ROV2026

Generated by Doxygen 1.9.8

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

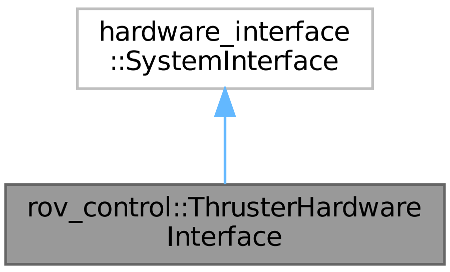
Here is a list of all documented files with brief descriptions:

Chapter 4

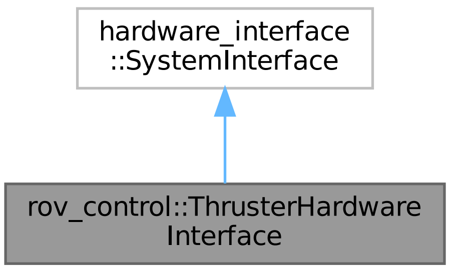
Class Documentation

4.1 rov_control::ThrusterHardwareInterface Class Reference

Inheritance diagram for rov_control::ThrusterHardwareInterface:


hardware_interface ::SystemInterface
rov_control::ThrusterHardware Interface

Collaboration diagram for rov_control::ThrusterHardwareInterface:


hardware_interface ::SystemInterface
rov_control::ThrusterHardware Interface

Public Member Functions

- hardware_interface::CallbackReturn on_init (const hardware_interface::HardwareInfo &info) override

  Initialize the thruster hardware interface with hardware information.
- std::vector< hardware_interface::StateInterface > export_state_interfaces () override

  Export state interfaces for the thruster hardware.
- std::vector< hardware_interface::CommandInterface > export_command_interfaces () override

  Export command interfaces for the thruster hardware.
- hardware_interface::return_type read ()

  Read the current state of the thruster hardware.
- hardware_interface::return_type write ()

  Write the current command values to the thruster hardware.
- hardware_interface::CallbackReturn on_configure ()

  Configure the thruster hardware interface.
- hardware_interface::CallbackReturn on_cleanup ()

  Cleanup the thruster hardware interface.
- hardware_interface::CallbackReturn on_shutdown ()

  Shutdown the thruster hardware interface.
- hardware_interface::CallbackReturn on_activate ()

  Activate the thruster hardware interface.
- hardware_interface::CallbackReturn on_deactivate ()

  Deactivate the thruster hardware interface.
- hardware_interface::CallbackReturn on_error ()

  Handle error state for the thruster hardware interface.

Private Member Functions

- void load_parameters (const hardware_interface::HardwareInfo &info)

  Loads hardware info.
- uint16_t command_to_ticks (double command, uint16_t pwm_min_µs, uint16_t pwm_max_µs, uint16_↩
t pwm_mid_µs, uint16_t pwm_freq_hz)

  Convert a normalized thruster command to PCA9685 ticks.

Private Attributes

- std::vector< double > **command_**
- std::vector< double > **state_**
- uint16_t **pwm_freq_hz_** {50}
- uint16_t **pwm_min_µs_** {1000}
- uint16_t **pwm_max_µs_** {2000}
- uint16_t **pwm_mid_µs_** {1500}

4.1.1 Member Function Documentation

4.1.1.1 command_to_ticks()

```
uint16_t rov_control::ThrusterHardwareInterface::command_to_ticks (
             double command,
             uint16_t pwm_min_μs,
             uint16_t pwm_max_μs,
             uint16_t pwm_mid_μs,
             uint16_t pwm_freq_hz )  [private]
```

Convert a normalized thruster command to PCA9685 ticks.

This function maps a normalized command value (-1 to 1) to a PWM pulse width in microseconds, then converts that pulse width to PCA9685 ticks. The mapping uses the configured minimum, maximum, and midpoint pulse widths.

The mapping formula is: pulse_μs = pwm_mid_μs + command $*$ (command $>=$ 0 ? (pwm_max_μs - pwm_mid_μs) : (pwm_mid_μs - pwm_min_μs))

- For command = 0.0, returns the midpoint pulse width (neutral/stop).
- For command $>$ 0.0, linearly interpolates between midpoint and maximum (forward thrust).
- For command $<$ 0.0, linearly interpolates between midpoint and minimum (reverse thrust).
- The command value is clamped to [-1.0, 1.0] for safety.

For our Blue Robotics Basic ESCs, and with most other ESCs, the mapping is:

- Minimum pulse (e.g., 1100 μs) = full reverse
- Midpoint pulse (e.g., 1500 μs) = stop
- Maximum pulse (e.g., 1900 μs) = full forward

Parameters

command	The normalized thruster command [-1.0, 1.0].
pwm_min_μs	Minimum pulse width in microseconds.
pwm_max_μs	Maximum pulse width in microseconds.
pwm_mid_μs	Midpoint pulse width in microseconds.
pwm_freq_hz	PWM frequency in Hertz.

Returns

The number of PCA9685 ticks corresponding to the command.

4.1.1.2 export_command_interfaces()

```
std::vector< hardware_interface::CommandInterface > rov_control::ThrusterHardwareInterface↩
::export_command_interfaces ( )  [override]
```

Export command interfaces for the thruster hardware.

This method creates and returns a vector of command interfaces for each thruster joint, as defined in the description/urdf/ROV2026.urdf.xacro. Each interface allows the ROS 2 control framework to send effort (command) values to the corresponding thruster.

Returns

std::vector<hardware_interface::CommandInterface> A vector containing the command interfaces for all thrusters.

4.1.1.3 export_state_interfaces()

```
std::vector< hardware_interface::StateInterface > rov_control::ThrusterHardwareInterface↩
::export_state_interfaces ( )  [override]
```

Export state interfaces for the thruster hardware.

This method creates and returns a vector of state interfaces for each thruster joint, as defined in the description/urdf/ROV2026.urdf.xacro. Each interface allows the ROS 2 control framework to read the current effort (state) values from the corresponding thruster.

Returns

std::vector<hardware_interface::StateInterface> A vector containing the state interfaces for all thrusters.

4.1.1.4 load_parameters()

```
void rov_control::ThrusterHardwareInterface::load_parameters (
             const hardware_interface::HardwareInfo & info )  [private]
```

Loads hardware info.

This method reads the PWM-related parameters (frequency, minimum, maximum, and midpoint pulse widths) from the provided hardware_interface::HardwareInfo structure. If a parameter is present in the hardware parameters map, its value is parsed and assigned to the corresponding member variable. If a parameter is not present, the existing value (typically the default) is retained.

Parameters

info	The hardware information structure containing hardware parameters.

4.1.1.5 on_activate()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_activate ( )
```

Activate the thruster hardware interface.

This method is called during the transition from the inactive state to the active state in the ROS 2 lifecycle. It is responsible for preparing the hardware interface to start accepting and executing commands. TODO: Activate or enable hardware.

Returns

hardware_interface::CallbackReturn Returns SUCCESS if activation was successful, ERROR otherwise.

4.1.1.6 on_cleanup()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_cleanup ( )
```

Cleanup the thruster hardware interface.

This method is called during the transition from the inactive state to the unconfigured state in the ROS 2 lifecycle. It resets the command and state vectors to zero and releases any resources allocated during configuration or activation. This prepares the hardware interface for possible reconfiguration or safe shutdown. TODO: Cleanup or release hardware resources in preparation for reconfiguration or shutdown.

Returns

hardware_interface::CallbackReturn Returns SUCCESS if cleanup was successful, ERROR otherwise.

4.1.1.7 on_configure()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_configure ( )
```

Configure the thruster hardware interface.

This method is called during the transition from the unconfigured state to the inactive state in the ROS 2 lifecycle. It resets the command and state vectors to zero and prepares the hardware interface for activation. TODO↩
: Initialization or setup hardware required before activation.

Returns

hardware_interface::CallbackReturn Returns SUCCESS if configuration was successful, ERROR otherwise.

4.1.1.8 on_deactivate()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_deactivate ( )
```

Deactivate the thruster hardware interface.

This method is called during the transition from the active state to the inactive state in the ROS 2 lifecycle. It is responsible for stopping the hardware interface from accepting and executing commands. We reset the state and command vectors to zero to ensure safe deactivation. TODO: Deactivate or disable hardware.

Returns

hardware_interface::CallbackReturn Returns SUCCESS if deactivation was successful, ERROR otherwise.

4.1.1.9 on_error()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_error ( )
```

Handle error state for the thruster hardware interface.

This method is called when the hardware interface enters the error state in the ROS 2 lifecycle. It is responsible for putting the hardware into a safe state, such as stopping all thrusters by setting the command and state vectors to zero, and logging the error. This ensures that the hardware does not continue operating in an unsafe or undefined state after an error has occurred.

Returns

hardware_interface::CallbackReturn Returns ERROR to indicate the hardware is in an error state.

4.1.1.10 on_init()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_init (
            const hardware_interface::HardwareInfo & info )  [override]
```

Initialize the thruster hardware interface with hardware information.

This method is called during the initialization phase of the hardware interface lifecycle. It checks if all required parameters are set and valid by calling the base class implementation. If successful, it initializes the command and state vectors to zero, with a length equal to the number of thruster joints.

Parameters

info	The hardware information structure containing joint and interface definitions.

Returns

hardware_interface::CallbackReturn Returns SUCCESS if initialization was successful, ERROR otherwise.

4.1.1.11 on_shutdown()

```
hardware_interface::CallbackReturn rov_control::ThrusterHardwareInterface::on_shutdown ( )
```

Shutdown the thruster hardware interface.

This method is called during the transition to the finalized (shutdown) state in the ROS 2 lifecycle. It is responsible for safely stopping all thrusters, releasing hardware resources, and resetting the command and state vectors to zero. This ensures the hardware is left in a safe state before the node is destroyed or the process exits. TODO: Shutdown hardware.

Returns

hardware_interface::CallbackReturn Returns SUCCESS if shutdown was successful, ERROR otherwise.

4.1.1.12 read()

```
hardware_interface::return_type rov_control::ThrusterHardwareInterface::read ( )
```

Read the current state of the thruster hardware.

This method updates the internal state vector to reflect the current state of each thruster. Since the ESCs do not provide feedback, the state is set to match the last command sent. This allows the ROS 2 control framework to assume the thrusters are following the commanded values.

Returns

hardware_interface::return_type Returns OK after updating the state.

4.1.1.13 write()

```
hardware_interface::return_type rov_control::ThrusterHardwareInterface::write ( )
```

Write the current command values to the thruster hardware.

This method sends the command values stored in the internal command vector to the thruster hardware. For each thruster, the corresponding command value is converted to PCA9685 ticks using the configured PWM parameters (min, max, mid pulse widths, and frequency). The pulse is always set to start at the beginning of the PWM cycle (on=0), and the width is set by the calculated ticks value (off=ticks).

The function calls pca9685_write_channel() for each thruster channel:

- The third argument (on=0) means the PWM pulse starts at the beginning of the cycle.
- The fourth argument (off=ticks) sets the pulse width. If writing to any channel fails, an error is logged and the function returns ERROR. Otherwise, it logs the command sent to each thruster and returns OK.

Returns

hardware_interface::return_type Returns OK after sending the commands, or ERROR if any write fails.

The documentation for this class was generated from the following file:

- src/rov_control/include/rov_control/thruster_interface.hpp

Chapter 5

File Documentation

5.1 thruster_interface.hpp

```
#ifndef ROV_CONTROL_THRUSTER_INTERFACE_HPP
#define ROV_CONTROL_THRUSTER_INTERFACE_HPP

#pragma once
#include "hardware_interface/system_interface.hpp"
#include "hardware_interface/hardware_info.hpp"
#include "rclcpp/macros.hpp"
#include <vector>

namespace rov_control
{
  // Hardware interface for controlling thrusters, inheriting from SystemInterface
  class ThrusterHardwareInterface : public hardware_interface::SystemInterface
  {
  public:
    // Macro to define shared pointer stuff
    RCLCPP_SHARED_PTR_DEFINITIONS(ThrusterHardwareInterface)


    hardware_interface::CallbackReturn on_init(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 read(); // Add override when needed.

    hardware_interface::return_type write(); // Add override when needed.

    hardware_interface::CallbackReturn on_configure();

    hardware_interface::CallbackReturn on_cleanup();

    hardware_interface::CallbackReturn on_shutdown();

    hardware_interface::CallbackReturn on_activate();

    hardware_interface::CallbackReturn on_deactivate();

    hardware_interface::CallbackReturn on_error();

  private:
    // Stores the latest command values for each thruster
    std::vector<double> command_;

    // Stores the latest state values for each thruster
    std::vector<double> state_;

    uint16_t pwm_freq_hz_{50}; // PWM frequency in Hz

    uint16_t pwm_min_µs_{1000}; // Minimum pulse width in microseconds
    uint16_t pwm_max_µs_{2000}; // Maximum pulse width in microseconds
    uint16_t pwm_mid_µs_{1500}; // Mid pulse width in microseconds

    void load_parameters(const hardware_interface::HardwareInfo &info);

    uint16_t command_to_ticks(
        double command,
        uint16_t pwm_min_µs,
        uint16_t pwm_max_µs,
        uint16_t pwm_mid_µs,
        uint16_t pwm_freq_hz);
  };
}

#endif // ROV_CONTROL_THRUSTER_INTERFACE_HPP
```

Index