools.

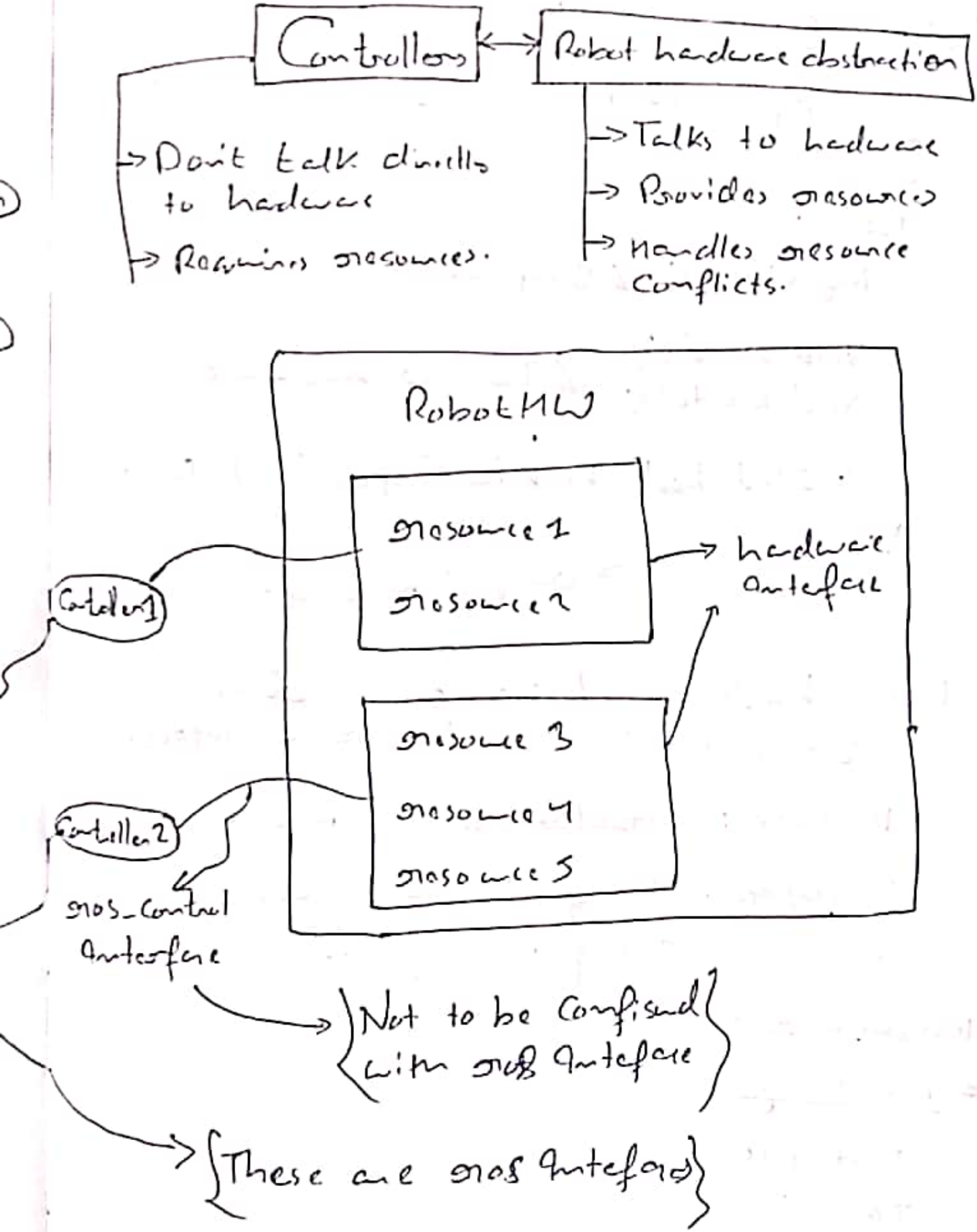
ros-Controls (github).

→ Control_msgs

→ Control_toolbox

→ ros_control (core)

→ ros_controllers.



① Setting up a robot

```
Class MyRobot: public hardware_interface::RobotHW
{
Public:
  MyRobot();  // Setup robot

  Void read();
  Void write();   // Talk to hardware

  Virtual bool checkForConflict(...) const;
};
```

→ only if needed.

⇒ Robot hardware abstraction is done using a package called hardware-interface.

Resource : actuator, joints, sensor

Interface : Set of similar resource

Robot : Set of Interface.

⇒ Resource and Interfaces:

* Read-only

Joint state

IMU

Force-torque sensor

Read-write

Position joint

Velocity joint

Effort joint.

gazebo_ros_control

→ Gazebo plugins for ros_control:

→ Default plugin

→ Populate joint interface from URDF

→ Reads transmission and joint limits specs.

→ Custom plugin

→ up to you

② Controllers

Non real-time operations

Load

→ load + initialize plugin

→ check requisites {hardware resource existence, configuration}

→ setup ROS interface.

* Unload
  → destroy + unload plugin

Real-time safe operations
* start
  → executed before first update
  → resource conflict handling.

* stop
  → executed after last update
  → typical policy: cancel goals.

* update
  → executed periodically

Non real-time operations.

* callbacks
  → non real-time computation
  → executed asynchronously.

Controllers are dynamically loadable plugins.

③ Controller_manager

⇒ Controller_manager has two main purpose:

① Robot resource management

② Controller lifecycle management.

⇒ ROS service API

- Controller lifecycle Management
  - load_controller
  - unload_controller
  - switch_controller
- Queries
  - list_controllers
  - list_controller_types
- Other
  - reload_controller_libraries.

④ transmission_interface

→ Propagate between actuator and joint spaces.

→ Available transmissions:

- Simple reducer
- Four-bar linkage
- Differential.

⑤ joint_limits_interface

⇒ Last line of defense before commanding HW.

⇒ Simple, fast & local strategy.

It Contains

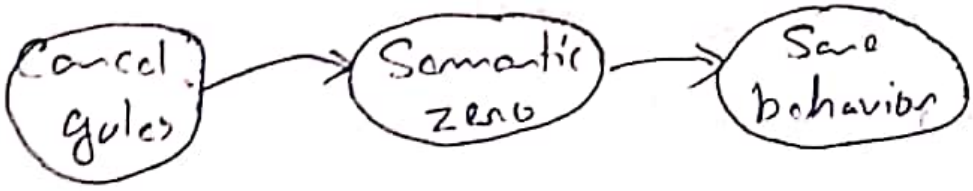
- ① Data structure for representing joint limits
- ② Method of populating them
  - → URDF
  - → yaml
- ③ Methods for enforcing joint limits
  - → soft limits
  - → clamping

⑥ E-stop handling

→ Robot-specific

→ Recovery behavior

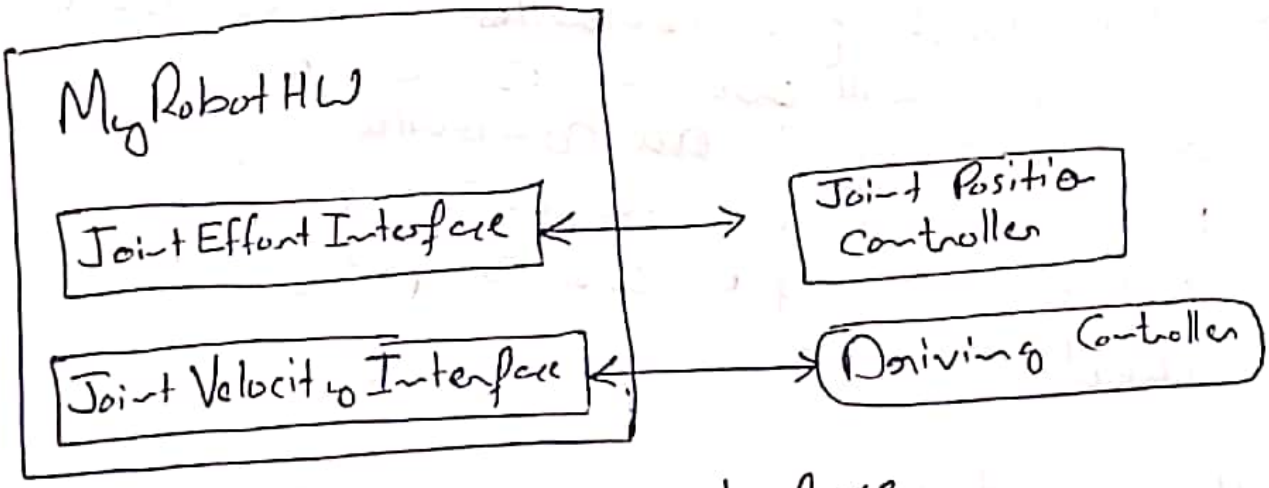
→ Reset controllers.



hardware_interface

① Setting up a new robot

⇒ When you make your robot support one or more of the standard interfaces, you will be able to take advantage of a large library of controllers that works on the standard interface.



② Using an existing interface

Controller manager

⇒ The Controller_manager provides a hard-realtime-compatible loop to control a robot mechanism, which is represented by a hardware_interface::RobotHW instance.

⇒ The Controller_manager provides the infrastructure to load, unload, start and stop controllers.

⇒ When loading a controller, the controller_manager will use the controller name as the root for all controller specific parameters, most importantly, type which identifies which plugin to load.

⇒ The controller manager provides the infrastructure to interact with controllers.

- → Using Launch file
- → From Command line
- → From Node.

① Command-line tools

ⓐ Controller_manager

rosrun controller_manager controller_manager <command> <name1> <name2> ...

- load
- Unload
- start
- stop
- Spawn
- Kill

⇒ To get the state of the controllers use:

rosrun controller_manager controller_manager <command>

- list
- list-types
- reload-libraries
- reload-libraries --restore

ⓑ Spawner

rosrun controller_manager spawner [--stopped] <name1> <name2> ...

⇒ To automatically load and start a set of controllers at once.

⇒ and automatically stop and unload those same controllers at once.

ⓒ Unspawner

⇒ To automatically stop a set of controllers, and restart them later, you can use the unspawner tool.

rosrun controller_manager unspawner <name1> <name2> - - -

ⓓ Controller_group

→ New in melodic.

→ Convenient when you want to switch from a group of controllers to another for some specific purposes.

Example

```
Controller_group:
  Production:
    - Prod_controller_1
    - Prod_controller_2
  development:
    - devel_controller_1
    - devel_controller_2
    - devel_controller_3
```

→ To run the controller_group script:

rosrun controller_manager controller_group
<command> <args>

- list
- Spawn <group>
- Switch <group>

② Creating launch files

```
<launch>
  <node pkg="controller_manager"
        type="spawner"
        args="controller_name1 controller_name2" />
</launch>
```

③ Graphical tools

⇒ The rqt_controller_manager is a rqt plugin that allows to graphically load, unload, start and stop controllers, as well as to display information about loaded controllers.

rosrun rqt_controller_manager rqt_controller_manager

(4) ROS API

⇒ To interact with controllers from another ROS node, the controller manager provides five service calls.

(a) controller_manager/load_controller.

(b) controller_manager/unload_controller

(c) controller_manager/switch_controller

(d) controller_manager/list_controllers.

(e) controller_manager/list_controller_types

(f) controller_manager/reload_controller_libraries.