Bioscara

Generated by Doxygen 1.9.8

Chapter 1

Documentation

This documentation currently documents the following:

- The joint firmware in the `/Arduino` directory
- The interfacing library used for communicating with the joins in the `/ROS2` directory.

Chapter 2

Todo List

- **Member Joint::read (const stp_reg_t reg, T &data, u_int8_t &flags)**
  Implement a return code for read only functions
  - Implement clearStall function
- **Member Joint_comms::addJoint (const int address, const std::string name, const float gearRatio, const float offset)**
  Measure joint ranges
  - Investigate if possible to make independent of homing
- **Member Joint_comms::checkOrientations (std::vector< float > angle_v)**
  Only execute if not performed before
  - save in private flag and inhibit movement if this has not been executed.
- **Member loop (void)**
  why are BIT2 and BIT3 constantly checked and set? Would it be sufficient to do this only invoking the actual functions?

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

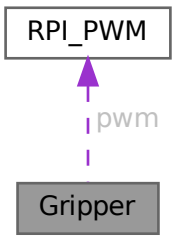
Class Documentation

5.1 Gripper Class Reference

Gripper object to interact with the robot gripper.

`#include <mGripper.h>`

Collaboration diagram for Gripper:



Public Member Functions

- Gripper (void)
- int init (void)

  Placeholder, does nothing.
- int deinit (void)

  Placeholder, does nothing.
- int enable (void)

  Prepares the servo for use.
- int disable (void)

  Disables the servo.
- int setPosition (float width)

  Sets the gripper width in mm from the closed position.

Private Attributes

- RPI_PWM pwm

5.1.1 Detailed Description

Gripper object to interact with the robot gripper.

The gripper must be a PWM controlled servo.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 Gripper()

```
Gripper::Gripper (
            void  )
```

5.1.3 Member Function Documentation

5.1.3.1 deinit()

```
int Gripper::deinit (
            void  )
```

Placeholder, does nothing.

Returns

0

5.1.3.2 disable()

```
int Gripper::disable (
            void  )
```

Disables the servo.

Stops the servo and disables the PWM generation.

Returns

non-zero error code.

5.1.3.3 enable()

```
int Gripper::enable (
            void  )
```

Prepares the servo for use.

Starts the PWM generation but does not set a position. Must be called before a position is set. The PWM pin is GPIO18. PWM chip is 0, channel 0. ∗

Returns

non-zero error code.

5.1.3.4 init()

```
int Gripper::init (
            void  )
```

Placeholder, does nothing.

Returns

0

5.1.3.5 setPosition()

```
int Gripper::setPosition (
            float width )
```

Sets the gripper width in mm from the closed position.

Arguments outside the allowed range are bounded to limit.

Parameters

width	width in mm. 30 - 85 mm are currently allowed. With a new gripper this should be changed.

5.1.4 Member Data Documentation

5.1.4.1 pwm

```
RPI_PWM Gripper::pwm  [private]
```

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

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mGripper.h
- ROS2/ros2_scara_ws/src/joint_communication/src/mGripper.cpp

5.2 Joint Class Reference

Representing a single joint on the I2C bus.

`#include <mJoint.h>`

Public Member Functions

- Joint (const int address, const std::string name, const float gearRatio, const float offset)
- int init (void)
- int deinit (void)
- int printInfo (void)
- int getPosition (float &angle)
- int setPosition (float angle)
- int getVelocity (float &degps)
- int setVelocity (float degps)
- int checkOrientation (float angle=10.0)
- int enable (u_int8_t driveCurrent, u_int8_t holdCurrent)

  Initialize the joint and engages motor.
- int disable (void)

  disenganges the joint motor without closing i2c handle
- int home (u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current)

  Homes the motor.
- int stop (bool mode)

  Stops the motor.
- int disableCL (void)

  Disables the Closed-Loop PID Controller.
- int setDriveCurrent (u_int8_t current)

  Set the Drive Current.
- int setHoldCurrent (u_int8_t current)

  Set the Hold Current.
- int setBrakeMode (u_int8_t mode)

  Set Brake Mode.
- int getStall (u_int8_t &stall)

  checks if the motor is stalled
- int enableStallguard (u_int8_t sensitivity)

  Enable encoder stall detection. A detected stall can be reset by homeing.
- int getIsHomed (u_int8_t &homed)

  retrieves the status flags from the joint and checks if the joint is homed.
- int getIsHomed (void)

  retrieves the status flags from the joint.
- bool isHomed (void)

  Get the isHomed state variable saved locally.
- int getIsSetup (u_int8_t &setup)

  checks if the joint is setup from the joint
- int getIsSetup (void)

  checks if the joint is setup from the joint
- bool isSetup (void)
- int moveSteps (int32_t steps)
- int checkCom (void)
- u_int8_t getFlags (void)

Public Attributes

- std::string name

Private Types

- enum stp_reg_t {
  PING = 0x0f , SETUP = 0x10 , SETRPM = 0x11 , GETDRIVERRPM = 0x12 ,
  MOVESTEPS = 0x13 , MOVEANGLE = 0x14 , MOVETOANGLE = 0x15 , GETMOTORSTATE = 0x16 ,
  RUNCOTINOUS = 0x17 , ANGLEMOVED = 0x18 , SETCURRENT = 0x19 , SETHOLDCURRENT = 0x1A ,
  SETMAXACCELERATION = 0x1B , SETMAXDECELERATION = 0x1C , SETMAXVELOCITY = 0x1D ,
  ENABLESTALLGUARD = 0x1E ,
  DISABLESTALLGUARD = 0x1F , CLEARSTALL = 0x20 , ISSTALLED = 0x21 , SETBRAKEMODE = 0x22 ,
  ENABLEPID = 0x23 , DISABLEPID = 0x24 , ENABLECLOSEDLOOP = 0x25 , DISABLECLOSEDLOOP = 0x26 ,
  SETCONTROLTHRESHOLD = 0x27 , MOVETOEND = 0x28 , STOP = 0x29 , GETPIDERROR = 0x2A ,
  CHECKORIENTATION = 0x2B , GETENCODERRPM = 0x2C , HOME = 0x2D , ISHOMED = 0x2E ,
  ISSETUP = 0x2F }

  register and command definitions

Private Member Functions

- template<typename T >
  int read (const stp_reg_t reg, T &data, u_int8_t &flags)

  Wrapper function to request data from the I2C slave.
- template<typename T >
  int write (const stp_reg_t reg, T data, u_int8_t &flags)

  Wrapper function to send command to the I2C slave.

Private Attributes

- u_int8_t flags = 0x00

  State flags transmitted with every I2C transaction.
- u_int8_t ishomed = 0

  flag if homed
- u_int8_t issetup = 0

  flag is setup
- int address

  I2C adress.
- float gearRatio = 1

  gear ratio from encoder units to joint units
- float offset = 0

  offset in degrees or mm from encoder zero to joint zero.
- int handle = -1

  I2C bus handle.

5.2.1 Detailed Description

Representing a single joint on the I2C bus.

5.2.2 Member Enumeration Documentation

5.2.2.1 stp_reg_t

```
enum Joint::stp_reg_t  [private]
```

register and command definitions

a register can be read (R) or written (W), each register has a size in bytes. The payload can be split into multiple values or just be a single value. Note that not all functions are implemented.

Enumerator

PING	R; Size: 1; [(char) ACK].
SETUP	W; Size: 2; [(uint8) holdCurrent, (uint8) driveCurrent].
SETRPM	W; Size: 4; [(float) RPM].
GETDRIVERRPM	
MOVESTEPS	W; Size: 4; [(int32) steps].
MOVEANGLE	
MOVETOANGLE	W; Size: 4; [(float) degrees].
GETMOTORSTATE	
RUNCOTINOUS	
ANGLEMOVED	R; Size: 4; [(float) degrees].
SETCURRENT	W; Size: 1; [(uint8) driveCurrent].
SETHOLDCURRENT	W; Size: 1; [(uint8) holdCurrent].
SETMAXACCELERATION	
SETMAXDECELERATION	
SETMAXVELOCITY	
ENABLESTALLGUARD	W; Size: 1; [(uint8) threshold].
DISABLESTALLGUARD	
CLEARSTALL	
ISSTALLED	R; Size: 1; [(uint8) isStalled].
SETBRAKEMODE	W; Size: 1; [(uint8) mode].
ENABLEPID	
DISABLEPID	
ENABLECLOSEDLOOP	
DISABLECLOSEDLOOP	W; Size: 1; [(uint8) 0].
SETCONTROLTHRESHOLD	
MOVETOEND	
STOP	W; Size: 1; [(uint8) mode].
GETPIDERROR	
CHECKORIENTATION	W; Size: 4; [(float) degrees].
GETENCODERRPM	R; Size: 4; [(float) RPM].
HOME	W; Size: 4; [(uint8) current, (int8) sensitivity, (uint8) speed, (uint8) direction].
ISHOMED	R; Size: 1; [(uint8) isStalled].
ISSETUP	R; Size: 1; [(uint8) isStalled].

5.2.3 Constructor & Destructor Documentation

5.2.3.1 Joint()

```
Joint::Joint (
            const int address,
            const std::string name,
            const float gearRatio,
            const float offset )
```

5.2.4 Member Function Documentation

5.2.4.1 checkCom()

```
int Joint::checkCom (
            void  )
```

5.2.4.2 checkOrientation()

```
int Joint::checkOrientation (
            float angle = 10.0 )
```

5.2.4.3 deinit()

```
int Joint::deinit (
            void  )
```

5.2.4.4 disable()

```
int Joint::disable (
            void  )
```

disenganges the joint motor without closing i2c handle

Returns

error code.

5.2.4.5 disableCL()

```
int Joint::disableCL (
            void  )
```

Disables the Closed-Loop PID Controller.

Returns

error code.

5.2.4.6 enable()

```
int Joint::enable (
            u_int8_t driveCurrent,
            u_int8_t holdCurrent )
```

Initialize the joint and engages motor.

Parameters

driveCurrent	drive current in 0-100 % of 2.5A output (check uStepper doc.)
holdCurrent	hold current in 0-100 % of 2.5A output (check uStepper doc.)

Returns

error code.

5.2.4.7 enableStallguard()

```
int Joint::enableStallguard (
            u_int8_t sensitivity )
```

Enable encoder stall detection. A detected stall can be reset by homeing.

Parameters

sensitivity	Encoder stalldetect sensitivity - From -100 to 10 where lower number is less sensitive and higher is more sensitive

5.2.4.8 getFlags()

```
u_int8_t Joint::getFlags (
            void  )
```

get driver state flags

Returns

flags.

5.2.4.9 getIsHomed() [1/2]

```
int Joint::getIsHomed (
            u_int8_t & homed )
```

retrieves the status flags from the joint and checks if the joint is homed.

Parameters

homed	not homed: 0, homed: 1

Returns

error code.

5.2.4.10 getIsHomed() [2/2]

```
int Joint::getIsHomed (
            void  )
```

retrieves the status flags from the joint.

This overload does not return the value of isHomed variable, use Joint::isHomed() instead.

Returns

error code.

5.2.4.11 getIsSetup() [1/2]

```
int Joint::getIsSetup (
            u_int8_t & setup )
```

checks if the joint is setup from the joint

Parameters

setup	not setup: 0, setup: 1

Returns

error code.

5.2.4.12 getIsSetup() [2/2]

```
int Joint::getIsSetup (
            void  )
```

checks if the joint is setup from the joint

Returns

error code.

5.2.4.13 getPosition()

```
int Joint::getPosition (
            float & angle )
```

5.2.4.14 getStall()

```
int Joint::getStall (
            u_int8_t & stall )
```

checks if the motor is stalled

Parameters

stall	not stalled: 0, stalled: 1

Returns

error code.

5.2.4.15 getVelocity()

```
int Joint::getVelocity (
            float & degps )
```

5.2.4.16 home()

```
int Joint::home (
            u_int8_t direction,
            u_int8_t rpm,
            u_int8_t sensitivity,
            u_int8_t current )
```

Homes the motor.

Parameters

direction	CCW: 0, CW: 1.
rpm	speed of motor in rpm $>$ 10.
sensitivity	Encoder pid error threshold 0 to 255.
current	homeing current, determines how easy it is to stop the motor and thereby provoke a stall

Returns

error code.

5.2.4.17 init()

```
int Joint::init (
            void  )
```

5.2.4.18 isHomed()

```
bool Joint::isHomed (
            void  )
```

Get the isHomed state variable saved locally.

To retrieve the actual state call Joint::getIsHomed()

Returns

local isHomed state variable.

5.2.4.19 isSetup()

```
bool Joint::isSetup (
            void  )
```

Returns

the isSetup state variable.

5.2.4.20 moveSteps()

```
int Joint::moveSteps (
            int32_t steps )
```

5.2.4.21 printInfo()

```
int Joint::printInfo (
            void  )
```

5.2.4.22 read()

```
template<typename T >
int Joint::read (
            const stp_reg_t reg,
            T & data,
            u_int8_t & flags )  [private]
```

Wrapper function to request data from the I2C slave.

Allocates a buffer of size sizeof(T) + RFLAGS_SIZE. invokes readFromI2CDev(), and copies the received payload to *data* and the transmisison flags to *flags*. See Joint::flags for details.

Todo
- Implement a return code for read only functions
- Implement clearStall function

Template Parameters

T	Datatype of value to be transmitted

Parameters

reg	stp_reg_t register to read
data	reference to store payload.
flags	reference to a byte which stores the return flags

Returns

0 on OK, negative on error

5.2.4.23 setBrakeMode()

```
int Joint::setBrakeMode (
            u_int8_t mode )
```

Set Brake Mode.

Parameters

mode	Freewheel: 0, Coolbrake: 1, Hardbrake: 2

Returns

error code.

5.2.4.24 setDriveCurrent()

```
int Joint::setDriveCurrent (
            u_int8_t current )
```

Set the Drive Current.

Warning

This function is unreliable and not well tested. Use enable() instead!

Parameters

current	0% - 100% of driver current

Returns

error code.

5.2.4.25 setHoldCurrent()

```
int Joint::setHoldCurrent (
            u_int8_t current )
```

Set the Hold Current.

Warning

This function is unreliable and not well tested. Use enable() instead!

Parameters

current	0% - 100% of driver current

Returns

error code.

5.2.4.26 setPosition()

```
int Joint::setPosition (
            float angle )
```

5.2.4.27 setVelocity()

```
int Joint::setVelocity (
            float degps )
```

5.2.4.28 stop()

```
int Joint::stop (
            bool mode )
```

Stops the motor.

Note

When stopping the motor in soft mode, wait sufficiently long until the motor has stopped. Since the stop() function in the motor controller is blocking. Continously checking the busy flag also might interfere with the stop() function on the controller side.

Parameters

mode	Hard: 0, Soft: 1

Returns

error code.

5.2.4.29 write()

```
template<typename T >
int Joint::write (
            const stp_reg_t reg,
            T data,
            u_int8_t & flags )  [private]
```

Wrapper function to send command to the I2C slave.

Allocates a buffer of size sizeof(T) + RFLAGS_SIZE. Copyies *data* to the buffer and invokes writeToI2CDev(). The flags received from the transaction are copied to *flags*. The flags are described in Joint::read().

Template Parameters

T	Datatype of value to be transmitted

Parameters

reg	stp_reg_t command to execute
data	payload to transmit. It is the users responsibility to populate the right amount of data for the relevant register
flags	reference to a byte which stores the return flags

Returns

0 on OK, negative on error

5.2.5 Member Data Documentation

5.2.5.1 address

```
int Joint::address  [private]
```

I2C adress.

5.2.5.2 flags

```
u_int8_t Joint::flags = 0x00  [private]
```

State flags transmitted with every I2C transaction.

The transmission flags purpose are to transmit the joints current state. Note: They can not be used as error indication of the execution of a transmitted write command, since commands are executed after the I2C transaction is completed. The status flags are one byte with following structure:

BIT7	BIT6	BIT5	BIT4	BIT3	BIT2	BIT1	BIT0
reserved	reserved	reserved	reserved	SETUP	HOMED	BUSY	STALL

STALL is set if a stall from the stall detection is sensed and the joint is stopped. The flag is cleared when the joint is homed.
BUSY is set if the slave is busy processing a previous command.
HOMED is set if the joint is homed. Movement is only allowed if this flag is clear
SETUP is set if the joint is setup after calling Joint::enable()

5.2.5.3 gearRatio

```
float Joint::gearRatio = 1  [private]
```

gear ratio from encoder units to joint units

5.2.5.4 handle

```
int Joint::handle = -1  [private]
```

I2C bus handle.

5.2.5.5 ishomed

```
u_int8_t Joint::ishomed = 0  [private]
```

flag if homed

5.2.5.6 issetup

```
u_int8_t Joint::issetup = 0  [private]
```

flag is setup

5.2.5.7 name

```
std::string Joint::name
```

5.2.5.8 offset

```
float Joint::offset = 0  [private]
```

offset in degrees or mm from encoder zero to joint zero.

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

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.h
- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.hpp
- ROS2/ros2_scara_ws/src/joint_communication/src/mJoint.cpp

5.3 Joint_comms Class Reference

Communication object for all joints.

```
#include <mJointCom.h>
```

Public Member Functions

- Joint_comms (void)
- ~Joint_comms ()
- int init (void)

  Initializes all joints.
- int deinit (void)

  Frees all joints from the I2C bus.
- void addJoint (const int address, const std::string name, const float gearRatio, const float offset)

  add Joints.
- int enables (std::vector< u_int8_t > driveCurrent_v, std::vector< u_int8_t > holdCurrent_v)

  Engages the joints.
- int enables (u_int8_t driveCurrent, u_int8_t holdCurrent)

  Engages the joints with the same current settings for all joints.
- int disables (void)

  Disenganges the joint without closing i2c handle.
- int home (std::string name, u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current)

  Executes the homing sequence of a joint.
- int getPositions (std::vector< float > &angle_v)

  Get the positions of all joints.
- int setPositions (std::vector< float > angle_v)

  Set the positions of all joints.
- int getVelocities (std::vector< float > &degps_v)

  Get the velocities of all joints.
- int setVelocities (std::vector< float > degps_v)

  Set the velocities of all joints.
- int checkOrientations (std::vector< float > angle_v)

  Sequentially checks the orientations of each joint.
- int checkOrientations (float angle=10.0)

  Overload to use standard angle of 10 degrees.
- int stops (bool mode)

  Stops the motors.
- int disableCLs (void)

  Disables the Closed-Loop PID Controllers.
- int setDriveCurrents (std::vector< u_int8_t > current)

  Set the drive Currents.
- int setDriveCurrents (u_int8_t current)

  Overload to set all drive currents to the same value.
- int setHoldCurrents (std::vector< u_int8_t > current)

  Set the Hold Currents.
- int setHoldCurrents (u_int8_t current)

  Overload to set all hold currents to the same value.
- int setBrakeModes (u_int8_t mode)

  Set Brake Modes.
- int enableStallguards (std::vector< u_int8_t > thresholds)

  Enable encoder stall detection.

Public Attributes

- std::vector< Joint > joints

  Internal vector storing the Joint objects.

5.3.1 Detailed Description

Communication object for all joints.

CLass handling interfacing with the joints.

5.3.2 Constructor & Destructor Documentation

5.3.2.1 Joint_comms()

```
Joint_comms::Joint_comms (
            void  )
```

5.3.2.2 ~Joint_comms()

```
Joint_comms::~Joint_comms ( )
```

5.3.3 Member Function Documentation

5.3.3.1 addJoint()

```
void Joint_comms::addJoint (
            const int address,
            const std::string name,
            const float gearRatio,
            const float offset )
```

add Joints.

Appends a joint to internal vector.

Parameters

addresses	1-byte I2C device adress (0x11 ... 0x14) for J1 ... J4
names	string device name for output logs
gearRatio	gear ratio of joint. This is used to transform position and velocity commands in joint units to the stepper units. Signed: sign depends if homed CW or CCW. J1: 35; J2: -360/4 (4 mm per revolution); J3: 24; J4: 12;
offset	offset between encoder zero and joint zero (in joint units). J1: TBD; J2: -TBD (negative because homed at top); J3: TBD; J4: TBD;

Todo
- Measure joint ranges
- Investigate if possible to make independent of homing

5.3.3.2 checkOrientations() [1/2]

```
int Joint_comms::checkOrientations (
            float angle = 10.0 )
```

Overload to use standard angle of 10 degrees.

Returns

error code.

5.3.3.3 checkOrientations() [2/2]

```
int Joint_comms::checkOrientations (
            std::vector< float > angle_v )
```

Sequentially checks the orientations of each joint.

This function must only be called after the joint has been powered down. This function must be called after the joint has been enabled with enables() and before any movement.

Parameters

angle_v	vector of degrees to rotate to check the orientation. Should be small values of a few degrees.

Returns

error code.

Todo

- Only execute if not performed before
- save in private flag and inhibit movement if this has not been executed.

5.3.3.4 deinit()

```
int Joint_comms::deinit (
            void  )
```

Frees all joints from the I2C bus.

Deinitializes all joints by removing them from the I2C bus.

Returns

0 on success, non-zero otherwise

5.3.3.5 disableCLs()

```
int Joint_comms::disableCLs (
            void  )
```

Disables the Closed-Loop PID Controllers.

Returns

error code.

5.3.3.6 disables()

```
int Joint_comms::disables (
            void  )
```

Disenganges the joint without closing i2c handle.

Call this function when the joint should be in freedrive mode.

Returns

error code.

5.3.3.7 enables() [1/2]

```
int Joint_comms::enables (
            std::vector< u_int8_t > driveCurrent_v,
            std::vector< u_int8_t > holdCurrent_v )
```

Engages the joints.

Sets the drive and hold currents for each joint and engages the motor. Currents are in percent of driver max. output (2.5A, check with datasheet)

Parameters

driveCurrent_v	vector of drive currents 0-100. the i'th vector entry sets the current for the i'th added joint.
holdCurrent_v	vector of hold currents 0-100. the i'th vector entry sets the current for the i'th added joint.

Returns

error code.

5.3.3.8 enables() [2/2]

```
int Joint_comms::enables (
```

```
            u_int8_t driveCurrent,
            u_int8_t holdCurrent )
```

Engages the joints with the same current settings for all joints.

In this overload the same drive and hold currents are written to every joint.

Parameters

driveCurrent	drive current 0-100.
holdCurrent	hold current 0-100.

Returns

error code.

5.3.3.9 enableStallguards()

```
int Joint_comms::enableStallguards (
            std::vector< u_int8_t > thresholds )
```

Enable encoder stall detection.

If the PID error exceeds the set threshold a stall is triggered and the motor disabled. A detected stall can be reset by homeing.

Parameters

thresholds	Vector of thresholds. 0 - 255 where lower is more sensitive.

5.3.3.10 getPositions()

```
int Joint_comms::getPositions (
            std::vector< float > & angle_v )
```

Get the positions of all joints.

The current positions of all joints are returned. The units are degrees and mm for revolute and prismatic joints respectively.

Parameters

angle_v	Reference to allocated vector of appropriate size to hold all joint positions.

Returns

error code.

5.3.3.11 getVelocities()

```
int Joint_comms::getVelocities (
            std::vector< float > & degps_v )
```

Get the velocities of all joints.

The current velocities of all joints are returned. The units are degrees/s and mm/s for revolute and prismatic joints respectively.

Parameters

degps_v	Reference to allocated vector of appropriate size to hold all joint velocities.

Returns

error code.

5.3.3.12 home()

```
int Joint_comms::home (
            std::string name,
            u_int8_t direction,
            u_int8_t rpm,
            u_int8_t sensitivity,
            u_int8_t current )
```

Executes the homing sequence of a joint.

The joint will drive in the specified direction with the specified speed until a resistance which drives the current above the specified threshold is encountered. At this point the stepper stops and zeros the encoder.

Parameters

name	joint name.
direction	CCW: 0, CW: 1.
rpm	speed of motor in rpm $>$ 10
sensitivity	PID error threshold, 0 to 255.
current	homeing current, determines how easy it is to stop the motor and thereby provoke a stall

Returns

error code.

5.3.3.13 init()

```
int Joint_comms::init (
            void  )
```

Initializes all joints.

Warning

Add some joints using addJoint() before calling this function. Iterates over all joints and initializes them on the I2C bus and tests if they are responsive.

Returns

0 on success, non-zero otherwise

5.3.3.14 setBrakeModes()

```
int Joint_comms::setBrakeModes (
            u_int8_t mode )
```

Set Brake Modes.

Applies the same brake modes to all joints. usefull to disengage all motors.

Parameters

mode	Freewheel: 0, Coolbrake: 1, Hardbrake: 2

Returns

error code.

5.3.3.15 setDriveCurrents() [1/2]

```
int Joint_comms::setDriveCurrents (
            std::vector< u_int8_t > current )
```

Set the drive Currents.

Warning

This function is unreliable and not well tested. Use enables() instead!

Parameters

current	0% - 100% of driver current

Returns

error code.

5.3.3.16 setDriveCurrents() [2/2]

```
int Joint_comms::setDriveCurrents (
            u_int8_t current )
```

Overload to set all drive currents to the same value.

Warning

This function is unreliable and not well tested. Use enables() instead!

Parameters

current	0% - 100% of driver current

Returns

error code.

5.3.3.17 setHoldCurrents() [1/2]

```
int Joint_comms::setHoldCurrents (
            std::vector< u_int8_t > current )
```

Set the Hold Currents.

Warning

This function is unreliable and not well tested. Use enables() instead!

Parameters

current	0% - 100% of driver current

Returns

error code.

5.3.3.18 setHoldCurrents() [2/2]

```
int Joint_comms::setHoldCurrents (
            u_int8_t current )
```

Overload to set all hold currents to the same value.

Warning

This function is unreliable and not well tested. Use enables() instead!

Parameters

current	0% - 100% of driver current

Returns

error code.

5.3.3.19 setPositions()

```
int Joint_comms::setPositions (
            std::vector< float > angle_v )
```

Set the positions of all joints.

Set new target positons of all joints. The units are degrees and mm for revolute and prismatic joints respectively.

Parameters

angle_v	Vector of new target positions.

Returns

error code.

5.3.3.20 setVelocities()

```
int Joint_comms::setVelocities (
            std::vector< float > degps_v )
```

Set the velocities of all joints.

Set new target positons of all joints. The units are degrees and mm for revolute and prismatic joints respectively.

Parameters

degps_v	Vector of new target velocities.

Returns

error code.

5.3.3.21 stops()

```
int Joint_comms::stops (
            bool mode )
```

Stops the motors.

Stops all motors either soft or hard.

Parameters

mode	Hard: 0, Soft: 1

Returns

error code.

5.3.4 Member Data Documentation

5.3.4.1 joints

```
std::vector<Joint> Joint_comms::joints
```

Internal vector storing the Joint objects.

A Joint can be added by invoking addJoint()

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

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJointCom.h
- ROS2/ros2_scara_ws/src/joint_communication/src/mJointCom.cpp

5.4 RPI_PWM Class Reference

PWM class for the Raspberry PI 4 and 5.

```
#include <uPWM.h>
```

Public Member Functions

- int start (int channel, int frequency, float duty_cycle=0, int chip=2)
- void stop ()
- ~RPI_PWM ()
- int setDutyCycle (float v) const

Private Member Functions

- void setPeriod (int ns) const
- int setDutyCycleNS (int ns) const
- void enable () const
- void disable () const
- int writeSYS (std::string filename, int value) const

Private Attributes

- int per = 0
- std::string chippath
- std::string pwmpath

5.4.1 Detailed Description

PWM class for the Raspberry PI 4 and 5.

5.4.2 Constructor & Destructor Documentation

5.4.2.1 ~RPI_PWM()

```
RPI_PWM::~RPI_PWM ( )  [inline]
```

5.4.3 Member Function Documentation

5.4.3.1 disable()

```
void RPI_PWM::disable ( ) const  [inline], [private]
```

5.4.3.2 enable()

```
void RPI_PWM::enable ( ) const  [inline], [private]
```

5.4.3.3 setDutyCycle()

```
int RPI_PWM::setDutyCycle (
            float v ) const  [inline]
```

Sets the duty cycle.

Parameters

v	The duty cycle in percent.
return	>0 on success and -1 after an error.

5.4.3.4 setDutyCycleNS()

```
int RPI_PWM::setDutyCycleNS (
            int ns ) const  [inline], [private]
```

5.4.3.5 setPeriod()

```
void RPI_PWM::setPeriod (
            int ns ) const  [inline], [private]
```

5.4.3.6 start()

```
int RPI_PWM::start (
            int channel,
            int frequency,
            float duty_cycle = 0,
            int chip = 2 )  [inline]
```

Starts the PWM

Parameters

channel	The GPIO channel which is 2 or 3 for the RPI5
frequency	The PWM frequency
duty_cycle	The initial duty cycle of the PWM (default 0)
chip	The chip number (for RPI5 it's 2)
return	>0 on success and -1 if an error has happened.

5.4.3.7 stop()

```
void RPI_PWM::stop ( )  [inline]
```

Stops the PWM

5.4.3.8 writeSYS()

```
int RPI_PWM::writeSYS (
            std::string filename,
            int value ) const  [inline], [private]
```

5.4.4 Member Data Documentation

5.4.4.1 chippath

```
std::string RPI_PWM::chippath  [private]
```

5.4.4.2 per

```
int RPI_PWM::per = 0  [private]
```

5.4.4.3 pwmpath

```
std::string RPI_PWM::pwmpath  [private]
```

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

- ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/uPWM.h

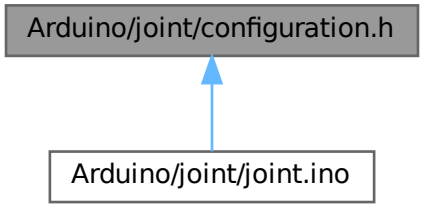
Chapter 6

File Documentation

6.1 Arduino/joint/configuration.h File Reference

Configuration definitions for Joint 1 to Joint 4.

This graph shows which files directly or indirectly include this file:



Macros

- #define ADR 0x11

  I2C adress of joint n is 0x1n.
- #define MAXACCEL 10000

  Maximum acceleration in steps/s^2. Can be set for each joint depending on inertia. If set to high stalls might trigger since PID error grows too large.
- #define MAXVEL 800

  Maximum velocity in steps/s. Can be set for each joint. If set to high stalls might trigger since PID error grows too large.

6.1.1 Detailed Description

Configuration definitions for Joint 1 to Joint 4.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

This file shall be included AFTER one of J1, J2, J3 or J4 have been defined.

6.1.2 Macro Definition Documentation

6.1.2.1 ADR

```
#define ADR 0x11
```

I2C adress of joint n is 0x1n.

6.1.2.2 MAXACCEL

```
#define MAXACCEL 10000
```

Maximum acceleration in steps/s^2. Can be set for each joint depending on inertia. If set to high stalls might trigger since PID error grows too large.

6.1.2.3 MAXVEL

```
#define MAXVEL 800
```

Maximum velocity in steps/s. Can be set for each joint. If set to high stalls might trigger since PID error grows too large.

6.2 configuration.h

Go to the documentation of this file.

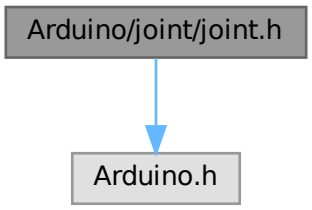
```
00001
00014 #ifndef CONFIGURATION_H
00015 #define CONFIGURATION_H
00016
00017 #if defined(J1)
00019 #define ADR 0x11
00020 #define MAXACCEL 10000
00021 #define MAXVEL 800
00022
00023 #elif defined(J2)
00024 #define ADR 0x12
00025 #define MAXACCEL 10000
00026 #define MAXVEL 800
00027
00028 #elif defined(J3)
00029 #define ADR 0x13
00030 #define MAXACCEL 10000
00031 #define MAXVEL 800
00032
00033 #elif defined(J4)
00034 #define ADR 0x14
00035 #define MAXACCEL 10000
00036 #define MAXVEL 800
00037 #else
00038
00039 /* Below only defined for documentation */
00043 #define ADR 0x11
00044
00049 #define MAXACCEL 10000
00050
00055 #define MAXVEL 800
00056 #error "No Joint has been defined. Define one of 'JX' where X 1,2,3,4"
00057 #endif
00058
00059 #endif
```

6.3 Arduino/joint/joint.h File Reference

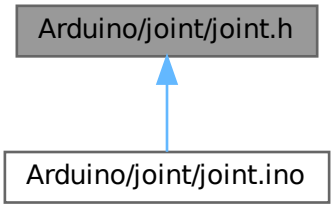
joint firmware header

`#include <Arduino.h>`

Include dependency graph for joint.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ACK 'O'
- #define NACK 'N'
- #define MAX_BUFFER 4

  Maximum size of I2C Payload in bytes.
- #define RFLAGS_SIZE 1

  Size of the return flags in bytes.
- #define DUMP_BUFFER(buffer, size)

  Macro to dump a buffer to the serial console.

Enumerations

- enum stp_reg_t {
  PING = 0x0f , SETUP = 0x10 , SETRPM = 0x11 , GETDRIVERRPM = 0x12 ,
  MOVESTEPS = 0x13 , MOVEANGLE = 0x14 , MOVETOANGLE = 0x15 , GETMOTORSTATE = 0x16 ,
  RUNCOTINOUS = 0x17 , ANGLEMOVED = 0x18 , SETCURRENT = 0x19 , SETHOLDCURRENT = 0x1A ,
  SETMAXACCELERATION = 0x1B , SETMAXDECELERATION = 0x1C , SETMAXVELOCITY = 0x1D ,
  ENABLESTALLGUARD = 0x1E ,
  DISABLESTALLGUARD = 0x1F , CLEARSTALL = 0x20 , ISSTALLED = 0x21 , SETBRAKEMODE = 0x22 ,
  ENABLEPID = 0x23 , DISABLEPID = 0x24 , ENABLECLOSEDLOOP = 0x25 , DISABLECLOSEDLOOP = 0x26 ,
  SETCONTROLTHRESHOLD = 0x27 , MOVETOEND = 0x28 , STOP = 0x29 , GETPIDERROR = 0x2A ,
  CHECKORIENTATION = 0x2B , GETENCODERRPM = 0x2C , HOME = 0x2D , ISHOMED = 0x2E ,
  ISSETUP = 0x2F }

  register and command definitions

Functions

- template<typename T >
  void readValue (T &val, uint8_t *rxBuf, size_t rx_length)

  Reads a value from a buffer to a value of the specified type.
- template<typename T >
  int writeValue (const T val, uint8_t *txBuf, size_t &tx_length)

  Writes a value of the specified type to a buffer.

6.3.1 Detailed Description

joint firmware header

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

This file contains definitions and macros for the joint firmware.

6.3.2 Macro Definition Documentation

6.3.2.1 ACK

```
#define ACK 'O'
```

6.3.2.2 DUMP_BUFFER

```
#define DUMP_BUFFER(
              buffer,
              size )
```

Value:

```
  {                                   \
    Serial.print("Buffer dump: ");    \
    for (size_t i = 0; i < size; i++) \
    {                                 \
      Serial.print(buffer[i], HEX);   \
      Serial.print(" ");              \
    }                                 \
    Serial.println();                 \
  }
```

Macro to dump a buffer to the serial console.

Parameters

buffer	pointer to a buffer to dump to the console
size	number of bytes to dump

6.3.2.3 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

6.3.2.4 NACK

```
#define NACK 'N'
```

6.3.2.5 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

6.3.3 Enumeration Type Documentation

6.3.3.1 stp_reg_t

```
enum stp_reg_t
```

register and command definitions

a register can be read (R) or written (W), each register has a size in bytes. The payload can be split into multiple values or just be a single value. Note that not all functions are implemented.

Enumerator

PING	R; Size: 1; [(char) ACK].
SETUP	W; Size: 2; [(uint8) holdCurrent, (uint8) driveCurrent].
SETRPM	W; Size: 4; [(float) RPM].
GETDRIVERRPM	
MOVESTEPS	W; Size: 4; [(int32) steps].
MOVEANGLE	
MOVETOANGLE	W; Size: 4; [(float) degrees].
GETMOTORSTATE	
RUNCOTINOUS	
ANGLEMOVED	R; Size: 4; [(float) degrees].
SETCURRENT	W; Size: 1; [(uint8) driveCurrent].
SETHOLDCURRENT	W; Size: 1; [(uint8) holdCurrent].
SETMAXACCELERATION	
SETMAXDECELERATION	
SETMAXVELOCITY	
ENABLESTALLGUARD	W; Size: 1; [(uint8) threshold].

Enumerator

DISABLESTALLGUARD	
CLEARSTALL	
ISSTALLED	R; Size: 1; [(uint8) isStalled].
SETBRAKEMODE	W; Size: 1; [(uint8) mode].
ENABLEPID	
DISABLEPID	
ENABLECLOSEDLOOP	
DISABLECLOSEDLOOP	W; Size: 1; [(uint8) 0].
SETCONTROLTHRESHOLD	
MOVETOEND	
STOP	W; Size: 1; [(uint8) mode].
GETPIDERROR	
CHECKORIENTATION	W; Size: 4; [(float) degrees].
GETENCODERRPM	R; Size: 4; [(float) RPM].
HOME	W; Size: 4; [(uint8) current, (uint8) sensitivity, (uint8) speed, (uint8) direction].
ISHOMED	R; Size: 1; [(uint8) isStalled].
ISSETUP	R; Size: 1; [(uint8) isStalled].

6.3.4 Function Documentation

6.3.4.1 readValue()

```
template<typename T >
void readValue (
            T & val,
            uint8_t * rxBuf,
            size_t rx_length )
```

Reads a value from a buffer to a value of the specified type.

Parameters

val	Reference to output variable
rxBuf	Buffer to read value from
rx_length	Length of the buffer

6.3.4.2 writeValue()

```
template<typename T >
int writeValue (
            const T val,
            uint8_t * txBuf,
            size_t & tx_length )
```

Writes a value of the specified type to a buffer.

Parameters

val	Reference to input variable
txBuf	pointer to tx buffer
tx_length	Length of the buffer returne

Returns

0 On success

6.4 joint.h

Go to the documentation of this file.

```
00001
00014 #ifndef JOINT_H
00015 #define JOINT_H
00016 #include <Arduino.h>
00017
00018 #define ACK 'O'
00019 #define NACK 'N'
00020
00026 #define MAX_BUFFER 4 // Bytes
00027
00033 #define RFLAGS_SIZE 1
00034
00041 #define DUMP_BUFFER(buffer, size)      \
00042   {                                    \
00043     Serial.print("Buffer dump: ");     \
00044     for (size_t i = 0; i < size; i++) \
00045     {                                  \
00046       Serial.print(buffer[i], HEX);    \
00047       Serial.print(" ");               \
00048     }                                  \
00049     Serial.println();                  \
00050   }
00051
00060 enum stp_reg_t
00061 {
00062   PING = 0x0f,
00063   SETUP = 0x10,
00064   SETRPM = 0x11,
00065   GETDRIVERRPM = 0x12,
00066   MOVESTEPS = 0x13,
00067   MOVEANGLE = 0x14,
00068   MOVETOANGLE = 0x15,
00069   GETMOTORSTATE = 0x16,
00070   RUNCOTINOUS = 0x17,
00071   ANGLEMOVED = 0x18,
00072   SETCURRENT = 0x19,
00073   SETHOLDCURRENT = 0x1A,
00074   SETMAXACCELERATION = 0x1B,
00075   SETMAXDECELERATION = 0x1C,
00076   SETMAXVELOCITY = 0x1D,
00077   ENABLESTALLGUARD = 0x1E,
00078   DISABLESTALLGUARD = 0x1F,
00079   CLEARSTALL = 0x20,
00080   ISSTALLED = 0x21,
00081   SETBRAKEMODE = 0x22,
00082   ENABLEPID = 0x23,
00083   DISABLEPID = 0x24,
00084   ENABLECLOSEDLOOP = 0x25,
00085   DISABLECLOSEDLOOP = 0x26,
00086   SETCONTROLTHRESHOLD = 0x27,
00087   MOVETOEND = 0x28,
00088   STOP = 0x29,
00089   GETPIDERROR = 0x2A,
00090   CHECKORIENTATION = 0x2B,
00091   GETENCODERRPM = 0x2C,
00092   HOME = 0x2D,
00093   ISHOMED = 0x2E,
00094   ISSETUP = 0x2F
00095 };
00096
00103 template <typename T>
00104 void readValue(T &val, uint8_t *rxBuf, size_t rx_length)
```

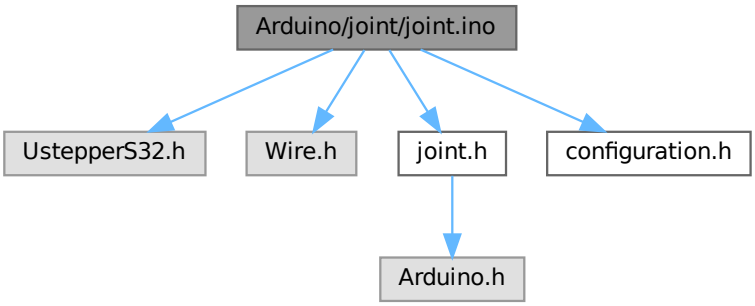
```
00105 {
00106   memcpy(&val, rxBuf, rx_length);
00107 }
00108
00116 template <typename T>
00117 int writeValue(const T val, uint8_t *txBuf, size_t &tx_length)
00118 {
00119   tx_length = sizeof(T);
00120   memcpy(txBuf, &val, tx_length);
00121   return 0;
00122 }
00123
00124 #endif
```

6.5 Arduino/joint/joint.ino File Reference

joint firmware

```
#include <UstepperS32.h>
#include <Wire.h>
#include "joint.h"
#include "configuration.h"
```

Include dependency graph for joint.ino:



Macros

- #define J1

  Define either joint that is to be flashed.

Functions

- void stepper_receive_handler (uint8_t reg)

  Handles commands received via I2C.
- void stepper_request_handler (uint8_t reg)

  Handles read request received via I2C.
- void receiveEvent (int n)

  I2C receive event Handler.
- void requestEvent ()

  I2C request event Handler.
- void setup (void)

  Setup Peripherals.
- void loop (void)

  Main loop.

Variables

- UstepperS32 stepper
- uint8_t reg = 0
- uint8_t rx_buf [MAX_BUFFER] = { 0 }
- uint8_t tx_buf [MAX_BUFFER+RFLAGS_SIZE] = { 0 }
- bool tx_data_ready = 0
- bool rx_data_ready = 0
- size_t tx_length = 0
- size_t rx_length = 0

6.5.1 Detailed Description

joint firmware

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

This file contains the joint firmware.

6.5.2 Macro Definition Documentation

6.5.2.1 J1

```
#define J1
```

Define either joint that is to be flashed.

Define either J1, J2, J3 or J4 and subsequently include configuration.h

6.5.3 Function Documentation

6.5.3.1 loop()

```
void loop (
            void  )
```

Main loop.

Executes the following:
1) if isStallguardEnabled: compares stepper.getPidError() with stallguardThreshold and sets BIT0 of the state byte.
2) sets/clears BIT2 of the state byte if the joint is homed or not.
3) sets/clears BIT3 of the state byte if the joint is setup or not.
4) if rx_data_ready: set BIT1 of the state byte to indicate device is busy. Invoke stepper_receive_handler. Clear BIT1 of the state byte to indicate device is no longer busy

Todo
- why are BIT2 and BIT3 constantly checked and set? Would it be sufficient to do this only invoking the actual functions?

6.5.3.2 receiveEvent()

```
void receiveEvent (
            int n )
```

I2C receive event Handler.

Reads the content of the received message. Saves the reg so it can be used in the main loop. If the master invokes the read() function the message contains only the register byte and no payload. If the master invokes the write() the message has a payload of appropriate size for the command. For a read request the message looks like this:
< [REG]
> [TXBUFn]...[TXBUF2][TXBUF1][TXBUF0][FLAGS]
For a command the message looks like this:
< [REG][RXBUFn]...[RXBUF2][RXBUF1][RXBUF0]
> [FLAGS]
The payload is read into the rx_buf, rx_length is set to the payload length and the rx_data_ready flag is set.

Parameters

n	the number of bytes read from the controller device: MAX_BUFFER

6.5.3.3 requestEvent()

```
void requestEvent ( )
```

I2C request event Handler.

Sends the response data to the master. Every transaction begins with a receive event. This function is only called when the master calls the read() function. Hence this function is only invoked after the receiveEvent() handler has been called. The function calls the stepper_request_handler() which is non-blocking. stepper_request_handler() populates the tx_buf, the current state flags are appended to the tx_buf and then it is send to the master.

6.5.3.4 setup()

```
void setup (
            void  )
```

Setup Peripherals.

Setup I2C with the address ADR, and begin Serial for debugging with baudrate 9600.

6.5.3.5 stepper_receive_handler()

```
void stepper_receive_handler (
            uint8_t reg )
```

Handles commands received via I2C.

Warning

This is a blocking function which may take some time to execute. This function must not be called from an ISR or callback! Call from main loop instead.

All the registers that are inside this handler are considered command which require an action.

Parameters

reg	command that should be executed.

6.5.3.6 stepper_request_handler()

```
void stepper_request_handler (
            uint8_t reg )
```

Handles read request received via I2C.

Can be invoked from the I2C ISR since reads from the stepper are non-blocking. Also Handling reads and the subsequent wire.write(), did not work from the main loop.

All registers inside this function are regarded as read only.

Parameters

reg	register to read.

6.5.4 Variable Documentation

6.5.4.1 reg

```
uint8_t reg = 0
```

6.5.4.2 rx_buf

```
uint8_t rx_buf[MAX_BUFFER] = { 0 }
```

6.5.4.3 rx_data_ready

```
bool rx_data_ready = 0
```

6.5.4.4 rx_length

```
size_t rx_length = 0
```

6.5.4.5 stepper

```
UstepperS32 stepper
```

6.5.4.6 tx_buf

```
uint8_t tx_buf[MAX_BUFFER+RFLAGS_SIZE] = { 0 }
```

6.5.4.7 tx_data_ready

```
bool tx_data_ready = 0
```

6.5.4.8 tx_length

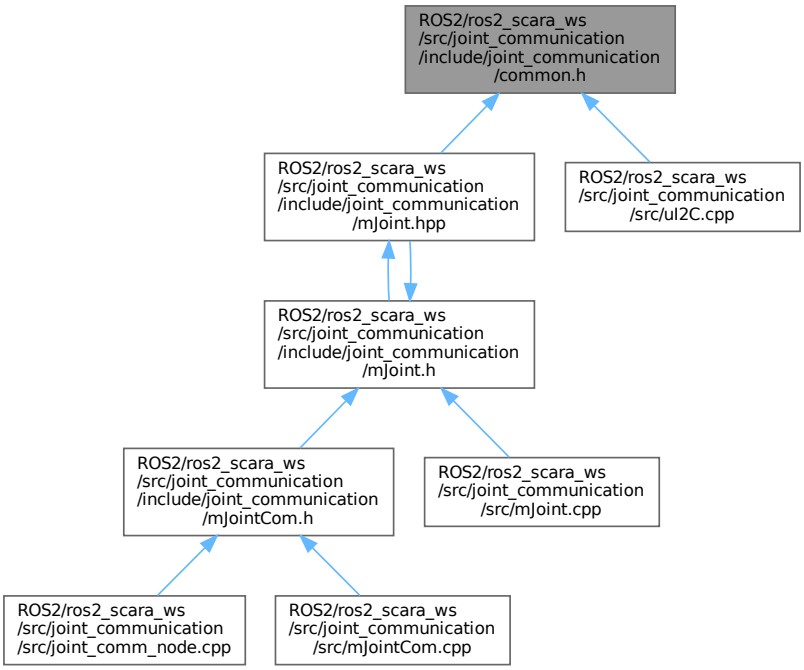
```
size_t tx_length = 0
```

6.6 docs/DOCS_README.md File Reference

6.7 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↩ communication/common.h File Reference

A file containing utility macros and functions.

This graph shows which files directly or indirectly include this file:



Macros

- #define DUMP_BUFFER(buffer, size)

  Macro to dump a buffer to cout.

6.7.1 Detailed Description

A file containing utility macros and functions.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

6.7.2 Macro Definition Documentation

6.7.2.1 DUMP_BUFFER

```
#define DUMP_BUFFER(
              buffer,
              size )
```

Value:

```
  {                                       \
    std::cout « "Buffer dump: ";      \
    for (size_t i = 0; i < size; i++) \
    {                                       \
      printf("%#x ", buffer[i]);        \
    }                                       \
    std::cout « std::endl;            \
  }
```

Macro to dump a buffer to cout.

Parameters

buffer	pointer to a buffer to dump to the console
size	number of bytes to dump

6.8 common.h

Go to the documentation of this file.

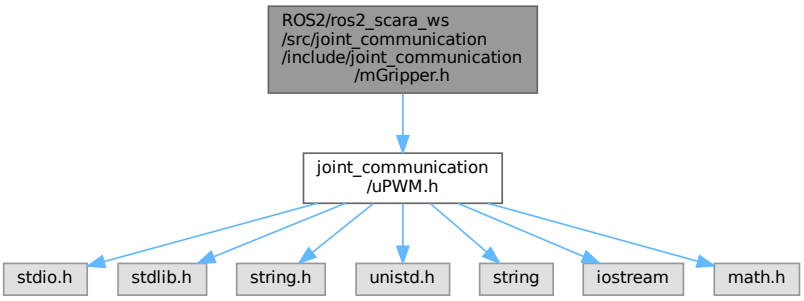
```
00001
00011 #ifndef COMMON_H
00012 #define COMMON_H
00013
00014
00021 #define DUMP_BUFFER(buffer, size)      \
00022   {                                       \
00023     std::cout « "Buffer dump: ";      \
00024     for (size_t i = 0; i < size; i++) \
00025     {                                       \
00026       printf("%#x ", buffer[i]);        \
00027     }                                       \
00028     std::cout « std::endl;            \
00029   }
00030
00031 #endif // COMMON_H
```

6.9 ROS2/ros2_scara_ws/src/joint_communication/include/joint_↩ communication/mGripper.h File Reference

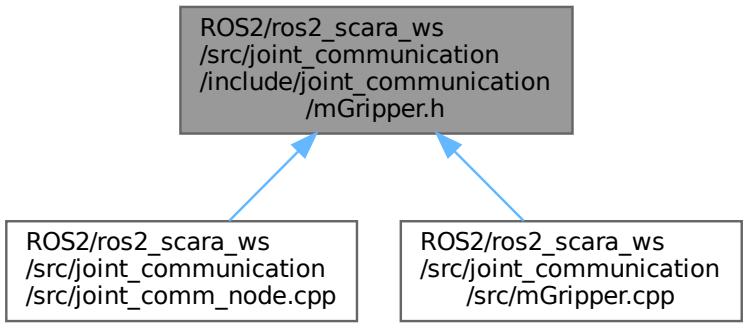
File containing the Gripper class.

```
#include "joint_communication/uPWM.h"
```

Include dependency graph for mGripper.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Gripper

  Gripper object to interact with the robot gripper.

6.9.1 Detailed Description

File containing the Gripper class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Include this file for API functions to interact with the gripper.

6.10 mGripper.h

Go to the documentation of this file.

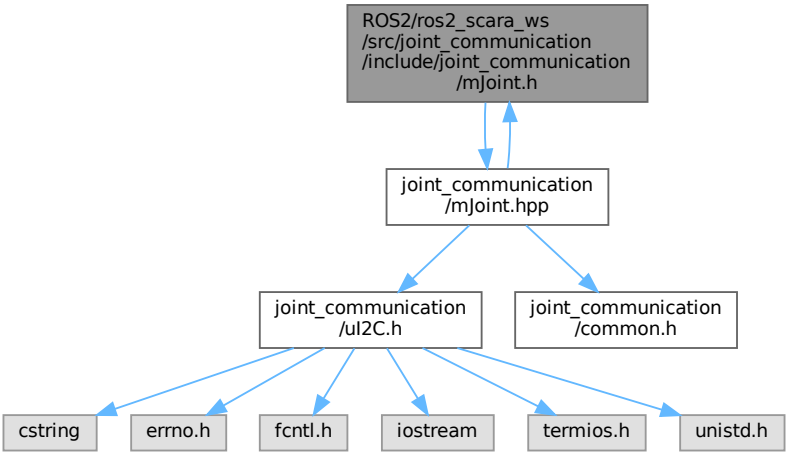
```
00001
00013 #ifndef MGRIPPER_H
00014 #define MGRIPPER_H
00015 #include "joint_communication/uPWM.h"
00016
00023 class Gripper
00024 {
00025 public:
00026     Gripper(void);
00027
00033     int init(void);
00034
00040     int deinit(void);
00041
00050     int enable(void);
00051
00059     int disable(void);
00060
00067     int setPosition(float width);
00068
00069 private:
00070     RPI_PWM pwm;
00071 };
00072 #endif // MGRIPPER_H
```

6.11 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.h File Reference

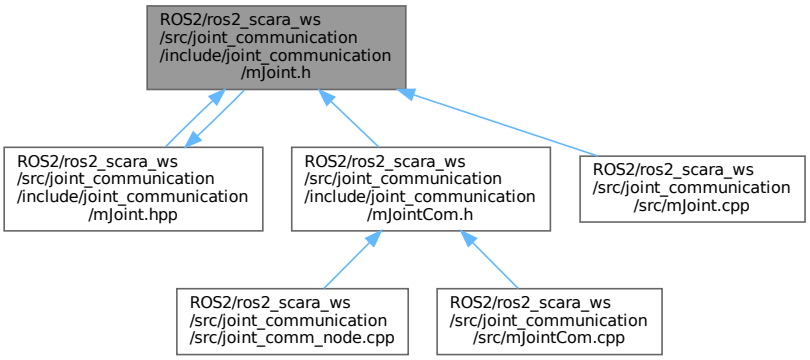
File including the Joint class.

```
#include "joint_communication/mJoint.hpp"
```

Include dependency graph for mJoint.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Joint

  Representing a single joint on the I2C bus.

Macros

- #define JOINT2ENCODERANGLE(jointAngle, gearRatio, offset) (gearRatio ∗ (jointAngle + offset))
- #define ENCODER2JOINTANGLE(encoderAngle, gearRatio, offset) (encoderAngle / gearRatio - offset)

6.11.1 Detailed Description

File including the Joint class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-29

Copyright

6.11.2 Macro Definition Documentation

6.11.2.1 ENCODER2JOINTANGLE

```
#define ENCODER2JOINTANGLE(
              encoderAngle,
              gearRatio,
              offset ) (encoderAngle / gearRatio - offset)
```

6.11.2.2 JOINT2ENCODERANGLE

```
#define JOINT2ENCODERANGLE(
              jointAngle,
              gearRatio,
              offset ) (gearRatio * (jointAngle + offset))
```

6.12 mJoint.h

Go to the documentation of this file.

```
00001
00012 #ifndef MJOINT_H
00013 #define MJOINT_H
00014
00015 #define JOINT2ENCODERANGLE(jointAngle, gearRatio, offset) (gearRatio * (jointAngle + offset))
00016 #define ENCODER2JOINTANGLE(encoderAngle, gearRatio, offset) (encoderAngle / gearRatio - offset)
00017
00022 class Joint
00023 {
00024 public:
00025   Joint(const int address, const std::string name, const float gearRatio, const float offset);
00026   // ~Joint();
00027
00028   int init(void);
00029   int deinit(void);
```

```
00030   int printInfo(void);
00031   int getPosition(float &angle);
00032   int setPosition(float angle);
00033   int getVelocity(float &degps);
00034   int setVelocity(float degps);
00035   int checkOrientation(float angle = 10.0);
00036
00043   int enable(u_int8_t driveCurrent, u_int8_t holdCurrent);
00044
00049   int disable(void);
00050
00060   int home(u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t current);
00061
00070   int stop(bool mode);
00075   int disableCL(void);
00076
00083   int setDriveCurrent(u_int8_t current);
00084
00091   int setHoldCurrent(u_int8_t current);
00092
00098   int setBrakeMode(u_int8_t mode);
00099
00105   int getStall(u_int8_t &stall);
00106
00111   int enableStallguard(u_int8_t sensitivity);
00112
00118   int getIsHomed(u_int8_t &homed);
00119
00126   int getIsHomed(void);
00127
00134   bool isHomed(void);
00135
00141   int getIsSetup(u_int8_t &setup);
00142
00147   int getIsSetup(void);
00148
00152   bool isSetup(void);
00153
00154   int moveSteps(int32_t steps);
00155   int checkCom(void);
00156
00161   u_int8_t getFlags(void);
00162
00163   std::string name;
00164
00165 protected:
00166 private:
00176   enum stp_reg_t
00177   {
00178     PING = 0x0f,
00179     SETUP = 0x10,
00180     SETRPM = 0x11,
00181     GETDRIVERRPM = 0x12,
00182     MOVESTEPS = 0x13,
00183     MOVEANGLE = 0x14,
00184     MOVETOANGLE = 0x15,
00185     GETMOTORSTATE = 0x16,
00186     RUNCOTINOUS = 0x17,
00187     ANGLEMOVED = 0x18,
00188     SETCURRENT = 0x19,
00189     SETHOLDCURRENT = 0x1A,
00190     SETMAXACCELERATION = 0x1B,
00191     SETMAXDECELERATION = 0x1C,
00192     SETMAXVELOCITY = 0x1D,
00193     ENABLESTALLGUARD = 0x1E,
00194     DISABLESTALLGUARD = 0x1F,
00195     CLEARSTALL = 0x20,
00196     ISSTALLED = 0x21,
00197     SETBRAKEMODE = 0x22,
00198     ENABLEPID = 0x23,
00199     DISABLEPID = 0x24,
00200     ENABLECLOSEDLOOP = 0x25,
00201     DISABLECLOSEDLOOP = 0x26,
00202     SETCONTROLTHRESHOLD = 0x27,
00203     MOVETOEND = 0x28,
00204     STOP = 0x29,
00205     GETPIDERROR = 0x2A,
00206     CHECKORIENTATION = 0x2B,
00207     GETENCODERRPM = 0x2C,
00208     HOME = 0x2D,
00209     ISHOMED = 0x2E,
00210     ISSETUP = 0x2F
00211   };
00212
00213   template <typename T>
00214   int read(const stp_reg_t reg, T &data, u_int8_t &flags);
00215
```

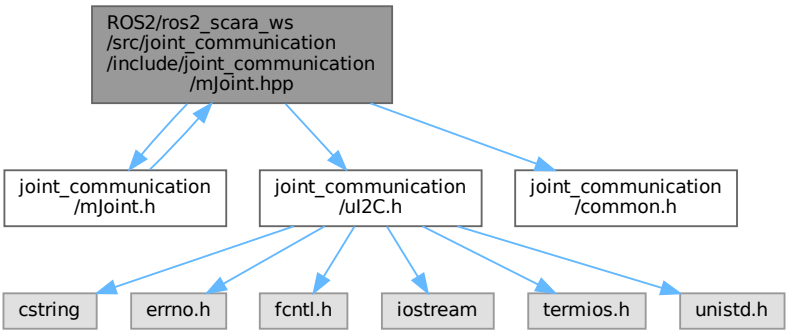
```
00216   template <typename T>
00217   int write(const stp_reg_t reg, T data, u_int8_t &flags);
00218
00237   u_int8_t flags = 0x00;
00238
00239   u_int8_t ishomed = 0;
00240   u_int8_t issetup = 0;
00241
00242   int address;
00243   float gearRatio = 1;
00244   float offset = 0;
00245
00246   int handle = -1;
00247 };
00248
00249 #include "joint_communication/mJoint.hpp"
00250
00251 #endif
```

6.13 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJoint.hpp File Reference

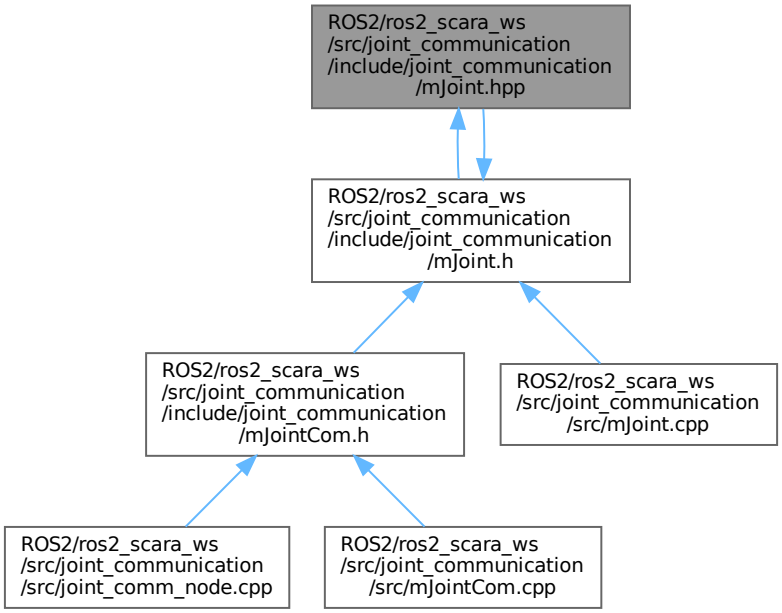
Templated functions for the Joint class.

```
#include "joint_communication/mJoint.h"
#include "joint_communication/uI2C.h"
#include "joint_communication/common.h"
```

Include dependency graph for mJoint.hpp:



This graph shows which files directly or indirectly include this file:



6.13.1 Detailed Description

Templated functions for the Joint class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-29

Copyright

Copyright (c) 2025

This header must be included at the END of the mJoint.h file.

6.14 mJoint.hpp

Go to the documentation of this file.

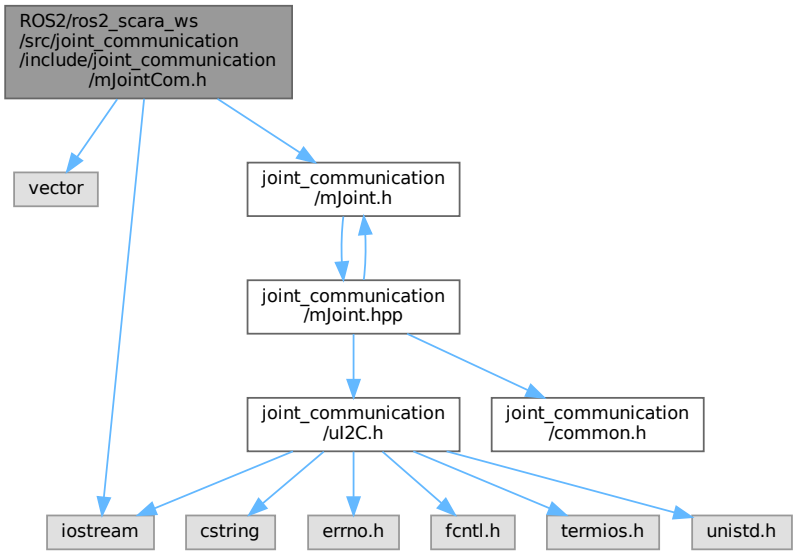
```
00001
00012 #include "joint_communication/mJoint.h"
00013 #include "joint_communication/uI2C.h"
00014 #include "joint_communication/common.h"
00015
00031 template <typename T>
00032 int Joint::read(const stp_reg_t reg, T &data, u_int8_t &flags)
00033 {
00034     size_t size = sizeof(T) + RFLAGS_SIZE;
00035     char *buf = new char[size];
00036     int n = readFromI2CDev(this->handle, reg, buf, size);
00037     if (n != static_cast<int>(size))
00038     {
00039         delete[] buf;
00040         return -1;
00041     }
00042     memcpy(&data, buf, size - RFLAGS_SIZE);
00043     memcpy(&flags, buf + size - RFLAGS_SIZE, RFLAGS_SIZE);
00044     delete[] buf;
00045     return 0;
00046 }
00047
00063 template <typename T>
00064 int Joint::write(const stp_reg_t reg, T data, u_int8_t &flags)
00065 {
00066     size_t size = sizeof(T) + RFLAGS_SIZE;
00067     char *buf = new char[size];
00068     memcpy(buf, &data, size - RFLAGS_SIZE);
00069     int rc = writeToI2CDev(this->handle, reg, buf, size - RFLAGS_SIZE, buf + size - RFLAGS_SIZE);
00070     rc = rc > 0 ? 0 : rc;
00071
00072     memcpy(&flags, buf + size - RFLAGS_SIZE, RFLAGS_SIZE);
00073     delete[] buf;
00074     return rc;
00075 }
```

6.15 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/mJointCom.h File Reference

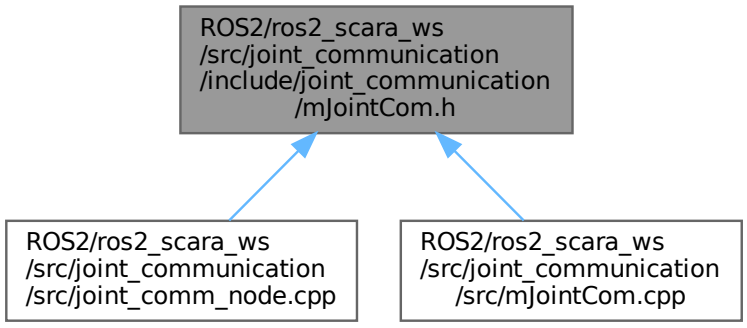
File containing the Joint_comms class.

```
#include <vector>
#include <iostream>
#include "joint_communication/mJoint.h"
```

Include dependency graph for mJointCom.h:



This graph shows which files directly or indirectly include this file:



Classes

- class Joint_comms

  Communication object for all joints.

6.15.1 Detailed Description

File containing the Joint_comms class.

Author

Sebastian Storz

Version

0.1

Date

2025-05-27

Copyright

Include this file for API functions to interact with the stepper motors.

6.16 mJointCom.h

Go to the documentation of this file.

```
00001
00014 #ifndef MJOINTCOM_H
00015 #define MJOINTCOM_H
00016
00017 #include <vector>
00018 #include <iostream>
00019 #include "joint_communication/mJoint.h"
00020
00027 class Joint_comms
00028 {
00029 public:
00030   Joint_comms(void);
00031   ~Joint_comms();
00032
00041   int init(void);
00042
00050   int deinit(void);
00051
00068   void addJoint(const int address, const std::string name, const float gearRatio, const float offset);
00069
00083   int enables(std::vector<u_int8_t> driveCurrent_v, std::vector<u_int8_t> holdCurrent_v);
00084
00093   int enables(u_int8_t driveCurrent, u_int8_t holdCurrent);
00094
00101   int disables(void);
00102
00117   int home(std::string name, u_int8_t direction, u_int8_t rpm, u_int8_t sensitivity, u_int8_t
      current);
00118
00128   int getPositions(std::vector<float> &angle_v);
00129
00139   int setPositions(std::vector<float> angle_v);
00140
00150   int getVelocities(std::vector<float> &degps_v);
00151
00161   int setVelocities(std::vector<float> degps_v);
00162
00175   int checkOrientations(std::vector<float> angle_v);
00176
00182   int checkOrientations(float angle = 10.0);
00183
00191   int stops(bool mode);
00192
00197   int disableCLs(void);
00198
00207   int setDriveCurrents(std::vector<u_int8_t> current);
00208
00216   int setDriveCurrents(u_int8_t current);
00217
```

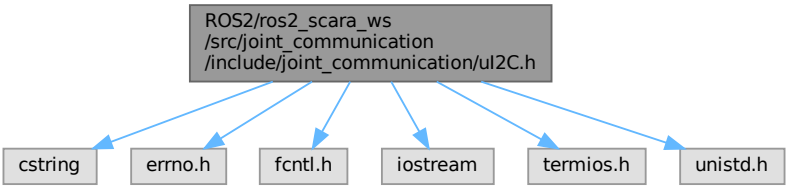
```
00225   int setHoldCurrents(std::vector<u_int8_t> current);
00226
00234   int setHoldCurrents(u_int8_t current);
00235
00243   int setBrakeModes(u_int8_t mode);
00244
00252   int enableStallguards(std::vector<u_int8_t> thresholds);
00253
00259   std::vector<Joint> joints;
00260
00261 protected:
00262 private:
00263 };
00264
00265 #endif
```

6.17 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/uI2C.h File Reference

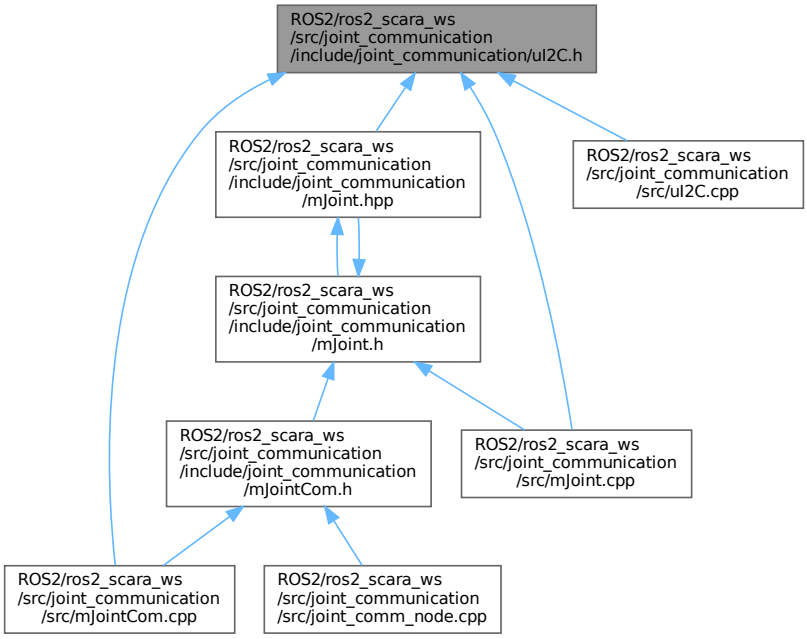
Low level utility for I2C communication on Raspberry Pi using lgpio library.

```
#include <cstring>
#include <errno.h>
#include <fcntl.h>
#include <iostream>
#include <termios.h>
#include <unistd.h>
```

Include dependency graph for uI2C.h:



This graph shows which files directly or indirectly include this file:



Macros

- #define ACK 'O'
- #define NACK 'N'
- #define RFLAGS_SIZE 1

  Size of the return flags in bytes.
- #define MAX_BUFFER 4

  Maximum size of I2C Payload in bytes.

Functions

- int openI2CDevHandle (const int dev_addr)

  Initiates an I2C device on the bus.
- int readFromI2CDev (const int dev_handle, const int reg, char ∗buffer, const int data_length)

  reads block of bytes from device to buffer
- int writeToI2CDev (const int dev_handle, const int reg, char ∗tx_buffer, const int data_length, char ∗RFLAGS_buffer)

  writes block of bytes from buffer to device
- int closeI2CDevHandle (const int dev_handle)

  close an I2C device on the bus

6.17.1 Detailed Description

Low level utility for I2C communication on Raspberry Pi using lgpio library.

Author

Sebastian Storz

Version

0.1

Date

2025-05-28

Copyright

Copyright (c) 2025

lgpio needs to be installed and linked! Installation:

```
cd ~
sudo apt update
sudo apt install -y swig
wget https://github.com/joan2937/lg/archive/master.zip
unzip master.zip
cd lg-master
make
sudo make install
cd ..
sudo rm -rf lg-master
rm master.zip
```

bash

6.17.2 Macro Definition Documentation

6.17.2.1 ACK

```
#define ACK 'O'
```

6.17.2.2 MAX_BUFFER

```
#define MAX_BUFFER 4
```

Maximum size of I2C Payload in bytes.

4 bytes used to transmit floats and int32_t

6.17.2.3 NACK

```
#define NACK 'N'
```

6.17.2.4 RFLAGS_SIZE

```
#define RFLAGS_SIZE 1
```

Size of the return flags in bytes.

Only one byte used and hence set to 1.

6.17.3 Function Documentation

6.17.3.1 closeI2CDevHandle()

```
int closeI2CDevHandle (
            const int dev_handle )
```

close an I2C device on the bus

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`

Returns

0 on OK, negative on error.

6.17.3.2 openI2CDevHandle()

```
int openI2CDevHandle (
            const int dev_addr )
```

Initiates an I2C device on the bus.

Parameters

dev_addr	7-bit device adress [0 - 0x7F]

Returns

the device handle, negative on error.

6.17.3.3 readFromI2CDev()

```
int readFromI2CDev (
            const int dev_handle,
            const int reg,
            char * buffer,
            const int data_length )
```

reads block of bytes from device to buffer

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`
reg	the command/data register
buffer	pointer to data buffer to hold received values
data_length	number of bytes to read

Returns

number of bytes read, negative on error.

6.17.3.4 writeToI2CDev()

```
int writeToI2CDev (
            const int dev_handle,
            const int reg,
            char * tx_buffer,
            const int data_length,
            char * RFLAGS_buffer )
```

writes block of bytes from buffer to device

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`
reg	the command/data register
tx_buffer	pointer to data buffer holding the data to send
data_length	number of bytes to send
RFLAGS_buffer	buffer to hold returned flags

Returns

0 on OK, negative on error.

6.18 uI2C.h

Go to the documentation of this file.

```
00001
00028 #ifndef USERIAL_H
00029 #define USERIAL_H
00030 #include <cstring>
00031 #include <errno.h>
00032 #include <fcntl.h>
00033 #include <iostream>
00034 #include <termios.h>
00035 #include <unistd.h>
00036
00037 #define ACK 'O'
00038 #define NACK 'N'
00039
00043 #define RFLAGS_SIZE 1
00044
00048 #define MAX_BUFFER 4 // Bytes
00049
00055 int openI2CDevHandle(const int dev_addr);
```

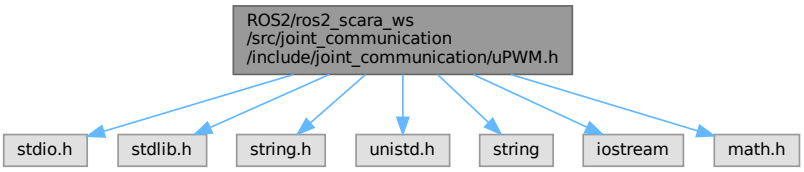
```
00056
00065 int readFromI2CDev(const int dev_handle, const int reg, char *buffer, const int data_length);
00066
00076 int writeToI2CDev(const int dev_handle, const int reg, char *tx_buffer, const int data_length, char
      *RFLAGS_buffer);
00077
00083 int closeI2CDevHandle(const int dev_handle);
00084
00085
00086 #endif
```

6.19 ROS2/ros2_scara_ws/src/joint_communication/include/joint_communication/uPWM.h File Reference

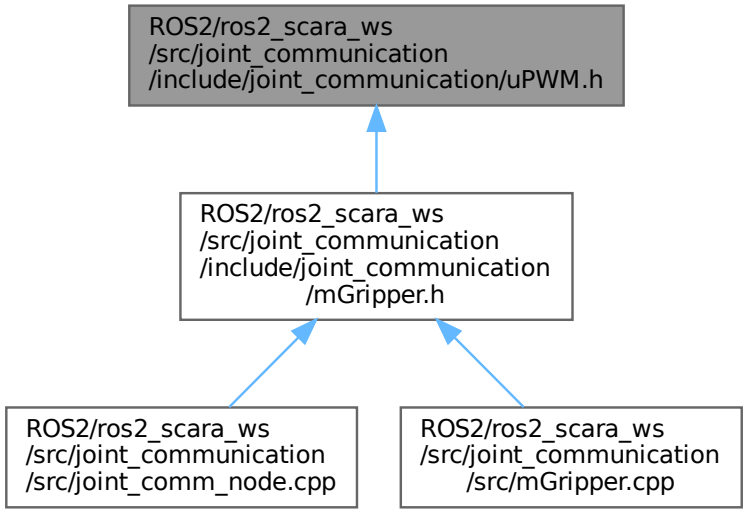
Includes source code for Hardware PWM generation on Raspberry Pi 4.

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <string>
#include <iostream>
#include <math.h>
```

Include dependency graph for uPWM.h:



This graph shows which files directly or indirectly include this file:



Classes

- class RPI_PWM

  PWM class for the Raspberry PI 4 and 5.

6.19.1 Detailed Description

Includes source code for Hardware PWM generation on Raspberry Pi 4.

Author

Sebastian Storz and Bernd Porr, bernd.porr@glasgow.ac.uk

Version

0.1

Date

2025-05-27

I copied this from: https://github.com/berndporr/rpi_pwm/blob/main/rpi_pwm.h and slightly modified it.

lgpio, the library used for I2C access can only generate soft PWM, The timing jitter will cause the servo to fidget. This may cause it to overheat and wear out prematurely.

Copyright

6.20 uPWM.h

Go to the documentation of this file.

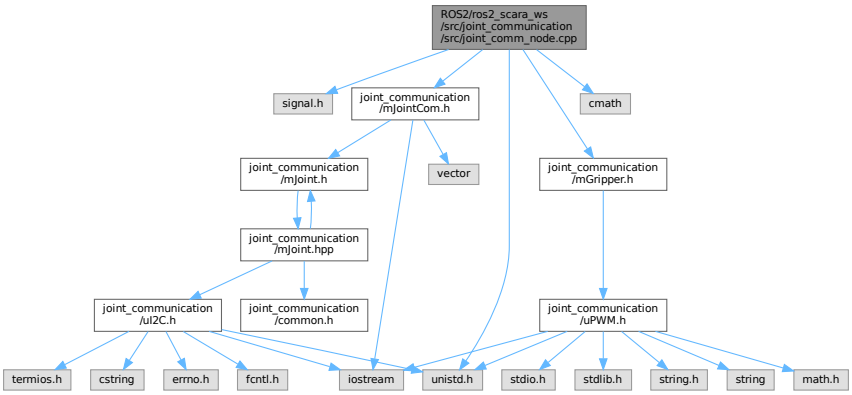
```
00001
00019 #ifndef __RPIPWM
00020 #define __RPIPWM
00021
00022 #include <stdio.h>
00023 #include <stdlib.h>
00024 #include <string.h>
00025 #include <unistd.h>
00026 #include <string>
00027 #include <iostream>
00028 #include <math.h>
00029
00033 class RPI_PWM
00034 {
00035 public:
00044     int start(int channel, int frequency, float duty_cycle = 0, int chip = 2)
00045     {
00046         chippath = "/sys/class/pwm/pwmchip" + std::to_string(chip);
00047         pwmpath = chippath + "/pwm" + std::to_string(channel);
00048         std::string p = chippath + "/export";
00049         FILE *const fp = fopen(p.c_str(), "w");
00050         if (NULL == fp)
00051         {
00052             std::cerr « "PWM device does not exist. Make sure to add 'dtoverlay=pwm-2chan' to
      /boot/firmware/config.txt.\n";
00053             return -1;
00054         }
00055         const int r = fprintf(fp, "%d", channel);
00056         fclose(fp);
00057         if (r < 0)
00058             return r;
00059         usleep(100000); // it takes a while till the PWM subdir is created
00060         per = (int)1E9 / frequency;
00061         setPeriod(per);
00062         setDutyCycle(duty_cycle);
00063         enable();
00064         return r;
00065     }
00066
00070     void stop()
00071     {
00072         disable();
00073     }
00074
00075     ~RPI_PWM()
00076     {
00077         disable();
00078     }
00079
00085     inline int setDutyCycle(float v) const
00086     {
00087         const int dc = (int)round((float)per * (v / 100.0));
00088         const int r = setDutyCycleNS(dc);
00089         return r;
00090     }
00091
00092 private:
00093     void setPeriod(int ns) const
00094     {
00095         writeSYS(pwmpath + "/" + "period", ns);
00096     }
00097
00098     inline int setDutyCycleNS(int ns) const
00099     {
00100         const int r = writeSYS(pwmpath + "/" + "duty_cycle", ns);
00101         return r;
00102     }
00103
00104     void enable() const
00105     {
00106         writeSYS(pwmpath + "/" + "enable", 1);
00107     }
00108
00109     void disable() const
00110     {
00111         writeSYS(pwmpath + "/" + "enable", 0);
00112     }
00113
00114     int per = 0;
00115
00116     std::string chippath;
00117     std::string pwmpath;
```

```
00118
00119     inline int writeSYS(std::string filename, int value) const
00120     {
00121         FILE *const fp = fopen(filename.c_str(), "w");
00122         if (NULL == fp)
00123         {
00124             return -1;
00125         }
00126         const int r = fprintf(fp, "%d", value);
00127         fclose(fp);
00128         return r;
00129     }
00130 };
00131
00132 #endif
```

6.21 ROS2/ros2_scara_ws/src/joint_communication/src/joint_comm_↩ node.cpp File Reference

```
#include <signal.h>
#include <unistd.h>
#include "joint_communication/mJointCom.h"
#include "joint_communication/mGripper.h"
#include <cmath>
```

Include dependency graph for joint_comm_node.cpp:



Functions

- void INT_handler (int s)
- int main (int argc, char **argv)

Variables

- Joint_comms _Joints
- Gripper _Gripper

6.21.1 Function Documentation

6.21.1.1 INT_handler()

```
void INT_handler (
            int s )
```

6.21.1.2 main()

```
int main (
            int argc,
            char ** argv )
```

6.21.2 Variable Documentation

6.21.2.1 _Gripper

```
Gripper _Gripper
```

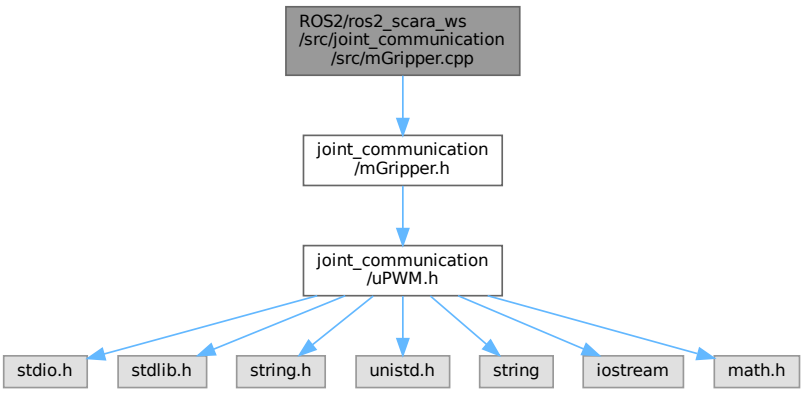
6.21.2.2 _Joints

```
Joint_comms _Joints
```

6.22 ROS2/ros2_scara_ws/src/joint_communication/src/mGripper.cpp File Reference

```
#include "joint_communication/mGripper.h"
```

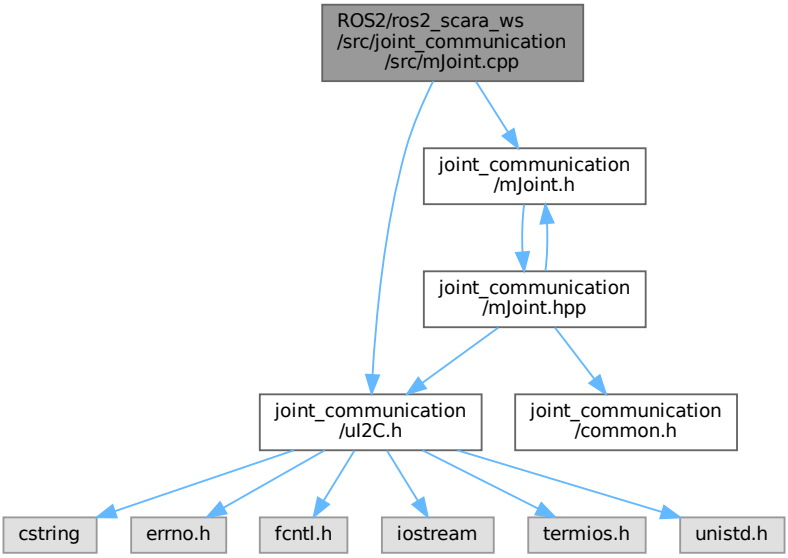
Include dependency graph for mGripper.cpp:



6.23 ROS2/ros2_scara_ws/src/joint_communication/src/mJoint.cpp File Reference

```
#include "joint_communication/uI2C.h"
#include "joint_communication/mJoint.h"
```

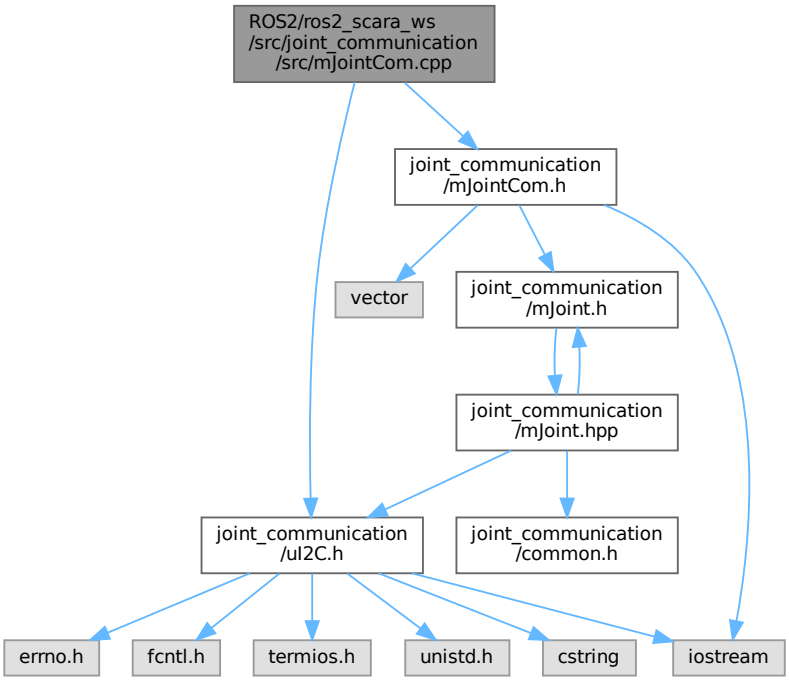
Include dependency graph for mJoint.cpp:



6.24 ROS2/ros2_scara_ws/src/joint_communication/src/mJointCom.cpp File Reference

```
#include "joint_communication/uI2C.h"
#include "joint_communication/mJointCom.h"
```

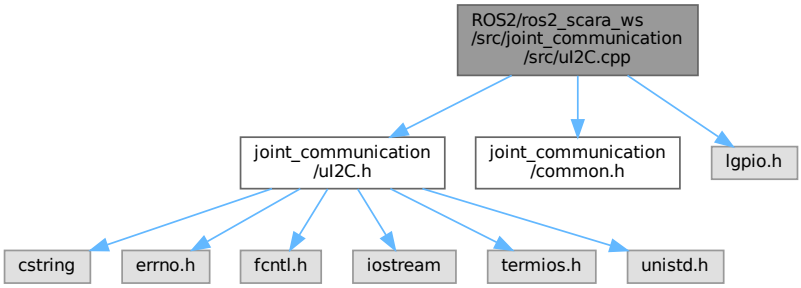
Include dependency graph for mJointCom.cpp:



6.25 ROS2/ros2_scara_ws/src/joint_communication/src/uI2C.cpp File Reference

```
#include "joint_communication/uI2C.h"
#include "joint_communication/common.h"
#include <lgpio.h>
```

Include dependency graph for uI2C.cpp:



Functions

- int openI2CDevHandle (const int dev_addr)

Initiates an I2C device on the bus.

- int readFromI2CDev (const int dev_handle, const int reg, char *buffer, const int data_length)

  reads block of bytes from device to buffer

- int writeToI2CDev (const int dev_handle, const int reg, char *tx_buffer, const int data_length, char *RFLAGS_buffer)

  writes block of bytes from buffer to device

- int closeI2CDevHandle (const int dev_handle)

  close an I2C device on the bus

6.25.1 Function Documentation

6.25.1.1 closeI2CDevHandle()

```
int closeI2CDevHandle (
            const int dev_handle )
```

close an I2C device on the bus

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`

Returns

0 on OK, negative on error.

6.25.1.2 openI2CDevHandle()

```
int openI2CDevHandle (
            const int dev_addr )
```

Initiates an I2C device on the bus.

Parameters

dev_addr	7-bit device adress [0 - 0x7F]

Returns

the device handle, negative on error.

6.25.1.3 readFromI2CDev()

```
int readFromI2CDev (
            const int dev_handle,
            const int reg,
```

```
            char * buffer,
            const int data_length )
```

reads block of bytes from device to buffer

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`
reg	the command/data register
buffer	pointer to data buffer to hold received values
data_length	number of bytes to read

Returns

number of bytes read, negative on error.

6.25.1.4 writeToI2CDev()

```
int writeToI2CDev (
            const int dev_handle,
            const int reg,
            char * tx_buffer,
            const int data_length,
            char * RFLAGS_buffer )
```

writes block of bytes from buffer to device

Parameters

dev_handle	device handle obtained from `openI2CDevHandle`
reg	the command/data register
tx_buffer	pointer to data buffer holding the data to send
data_length	number of bytes to send
RFLAGS_buffer	buffer to hold returned flags

Returns

0 on OK, negative on error.

Index