Driver for Black Betal robot in ROS2

3.0.0

Generated by Doxygen 1.9.1

Sun May 28 2023 19:24:50

1 Bakalárska práca

Tento projekt je kód písaný pre potreby bakalárskej prace.

Názov: Vytvorenie ovládača v prostredí ROS pre mobilného robota
Autor: Filip Lobpreis
ID študenta: 111124

1.1 Kód

Pre jednoduchosť zaobchádzania sú s balíkom priložené štyri súbory:

Skript	Opis
compile	Kopiluje kod pomocou prikazu `colcon build`. Zisti kolko jadier ma pocitac uzivatela a pouzije dvojnasobok vlakien tohto poctu na kopilaciu. Je tu moznost pouzit vlajku `-d` popripade `--doc`, ktora zabezpeci generovanie dokumentacie kodu pomocou spustitelneho suboru `doxygen`. `-h / --help` zobrazi help prikazy. Pri zadani vlajky `-c / --clean` sa vymaze build adresar a spusti kompilaciu z praznej konfiguracie.
run	Pouziva subor `compile`. Ak prebehne kompilacia bez chyb tak spusti program. Tento program vie sprostredkovat vlajku `-d` respektive `--doc` suboru `compile`.
test	Taktiez pouziva subor `compile`. Namiesto spustenia programu spusti testy.
clearLogs	Tento skript vytvori priecinok `backupLogs`. Tam presunie vsetky logy z priecinka `log`.
square	Subor `square` mozeme spustit az po spusteni suboru `run`. Tento subor posiela prikazy robtu tak, aby spravil stvorec. Pre blizsie informacie zadajte prikaz `square --help`

> ***NOTE***: Dokumentácia k programu je vygenerovaná v anglickom jazyku, aby si ju mohlo precitať väčšie spektrum lúdi.

Celý projekt je stavaný okolo TCP/IP klienta. Ten komunikuje s robotom pomocou správ typu JSON. Komunikácia prebieha z nasej strany vo viacerých vrstvách. Keď pošleme nejaký request robotu, ten sa najprv uloží do rady. Z nej si náš klient vyťahuje správy a následne ich posiela. Potom príjme odpoveď od robota. Ak správa, ktorú sme poslali je typu požiadavky (request) na získanie rýchlostí kolies, tak príjme túto správu a uloží ju do ďalšej rady. Z nej si ju vyťahuje objekt odometry. Následne ju spracováva. Pre podrobnejšie fungovanie programu si precitajte dokumentáciu.

1.2 Referencie

https://www.github.com/Fildo7525/Bakalarsky-projekt

2 Namespace Index

2.1 Namespace List

Here is a list of all namespaces with brief descriptions:

3 Hierarchical Index

3.1 Class Hierarchy

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

4 Class Index

4.1 Class List

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

5 Namespace Documentation

5.1 bm Namespace Reference

namespace containing the enums for blackmetal constants.

Enumerations

- enum class Command {
  EMPTY , EMG_STOP , NORMAL_STOP , SET_LR_WHEEL_VELOCITY ,
  SET_LR_WHEEL_POSITION , NONE_5 , GET_LR_WHEEL_VELOCITY , PREPARE_WHEEL_CONTROLLER
  ,
  GET_LR_WHEEL_POSITION }

Commands defined in the blackmetal robot documentation.

- enum class Status {
  OK , FULL_BUFFER , SEND_ERROR , RECEIVE_ERROR ,
  TIMEOUT_ERROR , MULTIPLE_RECEIVE , ODOMETRY_SPEED_DATA }

  Status returned by the server.

Functions

- std::string toString (const bm::Command command)

  Get the string representation of the `bm::Command` code.
- std::string toString (const bm::Status status)

  Get the string representation of the `bm::Status` code.

5.1.1 Detailed Description

namespace containing the enums for blackmetal constants.

5.1.2 Enumeration Type Documentation

5.1.2.1 Command `enum bm::Command [strong]`

Commands defined in the blackmetal robot documentation.

Enumerator

EMPTY	Empty command.
EMG_STOP	Emergency stop.
NORMAL_STOP	Normal stop.
SET_LR_WHEEL_VELOCITY	Sets Left and right wheel velocity. The velocities are required for this command.
SET_LR_WHEEL_POSITION	Set the position to which should the wheels turn to.
NONE_5	Not implemented.
GET_LR_WHEEL_VELOCITY	Get left and right wheel actual velocities.
PREPARE_WHEEL_CONTROLLER	Prepare the controller of left and right wheel.
GET_LR_WHEEL_POSITION	Get left and right wheel actual positions from robot.

5.1.2.2 Status `enum bm::Status [strong]`

Status returned by the server.

The robot receives the json string and returns a json string with a specific return status. This status is than mapped on `BlackMetal::Status` enum type.

Enumerator

OK	Server processed the request.
FULL_BUFFER	Server could not process the request. The buffer is full.
SEND_ERROR	The client could not send data to the server.
RECEIVE_ERROR	The response could not be received.
TIMEOUT_ERROR	Error emitted on timeout while receiving or sending.
MULTIPLE_RECEIVE	Special case when we receive multiple responses on one receive.
ODOMETRY_SPEED_DATA	This flag represents whether the received data are meant for odometry or not.

5.1.3 Function Documentation

5.1.3.1 toString() [1/2]

```
std::string bm::toString (
        const bm::Command command )
```

Get the string representation of the `bm::Command` code.

Parameters

command	Command to be transformed.

5.1.3.2 toString() [2/2]

```
std::string bm::toString (
        const bm::Status status )
```

Get the string representation of the `bm::Status` code.

Parameters

status	Status to be transformed.

5.2 ts Namespace Reference

ts is a namespace grouping thread safe classes and functions.

Classes

- class Queue

  Thread safe dynamic templated priority queue.

5.2.1 Detailed Description

ts is a namespace grouping thread safe classes and functions.

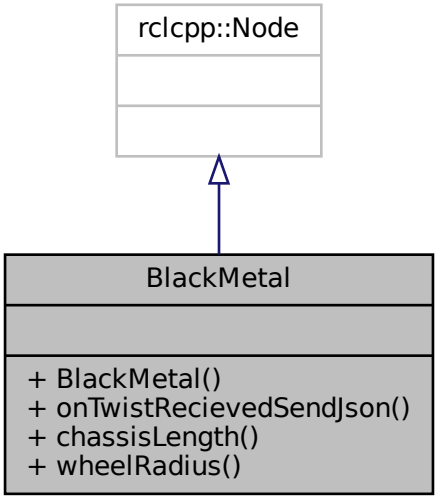
6 Class Documentation

6.1 BlackMetal Class Reference

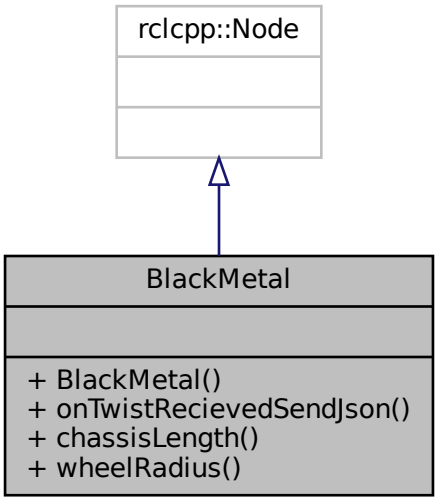
Class for communication with the Blackmetal robot.

`#include <BlackMetal.hpp>`

Inheritance diagram for BlackMetal:



Collaboration diagram for BlackMetal:



Public Member Functions

- BlackMetal ()

  Constructor.
- void onTwistRecievedSendJson (const geometry_msgs::msg::Twist &msg)

  Convert Twist messages and send them to the Blackmetal robot.
- double chassisLength () const

  Retrieve the chassis length set in the config file.
- double wheelRadius () const

  Retrieve the wheel radius set from the config file.

6.1.1 Detailed Description

Class for communication with the Blackmetal robot.

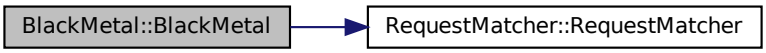
This class inherits the functionality of Client class and implements the methods for communication with the mentioned Black Metal robot. The left and right wheel velocities are converted from `geometry_msgs::msg::Twist` message type.

6.1.2 Constructor & Destructor Documentation

6.1.2.1 BlackMetal() `BlackMetal::BlackMetal ( )`

Constructor.

Here is the call graph for this function:



6.1.3 Member Function Documentation

6.1.3.1 chassisLength() `double BlackMetal::chassisLength ( ) const`

Retrieve the chassis length set in the config file.

Here is the call graph for this function:

6.1.3.2 onTwistRecievedSendJson() `void BlackMetal::onTwistRecievedSendJson ( const geometry_msgs::msg::Twist & msg )`

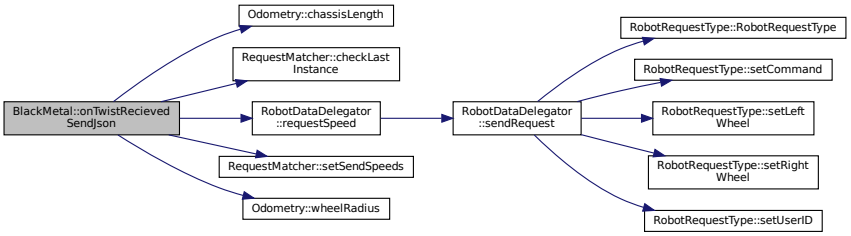
Convert Twist messages and send them to the Blackmetal robot.

This function waits for the Twist messages. Transfers them to the json type of string and sends them to the robot. The linear and angular parameters have to be calculated for wheel for the json.

Parameters

msg	The received message.

Here is the call graph for this function:



6.1.3.3 wheelRadius() `double BlackMetal::wheelRadius ( ) const`

Retrieve the wheel radius set from the config file.

Here is the call graph for this function:

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

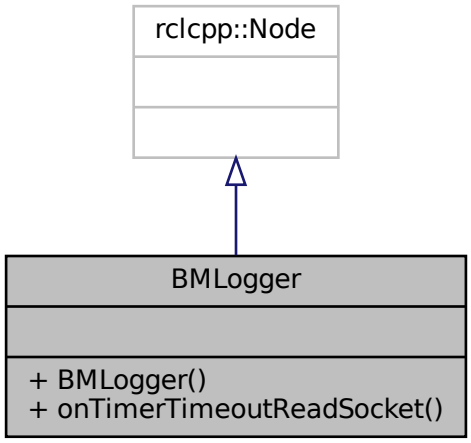
- BlackMetal.hpp
- BlackMetal.cpp

6.2 BMLogger Class Reference

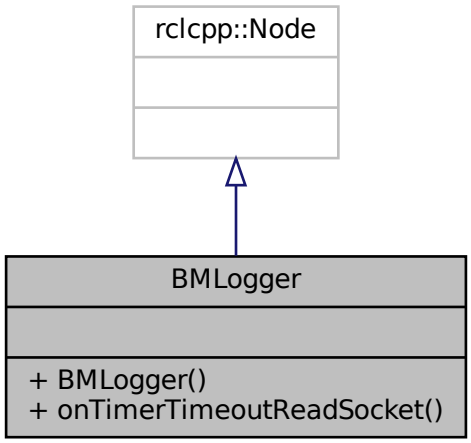
This class represents a client, which connects to robot and handles the messages that are handed to him as logs. The received messages are in JSON format. These messages contain two key.

`#include <BMLogger.hpp>`

Inheritance diagram for BMLogger:



Collaboration diagram for BMLogger:



Public Member Functions

- BMLogger ()

  Constructor.
- void onTimerTimeoutReadSocket ()

  A Callback function called every second in the timer to read the data from the socket.

6.2.1 Detailed Description

This class represents a client, which connects to robot and handles the messages that are handed to him as logs. The received messages are in JSON format. These messages contain two key.

- state
- direction Both of these keys have values represented by integers.

NOTE: This is written in robot's documentation. However the robot only sends STATE_READY and DIRECTION↩ _NORTH. No other value is sent.

The state can have five states:

- 0: STATE_UNKNOWN
- 1: STATE_READY
- 2: STATE_ERROR
- 3: STATE_RUN
- 4: STATE_SHUTDOWN

The direction can have four states:

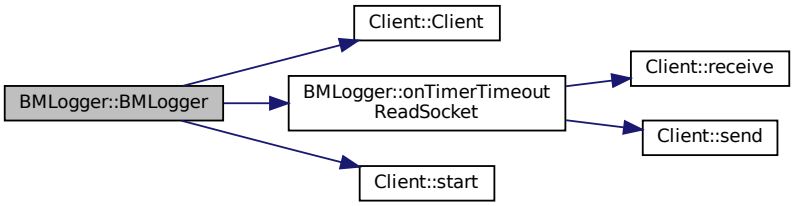
- 1: DIRECTION_NORTH
- 2: DIRECTION_SOUTH
- 3: DIRECTION_WEST
- 4: DIRECTION_EAST

6.2.2 Constructor & Destructor Documentation

6.2.2.1 BMLogger() `BMLogger::BMLogger ( )`

Constructor.

Here is the call graph for this function:

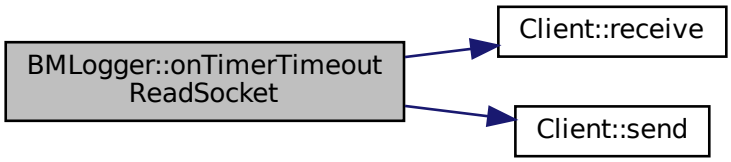


6.2.3 Member Function Documentation

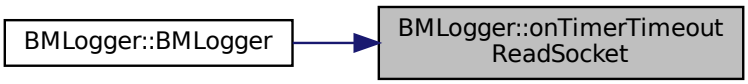
6.2.3.1 onTimerTimeoutReadSocket() `void BMLogger::onTimerTimeoutReadSocket ( )`

A Callback function called every second in the timer to read the data from the socket.

The calling time is set to one second. However, if the server does not send data during that period, the function blocks until the server does not send us some string for too log. Here is the call graph for this function:



Here is the caller graph for this function:



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

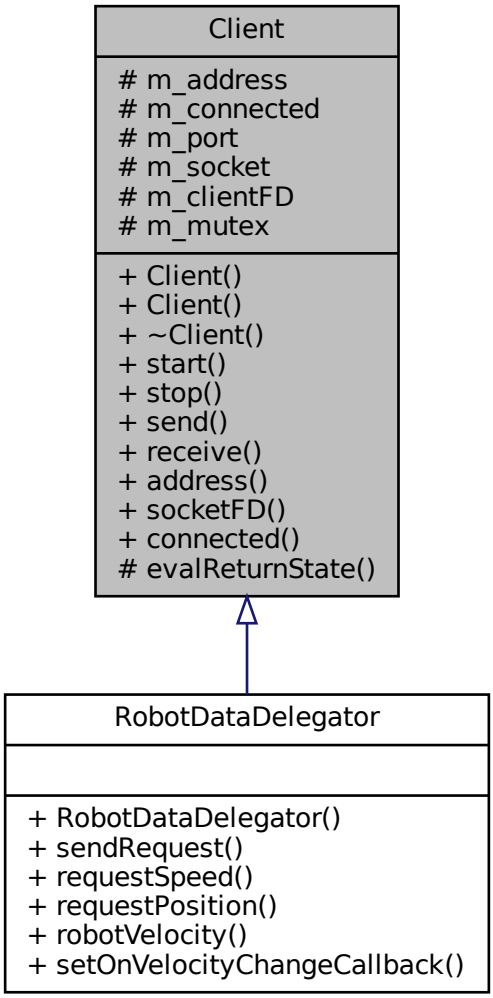
- BMLogger.hpp
- BMLogger.cpp

6.3 Client Class Reference

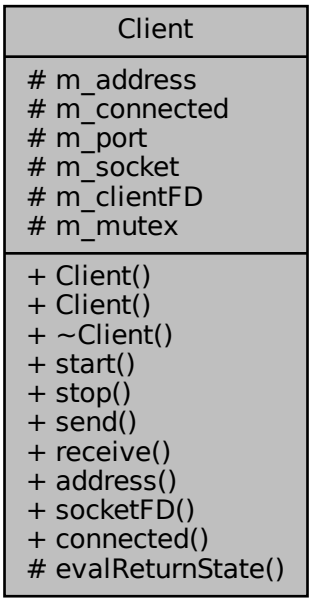
Class handling TCP/IPv4 connections.

`#include <Client.hpp>`

Inheritance diagram for Client:



Collaboration diagram for Client:



Public Member Functions

- Client ()=default

  Default constructor.
- Client (const std::string &address, int port, long wateTime_usec=WAIT_TIME)

  Constructs and initializes the client for communication.
- virtual ~Client ()

  Destructor.
- void start (const std::string &address, int port, long wateTime_usec)

  Starts the client and connects to the specified server at address:port.
- void stop ()

  Disconnects the client from the server and closes the opened socket.
- virtual bm::Status send (const std::string &msg)

  Send a desired string message to the server.
- virtual bm::Status receive (std::string &msg)

  Receive a message from the server.
- std::string address ()

  Returns copy of the IP address of the server we are currently connected to.
- int socketFD ()

  Returns a copy of socket file descriptor.
- bool connected ()

  Indicates if we are or are not connected.

Protected Member Functions

- virtual bm::Status evalReturnState (const std::string &returnJson)

  Virtual function that evaluates the request status.

Protected Attributes

- std::string m_address

  IPv4 address to which we tried or are connected to.
- bool m_connected

  Flag checking the client's connection.
- int m_port

  Port to which is the client connected to.
- int m_socket

  Socket for biding to server, sending and receiving data.
- int m_clientFD

  Client file descriptor.
- std::mutex m_mutex

  Mutex called every time we want to get data from the client.

6.3.1 Detailed Description

Class handling TCP/IPv4 connections.

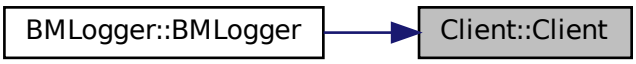
This class is a wrapper arround linux server-client interface. The communication is done using TCP/IP protocol. You can define the time, how much should the send and receive functions wait for the server.

6.3.2 Constructor & Destructor Documentation

6.3.2.1 Client() [1/2] Client::Client ( ) [default]

Default constructor.

Here is the caller graph for this function:



6.3.2.2 Client() [2/2]

```
Client::Client (
        const std::string & address,
        int port,
        long wateTime_usec = WAIT_TIME )
```

Constructs and initializes the client for communication.

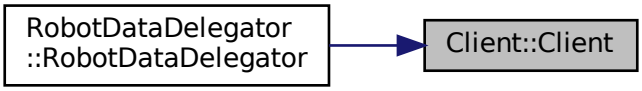
If -1 is supplied as a wateTime_usec the functions will stay blocking.

Parameters

address	IPv4 Address of the server.
port	On which to start the communication.
wateTime_usec	How many seconds should the client wait on receive and send functions.

Here is the call graph for this function:

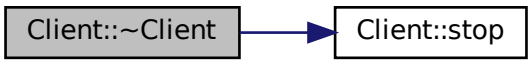
Here is the caller graph for this function:



6.3.2.3 ~Client() `Client::~Client ( ) [virtual]`

Destructor.

Here is the call graph for this function:



6.3.3 Member Function Documentation

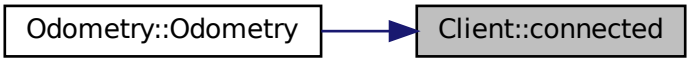
6.3.3.1 address() `std::string Client::address ( )`

Returns copy of the IP address of the server we are currently connected to.

6.3.3.2 connected() `bool Client::connected ( )`

Indicates if we are or are not connected.

Here is the caller graph for this function:



6.3.3.3 evalReturnState() `bm::Status Client::evalReturnState ( const std::string & returnJson ) [protected], [virtual]`

Virtual function that evaluates the request status.

This function is called in case that all the communication was successful and we received data from the server.

Parameters

returnJson	Json string returned from the communication.

6.3.3.4 receive() `bm::Status Client::receive ( std::string & msg ) [virtual]`

Receive a message from the server.

Be aware this is a blocking function. The execution will stop until the server does not send a message to us.

Parameters

msg	Variable where the received message should be saved.

Returns

bm::Status::RECEIVE_ERROR when the ::read function crashes, bm::Status::TIMEOUT_ERROR when the wateTime_usec is exceeded, bm::Status::OK otherwise.

Here is the caller graph for this function:

6.3.3.5 send() bm::Status Client::send (const std::string & *msg*) [virtual]

Send a desired string message to the server.

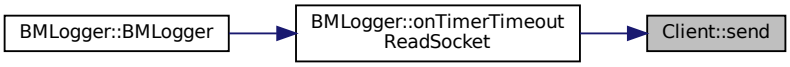
Parameters

msg	message to be send.

Returns

bm::Status::RECEIVE_ERROR when the ::send function crashes, bm::Status::TIMEOUT_ERROR when the wateTime_usec is exceeded, bm::Status::OK otherwise.

Here is the caller graph for this function:



6.3.3.6 socketFD() int Client::socketFD ( )

Returns a copy of socket file descriptor.

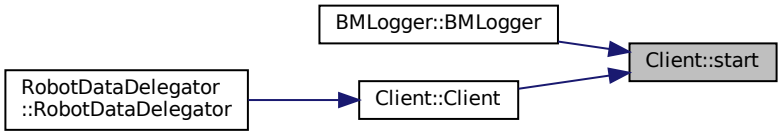
6.3.3.7 start() void Client::start (const std::string & *address*, int *port*, long *wateTime_usec*)

Starts the client and connects to the specified server at address:port.

Parameters

address	Address to connect to.
port	Port to connect to.
wateTime_usec	How many seconds should the client wait on receive and send functions. If -1 is supplied the functions will stay blocking.

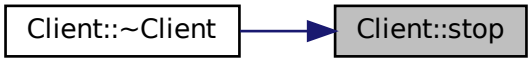
Here is the caller graph for this function:



6.3.3.8 stop() `void Client::stop ( )`

Disconnects the client from the server and closes the opened socket.

Here is the caller graph for this function:



6.3.4 Member Data Documentation

6.3.4.1 m_address `std::string Client::m_address [protected]`

IPv4 address to which we tried or are connected to.

6.3.4.2 m_clientFD `int Client::m_clientFD [protected]`

Client file descriptor.

6.3.4.3 m_connected `bool Client::m_connected [mutable], [protected]`

Flag checking the client's connection.

6.3.4.4 m_mutex `std::mutex Client::m_mutex [protected]`

Mutex called every time we want to get data from the client.

6.3.4.5 m_port `int Client::m_port [protected]`

Port to which is the client connected to.

6.3.4.6 m_socket `int Client::m_socket [protected]`

Socket for biding to server, sending and receiving data.

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

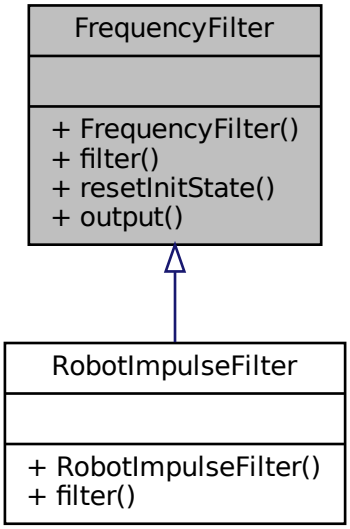
- Client.hpp
- Client.cpp

6.4 FrequencyFilter Class Reference

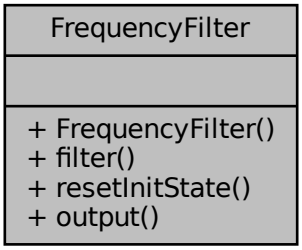
Class implementing a frequency filter.

`#include <FrequencyFilter.hpp>`

Inheritance diagram for FrequencyFilter:



Collaboration diagram for FrequencyFilter:



Public Member Functions

- FrequencyFilter (double alpha, double initState=0)

  Constructor of the low pass filter.
- virtual double filter (double input)

  Add a new value to the output of the filter.
- double resetInitState (double newValue)

  Resets the output state of the filter to the supplied value.
- double output ()

  Returns the state of the output of the filter.

6.4.1 Detailed Description

Class implementing a frequency filter.

This class can be used to filter out the high frequency of a signal. To initialize an object You have to supply the `alpha` value. It represents the percentage of how much should the old output value effect the new output value.

6.4.2 Constructor & Destructor Documentation

6.4.2.1 FrequencyFilter() `FrequencyFilter::FrequencyFilter (`

```
double alpha,
double initState = 0 )  [explicit]
```

Constructor of the low pass filter.

The filter setups the initial values. These supers the high frequency of the supplied signal.

Parameters

alpha	Is a value in range [0;1). Specifies the percentage of how much should the old value effect the output state.
initState	State at which should the filter start.

Here is the caller graph for this function:

6.4.3 Member Function Documentation

6.4.3.1 filter() `double FrequencyFilter::filter (`

```
double input )  [virtual]
```

Add a new value to the output of the filter.

Parameters

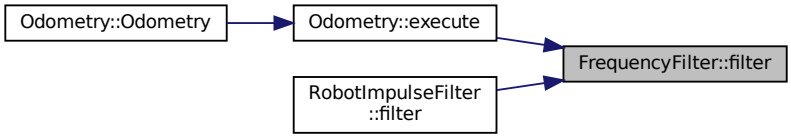
input	The new sample from the input signal.

Returns

New output value already counted with the input.

Reimplemented in RobotImpulseFilter.

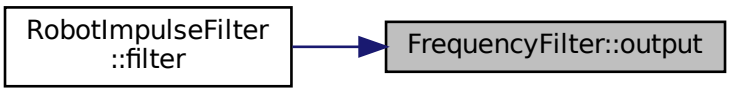
Here is the caller graph for this function:



6.4.3.2 output() `double FrequencyFilter::output ( )`

Returns the state of the output of the filter.

Here is the caller graph for this function:



6.4.3.3 resetInitState() `double FrequencyFilter::resetInitState ( double newValue )`

Resets the output state of the filter to the supplied value.

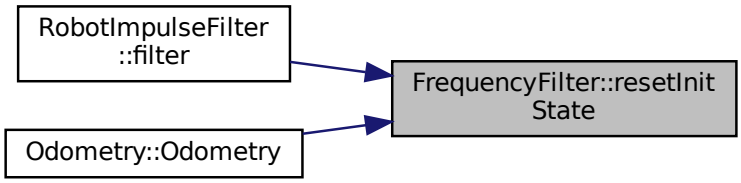
Parameters

newValue	State the filter will be after reset.

Returns

The old state of the filter.

Here is the caller graph for this function:



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

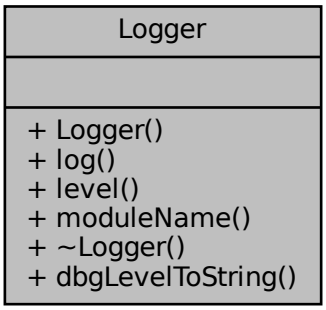
- FrequencyFilter.hpp
- FrequencyFilter.cpp

6.5 Logger Class Reference

Class handling all the debugging from the macros in log.hpp.

`#include <Logger.hpp>`

Collaboration diagram for Logger:



Public Types

- enum class level {
  DBG , INFO , WARN , ERR ,
  FATAL , SUCCESS , OFF }

  Logging levels ordered from the most verbose to the least verbose.

Public Member Functions

- Logger (const char *module, level lvl=level::INFO)

  Constructor.
- void log (const level dbgLevel, const char *codePath, pid_t pid, const char *message, const char *color="")

  Log message at a specific codePath with a colour to the screen.
- level level ()
- std::string moduleName ()
- ~Logger ()

Static Public Member Functions

- static const char * dbgLevelToString (const level level)

  Converts the log level to a specified string.

6.5.1 Detailed Description

Class handling all the debugging from the macros in log.hpp.

The ros2 logger does not support the functionality that is in this file. The builtin logger does not define modules but writes all the logs to the same file. The structure is different, too. Therefore I have created a Logger class that suits my needs.

TODO: The logging level could be changed during the runtime. There could be a file containing names of all the modules with a default logging level. The logging level could be taken from the file on every print or just on initialization.

WARN: This will slower the program significantly. This would be better with QT5 and FileWatcher.

6.5.2 Member Enumeration Documentation

6.5.2.1 level `enum enum Logger::level Logger::level [strong]`

Logging levels ordered from the most verbose to the least verbose.

Enumerator

DBG	
INFO	
WARN	
ERR	
FATAL	
SUCCESS	
OFF	

6.5.3 Constructor & Destructor Documentation

6.5.3.1 Logger()

```
Logger::Logger (
        const char * module,
        enum Logger::level lvl = level::INFO )  [explicit]
```

Constructor.

6.5.3.2 ~Logger()

```
Logger::~Logger ( )
```

Free the resources.

6.5.4 Member Function Documentation

6.5.4.1 dbgLevelToString()

```
const char * Logger::dbgLevelToString (
        const level level )  [static]
```

Converts the log level to a specified string.

Parameters

level	Log level to be converted.

Returns

log level name in string type.

Here is the caller graph for this function:

6.5.4.2 level() `level Logger::level ( )`

Prints info about the class.

Returns

The logging level for the module.

6.5.4.3 log()

```
void Logger::log (
        const level dbgLevel,
        const char * codePath,
        pid_t pid,
        const char * message,
        const char * color = "" )
```

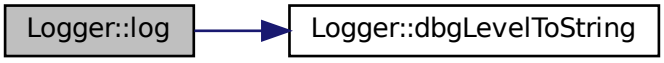
Log message at a specific codePath with a colour to the screen.

The same log is printed to the file named according to the module name. In the file is a timestamp at the beginning of the logging message. This method is counting with multiple threads writing to the standard output and to the same file. Thus the printing is guarded with scoped mutex.

Parameters

dbgLevel	Level of the log that is to be printed.
codePath	Function name where the logging message is supported.
pid	Process printing the message.
message	Message to be printed.
color	Color of the text based on the logging level.

Here is the call graph for this function:



6.5.4.4 moduleName() `std::string Logger::moduleName ( )`

Returns the name of the module.

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

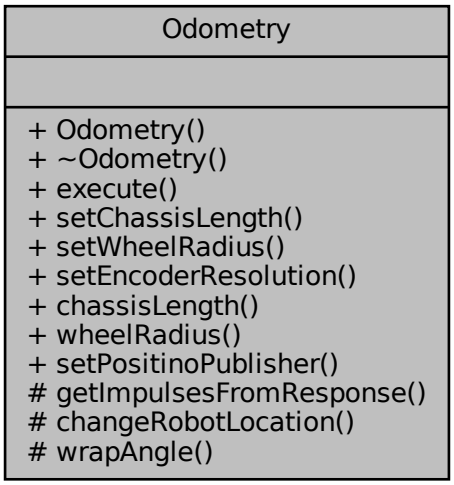
- Logger.hpp
- Logger.cpp

6.6 Odometry Class Reference

Manages the received data containing left and right wheel speed using the control server.

`#include <Odometry.hpp>`

Collaboration diagram for Odometry:



Classes

- class Speed

  Stores the speeds of the left and right wheel obtained from the robot.

Public Member Functions

- Odometry (std::shared_ptr< RobotDataDelegator > robotDataDelegator)

  Creates an odometry object. This object polls the server for its velocity. Therefore the object should call the execute function in different thread.
- ~Odometry ()

  Destructor.
- void execute ()

  Function polling the robot for its left and right wheel speed. WARN: The function WILL block the thread, thus it must be run in different thread.
- void setChassisLength (double chassisLength)

  Sets the length of the robot chassis.
- void setWheelRadius (double wheelRadius)

  Sets the radius of left and right wheel speed.
- void setEncoderResolution (int encoderResolution)

  Sets the encoder resolution.
- double chassisLength () const

  Returns the length of the chassis.
- double wheelRadius () const

  Returns the radius of the wheels.
- void setPositinoPublisher (rclcpp::Publisher< nav_msgs::msg::Odometry >::SharedPtr positionPublisher)

  Sets the position publisher created by `rclcpp::Node`.

Protected Member Functions

- Speed getImpulsesFromResponse (RobotResponseType &&response) const

  Retrieve the left and right wheel speed from the received json message.
- void changeRobotLocation (Speed &&speed)

  Changes the robot location based on the left and right wheel velocity.
- double wrapAngle (double angle) const

  Computes the angle of the robot in interval [-pi, pi] on Cartesian plain.

6.6.1 Detailed Description

Manages the received data containing left and right wheel speed using the control server.

With a set frequency sends requests to server and evaluates the position, of where it is located in the Cartesian plain. The beginning position after turning on the robot is [0, 0, 0]. The coordinates are represented as [x, y, quaternion].

6.6.2 Constructor & Destructor Documentation

6.6.2.1 Odometry() `Odometry::Odometry (`

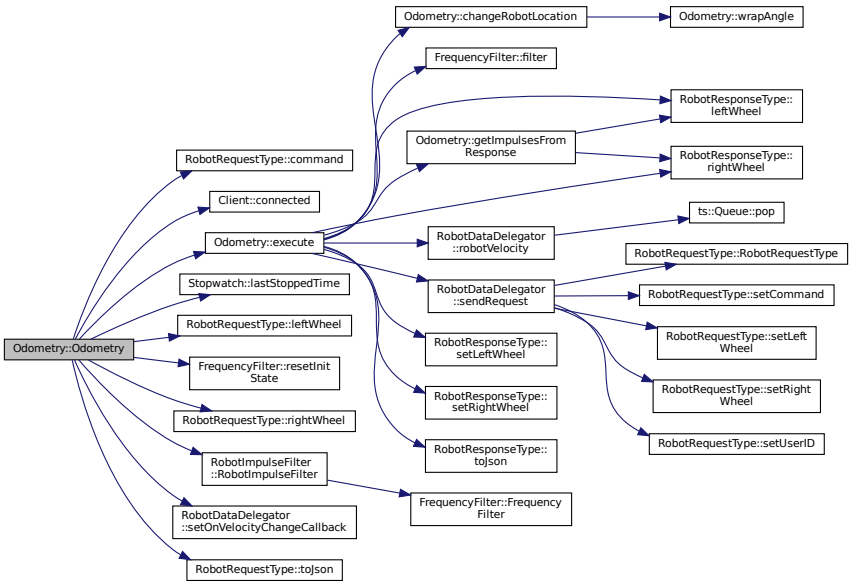
`std::shared_ptr< RobotDataDelegator > robotDataDelegator ) [explicit]`

Creates an odometry object. This object polls the server for its velocity. Therefore the object should call the execute function in different thread.

Parameters

controlClient	Reference to the control software connected to the robot.

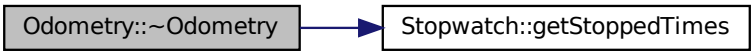
Here is the call graph for this function:



6.6.2.2 ~Odometry() `Odometry::~Odometry ( )`

Destructor.

Here is the call graph for this function:



6.6.3 Member Function Documentation

6.6.3.1 changeRobotLocation() `void Odometry::changeRobotLocation ( Speed && speed ) [protected]`

Changes the robot location based on the left and right wheel velocity.

The location is calculated from the period of the polling execute function and the time that takes to obtain the velocities of the wheels.

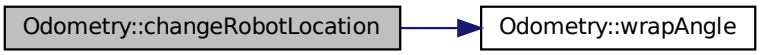
See also

execute The polling function.

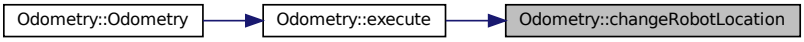
Parameters

speed	Rvalue reference to a structure containing the left and right wheel velocity.

Here is the call graph for this function:



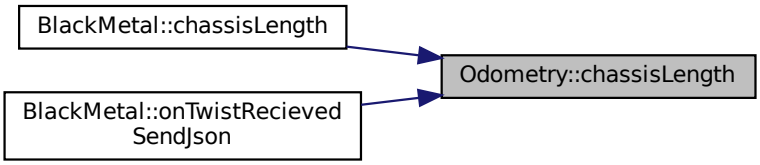
Here is the caller graph for this function:



6.6.3.2 chassisLength() `double Odometry::chassisLength ( ) const`

Returns the length of the chassis.

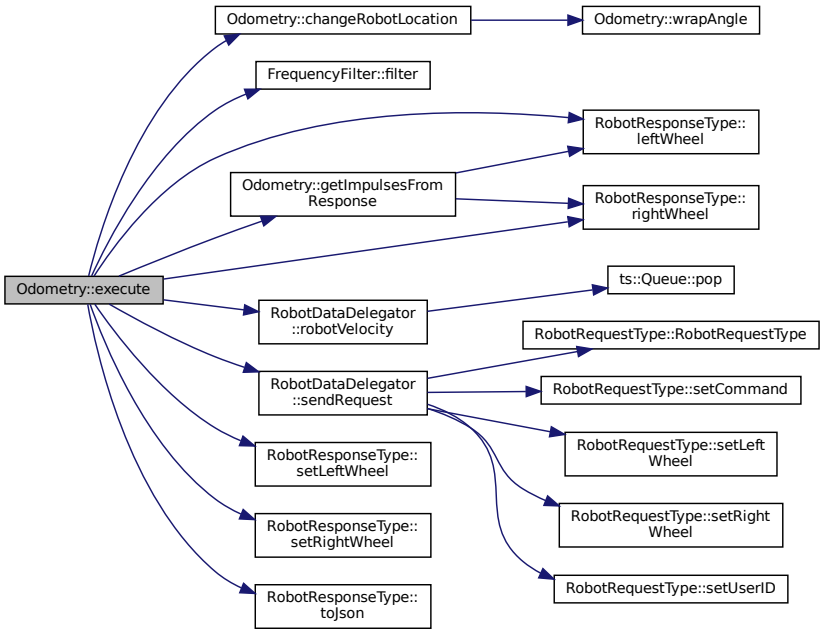
Here is the caller graph for this function:



6.6.3.3 execute() `void Odometry::execute ( )`

Function polling the robot for its left and right wheel speed. WARN: The function WILL block the thread, thus it must be run in different thread.

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.3.4 getImpulsesFromResponse() `Odometry::Speed Odometry::getImpulsesFromResponse ( RobotResponseType && response ) const [protected]`

Retrieve the left and right wheel speed from the received json message.

See also

m_leftWheelImpulseFilter Low pass filter filtering the impulses of the left wheel.

m_rightWheelImpulseFilter Low pass filter filtering the impulses of the right wheel.

@warn This method inverts the right wheel speed so that we could calculate the position of the robot.

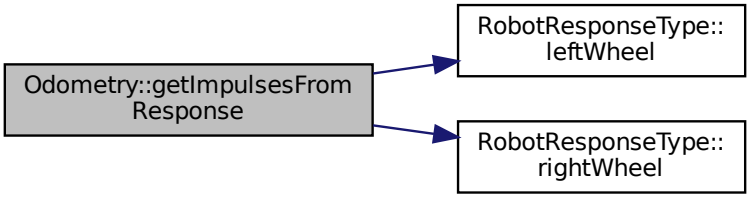
Parameters

response	Message received from the server.

Returns

`Speed` structure of left and right wheel speed.

Here is the call graph for this function:



Here is the caller graph for this function:

6.6.3.5 setChassisLength()

```
void Odometry::setChassisLength (
        double chassisLength )
```

Sets the length of the robot chassis.

Parameters

chassisLength	Value to be set.

6.6.3.6 setEncoderResolution()

```
void Odometry::setEncoderResolution (
        int encoderResolution )
```

Sets the encoder resolution.

Parameters

encoderResolution	Value to be set.

6.6.3.7 setPositinoPublisher() `void Odometry::setPositinoPublisher ( rclcpp::Publisher< nav_msgs::msg::Odometry >::SharedPtr positionPublisher )`

Sets the position publisher created by `rclcpp::Node`.

Parameters

positionPublisher	Publisher of the odometry messages.

6.6.3.8 setWheelRadius() `void Odometry::setWheelRadius ( double wheelRadius )`

Sets the radius of left and right wheel speed.

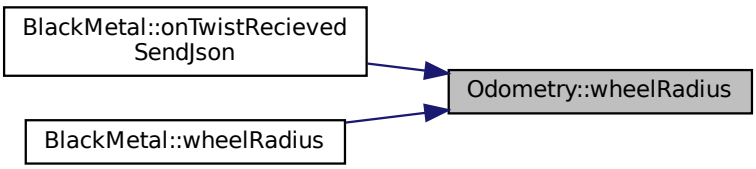
Parameters

wheelRadius	Value to be set.

6.6.3.9 wheelRadius() `double Odometry::wheelRadius ( ) const`

Returns the radius of the wheels.

Here is the caller graph for this function:



6.6.3.10 wrapAngle() double Odometry::wrapAngle (double *angle*) const [protected]

Computes the angle of the robot in interval [-pi, pi] on Cartesian plain.

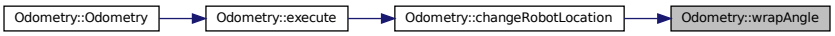
Parameters

angle	Angle to be wrapped in radians.

Returns

Wrapped angle in radians.

Here is the caller graph for this function:



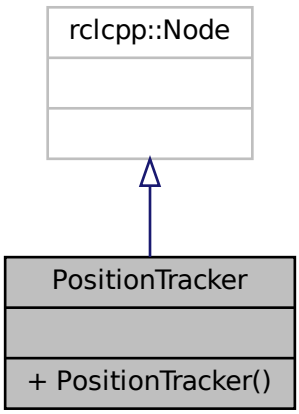
The documentation for this class was generated from the following files:

- Odometry.hpp
- Odometry.cpp

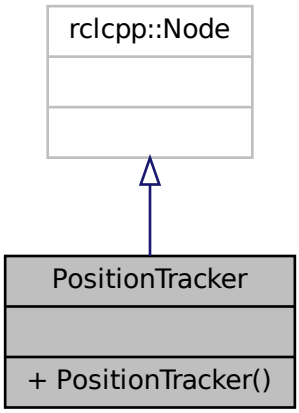
6.7 PositionTracker Class Reference

Class for tracking and logging robot location.

Inheritance diagram for PositionTracker:



Collaboration diagram for PositionTracker:



Public Member Functions

- PositionTracker ()

6.7.1 Detailed Description

Class for tracking and logging robot location.

This class connects to the topic "position" and logs the received messages.

6.7.2 Constructor & Destructor Documentation

6.7.2.1 PositionTracker() `PositionTracker::PositionTracker ( ) [inline]`

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

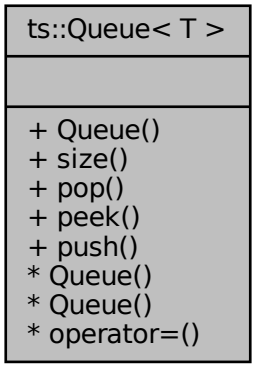
- position_tracker.cpp

6.8 ts::Queue< T > Class Template Reference

Thread safe dynamic templated priority queue.

`#include <Queue.hpp>`

Collaboration diagram for ts::Queue< T >:



Public Member Functions

- Queue (const std::string &name)
  Default constructor.
- unsigned long size ()
  Returns the size of the priority queue.
- T pop ()
  Returns the top element in the priority queue and removes it from the internal structure.
- T peek ()
  Returns the copy of the top element in the priority queue.
- void push (const T &item)
  Add a new element to the priority queue.

6.8.1 Detailed Description

template<typename T>
class ts::Queue< T >

Thread safe dynamic templated priority queue.

The queue is implemented using `pqueue<T>`, `std::mutex` and `std::condition_variable`. When you want to pop an element from the queue the thread will be blocked until you push an element from other thread.

See also

pqueue<T> For more information about the priority queue.

Template Parameters

T	The type of the elements stored in the queue.

6.8.2 Constructor & Destructor Documentation

6.8.2.1 Queue() `template<typename T >`

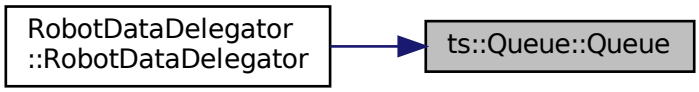
```
ts::Queue< T >::Queue (
            const std::string & name )  [explicit]
```

Default constructor.

Parameters

name	The name of the queue used for debugging.

Here is the caller graph for this function:



6.8.3 Member Function Documentation

6.8.3.1 peek() `template<typename T >`

```
T ts::Queue< T >::peek
```

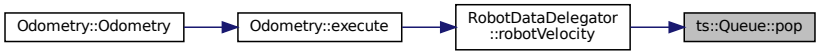
Returns the copy of the top element in the priority queue.

6.8.3.2 pop() template<typename T >

T ts::Queue< T >::pop

Returns the top element in the priority queue and removes it from the internal structure.

Here is the caller graph for this function:



6.8.3.3 push() template<typename T >

void ts::Queue< T >::push (
const T & *item*)

Add a new element to the priority queue.

Inserts the element to the queue. If the inserted item is evaluated by the std::greater<T>() as grater the item is prioritized.

Parameters

item	to be added to the queue.

6.8.3.4 size() template<typename T >

unsigned long ts::Queue< T >::size

Returns the size of the priority queue.

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

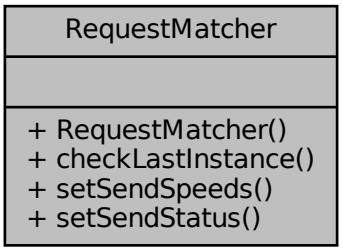
- Queue.hpp

6.9 RequestMatcher Class Reference

Class used for filtering out the sending request data.

#include <RequestMatcher.hpp>

Collaboration diagram for RequestMatcher:



Public Member Functions

- RequestMatcher (const std::pair< double, double > &speeds)

  Constructor.
- bool checkLastInstance (const std::pair< double, double > &speeds)

  Checks if the message matches the last send message.
- RequestMatcher & setSendSpeeds (const std::pair< double, double > &speeds)

  Sets the last send message to be matched.
- RequestMatcher & setSendStatus (bm::Status status)

  Sets the last send status to be matched.

6.9.1 Detailed Description

Class used for filtering out the sending request data.

If the requests parameters to be send are the same as the last send data and the last send was successful. The message is not send. The robot has set timer to stop execution after 10 seconds, therefore we set a timelimit to 9 seconds. If this limit is overstepped we send the next message even if it is a duplicate.

6.9.2 Constructor & Destructor Documentation

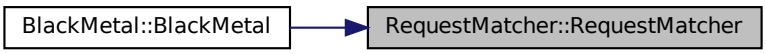
6.9.2.1 RequestMatcher() `RequestMatcher::RequestMatcher ( const std::pair< double, double > & speeds ) [explicit]`

Constructor.

Parameters

speeds	Pair of doubles to be set as a default request parameters to be matched.

Here is the caller graph for this function:



6.9.3 Member Function Documentation

6.9.3.1 checkLastInstance() `bool RequestMatcher::checkLastInstance ( const std::pair< double, double > & speeds )`

Checks if the message matches the last send message.

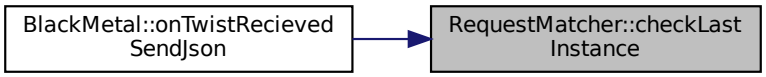
Parameters

msg	Message to be matched.

Returns

True if the `msg` is the last send message, false otherwise.

Here is the caller graph for this function:



6.9.3.2 setSendSpeeds() `RequestMatcher & RequestMatcher::setSendSpeeds ( const std::pair< double, double > & speeds )`

Sets the last send message to be matched.

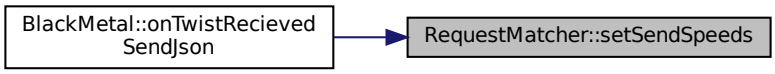
Parameters

msg	String to be set as a last send message.

Returns

Reference to this object.

Here is the caller graph for this function:



6.9.3.3 setSendStatus() `RequestMatcher & RequestMatcher::setSendStatus ( bm::Status status )`

Sets the last send status to be matched.

Parameters

status	Status to be set as a last send status.

Returns

Reference to this object.

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

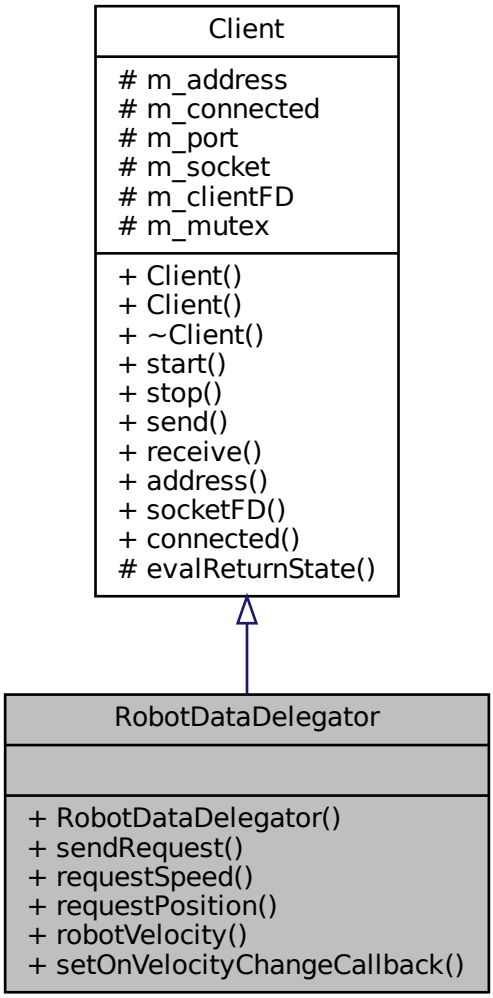
- RequestMatcher.hpp
- RequestMatcher.cpp

6.10 RobotDataDelegator Class Reference

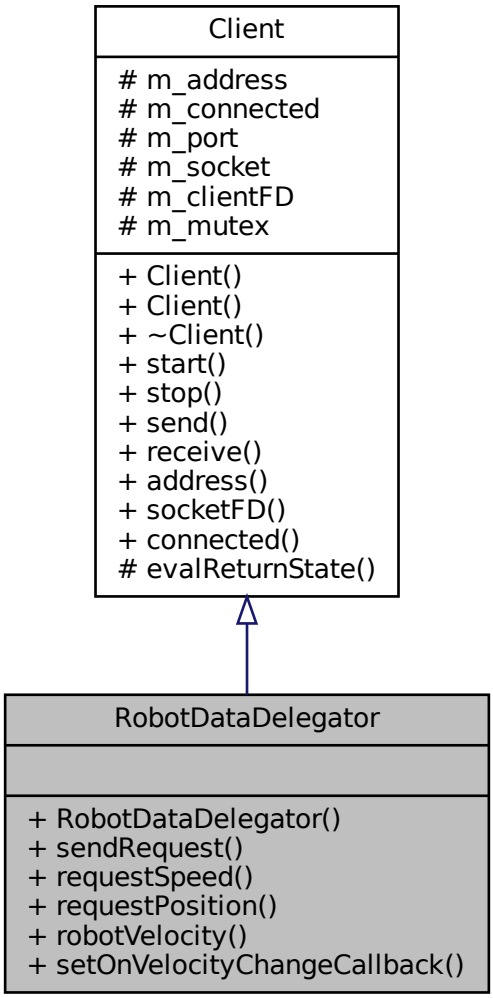
Child class implementing Client class and operating with robot messages.

`#include <RobotDataDelegator.hpp>`

Inheritance diagram for RobotDataDelegator:



Collaboration diagram for RobotDataDelegator:



Public Member Functions

- RobotDataDelegator (const std::string &address, int port, std::shared_ptr< RequestMatcher > matcher)

  Constructor implementing `Client` constructor.
- virtual bm::Status sendRequest (bm::Command cmd, RobotRequestType::WheelValueT rightWheel=0, RobotRequestType::WheelValueT leftWheel=0)

  Forms a json string out of supplied parameters and enqueues them.
- bm::Status requestSpeed (double rightWheel, double leftWheel)

  This function calls the `sendRequest` with `bm::Command::SET_LR_WHEEL_VELOCITY`.
- bm::Status requestPosition (long rightWheel, long leftWheel)

  This function calls the `sendRequest` with `bm::Command::SET_LR_WHEEL_POSITION`.
- RobotResponseType robotVelocity ()

  Returns the first json message that is located in the odometry queue.
- void setOnVelocityChangeCallback (std::function< void(RobotRequestType)> onVelocityChange)

  Sets the callabck function for resetting the filter when the robot velocity changes.

Additional Inherited Members

6.10.1 Detailed Description

Child class implementing Client class and operating with robot messages.

This class sends json requests via the `Client::send()` method. The parameters to send are supplied in method sendRequest. Method `receive` receives data from robot. Evaluetes the data and pushes them to separate ts::Queue so that Odometry class could take them. If the received data consist of multiple messages the message is split and every resonse is evaluated separatly. All this takes place in separate thread.

The order of send and received messages is organized using a thread safe queue.

See also

ts::Queue thread safe queue.

Client class used for communication with the robot.

RobotRequestType Class used for storing the request parameters.

RobotResponseType Class used for storing the response parameters.

Note

: The strings cannot be passed to parser, because the messages we receive are no according to the standard. We have to do it manually.

6.10.2 Constructor & Destructor Documentation

6.10.2.1 RobotDataDelegator() `RobotDataDelegator::RobotDataDelegator (`

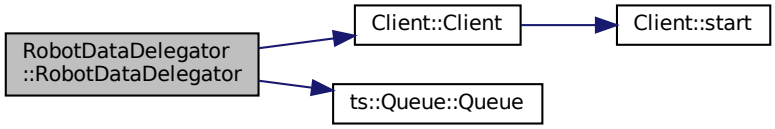
```
const std::string & address,
int port,
std::shared_ptr< RequestMatcher > matcher )
```

Constructor implementing `Client` constructor.

Parameters

address	IPv4 Address of the server.
port	Port to connect to.

Here is the call graph for this function:



6.10.3 Member Function Documentation

6.10.3.1 requestPosition() `bm::Status RobotDataDelegator::requestPosition ( long rightWheel, long leftWheel )`

This function calls the `sendRequest` with `bm::Command::SET_LR_WHEEL_POSITION`.

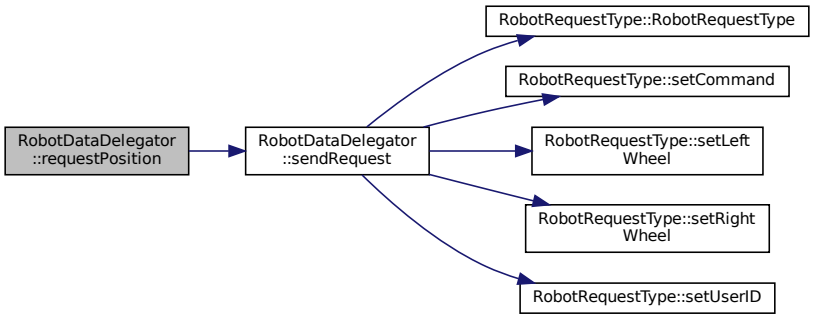
Parameters

rightWheel	Right wheel wished position.
leftWheel	Right wheel wished position.

Returns

Always returns `bm::Status::OK` after the request is enqueud.

Here is the call graph for this function:



6.10.3.2 requestSpeed()

```
bm::Status RobotDataDelegator::requestSpeed (
        double rightWheel,
        double leftWheel )
```

This function calls the `sendRequest` with `bm::Command::SET_LR_WHEEL_VELOCITY`.

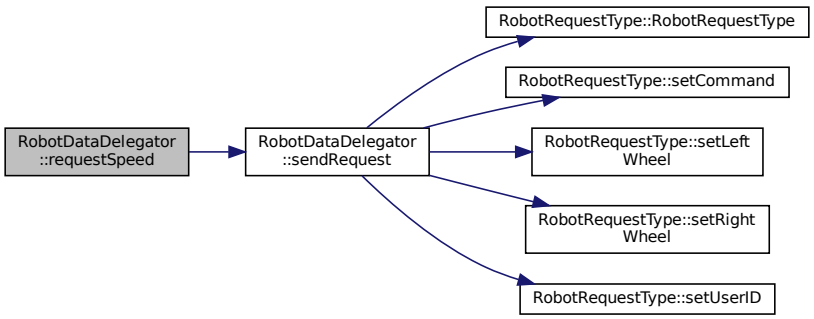
Parameters

rightWheel	Right wheel speed.
leftWheel	Right wheel speed.

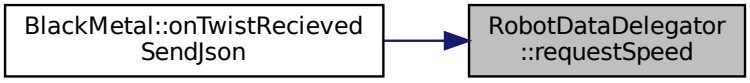
Returns

Always returns `bm::Status::OK` after the request is enqueud.

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3.3 robotVelocity()

```
RobotResponseType RobotDataDelegator::robotVelocity ( )
```

Returns the first json message that is located in the odometry queue.

Note

the queue is thread safe and will block until the message is available.

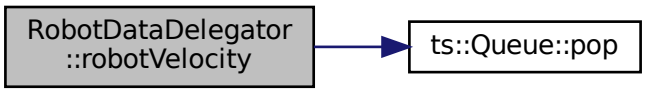
See also

ts::Queue Thread safe queue managing the recevied messages.

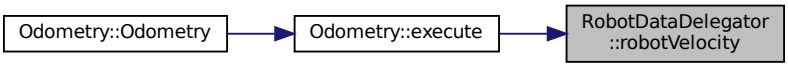
Returns

The first json message in the priority queue.

Here is the call graph for this function:



Here is the caller graph for this function:



6.10.3.4 sendRequest()

```
bm::Status RobotDataDelegator::sendRequest (
        bm::Command cmd,
        RobotRequestType::WheelValueT rightWheel = 0,
        RobotRequestType::WheelValueT leftWheel = 0 )  [virtual]
```

Forms a json string out of supplied parameters and enqueues them.

The request consists of either double or long parameters. The double parameters are used in set velocity request. The long parameters are used in the set position request.

See also

bm::Command The enumeration class of possible request commands.

RobotRequestType::WheelValueT The variant parameter of the request.

requestSpeed The function for setting the robot speed.

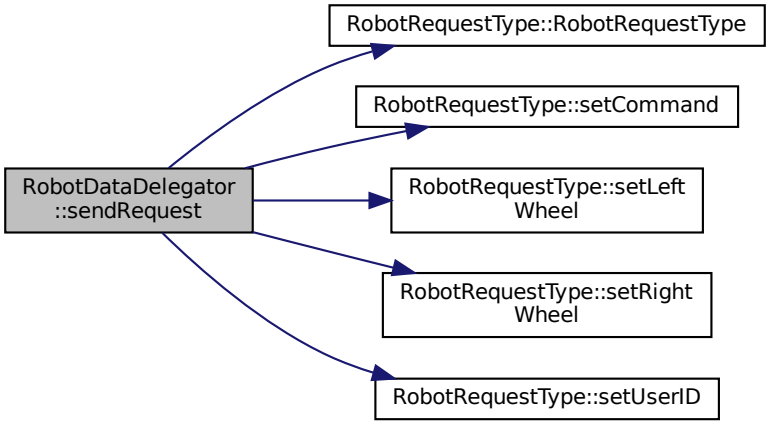
Parameters

cmd	Command which should the robot execute.
rightWheel	Right wheel speed or position. This parameter is needed only in `bm::Command::SET_LR_WHEEL_VELOCITY` and `bm::Command::SET_LR_WHEEL_POSITION`.
leftWheel	Right wheel speed or position. This parameter is needed only in `bm::Command::SET_LR_WHEEL_VELOCITY` and `bm::Command::SET_LR_WHEEL_POSITION`.

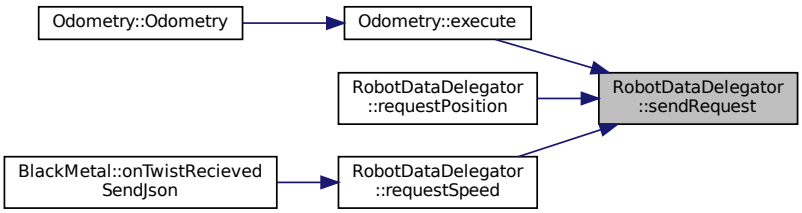
Returns

Always returns `bm::Status::OK` after the request is enqueud.

Here is the call graph for this function:



Here is the caller graph for this function:



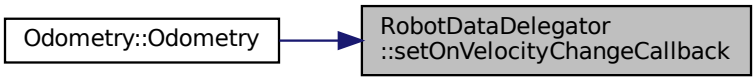
6.10.3.5 setOnVelocityChangeCallback() `void RobotDataDelegator::setOnVelocityChangeCallback ( std::function< void(RobotRequestType)> onVelocityChange )`

Sets the callabck function for resetting the filter when the robot velocity changes.

Parameters

onVelocityChange	Callabck function called when the robot velocity changes.

Here is the caller graph for this function:



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

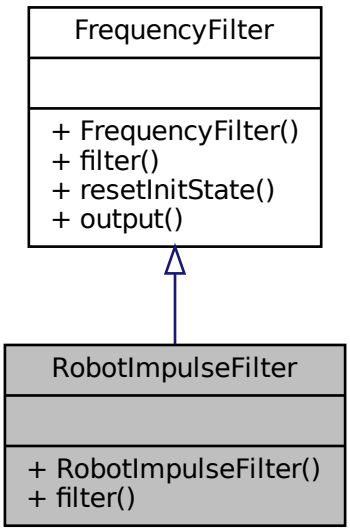
- RobotDataDelegator.hpp
- RobotDataDelegator.cpp

6.11 RobotImpulseFilter Class Reference

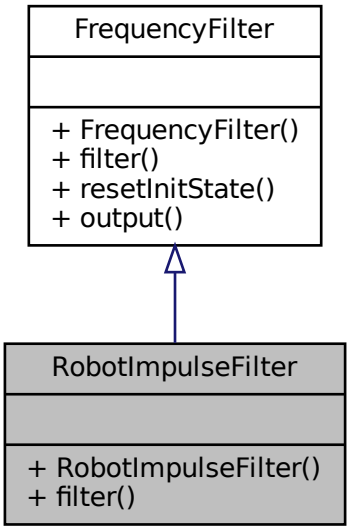
Filter used to filter the received motor impulses from the robot.

`#include <RobotImpulseFilter.hpp>`

Inheritance diagram for RobotImpulseFilter:



Collaboration diagram for RobotImpulseFilter:



Public Member Functions

- RobotImpulseFilter (double alpha)

  Constructor.
- double filter (double input) override

  The same as `FrequencyFilter::filter`, although the input state is firstly checked for the on/off switch.

6.11.1 Detailed Description

Filter used to filter the received motor impulses from the robot.

The impulses are checked for big fluctuations. Mainly the occasional '0' that sometimes comes up as a speed. This greatly breaks the filter. Thus, we do not use this values.

6.11.2 Constructor & Destructor Documentation

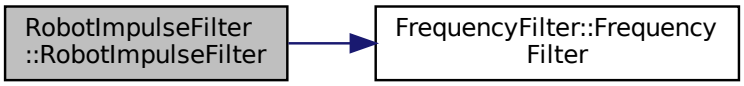
6.11.2.1 RobotImpulseFilter() `RobotImpulseFilter::RobotImpulseFilter (double alpha ) [explicit]`

Constructor.

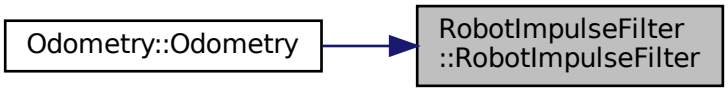
See also

FrequencyFilter Base class of this filter.

Here is the call graph for this function:



Here is the caller graph for this function:



6.11.3 Member Function Documentation

6.11.3.1 filter() `double RobotImpulseFilter::filter (double input ) [override], [virtual]`

The same as `FrequencyFilter::filter`, although the input state is firstly checked for the on/off switch.

If the input suddenly starts the high filtering would corrupt the output signal. Thus, if the input suddenly switches from the 'OFF' state (0) the output is reset to the first value. There may happen, that the input will drop to 0 for one sample. This state is filtered out. Finally because we have to calculate the output at real time, we cannot alter the older samples. Because of that the output is set to 'OFF' state (0) one sample after the real off switch is invoked.

See also

FrequencyFilter::filter method to filter the input. It is called internally in this method, too.

Parameters

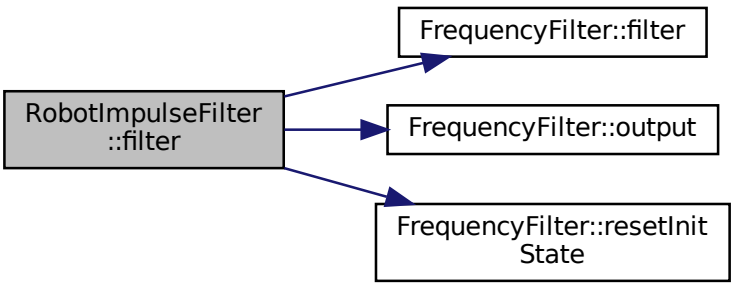
input	Sample of the input signal.

Returns

The real output of the filter.

Reimplemented from FrequencyFilter.

Here is the call graph for this function:



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

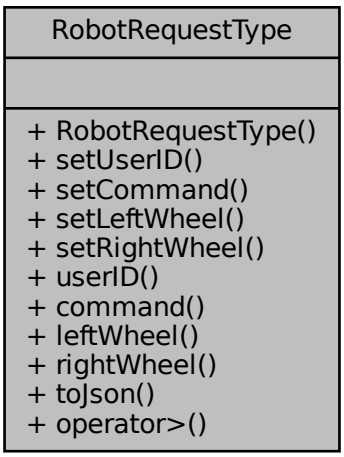
- RobotImpulseFilter.hpp
- RobotImpulseFilter.cpp

6.12 RobotRequestType Class Reference

Class for storing and managing the request parameters.

`#include <RobotRequestType.hpp>`

Collaboration diagram for RobotRequestType:



Public Types

- using WheelValueT = std::variant< long, double >

Public Member Functions

- RobotRequestType ()
  Default constructor.
- RobotRequestType & setUserID (int id)
- RobotRequestType & setCommand (bm::Command cmd)
- RobotRequestType & setLeftWheel (WheelValueT lw)
- RobotRequestType & setRightWheel (WheelValueT rw)
- int userID () const
- bm::Command command () const
- WheelValueT leftWheel () const
- WheelValueT rightWheel () const
- std::string toJson () const
- bool operator> (const RobotRequestType &other) const
  Compares two requests.

Friends

- std::ostream & operator<< (std::ostream &os, const RobotRequestType &request)

6.12.1 Detailed Description

Class for storing and managing the request parameters.

This class is used for storing the request parameters and their transformation to json string.

6.12.2 Member Typedef Documentation

6.12.2.1 WheelValueT `using RobotRequestType::WheelValueT = std::variant<long, double>`

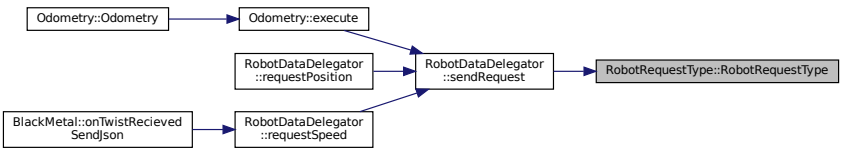
The sendRequest function takes arguments for left and right wheel for setting position and velocity. The velocity is set with double and the position is set with long. This is the generic solution for this issue.

6.12.3 Constructor & Destructor Documentation

6.12.3.1 RobotRequestType() `RobotRequestType::RobotRequestType ( ) [inline]`

Default constructor.

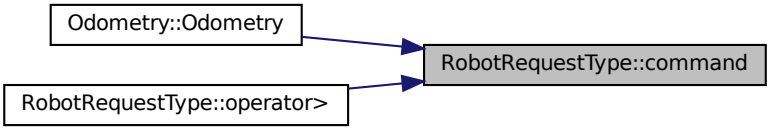
Here is the caller graph for this function:



6.12.4 Member Function Documentation

6.12.4.1 command() `bm::Command RobotRequestType::command ( ) const`

Returns the request command. Here is the caller graph for this function:



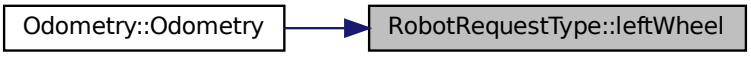
6.12.4.2 leftWheel() `RobotRequestType::WheelValueT RobotRequestType::leftWheel ( ) const`

Returns the left wheel value.

Returns

`WheelValueT` Left wheel value.

Here is the caller graph for this function:



6.12.4.3 operator>() `bool RobotRequestType::operator> ( const RobotRequestType & other ) const`

Compares two requests.

The requests are compared by their command. The lower command numbers are `EMEERGENCY_STOP`, `SET_LR_WHEEL_VELOCITY`, `SET_LR_WHEEL_POSITION`. When requests with this enqueued they have priority over the other requests.

Parameters

other	The other request to compare with.

Returns

true When the this request has higher priority.

Here is the call graph for this function:

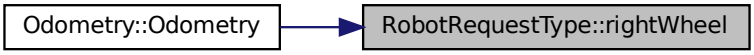
6.12.4.4 rightWheel() `RobotRequestType::WheelValueT RobotRequestType::rightWheel ( ) const`

Returns the right wheel value.

Returns

`WheelValueT` Right wheel value.

Here is the caller graph for this function:



6.12.4.5 setCommand() `RobotRequestType & RobotRequestType::setCommand ( bm::Command cmd )`

Sets the command.

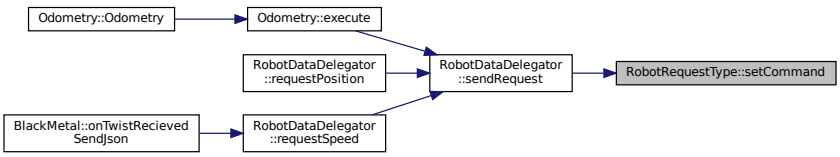
Parameters

cmd	Command to be set in the request.

Returns

RobotRequestType &Reference to this object.

Here is the caller graph for this function:



6.12.4.6 setLeftWheel() `RobotRequestType & RobotRequestType::setLeftWheel ( WheelValueT lw )`

Sets the left wheel value.

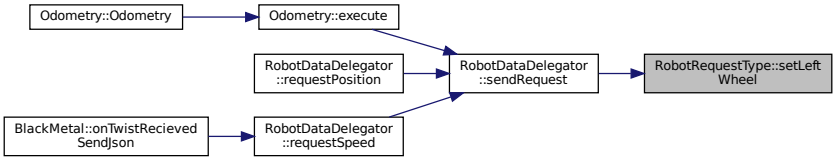
Parameters

lw	Left wheel value to be set in the request. This applies only in `bm::Command::SET_LR_WHEEL_VELOCITY` and `bm::Command::SET_LR_WHEEL_POSITION`.

Returns

RobotRequestType &Reference to this object.

Here is the caller graph for this function:



6.12.4.7 setRightWheel() `RobotRequestType & RobotRequestType::setRightWheel ( WheelValueT rw )`

Sets the right wheel value.

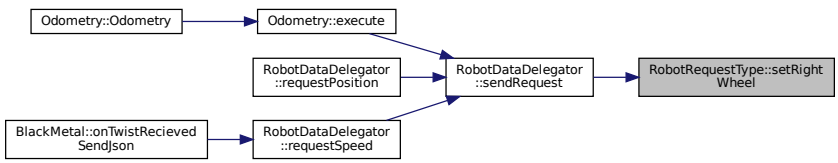
Parameters

rw	Right wheel value to be set in the request. This applies only in `bm::Command::SET_LR_WHEEL_VELOCITY` and `bm::Command::SET_LR_WHEEL_POSITION`.

Returns

RobotRequestType &Reference to this object.

Here is the caller graph for this function:



6.12.4.8 setUserID() `RobotRequestType & RobotRequestType::setUserID ( int id )`

Sets the user ID.

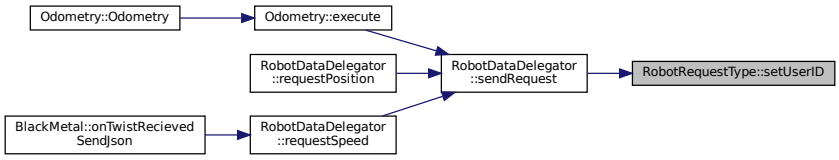
Parameters

id	User ID to be set in the request.

Returns

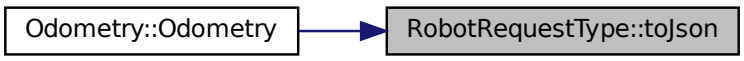
RobotRequestType &Reference to this object.

Here is the caller graph for this function:



6.12.4.9 toJson() `std::string RobotRequestType::toJson ( ) const`

Forms a json string out of the request. Here is the caller graph for this function:



6.12.4.10 userID() `int RobotRequestType::userID ( ) const`

Returns the user ID.

Returns

int User ID.

6.12.5 Friends And Related Function Documentation

6.12.5.1 operator<< std::ostream& operator<< (std::ostream & os, const RobotRequestType & request) [friend]

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

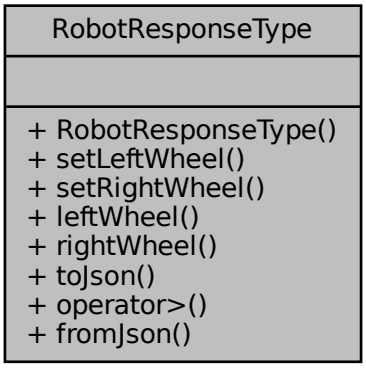
- RobotRequestType.hpp
- RobotRequestType.cpp

6.13 RobotResponseType Class Reference

Json response from the robot.

`#include <RobotResponseType.hpp>`

Collaboration diagram for RobotResponseType:



Public Member Functions

- RobotResponseType ()=default

  Default constructor.
- RobotResponseType & setLeftWheel (long leftWheel)
- RobotResponseType & setRightWheel (long rightWheel)
- long leftWheel () const
- long rightWheel () const
- std::string toJson () const
- bool operator> (const RobotResponseType &other) const

Static Public Member Functions

- static RobotResponseType fromJson (const std::string &json)

  Creates the object from the json representation.

6.13.1 Detailed Description

Json response from the robot.

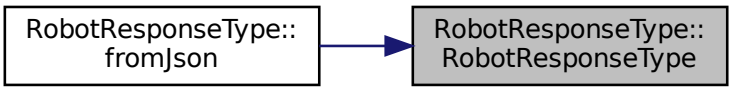
This class represents the json response from the robot. There is a possibility to set the left and right wheel speed. The class also provides a method for converting the object to json and vice versa.

6.13.2 Constructor & Destructor Documentation

6.13.2.1 RobotResponseType() `RobotResponseType::RobotResponseType ( ) [default]`

Default constructor.

Here is the caller graph for this function:



6.13.3 Member Function Documentation

6.13.3.1 fromJson() `RobotResponseType RobotResponseType::fromJson ( const std::string & json ) [static]`

Creates the object from the json representation.

The function parses the json and creates the object from it.

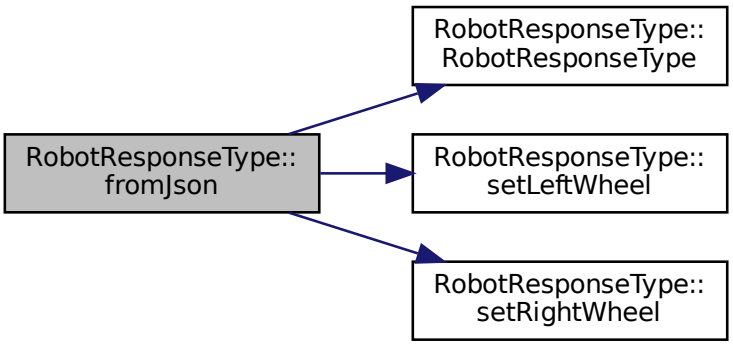
Parameters

json	Json representation of the object.

Returns

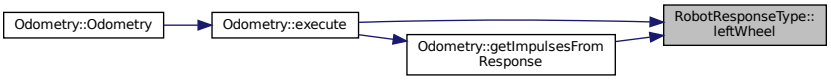
Object created from the json.

Here is the call graph for this function:



6.13.3.2 leftWheel() `long RobotResponseType::leftWheel ( ) const`

Get the left wheel speed. Here is the caller graph for this function:



6.13.3.3 operator>() `bool RobotResponseType::operator> ( const RobotResponseType & other ) const`

Compares the left and right wheel speed.

Parameters

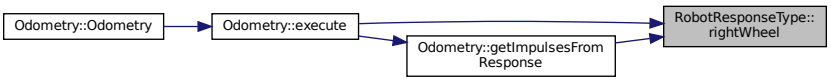
other	Object to be compared with.

Returns

Always false

6.13.3.4 rightWheel() `long RobotResponseType::rightWheel ( ) const`

Get the right wheel speed. Here is the caller graph for this function:



6.13.3.5 setLeftWheel() `RobotResponseType & RobotResponseType::setLeftWheel ( long leftWheel )`

Sets the left wheel speed in the json response representation.

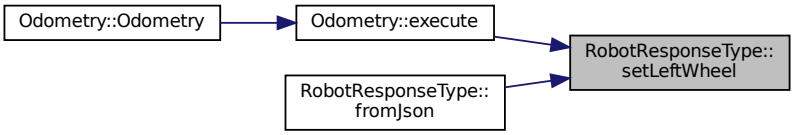
Parameters

leftWheel	Speed to be set in the json.

Returns

Reference to the object.

Here is the caller graph for this function:



6.13.3.6 setRightWheel() `RobotResponseType & RobotResponseType::setRightWheel ( long rightWheel )`

Sets the right wheel speed in the json response representation.

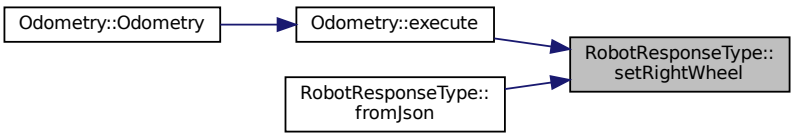
Parameters

rightWheel	Speed to be set in the json.

Returns

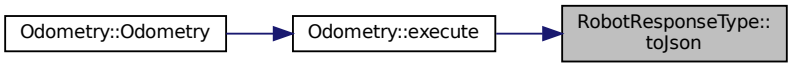
Reference to the object.

Here is the caller graph for this function:



6.13.3.7 toJson() `std::string RobotResponseType::toJson ( ) const`

Returns the representation of the class in json format. Here is the caller graph for this function:



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

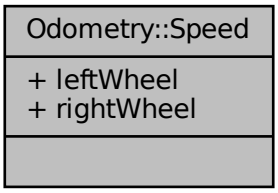
- RobotResponseType.hpp
- RobotResponseType.cpp

6.14 Odometry::Speed Class Reference

Stores the speeds of the left and right wheel obtained from the robot.

`#include <Odometry.hpp>`

Collaboration diagram for Odometry::Speed:



Public Attributes

- double leftWheel

  The velocity of left wheel.
- double rightWheel

  The velocity of right wheel.

6.14.1 Detailed Description

Stores the speeds of the left and right wheel obtained from the robot.

6.14.2 Member Data Documentation

6.14.2.1 leftWheel `double Odometry::Speed::leftWheel`

The velocity of left wheel.

6.14.2.2 rightWheel `double Odometry::Speed::rightWheel`

The velocity of right wheel.

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

- Odometry.hpp

6.15 Stopwatch Class Reference

Benchmark the program using the RAII procedure.

`#include <Stopwatch.hpp>`

Collaboration diagram for Stopwatch:

Public Member Functions

- Stopwatch (size_t maxLength=1000)
- ~Stopwatch ()

Static Public Member Functions

- static double lastStoppedTime ()
- static double stoppedTimeAt (const std::vector< double >::size_type index)

  Access a specified stopped time. If the index parameter is bigger than the size of the vector the last stopped time is returned.
- static std::vector< double > getStoppedTimes ()

6.15.1 Detailed Description

Benchmark the program using the RAII procedure.

Use this class in separate scope with what you want to measure. The class will than save the measured time in a static vector. You can later access the times separately or you can access the vector as a whole using static methods. TODO: The possibility of measuring the time in different units. Default microseconds. This can be done using templates while checking the type in constructor.

6.15.2 Constructor & Destructor Documentation

6.15.2.1 Stopwatch() `Stopwatch::Stopwatch ( size_t maxLength = 1000 )`

Constructor saves a starting timestamp.

Parameters

maxLength	The maximal number of timestamps this instance will allow.

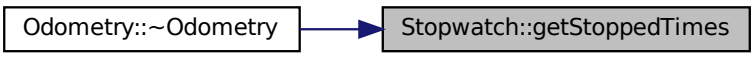
6.15.2.2 ~Stopwatch() `Stopwatch::~Stopwatch ( )`

Destructor calculates the time of its life and saves the time in microseconds (double) to static vector.

6.15.3 Member Function Documentation

6.15.3.1 getStoppedTimes() `std::vector< double > Stopwatch::getStoppedTimes ( ) [static]`

Get the copy of all the stopped times. The vector capturing them is cleared. Here is the caller graph for this function:



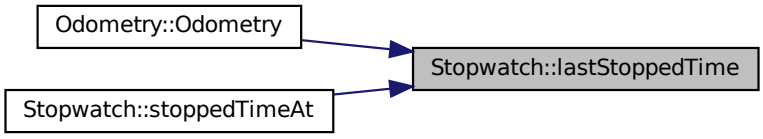
6.15.3.2 lastStoppedTime() `double Stopwatch::lastStoppedTime ( ) [static]`

Access the last stopped time in the vector of stopped times.

Returns

Copy of the last stopped time.

Here is the caller graph for this function:



6.15.3.3 stoppedTimeAt()

```
double Stopwatch::stoppedTimeAt (
        const std::vector< double >::size_type index )  [static]
```

Access a specified stopped time. If the index parameter is bigger than the size of the vector the last stopped time is returned.

Parameters

index	Index of the timestamp to access. To access the last parameter enter -1.

Returns

The const double reference to the timestamp.

Here is the call graph for this function:

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

- Stopwatch.hpp
- Stopwatch.cpp

Index