TTK4900 Driver documentation

Generated by Doxygen 1.9.1

Chapter 1

Class Index

1.1 Class List

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

Chapter 2

File Index

2.1 File List

Here is a list of all files with brief descriptions:

Chapter 3

Class Documentation

3.1 accelerometer_inData Struct Reference

Holds INCOMING accelerometer data, NOT part of the accelerometer driver.

```
#include <joint_controller.h>
```

Public Attributes

- int16_t xAcc

  X axis acceleration.
- int16_t xRot

  X axis rotation.
- int16_t yAcc

  Y axis acceleration.
- int16_t yRot

  Y axis rotation.
- int16_t zAcc

  Z axis acceleration.
- int16_t zRot

  Z axis rotation.
- uint8_t newXAcc

  Flags new X acceleration data on arrival.
- uint8_t newYAcc

  Flags new Y acceleration data on arrival.
- uint8_t newZAcc

  Flags new Z acceleration data on arrival.
- uint8_t newXRot

  Flags new X rotation data on arrival.
- uint8_t newYRot

  Flags new Y rotation data on arrival.
- uint8_t newZRot

  Flags new Z rotation data on arrival.

3.1.1 Detailed Description

Holds INCOMING accelerometer data, NOT part of the accelerometer driver.

3.1.2 Member Data Documentation

3.1.2.1 newXAcc

```
uint8_t accelerometer_inData::newXAcc
```

Flags new X acceleration data on arrival.

3.1.2.2 newXRot

```
uint8_t accelerometer_inData::newXRot
```

Flags new X rotation data on arrival.

3.1.2.3 newYAcc

```
uint8_t accelerometer_inData::newYAcc
```

Flags new Y acceleration data on arrival.

3.1.2.4 newYRot

```
uint8_t accelerometer_inData::newYRot
```

Flags new Y rotation data on arrival.

3.1.2.5 newZAcc

```
uint8_t accelerometer_inData::newZAcc
```

Flags new Z acceleration data on arrival.

3.1.2.6 newZRot

```
uint8_t accelerometer_inData::newZRot
```

Flags new Z rotation data on arrival.

3.1.2.7 xAcc

```
int16_t accelerometer_inData::xAcc
```

X axis acceleration.

3.1.2.8 xRot

```
int16_t accelerometer_inData::xRot
```

X axis rotation.

3.1.2.9 yAcc

```
int16_t accelerometer_inData::yAcc
```

Y axis acceleration.

3.1.2.10 yRot

```
int16_t accelerometer_inData::yRot
```

Y axis rotation.

3.1.2.11 zAcc

```
int16_t accelerometer_inData::zAcc
```

Z axis acceleration.

3.1.2.12 zRot

```
int16_t accelerometer_inData::zRot
```

Z axis rotation.

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

- joint_controller.h

3.2 can_mailbox Struct Reference

A virtual CAN mailbox for outgoing and incoming messages.

```
#include <can_driver.h>
```

Public Attributes

- uint8_t newMsg

  Flag signifying that the mailbox contains an unhandled message.
- uint32_t msgId

  CAN message ID, 11 bits (standard ID)
- uint8_t data [8]

  CAN message data.

3.2.1 Detailed Description

A virtual CAN mailbox for outgoing and incoming messages.

3.2.2 Member Data Documentation

3.2.2.1 data

```
uint8_t can_mailbox::data[8]
```

CAN message data.

3.2.2.2 msgId

```
uint32_t can_mailbox::msgId
```

CAN message ID, 11 bits (standard ID)

3.2.2.3 newMsg

```
uint8_t can_mailbox::newMsg
```

Flag signifying that the mailbox contains an unhandled message.

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

- can_driver.h

3.3 current_measurement_descriptor Struct Reference

Database with key information for the motor current sensing ADCs.

```
#include <adc_driver.h>
```

Public Attributes

- double VrefA

  Analog reference voltage, V.
- double Ripropi

  Current sense resistor, Ohm.
- double Aipropi

  Current sense proportional current, uA/A.
- uint32_t Nadc

  ADC saturation point.
- ADC_HandleTypeDef ∗ adc

  ADC for current measurement.
- double conversionConst

  Constant number for conversion of ADC value to Ampere.
- uint32_t lastReading

  Latest ADC raw value.
- double lastMeasurement

  Latest calculated current.

3.3.1 Detailed Description

Database with key information for the motor current sensing ADCs.

3.3.2 Member Data Documentation

3.3.2.1 adc

```
ADC_HandleTypeDef* current_measurement_descriptor::adc
```

ADC for current measurement.

3.3.2.2 Aipropi

```
double current_measurement_descriptor::Aipropi
```

Current sense proportional current, uA/A.

3.3.2.3 conversionConst

```
double current_measurement_descriptor::conversionConst
```

Constant number for conversion of ADC value to Ampere.

3.3.2.4 lastMeasurement

```
double current_measurement_descriptor::lastMeasurement
```

Latest calculated current.

3.3.2.5 lastReading

```
uint32_t current_measurement_descriptor::lastReading
```

Latest ADC raw value.

3.3.2.6 Nadc

```
uint32_t current_measurement_descriptor::Nadc
```

ADC saturation point.

3.3.2.7 Ripropi

```
double current_measurement_descriptor::Ripropi
```

Current sense resistor, Ohm.

3.3.2.8 VrefA

```
double current_measurement_descriptor::VrefA
```

Analog reference voltage, V.

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

- adc_driver.h

3.4 imu_descriptor Struct Reference

Key information about the IMUs.

```
#include <accelerometer_driver.h>
```

Public Attributes

- I2C_HandleTypeDef ∗ i2cHandle

  Pointer to the I2C bus peripheral.
- uint16_t readAddr

  Read address of the IMU on the I2C bus.
- uint16_t writeAddr

  Write address of the IMU on the I2C bus.
- uint8_t xAccAddr

  Start address of the IMU's X axis accelerometer register.
- uint8_t yAccAddr

  Start address of the IMU's Y axis accelerometer register.
- uint8_t zAccAddr

  Start address of the IMU's Z axis accelerometer register.
- uint8_t xRotAddr

  Start address of the IMU's X axis rotation rate register.
- uint8_t yRotAddr

  Start address of the IMU's Y axis rotation rate register.
- uint8_t zRotAddr

  Start address of the IMU's Z axis rotation rate register.

3.4.1 Detailed Description

Key information about the IMUs.

3.4.2 Member Data Documentation

3.4.2.1 i2cHandle

```
I2C_HandleTypeDef* imu_descriptor::i2cHandle
```

Pointer to the I2C bus peripheral.

3.4.2.2 readAddr

```
uint16_t imu_descriptor::readAddr
```

Read address of the IMU on the I2C bus.

3.4.2.3 writeAddr

```
uint16_t imu_descriptor::writeAddr
```

Write address of the IMU on the I2C bus.

3.4.2.4 xAccAddr

```
uint8_t imu_descriptor::xAccAddr
```

Start address of the IMU's X axis accelerometer register.

3.4.2.5 xRotAddr

```
uint8_t imu_descriptor::xRotAddr
```

Start address of the IMU's X axis rotation rate register.

3.4.2.6 yAccAddr

```
uint8_t imu_descriptor::yAccAddr
```

Start address of the IMU's Y axis accelerometer register.

3.4.2.7 yRotAddr

```
uint8_t imu_descriptor::yRotAddr
```

Start address of the IMU's Y axis rotation rate register.

3.4.2.8 zAccAddr

```
uint8_t imu_descriptor::zAccAddr
```

Start address of the IMU's Z axis accelerometer register.

3.4.2.9 zRotAddr

```
uint8_t imu_descriptor::zRotAddr
```

Start address of the IMU's Z axis rotation rate register.

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

- accelerometer_driver.h

3.5 joint_controller_descriptor Struct Reference

Joint controller information database.

```
#include <joint_controller.h>
```

Public Attributes

- uint8_t hasAccelerometer

  Whether the joint has an accelerometer for position control.
- float posSetpoint

  The joint's setpoint in radians/mm relative to its zero position.
- float posCurrent

  The joint's current position in radians/mm relative to its zero position.
- float prevPos

  The joint's position in the previous timestep.
- float posError

  The joint's position error in radians/mm, relative to its setpoint.
- float prevError

  The joint's positional error in the previous timestep.
- float power

  Current power setting of the joint.
- float prevPower

  The joint's power setting in the previous timestep.
- uint8_t isMoving

  Whether the joint is in a "moving" state.
- uint8_t motorNum

  Link to the corresponding motor_descriptor.
- uint8_t sigmoidIntGain

  Whether or not the joint should use sigmoid integral gain.
- float Kp

  PID controller Kp.
- float KpTi

  PID controller Kp/Ti.
- float Kd

  PID controller Kd.
- float intError

  Positional integral error.
- uint8_t ∗ jointName

  The joint's human readable name. This field must be last.

3.5.1 Detailed Description

Joint controller information database.

3.5.2 Member Data Documentation

3.5.2.1 hasAccelerometer

```
uint8_t joint_controller_descriptor::hasAccelerometer
```

Whether the joint has an accelerometer for position control.

3.5.2.2 intError

```
float joint_controller_descriptor::intError
```

Positional integral error.

3.5.2.3 isMoving

```
uint8_t joint_controller_descriptor::isMoving
```

Whether the joint is in a "moving" state.

3.5.2.4 jointName

```
uint8_t* joint_controller_descriptor::jointName
```

The joint's human readable name. This field must be last.

3.5.2.5 Kd

```
float joint_controller_descriptor::Kd
```

PID controller Kd.

3.5.2.6 Kp

```
float joint_controller_descriptor::Kp
```

PID controller Kp.

3.5.2.7 KpTi

```
float joint_controller_descriptor::KpTi
```

PID controller Kp/Ti.

3.5.2.8 motorNum

```
uint8_t joint_controller_descriptor::motorNum
```

Link to the corresponding motor_descriptor.

3.5.2.9 posCurrent

```
float joint_controller_descriptor::posCurrent
```

The joint's current position in radians/mm relative to its zero position.

3.5.2.10 posError

```
float joint_controller_descriptor::posError
```

The joint's position error in radians/mm, relative to its setpoint.

3.5.2.11 posSetpoint

```
float joint_controller_descriptor::posSetpoint
```

The joint's setpoint in radians/mm relative to its zero position.

3.5.2.12 power

```
float joint_controller_descriptor::power
```

Current power setting of the joint.

3.5.2.13 prevError

```
float joint_controller_descriptor::prevError
```

The joint's positional error in the previous timestep.

3.5.2.14 prevPos

```
float joint_controller_descriptor::prevPos
```

The joint's position in the previous timestep.

3.5.2.15 prevPower

```
float joint_controller_descriptor::prevPower
```

The joint's power setting in the previous timestep.

3.5.2.16 sigmoidIntGain

```
uint8_t joint_controller_descriptor::sigmoidIntGain
```

Whether or not the joint should use sigmoid integral gain.

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

- joint_controller.h

3.6 motor_descriptor Struct Reference

Contains static and state information relevant to the operation of a motor driver.

```
#include <motor_driver.h>
```

Public Attributes

- uint8_t motorId

  The motor's unique ID, essential for CAN messaging.
- uint8_t voltageLimit

  Safe voltage limit, as stated in the motor's datasheet.
- uint8_t voltagePctCap

  Safe voltage percentage cap, given by input voltage and safe limit.
- int8_t motorPolarity

  The motor's polarity, which pole is connected to +/- on the driver.
- TIM_TypeDef * motorTimer

  The MCU timer peripheral which drives the motor.
- TIM_TypeDef * encoderTimer

  The MCU timer peripheral which registers the motor's encoder.
- int32_t resolution

  Relation between number of encoder clicks per mm or rad of movement.

- float torqueConst

    Motor's torque constant in Nm/A, from datasheet.
- uint16_t encoderInitCount

    Number of encoder clicks counted on startup (nominally 0)
- int32_t encoderTotalInit

    Number of total encoder clicks counted on startup (nominally 0)
- int32_t encoderTotalSetpoint

    Setpoint for motor encoder count, relevant if circumventing joint controller.
- int32_t encoderTotalCount

    Total number of encoder clicks registered since startup.
- int32_t encoderPreviousCount

    The previous total encoder count, used for updating total.
- int32_t mostRecentDelta

    The most recently registered increment/decrement in encoder count, essentially movement rate.
- uint8_t isMoving

    Whether the motor is moving, assumed true if mostRecentDelta>50.
- char ∗ motorName

    Human readable name of the motor.

3.6.1 Detailed Description

Contains static and state information relevant to the operation of a motor driver.

3.6.2 Member Data Documentation

3.6.2.1 encoderInitCount

```
uint16_t motor_descriptor::encoderInitCount
```

Number of encoder clicks counted on startup (nominally 0)

3.6.2.2 encoderPreviousCount

```
int32_t motor_descriptor::encoderPreviousCount
```

The previous total encoder count, used for updating total.

3.6.2.3 encoderTimer

```
TIM_TypeDef* motor_descriptor::encoderTimer
```

The MCU timer peripheral which registers the motor's encoder.

3.6.2.4 encoderTotalCount

```
int32_t motor_descriptor::encoderTotalCount
```

Total number of encoder clicks registered since startup.

3.6.2.5 encoderTotalInit

```
int32_t motor_descriptor::encoderTotalInit
```

Number of total encoder clicks counted on startup (nominally 0)

3.6.2.6 encoderTotalSetpoint

```
int32_t motor_descriptor::encoderTotalSetpoint
```

Setpoint for motor encoder count, relevant if circumventing joint controller.

3.6.2.7 isMoving

```
uint8_t motor_descriptor::isMoving
```

Whether the motor is moving, assumed true if mostRecentDelta>50.

3.6.2.8 mostRecentDelta

```
int32_t motor_descriptor::mostRecentDelta
```

The most recently registered increment/decrement in encoder count, essentially movement rate.

3.6.2.9 motorId

```
uint8_t motor_descriptor::motorId
```

The motor's unique ID, essential for CAN messaging.

3.6.2.10 motorName

```
char* motor_descriptor::motorName
```

Human readable name of the motor.

3.6.2.11 motorPolarity

```
int8_t motor_descriptor::motorPolarity
```

The motor's polarity, which pole is connected to +/- on the driver.

3.6.2.12 motorTimer

```
TIM_TypeDef* motor_descriptor::motorTimer
```

The MCU timer peripheral which drives the motor.

3.6.2.13 resolution

```
int32_t motor_descriptor::resolution
```

Relation between number of encoder clicks per mm or rad of movement.

3.6.2.14 torqueConst

```
float motor_descriptor::torqueConst
```

Motor's torque constant in Nm/A, from datasheet.

3.6.2.15 voltageLimit

```
uint8_t motor_descriptor::voltageLimit
```

Safe voltage limit, as stated in the motor's datasheet.

3.6.2.16 voltagePctCap

```
uint8_t motor_descriptor::voltagePctCap
```

Safe voltage percentage cap, given by input voltage and safe limit.

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

- motor_driver.h

3.7 string_cmd_pair Struct Reference

Pairs a command string token with a function pointer.

```
#include <string_cmd_parser.h>
```

Public Attributes

- char ∗ cmdString

  String token.
- void(∗ cmdFuncPointer)()

  Corresponding handler function.

3.7.1 Detailed Description

Pairs a command string token with a function pointer.

3.7.2 Member Data Documentation

3.7.2.1 cmdFuncPointer

```
void(* string_cmd_pair::cmdFuncPointer) ()
```

Corresponding handler function.

3.7.2.2 cmdString

```
char* string_cmd_pair::cmdString
```

String token.

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

- string_cmd_parser.h

3.8 string_cmd_processor_args Struct Reference

Wrapper struct to enable a variable number of arguments to the string processor.

```
#include <string_cmd_parser.h>
```

Public Attributes

- char ∗ inputString [64]

  String to be processed.
- uint8_t stringLength

  Length of the string to be processed.

3.8.1 Detailed Description

Wrapper struct to enable a variable number of arguments to the string processor.

3.8.2 Member Data Documentation

3.8.2.1 inputString

```
char* string_cmd_processor_args::inputString[64]
```

String to be processed.

3.8.2.2 stringLength

```
uint8_t string_cmd_processor_args::stringLength
```

Length of the string to be processed.

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

- string_cmd_parser.h

Chapter 4

File Documentation

4.1 accelerometer_driver.h File Reference

This file contains all the function prototypes and struct definitions for the accelerometer_driver.c file.

```
#include "uart_driver.h"
#include "unit_config.h"
#include "i2c.h"
#include "stdint.h"
#include "string.h"
```

Classes

- struct imu_descriptor

  Key information about the IMUs.

Functions

- uint8_t accl_interface_read_byte (uint8_t regAddr)

  Module external interface function to read a byte register.
- uint16_t accl_interface_read_register (uint8_t regAddr)

  Module external interface function to read a two-byte register.
- int16_t accl_interface_get_x_acc ()

  Module external interface function to read the IMU's X axis acceleration.
- int16_t accl_interface_get_y_acc ()

  Module external interface function to read the IMU's Y axis acceleration.
- int16_t accl_interface_get_z_acc ()

  Module external interface function to read the IMU's Z axis acceleration.
- int16_t accl_interface_get_x_rot ()

  Module external interface function to read the IMU's X axis rotation.
- int16_t accl_interface_get_y_rot ()

  Module external interface function to read the IMU's Y axis rotation.
- int16_t accl_interface_get_z_rot ()

  Module external interface function to read the IMU's Z axis rotation.

- void accl_interface_set_byte (uint8_t regAddr, uint8_t data)

  Module external interface function to write a byte to an IMU register.
- uint8_t accl_driver_read_byte (imu_descriptor ∗imu, uint8_t regAddr)

  Read a single byte register from the IMU.
- uint16_t accl_driver_read_register (imu_descriptor ∗imu, uint8_t regAddr)

  Read a two byte register from the IMU.
- void accl_driver_set_byte (imu_descriptor ∗imu, uint8_t regAddr, uint8_t data)

  Write single byte register of the IMU.
- int16_t accl_driver_get_x_acc (imu_descriptor ∗imu)

  Read X axis acceleration register.
- int16_t accl_driver_get_y_acc (imu_descriptor ∗imu)

  Read Y axis acceleration register.
- int16_t accl_driver_get_z_acc (imu_descriptor ∗imu)

  Read X axis acceleration register.
- int16_t accl_driver_get_x_rot (imu_descriptor ∗imu)

  Read X axis rotation register.
- int16_t accl_driver_get_y_rot (imu_descriptor ∗imu)

  Read Y axis rotation register.
- int16_t accl_driver_get_z_rot (imu_descriptor ∗imu)

  Read Z axis rotation register.

4.1.1 Detailed Description

This file contains all the function prototypes and struct definitions for the accelerometer_driver.c file.

Attention

IMU driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver specifies a struct of relevant register addresses from the LSM6DSM IMU as well as functions using the I2C peripheral to read these addresses.

4.1.2 Function Documentation

4.1.2.1 accl_driver_get_x_acc()

```
int16_t accl_driver_get_x_acc (
            imu_descriptor * imu )
```

Read X axis acceleration register.

Parameters

imu	Pointer to the relevant IMU struct

Returns

Acceleration raw value

4.1.2.2 accl_driver_get_x_rot()

```
int16_t accl_driver_get_x_rot (
            imu_descriptor * imu )
```

Read X axis rotation register.

Parameters

imu	Pointer to the relevant IMU struct

Returns

Rotation raw value

4.1.2.3 accl_driver_get_y_acc()

```
int16_t accl_driver_get_y_acc (
            imu_descriptor * imu )
```

Read Y axis acceleration register.

Parameters

imu	Pointer to the relevant IMU struct

Returns

Acceleration raw value

4.1.2.4 accl_driver_get_y_rot()

```
int16_t accl_driver_get_y_rot (
            imu_descriptor * imu )
```

Read Y axis rotation register.

Parameters

imu	Pointer to the relevant IMU struct

Returns

Rotation raw value

4.1.2.5 accl_driver_get_z_acc()

```
int16_t accl_driver_get_z_acc (
            imu_descriptor * imu )
```

Read X axis acceleration register.

Parameters

imu	Pointer to the relevant IMU struct

Returns

Acceleration raw value

4.1.2.6 accl_driver_get_z_rot()

```
int16_t accl_driver_get_z_rot (
            imu_descriptor * imu )
```

Read Z axis rotation register.

Parameters

imu	Pointer to the relevant IMU struct

Returns

Rotation raw value

4.1.2.7 accl_driver_read_byte()

```
uint8_t accl_driver_read_byte (
            imu_descriptor * imu,
            uint8_t regAddr )
```

Read a single byte register from the IMU.

Parameters

imu	Pointer to the relevant IMU struct
regAddr	Address to read

Returns

Value of the register

4.1.2.8 accl_driver_read_register()

```
uint16_t accl_driver_read_register (
            imu_descriptor * imu,
            uint8_t regAddr )
```

Read a two byte register from the IMU.

Parameters

imu	Pointer to the relevant IMU struct
regAddr	Address to start read

Returns

Concatenated values of the two registers (left shift + bitwOR)

4.1.2.9 accl_driver_set_byte()

```
void accl_driver_set_byte (
            imu_descriptor * imu,
            uint8_t regAddr,
            uint8_t data )
```

Write single byte register of the IMU.

Parameters

imu	Pointer to the relevant IMU struct
regAddr	Address to write
data	Data to write

4.1.2.10 accl_interface_get_x_acc()

```
int16_t accl_interface_get_x_acc ( )
```

Module external interface function to read the IMU's X axis acceleration.

Returns

Raw acceleration value as determined by the IMU's configured acceleration resolution

4.1.2.11 accl_interface_get_x_rot()

```
int16_t accl_interface_get_x_rot ( )
```

Module external interface function to read the IMU's X axis rotation.

Returns

Raw rotation value as determined by the IMU's configured rotation resolution

4.1.2.12 accl_interface_get_y_acc()

```
int16_t accl_interface_get_y_acc ( )
```

Module external interface function to read the IMU's Y axis acceleration.

Returns

Raw acceleration value as determined by the IMU's configured acceleration resolution

4.1.2.13 accl_interface_get_y_rot()

```
int16_t accl_interface_get_y_rot ( )
```

Module external interface function to read the IMU's Y axis rotation.

Returns

Raw rotation value as determined by the IMU's configured rotation resolution

4.1.2.14 accl_interface_get_z_acc()

```
int16_t accl_interface_get_z_acc ( )
```

Module external interface function to read the IMU's Z axis acceleration.

Returns

Raw acceleration value as determined by the IMU's configured acceleration resolution

4.1.2.15 accl_interface_get_z_rot()

```
int16_t accl_interface_get_z_rot ( )
```

Module external interface function to read the IMU's Z axis rotation.

Returns

Raw rotation value as determined by the IMU's configured rotation resolution

4.1.2.16 accl_interface_read_byte()

```
uint8_t accl_interface_read_byte (
            uint8_t regAddr )
```

Module external interface function to read a byte register.

Parameters

regAddr	Address to read

Returns

Value of the register

4.1.2.17 accl_interface_read_register()

```
uint16_t accl_interface_read_register (
            uint8_t regAddr )
```

Module external interface function to read a two-byte register.

Parameters

regAddr	Start address of the read

Returns

Value of the register

4.1.2.18 accl_interface_set_byte()

```
void accl_interface_set_byte (
            uint8_t regAddr,
            uint8_t data )
```

Module external interface function to write a byte to an IMU register.

Parameters

regAddr	Address to write
data	Data to write

4.2 adc_driver.h File Reference

This file contains all the function prototypes and struct definitions for the adc_driver.c file.

```
#include "stdint.h"
#include "math.h"
#include "adc.h"
#include "unit_config.h"
#include "uart_driver.h"
```

Classes

- struct current_measurement_descriptor

  Database with key information for the motor current sensing ADCs.

Functions

- double adc_interface_get_current (uint8_t sensorSelect)

  Module external interface function to read the most recently calculated current.
- void adc_interface_update_current (uint8_t sensorSelect)

  Module external interface function to trigger a current calculation.
- double adc_driver_calculate_current (current_measurement_descriptor ∗sensor)

  Calculates current in Ampere based on an ADC raw value.
- void adc_driver_update_reading (current_measurement_descriptor ∗sensor)

  Trigger a reading of the relevant ADC, insert into sensor struct.
- void adc_driver_update_measurement (current_measurement_descriptor ∗sensor)

  Calculate lastest current measurement, insert into struct.

4.2.1 Detailed Description

This file contains all the function prototypes and struct definitions for the adc_driver.c file.

Attention

ADC driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. This driver is tailored for use with the current sense pin of the DRV8251A motor driver

4.2.2 Function Documentation

4.2.2.1 adc_driver_calculate_current()

```
double adc_driver_calculate_current (
             current_measurement_descriptor * sensor )
```

Calculates current in Ampere based on an ADC raw value.

Parameters

sensor	Pointer to the relevant sensor struct

Returns

Ampere

4.2.2.2 adc_driver_update_measurement()

```
void adc_driver_update_measurement (
             current_measurement_descriptor * sensor )
```

Calculate lastest current measurement, insert into struct.

Parameters

sensor	Pointer to the relevant sensor struct

4.2.2.3 adc_driver_update_reading()

```
void adc_driver_update_reading (
            current_measurement_descriptor * sensor )
```

Trigger a reading of the relevant ADC, insert into sensor struct.

Parameters

sensor	Pointer to the relevant sensor struct

4.2.2.4 adc_interface_get_current()

```
double adc_interface_get_current (
            uint8_t sensorSelect )
```

Module external interface function to read the most recently calculated current.

Parameters

sensorSelect	Select from one of two motor current sensor ADCs

Returns

Latest calculated current measurement

4.2.2.5 adc_interface_update_current()

```
void adc_interface_update_current (
            uint8_t sensorSelect )
```

Module external interface function to trigger a current calculation.

Parameters

sensorSelect	Select from of two motor current sensor ADCs

4.3 can_driver.h File Reference

This file contains all the function prototypes and struct definitions for the can_driver.c file.

```
#include "stdint.h"
#include "can.h"
```

```
#include "unit_config.h"
#include "motor_driver.h"
#include "accelerometer_driver.h"
#include "joint_controller.h"
#include "state_machine.h"
```

Classes

- struct can_mailbox

  A virtual CAN mailbox for outgoing and incoming messages.

Macros

- #define CAN_MOTOR_CMD_OFFSET 5
- #define CAN_ACC_CMD_OFFSET 8
- #define CAN_GBL_CMD_OFFSET 10

Enumerations

- enum can_message_type {
  ACC_X_TX , ACC_Y_TX , ACC_Z_TX , ACC_REG_RX ,
  ACC_REG_REQ , JOINT_POS_SP , MOTOR_VLT_SP , JOINT_POS_REQ ,
  JOINT_POS_TX , ACC_X_REQ , ACC_Y_REQ , ACC_Z_REQ ,
  GBL_ST_SET , WRIST_ELBOW_SP , PINCH_TWIST_SP , num_types }

  CAN message types.

Functions

- void can_interface_queue_tx (uint8_t mailbox, uint8_t ∗outData, uint32_t id)

  Module external function for queueing a CAN message for transmit.
- void can_interface_send_msg (uint8_t ∗data, uint32_t id, int dlc, uint8_t hwMailbox)

  Public function to send CAN message immediately.
- void can_rx_executive ()

  Looks for new incoming CAN messages and handles them. MAIN LOOP ONLY.
- void can_tx_executive ()

  Looks for new CAN messages to send, sends. MAIN LOOP ONLY.
- void can_mailbox_set_data (can_mailbox ∗mailbox, uint8_t ∗inData)

  Sets the data field of the given mailbox.
- void can_mailbox_get_data (can_mailbox ∗mailbox, uint8_t ∗target)

  Inserts the data field of the given mailbox to target. CAUTION: memcpy 8 bytes.
- void can_mailbox_set_flag (can_mailbox ∗mailbox)

  Sets the newmsg flag of the given mailbox.
- uint8_t can_mailbox_get_flag (can_mailbox ∗mailbox)

  Gets the newmsg flag of the given mailbox.
- void can_mailbox_clear_flag (can_mailbox ∗mailbox)

  Clears the newmsg flag of the given mailbox.
- void can_mailbox_set_id (can_mailbox ∗mailbox, uint32_t id)

  Sets the CAN message ID field of the given mailbox.

- uint32_t can_mailbox_get_id (can_mailbox ∗mailbox)

  Gets the CAN message ID of the given mailbox.
- void can_driver_queue_tx (can_mailbox ∗mailbox, uint8_t ∗outData, uint32_t id)

  Queues a can message for transmit, driver internal.
- void can_driver_send_msg (uint8_t ∗data, uint32_t stdId, int dlc, uint8_t hwMailbox)

  Sends a CAN message.
- void can_cmd_handle_yAcc (uint32_t id, uint8_t ∗inData)
- void can_cmd_handle_regVal (uint32_t id, uint8_t ∗inData)

  Handles an incoming accelerometer register message.
- void can_cmd_handle_regReq (uint32_t id, uint8_t ∗inData)

  Handles an incoming accelerometer read request.
- void can_cmd_handle_motorSp (uint32_t id, uint8_t ∗inData)

  Handles an incoming motor setpoint.
- void can_cmd_handle_axisReq (uint32_t id, uint8_t ∗inData)

  Handles an incoming request for accelerometer axis data.
- void can_cmd_handle_axisData (uint32_t id, uint8_t ∗inData)

  Handles incoming accelerometer axis data.
- void can_cmd_handle_inState (uint32_t id, uint8_t ∗inData)

  Handles incoming state information.
- void can_cmd_handle_dualJointSp (uint32_t id, uint8_t ∗inData)

  Handles incoming setpoints for both joints.
- void can_cmd_handle_wristElbowSp (uint32_t id, uint8_t ∗inData)

  Handles incoming setpoints for elbow and wrist joints.
- void can_cmd_handle_pinchTwistSp (uint32_t id, uint8_t ∗inData)

  Handles incoming setpoints for twist and pinch joints.
- void can_driver_cmd_rx0 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_X_TX
- void can_driver_cmd_rx1 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_Y_TX
- void can_driver_cmd_rx2 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_Z_TX
- void can_driver_cmd_rx3 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_REG_RX
- void can_driver_cmd_rx4 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_REG_REQ
- void can_driver_cmd_rx5 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type JOINT_POS_SP
- void can_driver_cmd_rx6 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type MOTOR_VLT_SP
- void can_driver_cmd_rx7 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_X_TX
- void can_driver_cmd_rx8 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type JOINT_POS_TX
- void can_driver_cmd_rx9 (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_X_REQ
- void can_driver_cmd_rxA (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_Y_REQ
- void can_driver_cmd_rxB (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type ACC_Z_REQ
- void can_driver_cmd_rxC (uint32_t id, uint8_t ∗inData)

  "Generic" function to handle CAN message type GBL_ST_SET

- void can_driver_cmd_rxD (uint32_t id, uint8_t ∗inData)

    "Generic" function to handle CAN message type WRIST_ELBOW_SP
- void can_driver_cmd_rxE (uint32_t id, uint8_t ∗inData)

    "Generic" function to handle CAN message type PINCH_TWIST_SP

4.3.1 Detailed Description

This file contains all the function prototypes and struct definitions for the can_driver.c file.

Attention

CAN driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver specifies CAN message types relevant to the project, as well as relevant functions for the handling of transmission and reception of messages.

4.3.2 Macro Definition Documentation

4.3.2.1 CAN_ACC_CMD_OFFSET

```
#define CAN_ACC_CMD_OFFSET 8
```

4.3.2.2 CAN_GBL_CMD_OFFSET

```
#define CAN_GBL_CMD_OFFSET 10
```

4.3.2.3 CAN_MOTOR_CMD_OFFSET

```
#define CAN_MOTOR_CMD_OFFSET 5
```

4.3.3 Enumeration Type Documentation

4.3.3.1 can_message_type

```
enum can_message_type
```

CAN message types.

Enumerator

ACC_X_TX	A message containting acc/rot X axis.
ACC_Y_TX	A message containting acc/rot Y axis.
ACC_Z_TX	A message containting acc/rot Z axis.
ACC_REG_RX	A message containting an arbitrary accelerometer register value.
ACC_REG_REQ	A message requesting an arbitrary accelerometer register value.
JOINT_POS_SP	A message containing a joint position setpoint.
MOTOR_VLT_SP	A message containting a motor voltage percent setpoint.
JOINT_POS_REQ	A message requesting the position of a joint.
JOINT_POS_TX	A message containing the position of a joint.
ACC_X_REQ	A message requesting acc/rot x axis.
ACC_Y_REQ	A message requesting acc/rot y axis.
ACC_Z_REQ	A message requesting acc/rot z axis.
GBL_ST_SET	Message contains a global state to set.
WRIST_ELBOW_SP	Message contains setpoints for the elbow and wrist joint controllers.
PINCH_TWIST_SP	Message contains setpoints for the twist and pinch joints controllers.
num_types	Dummy type for counting the number of message types, must always be last.

4.3.4 Function Documentation

4.3.4.1 can_cmd_handle_axisData()

```
void can_cmd_handle_axisData (
            uint32_t id,
            uint8_t * inData )
```

Handles incoming accelerometer axis data.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.2 can_cmd_handle_axisReq()

```
void can_cmd_handle_axisReq (
            uint32_t id,
            uint8_t * inData )
```

Handles an incoming request for accelerometer axis data.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.3 can_cmd_handle_dualJointSp()

```
void can_cmd_handle_dualJointSp (
            uint32_t id,
            uint8_t * inData )
```

Handles incoming setpoints for both joints.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.4 can_cmd_handle_inState()

```
void can_cmd_handle_inState (
            uint32_t id,
            uint8_t * inData )
```

Handles incoming state information.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.5 can_cmd_handle_motorSp()

```
void can_cmd_handle_motorSp (
            uint32_t id,
            uint8_t * inData )
```

Handles an incoming motor setpoint.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.6 can_cmd_handle_pinchTwistSp()

```
void can_cmd_handle_pinchTwistSp (
            uint32_t id,
            uint8_t * inData )
```

Handles incoming setpoints for twist and pinch joints.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.7 can_cmd_handle_regReq()

```
void can_cmd_handle_regReq (
            uint32_t id,
            uint8_t * inData )
```

Handles an incoming accelerometer read request.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.8 can_cmd_handle_regVal()

```
void can_cmd_handle_regVal (
            uint32_t id,
            uint8_t * inData )
```

Handles an incoming accelerometer register message.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.9 can_cmd_handle_wristElbowSp()

```
void can_cmd_handle_wristElbowSp (
            uint32_t id,
            uint8_t * inData )
```

Handles incoming setpoints for elbow and wrist joints.

Parameters

id	Incoming CAN ID
inData	Incoming CAN data

4.3.4.10 can_cmd_handle_yAcc()

```
void can_cmd_handle_yAcc (
            uint32_t id,
            uint8_t * inData )
```

4.3.4.11 can_driver_cmd_rx0()

```
void can_driver_cmd_rx0 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_X_TX

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.12 can_driver_cmd_rx1()

```
void can_driver_cmd_rx1 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_Y_TX

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.13 can_driver_cmd_rx2()

```
void can_driver_cmd_rx2 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_Z_TX

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.14 can_driver_cmd_rx3()

```
void can_driver_cmd_rx3 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_REG_RX

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.15 can_driver_cmd_rx4()

```
void can_driver_cmd_rx4 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_REG_REQ

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.16 can_driver_cmd_rx5()

```
void can_driver_cmd_rx5 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type JOINT_POS_SP

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.17 can_driver_cmd_rx6()

```
void can_driver_cmd_rx6 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type MOTOR_VLT_SP

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.18 can_driver_cmd_rx7()

```
void can_driver_cmd_rx7 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_X_TX

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.19 can_driver_cmd_rx8()

```
void can_driver_cmd_rx8 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type JOINT_POS_TX

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.20 can_driver_cmd_rx9()

```
void can_driver_cmd_rx9 (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_X_REQ

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.21 can_driver_cmd_rxA()

```
void can_driver_cmd_rxA (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_Y_REQ

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.22 can_driver_cmd_rxB()

```
void can_driver_cmd_rxB (
            uint32_t id,
            uint8_t * inData )
```

"Generic" function to handle CAN message type ACC_Z_REQ

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.23 can_driver_cmd_rxC()

```
void can_driver_cmd_rxC (
             uint32_t id,
             uint8_t * inData )
```

"Generic" function to handle CAN message type GBL_ST_SET

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.24 can_driver_cmd_rxD()

```
void can_driver_cmd_rxD (
             uint32_t id,
             uint8_t * inData )
```

"Generic" function to handle CAN message type WRIST_ELBOW_SP

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.25 can_driver_cmd_rxE()

```
void can_driver_cmd_rxE (
             uint32_t id,
             uint8_t * inData )
```

"Generic" function to handle CAN message type PINCH_TWIST_SP

Parameters

id	CAN message ID
inData	CAN message data field

4.3.4.26 can_driver_queue_tx()

```
void can_driver_queue_tx (
             can_mailbox * mailbox,
             uint8_t * outData,
             uint32_t id )
```

Queues a can message for transmit, driver internal.

Parameters

mailbox	Mailbox struct in which data will be placed
outData	Data to send (8 bytes)
id	CAN transmit ID

4.3.4.27 can_driver_send_msg()

```
void can_driver_send_msg (
             uint8_t * data,
             uint32_t stdId,
             int dlc,
             uint8_t hwMailbox )
```

Sends a CAN message.

Parameters

data	Data to send, max 8 bytes
stdId	Message ID required for rx handling
dlc	Number of bytes to send
hwMailbox	Hardware(!) mailbox to queue to

4.3.4.28 can_interface_queue_tx()

```
void can_interface_queue_tx (
             uint8_t mailbox,
             uint8_t * outData,
             uint32_t id )
```

Module external function for queueing a CAN message for transmit.

Parameters

mailbox	Mailbox number, must correspond to the correct message type
outData	Data to send
id	CAN transmit ID

4.3.4.29 can_interface_send_msg()

```
void can_interface_send_msg (
             uint8_t * data,
             uint32_t id,
             int dlc,
             uint8_t hwMailbox )
```

Public function to send CAN message immediately.

Parameters

data	Data to send, max 8 byte
id	CAN message ID
dlc	CAN message data length
hwMailbox	Hardware transmit mailbox to send from

4.3.4.30 can_mailbox_clear_flag()

```
void can_mailbox_clear_flag (
             can_mailbox * mailbox )
```

Clears the newmsg flag of the given mailbox.

Parameters

mailbox	Mailbox to clear flag from

4.3.4.31 can_mailbox_get_data()

```
void can_mailbox_get_data (
             can_mailbox * mailbox,
             uint8_t * target )
```

Inserts the data field of the given mailbox to target. CAUTION: memcpy 8 bytes.

Parameters

mailbox	Mailbox to retrieve data from
target	Pointer to data target

4.3.4.32 can_mailbox_get_flag()

```
uint8_t can_mailbox_get_flag (
            can_mailbox * mailbox )
```

Gets the newmsg flag of the given mailbox.

Parameters

mailbox	Mailbox to get

Returns

Mailbox' newmsg flag

4.3.4.33 can_mailbox_get_id()

```
uint32_t can_mailbox_get_id (
            can_mailbox * mailbox )
```

Gets the CAN message ID of the given mailbox.

Parameters

mailbox	Mailbox to get ID from

Returns

CAN message ID

4.3.4.34 can_mailbox_set_data()

```
void can_mailbox_set_data (
            can_mailbox * mailbox,
            uint8_t * inData )
```

Sets the data field of the given mailbox.

Parameters

mailbox	Mailbox to insert data into
inData	Data to insert

4.3.4.35 can_mailbox_set_flag()

```
void can_mailbox_set_flag (
            can_mailbox * mailbox )
```

Sets the newmsg flag of the given mailbox.

Parameters

mailbox	Mailbox to set flag

4.3.4.36 can_mailbox_set_id()

```
void can_mailbox_set_id (
            can_mailbox * mailbox,
            uint32_t id )
```

Sets the CAN message ID field of the given mailbox.

Parameters

mailbox	Mailbox to insert ID
id	CAN message ID to insert

4.3.4.37 can_rx_executive()

```
void can_rx_executive ( )
```

Looks for new incoming CAN messages and handles them. MAIN LOOP ONLY.

4.3.4.38 can_tx_executive()

```
void can_tx_executive ( )
```

Looks for new CAN messages to send, sends. MAIN LOOP ONLY.

4.4 gpio_driver.h File Reference

This file contains all the function prototypes for the gpio_driver.c file.

```
#include "stdint.h"
#include "gpio.h"
#include "can_driver.h"
#include "joint_controller.h"
#include "state_machine.h"
#include "unit_config.h"
```

Functions

- void gpio_end_switch_handler ()

  Handler function for the rail end switch.
- void gpio_twist_switch_handler ()

  Handler function for the twist joint end switch.

4.4.1 Detailed Description

This file contains all the function prototypes for the gpio_driver.c file.

Attention

GPIO driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver handles GPIO interrupts from the end switch and twist joint optical sensors. The functions are called from the HAL_GPIO_EXTI↵ _Callback function declared in stm32fxx_hal_gpio.h

4.4.2 Function Documentation

4.4.2.1 gpio_end_switch_handler()

```
void gpio_end_switch_handler ( )
```

Handler function for the rail end switch.

4.4.2.2 gpio_twist_switch_handler()

```
void gpio_twist_switch_handler ( )
```

Handler function for the twist joint end switch.

4.5 joint_controller.h File Reference

This file contains all the function prototypes and struct definitions for the joint_controller.c file.

```
#include "stdint.h"
#include "math.h"
#include "tim.h"
#include "unit_config.h"
#include "uart_driver.h"
#include "motor_driver.h"
#include "can_driver.h"
#include "accelerometer_driver.h"
```

Classes

- struct joint_controller_descriptor

  Joint controller information database.
- struct accelerometer_inData

  Holds INCOMING accelerometer data, NOT part of the accelerometer driver.

Functions

- void controller_interface_update_controller ()

  Main function to run the controller on both joints.
- float controller_interface_get_setpoint (uint8_t controllerSelect)

  Public function to get the current positional setpoint of a joint.
- void controller_interface_set_setpoint (uint8_t controllerSelect, float setPoint)

  Public function to set the positional setpoint of a joint.
- float controller_interface_get_position (uint8_t controllerSelect)

  Public function to get the position of a joint.
- void controller_interface_update_position (uint8_t controllerSelect)

  Public function to trigger an update of the joint's position.
- void controller_interface_set_position (uint8_t controllerSelect, float position)

  Public function to get the current position of a joint.
- float controller_interface_get_error (uint8_t controllerSelect)

  Public function to get the current positional error of a joint.
- void controller_interface_update_error (uint8_t controllerSelect)

  Public function to trigger a calculation of the joint's positional error.
- void controller_interface_set_error (uint8_t controllerSelect, float error)

  Public function to set the positional error of a joint.
- uint8_t controller_interface_get_moving (uint8_t controllerSelect)

  Public function to get the moving state value of the joint.
- void controller_interface_set_moving (uint8_t controllerSelect)

  Public function to set the moving state flag of the joint.
- void controller_interface_clear_moving (uint8_t controllerSelect)

  Public function to clear the moving state flag of the joint.
- void controller_interface_set_power (uint8_t controllerSelect, float power)

  Public function to set the power of the joint.
- void controller_interface_update_power (uint8_t controllerSelect)

Public function to trigger an update of the joint's power by PID.
- float controller_interface_clicks_to_pos (uint8_t controllerSelect)

  Public function to convert the joint's associated encoder count to joint position.
- void controller_interface_request_acc_axis (uint8_t controllerSelect, uint8_t accSelect, char axis)

  Public function to request an update from the joint's accelerometer via CAN bus; acc and rot.
- int16_t controller_interface_acc_getX (uint8_t accSelect)

  Public function to get the accelerometer X axis acceleration.
- int16_t controller_interface_acc_getY (uint8_t accSelect)

  Public function to get the accelerometer Y axis acceleration.
- int16_t controller_interface_acc_getZ (uint8_t accSelect)

  Public function to get the accelerometer Z axis acceleration.
- void controller_interface_acc_setX (uint8_t accSelect, int16_t accVal)

  Public function to set the accelerometer X axis acceleration.
- void controller_interface_acc_setY (uint8_t accSelect, int16_t accVal)

  Public function to set the accelerometer Y axis acceleration.
- void controller_interface_acc_setZ (uint8_t accSelect, int16_t accVal)

  Public function to set the accelerometer Z axis acceleration.
- int16_t controller_interface_rot_getX (uint8_t accSelect)

  Public function to get the accelerometer X axis rotation rate.
- int16_t controller_interface_rot_getY (uint8_t accSelect)

  Public function to get the accelerometer Y axis rotation rate.
- int16_t controller_interface_rot_getZ (uint8_t accSelect)

  Public function to get the accelerometer Z axis rotation rate.
- void controller_interface_rot_setX (uint8_t accSelect, int16_t rotVal)

  Public function to set the accelerometer X axis rotation rate.
- void controller_interface_rot_setY (uint8_t accSelect, int16_t rotVal)

  Public function to set the accelerometer Y axis rotation rate.
- void controller_interface_rot_setZ (uint8_t accSelect, int16_t rotVal)

  Public function to set the accelerometer Z axis rotation rate.
- uint8_t controller_interface_acc_get_newX (uint8_t accSelect)

  Public function to get the new X acceleration data flag.
- uint8_t controller_interface_acc_get_newY (uint8_t accSelect)

  Public function to get the new Y acceleration data flag.
- uint8_t controller_interface_acc_get_newZ (uint8_t accSelect)

  Public function to get the new Z acceleration data flag.
- void controller_interface_acc_set_newX (uint8_t accSelect)

  Public function to set the new X acceleration data flag.
- void controller_interface_acc_set_newY (uint8_t accSelect)

  Public function to set the new Y acceleration data flag.
- void controller_interface_acc_set_newZ (uint8_t accSelect)

  Public function to set the new Z acceleration data flag.
- void controller_interface_acc_clear_newX (uint8_t accSelect)

  Public function to clear the new X acceleration data flag.
- void controller_interface_acc_clear_newY (uint8_t accSelect)

  Public function to clear the new Y acceleration data flag.
- void controller_interface_acc_clear_newZ (uint8_t accSelect)

  Public function to clear the new Z acceleration data flag.
- uint8_t controller_interface_rot_get_newX (uint8_t accSelect)

  Public function to get the new X rotation data flag.
- uint8_t controller_interface_rot_get_newY (uint8_t accSelect)

  Public function to get the new Y rotation data flag.

- uint8_t controller_interface_rot_get_newZ (uint8_t accSelect)

  Public function to get the new Z rotation data flag.
- void controller_interface_rot_set_newX (uint8_t accSelect)

  Public function to set the new X rotation data flag.
- void controller_interface_rot_set_newY (uint8_t accSelect)

  Public function to set the new Y rotation data flag.
- void controller_interface_rot_set_newZ (uint8_t accSelect)

  Public function to set the new Z rotation data flag.
- void controller_interface_rot_clear_newX (uint8_t accSelect)

  Public function to clear the new X rotation data flag.
- void controller_interface_rot_clear_newY (uint8_t accSelect)

  Public function to clear the new Y rotation data flag.
- void controller_interface_rot_clear_newZ (uint8_t accSelect)

  Public function to clear the new Z rotation data flag.
- uint8_t controller_interface_get_acc_poll ()

  Public function to poll the timer driven accelerometer poll flag.
- uint8_t controller_interface_get_mtr_poll ()

  Public function to read the timer driven motor poll flag.
- void controller_interface_set_acc_poll ()

  Puclic function to set the accelerometer poll flag.
- void controller_interface_set_mtr_poll ()

  Puclic function to set the motor poll flag.
- void controller_interface_clear_acc_poll ()

  Public function to clear the accelerometer poll flag.
- void controller_interface_clear_mtr_poll ()

  Public function to clear the motor poll flag.
- uint8_t controller_interface_get_upd_ctrl ()
- void controller_interface_set_upd_ctrl ()
- void controller_interface_clear_upd_ctrl ()
- uint8_t controller_interface_get_upd_telemetry ()
- void controller_interface_set_upd_telemetry ()
- void controller_interface_clear_upd_telemetry ()
- float joint_controller_get_setpoint (joint_controller_descriptor ∗joint)
- void joint_controller_set_setpoint (joint_controller_descriptor ∗joint, float setpoint)
- float joint_controller_get_position (joint_controller_descriptor ∗joint)
- void joint_controller_update_position (joint_controller_descriptor ∗joint)
- void joint_controller_set_position (joint_controller_descriptor ∗joint, float position)
- float joint_controller_get_error (joint_controller_descriptor ∗joint)
- void joint_controller_update_error (joint_controller_descriptor ∗joint)
- void joint_controller_set_error (joint_controller_descriptor ∗joint, float error)
- uint8_t joint_controller_get_moving (joint_controller_descriptor ∗joint)
- void joint_controller_set_moving (joint_controller_descriptor ∗joint)
- void joint_controller_clear_moving (joint_controller_descriptor ∗joint)
- void joint_controller_set_power (joint_controller_descriptor ∗joint, float power)
- void joint_controller_update_power (joint_controller_descriptor ∗joint)
- float joint_controller_clicks_to_pos (joint_controller_descriptor ∗joint)
- void joint_controller_adjust_enc_sp (joint_controller_descriptor ∗joint)
- void joint_controller_request_acc_axis (joint_controller_descriptor ∗joint, uint8_t accSelect, char axis)
- float joint_controller_acceleration_to_angle (joint_controller_descriptor ∗joint)
- int16_t controller_acc_getX (accelerometer_inData ∗accSelect)
- int16_t controller_acc_getY (accelerometer_inData ∗accSelect)
- int16_t controller_acc_getZ (accelerometer_inData ∗accSelect)

- void controller_acc_setX (accelerometer_inData ∗accSelect, int16_t accVal)
- void controller_acc_setY (accelerometer_inData ∗accSelect, int16_t accVal)
- void controller_acc_setZ (accelerometer_inData ∗accSelect, int16_t accVal)
- int16_t controller_rot_getX (accelerometer_inData ∗accSelect)
- int16_t controller_rot_getY (accelerometer_inData ∗accSelect)
- int16_t controller_rot_getZ (accelerometer_inData ∗accSelect)
- void controller_rot_setX (accelerometer_inData ∗accSelect, int16_t rotVal)
- void controller_rot_setY (accelerometer_inData ∗accSelect, int16_t rotVal)
- void controller_rot_setZ (accelerometer_inData ∗accSelect, int16_t rotVal)
- uint8_t controller_acc_get_newX (accelerometer_inData ∗accSelect)
- uint8_t controller_acc_get_newY (accelerometer_inData ∗accSelect)
- uint8_t controller_acc_get_newZ (accelerometer_inData ∗accSelect)
- void controller_acc_set_newX (accelerometer_inData ∗accSelect)
- void controller_acc_set_newY (accelerometer_inData ∗accSelect)
- void controller_acc_set_newZ (accelerometer_inData ∗accSelect)
- void controller_acc_clear_newX (accelerometer_inData ∗accSelect)
- void controller_acc_clear_newY (accelerometer_inData ∗accSelect)
- void controller_acc_clear_newZ (accelerometer_inData ∗accSelect)
- uint8_t controller_rot_get_newX (accelerometer_inData ∗accSelect)
- uint8_t controller_rot_get_newY (accelerometer_inData ∗accSelect)
- uint8_t controller_rot_get_newZ (accelerometer_inData ∗accSelect)
- void controller_rot_set_newX (accelerometer_inData ∗accSelect)
- void controller_rot_set_newY (accelerometer_inData ∗accSelect)
- void controller_rot_set_newZ (accelerometer_inData ∗accSelect)
- void controller_rot_clear_newX (accelerometer_inData ∗accSelect)
- void controller_rot_clear_newY (accelerometer_inData ∗accSelect)
- void controller_rot_clear_newZ (accelerometer_inData ∗accSelect)

4.5.1 Detailed Description

This file contains all the function prototypes and struct definitions for the joint_controller.c file.

Attention

Joint controller for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The controller makes use of the motor driver to implement PID positional control of the two joints for which the relevant MCU is responsible. The accelerometer struct holds data from the joint's accelerometer, where applicable, and is always received via the CAN bus. This makes accelerometer data inherent to joint control, not the accelerometer driver itself.

Private functions are not described individually, but correspond to their public counterparts. They take a joint controller descriptor struct as argument, and are called by the public functions.

4.5.2 Function Documentation

4.5.2.1 controller_acc_clear_newX()

```
void controller_acc_clear_newX (
            accelerometer_inData * accSelect )
```

4.5.2.2 controller_acc_clear_newY()

```
void controller_acc_clear_newY (
            accelerometer_inData * accSelect )
```

4.5.2.3 controller_acc_clear_newZ()

```
void controller_acc_clear_newZ (
            accelerometer_inData * accSelect )
```

4.5.2.4 controller_acc_get_newX()

```
uint8_t controller_acc_get_newX (
            accelerometer_inData * accSelect )
```

4.5.2.5 controller_acc_get_newY()

```
uint8_t controller_acc_get_newY (
            accelerometer_inData * accSelect )
```

4.5.2.6 controller_acc_get_newZ()

```
uint8_t controller_acc_get_newZ (
            accelerometer_inData * accSelect )
```

4.5.2.7 controller_acc_getX()

```
int16_t controller_acc_getX (
            accelerometer_inData * accSelect )
```

4.5.2.8 controller_acc_getY()

```
int16_t controller_acc_getY (
             accelerometer_inData * accSelect )
```

4.5.2.9 controller_acc_getZ()

```
int16_t controller_acc_getZ (
             accelerometer_inData * accSelect )
```

4.5.2.10 controller_acc_set_newX()

```
void controller_acc_set_newX (
             accelerometer_inData * accSelect )
```

4.5.2.11 controller_acc_set_newY()

```
void controller_acc_set_newY (
             accelerometer_inData * accSelect )
```

4.5.2.12 controller_acc_set_newZ()

```
void controller_acc_set_newZ (
             accelerometer_inData * accSelect )
```

4.5.2.13 controller_acc_setX()

```
void controller_acc_setX (
             accelerometer_inData * accSelect,
             int16_t accVal )
```

4.5.2.14 controller_acc_setY()

```
void controller_acc_setY (
             accelerometer_inData * accSelect,
             int16_t accVal )
```

4.5.2.15 controller_acc_setZ()

```
void controller_acc_setZ (
            accelerometer_inData * accSelect,
            int16_t accVal )
```

4.5.2.16 controller_interface_acc_clear_newX()

```
void controller_interface_acc_clear_newX (
            uint8_t accSelect )
```

Public function to clear the new X acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.17 controller_interface_acc_clear_newY()

```
void controller_interface_acc_clear_newY (
            uint8_t accSelect )
```

Public function to clear the new Y acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.18 controller_interface_acc_clear_newZ()

```
void controller_interface_acc_clear_newZ (
            uint8_t accSelect )
```

Public function to clear the new Z acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.19 controller_interface_acc_get_newX()

```
uint8_t controller_interface_acc_get_newX (
            uint8_t accSelect )
```

Public function to get the new X acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

X axis acceleration new data flag

4.5.2.20 controller_interface_acc_get_newY()

```
uint8_t controller_interface_acc_get_newY (
            uint8_t accSelect )
```

Public function to get the new Y acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Y axis acceleration new data flag

4.5.2.21 controller_interface_acc_get_newZ()

```
uint8_t controller_interface_acc_get_newZ (
            uint8_t accSelect )
```

Public function to get the new Z acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Z axis acceleration new data flag

4.5.2.22 controller_interface_acc_getX()

```
int16_t controller_interface_acc_getX (
            uint8_t accSelect )
```

Public function to get the accelerometer X axis acceleration.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

X axis acceleration raw value

4.5.2.23 controller_interface_acc_getY()

```
int16_t controller_interface_acc_getY (
            uint8_t accSelect )
```

Public function to get the accelerometer Y axis acceleration.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Y axis acceleration raw value

4.5.2.24 controller_interface_acc_getZ()

```
int16_t controller_interface_acc_getZ (
            uint8_t accSelect )
```

Public function to get the accelerometer Z axis acceleration.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Z axis acceleration raw value

4.5.2.25 controller_interface_acc_set_newX()

```
void controller_interface_acc_set_newX (
            uint8_t accSelect )
```

Public function to set the new X acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.26 controller_interface_acc_set_newY()

```
void controller_interface_acc_set_newY (
            uint8_t accSelect )
```

Public function to set the new Y acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.27 controller_interface_acc_set_newZ()

```
void controller_interface_acc_set_newZ (
            uint8_t accSelect )
```

Public function to set the new Z acceleration data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.28 controller_interface_acc_setX()

```
void controller_interface_acc_setX (
            uint8_t accSelect,
            int16_t accVal )
```

Public function to set the accelerometer X axis acceleration.

Parameters

accSelect	The relevant accelerometer inData struct
accVal	X axis acceleration raw value

4.5.2.29 controller_interface_acc_setY()

```
void controller_interface_acc_setY (
            uint8_t accSelect,
            int16_t accVal )
```

Public function to set the accelerometer Y axis acceleration.

Parameters

accSelect	The relevant accelerometer inData struct
accVal	Y axis acceleration raw value

4.5.2.30 controller_interface_acc_setZ()

```
void controller_interface_acc_setZ (
            uint8_t accSelect,
            int16_t accVal )
```

Public function to set the accelerometer Z axis acceleration.

Parameters

accSelect	The relevant accelerometer inData struct
accVal	Z axis acceleration raw value

4.5.2.31 controller_interface_clear_acc_poll()

```
void controller_interface_clear_acc_poll ( )
```

Public function to clear the accelerometer poll flag.

4.5.2.32 controller_interface_clear_moving()

```
void controller_interface_clear_moving (
            uint8_t controllerSelect )
```

Public function to clear the moving state flag of the joint.

Parameters

controllerSelect	One of two joints available to the MCU

4.5.2.33 controller_interface_clear_mtr_poll()

```
void controller_interface_clear_mtr_poll ( )
```

Public function to clear the motor poll flag.

4.5.2.34 controller_interface_clear_upd_ctrl()

```
void controller_interface_clear_upd_ctrl ( )
```

4.5.2.35 controller_interface_clear_upd_telemetry()

```
void controller_interface_clear_upd_telemetry ( )
```

4.5.2.36 controller_interface_clicks_to_pos()

```
float controller_interface_clicks_to_pos (
            uint8_t controllerSelect )
```

Public function to convert the joint's associated encoder count to joint position.

Parameters

controllerSelect	One of two joints available to the MCU

Returns

Joint position in rads

4.5.2.37 controller_interface_get_acc_poll()

```
uint8_t controller_interface_get_acc_poll ( )
```

Public function to poll the timer driven accelerometer poll flag.

Returns

Status of the accelerometer poll flag

4.5.2.38 controller_interface_get_error()

```
float controller_interface_get_error (
            uint8_t controllerSelect )
```

Public function to get the current positional error of a joint.

Parameters

controllerSelect	One of two joints available to the MCU

Returns

Joint positional error in rads relative to its setpoint

4.5.2.39 controller_interface_get_moving()

```
uint8_t controller_interface_get_moving (
            uint8_t controllerSelect )
```

Public function to get the moving state value of the joint.

Parameters

controllerSelect	One of two joints available to the MCU

Returns

isMoving flag

4.5.2.40 controller_interface_get_mtr_poll()

```
uint8_t controller_interface_get_mtr_poll ( )
```

Public function to read the timer driven motor poll flag.

Returns

Status of the motor poll flag

4.5.2.41 controller_interface_get_position()

```
float controller_interface_get_position (
            uint8_t controllerSelect )
```

Public function to get the position of a joint.

Parameters

controllerSelect	One of two joints available to the MCU

Returns

Joint position in radians relative to its zero position

4.5.2.42 controller_interface_get_setpoint()

```
float controller_interface_get_setpoint (
            uint8_t controllerSelect )
```

Public function to get the current positional setpoint of a joint.

Parameters

controllerSelect	One of two joints available to the MCU

Returns

Joint positional setpoints in radians relative to its zero position

4.5.2.43 controller_interface_get_upd_ctrl()

```
uint8_t controller_interface_get_upd_ctrl ( )
```

4.5.2.44 controller_interface_get_upd_telemetry()

```
uint8_t controller_interface_get_upd_telemetry ( )
```

4.5.2.45 controller_interface_request_acc_axis()

```
void controller_interface_request_acc_axis (
             uint8_t controllerSelect,
             uint8_t accSelect,
             char axis )
```

Public function to request an update from the joint's accelerometer via CAN bus; acc and rot.

Parameters

controllerSelect	One of two joints available to the MCU
accSelect	(One of) the joint's accelerometer(s)
axis	The axis for which information is requested

4.5.2.46 controller_interface_rot_clear_newX()

```
void controller_interface_rot_clear_newX (
             uint8_t accSelect )
```

Public function to clear the new X rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.47 controller_interface_rot_clear_newY()

```
void controller_interface_rot_clear_newY (
             uint8_t accSelect )
```

Public function to clear the new Y rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.48 controller_interface_rot_clear_newZ()

```
void controller_interface_rot_clear_newZ (
             uint8_t accSelect )
```

Public function to clear the new Z rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.49 controller_interface_rot_get_newX()

```
uint8_t controller_interface_rot_get_newX (
             uint8_t accSelect )
```

Public function to get the new X rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Z axis rotation new data flag

4.5.2.50 controller_interface_rot_get_newY()

```
uint8_t controller_interface_rot_get_newY (
             uint8_t accSelect )
```

Public function to get the new Y rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Y axis rotation new data flag

4.5.2.51 controller_interface_rot_get_newZ()

```
uint8_t controller_interface_rot_get_newZ (
            uint8_t accSelect )
```

Public function to get the new Z rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Z axis rotation new data flag

4.5.2.52 controller_interface_rot_getX()

```
int16_t controller_interface_rot_getX (
            uint8_t accSelect )
```

Public function to get the accelerometer X axis rotation rate.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

X axis rotation rate raw value

4.5.2.53 controller_interface_rot_getY()

```
int16_t controller_interface_rot_getY (
            uint8_t accSelect )
```

Public function to get the accelerometer Y axis rotation rate.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Y axis rotation rate raw value

4.5.2.54 controller_interface_rot_getZ()

```
int16_t controller_interface_rot_getZ (
             uint8_t accSelect )
```

Public function to get the accelerometer Z axis rotation rate.

Parameters

accSelect	The relevant accelerometer inData struct

Returns

Z axis rotation rate raw value

4.5.2.55 controller_interface_rot_set_newX()

```
void controller_interface_rot_set_newX (
             uint8_t accSelect )
```

Public function to set the new X rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.56 controller_interface_rot_set_newY()

```
void controller_interface_rot_set_newY (
             uint8_t accSelect )
```

Public function to set the new Y rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.57 controller_interface_rot_set_newZ()

```
void controller_interface_rot_set_newZ (
            uint8_t accSelect )
```

Public function to set the new Z rotation data flag.

Parameters

accSelect	The relevant accelerometer inData struct

4.5.2.58 controller_interface_rot_setX()

```
void controller_interface_rot_setX (
            uint8_t accSelect,
            int16_t rotVal )
```

Public function to set the accelerometer X axis rotation rate.

Parameters

accSelect	The relevant accelerometer inData struct
rotVal	X axis rotation rate raw value

4.5.2.59 controller_interface_rot_setY()

```
void controller_interface_rot_setY (
            uint8_t accSelect,
            int16_t rotVal )
```

Public function to set the accelerometer Y axis rotation rate.

Parameters

accSelect	The relevant accelerometer inData struct
rotVal	Y axis rotation rate raw value

4.5.2.60 controller_interface_rot_setZ()

```
void controller_interface_rot_setZ (
            uint8_t accSelect,
            int16_t rotVal )
```

Public function to set the accelerometer Z axis rotation rate.

Parameters

accSelect	The relevant accelerometer inData struct
rotVal	Z axis rotation rate raw value

4.5.2.61 controller_interface_set_acc_poll()

```
void controller_interface_set_acc_poll ( )
```

Puclic function to set the accelerometer poll flag.

4.5.2.62 controller_interface_set_error()

```
void controller_interface_set_error (
            uint8_t controllerSelect,
            float error )
```

Public function to set the positional error of a joint.

Parameters

controllerSelect	One of two joints available to the MCU
error	Joint positional error in rads

4.5.2.63 controller_interface_set_moving()

```
void controller_interface_set_moving (
            uint8_t controllerSelect )
```

Public function to set the moving state flag of the joint.

Parameters

controllerSelect	One of two joints available to the MCU

4.5.2.64 controller_interface_set_mtr_poll()

```
void controller_interface_set_mtr_poll ( )
```

Puclic function to set the motor poll flag.

4.5.2.65 controller_interface_set_position()

```
void controller_interface_set_position (
            uint8_t controllerSelect,
            float position )
```

Public function to get the current position of a joint.

Parameters

controllerSelect	One of two joints available to the MCU
position	Joint position in rads relative to its zero position

4.5.2.66 controller_interface_set_power()

```
void controller_interface_set_power (
            uint8_t controllerSelect,
            float power )
```

Public function to set the power of the joint.

Parameters

controllerSelect	One of two joints available to the MCU
power	Percentage of maximum power

4.5.2.67 controller_interface_set_setpoint()

```
void controller_interface_set_setpoint (
            uint8_t controllerSelect,
            float setPoint )
```

Public function to set the positional setpoint of a joint.

Parameters

controllerSelect	One of two joints available to the MCU
setPoint	Joint positional setpoint in rads relative to its zero position

4.5.2.68 controller_interface_set_upd_ctrl()

```
void controller_interface_set_upd_ctrl ( )
```

4.5.2.69 controller_interface_set_upd_telemetry()

```
void controller_interface_set_upd_telemetry ( )
```

4.5.2.70 controller_interface_update_controller()

```
void controller_interface_update_controller ( )
```

Main function to run the controller on both joints.

4.5.2.71 controller_interface_update_error()

```
void controller_interface_update_error (
            uint8_t controllerSelect )
```

Public function to trigger a calculation of the joint's positional error.

Parameters

controllerSelect	One of two joints available to the MCU

4.5.2.72 controller_interface_update_position()

```
void controller_interface_update_position (
            uint8_t controllerSelect )
```

Public function to trigger an update of the joint's position.

Parameters

controllerSelect	One of two joints available to the MCU

4.5.2.73 controller_interface_update_power()

```
void controller_interface_update_power (
            uint8_t controllerSelect )
```

Public function to trigger an update of the joint's power by PID.

Parameters

controllerSelect	One of two joints available to the MCU

4.5.2.74 controller_rot_clear_newX()

```
void controller_rot_clear_newX (
            accelerometer_inData * accSelect )
```

4.5.2.75 controller_rot_clear_newY()

```
void controller_rot_clear_newY (
            accelerometer_inData * accSelect )
```

4.5.2.76 controller_rot_clear_newZ()

```
void controller_rot_clear_newZ (
            accelerometer_inData * accSelect )
```

4.5.2.77 controller_rot_get_newX()

```
uint8_t controller_rot_get_newX (
            accelerometer_inData * accSelect )
```

4.5.2.78 controller_rot_get_newY()

```
uint8_t controller_rot_get_newY (
             accelerometer_inData * accSelect )
```

4.5.2.79 controller_rot_get_newZ()

```
uint8_t controller_rot_get_newZ (
             accelerometer_inData * accSelect )
```

4.5.2.80 controller_rot_getX()

```
int16_t controller_rot_getX (
             accelerometer_inData * accSelect )
```

4.5.2.81 controller_rot_getY()

```
int16_t controller_rot_getY (
             accelerometer_inData * accSelect )
```

4.5.2.82 controller_rot_getZ()

```
int16_t controller_rot_getZ (
             accelerometer_inData * accSelect )
```

4.5.2.83 controller_rot_set_newX()

```
void controller_rot_set_newX (
             accelerometer_inData * accSelect )
```

4.5.2.84 controller_rot_set_newY()

```
void controller_rot_set_newY (
             accelerometer_inData * accSelect )
```

4.5.2.85 controller_rot_set_newZ()

```
void controller_rot_set_newZ (
            accelerometer_inData * accSelect )
```

4.5.2.86 controller_rot_setX()

```
void controller_rot_setX (
            accelerometer_inData * accSelect,
            int16_t rotVal )
```

4.5.2.87 controller_rot_setY()

```
void controller_rot_setY (
            accelerometer_inData * accSelect,
            int16_t rotVal )
```

4.5.2.88 controller_rot_setZ()

```
void controller_rot_setZ (
            accelerometer_inData * accSelect,
            int16_t rotVal )
```

4.5.2.89 joint_controller_acceleration_to_angle()

```
float joint_controller_acceleration_to_angle (
            joint_controller_descriptor * joint )
```

4.5.2.90 joint_controller_adjust_enc_sp()

```
void joint_controller_adjust_enc_sp (
            joint_controller_descriptor * joint )
```

4.5.2.91 joint_controller_clear_moving()

```
void joint_controller_clear_moving (
             joint_controller_descriptor * joint )
```

4.5.2.92 joint_controller_clicks_to_pos()

```
float joint_controller_clicks_to_pos (
             joint_controller_descriptor * joint )
```

4.5.2.93 joint_controller_get_error()

```
float joint_controller_get_error (
             joint_controller_descriptor * joint )
```

4.5.2.94 joint_controller_get_moving()

```
uint8_t joint_controller_get_moving (
             joint_controller_descriptor * joint )
```

4.5.2.95 joint_controller_get_position()

```
float joint_controller_get_position (
             joint_controller_descriptor * joint )
```

4.5.2.96 joint_controller_get_setpoint()

```
float joint_controller_get_setpoint (
             joint_controller_descriptor * joint )
```

Private functions are not described individually, but serve the same purpose as their public counterparts. The joint controller descriptors are static structs defined in the .c file and correspond to the unique joints of the robotic arm.

4.5.2.97 joint_controller_request_acc_axis()

```
void joint_controller_request_acc_axis (
             joint_controller_descriptor * joint,
             uint8_t accSelect,
             char axis )
```

4.5.2.98 joint_controller_set_error()

```
void joint_controller_set_error (
            joint_controller_descriptor * joint,
            float error )
```

4.5.2.99 joint_controller_set_moving()

```
void joint_controller_set_moving (
            joint_controller_descriptor * joint )
```

4.5.2.100 joint_controller_set_position()

```
void joint_controller_set_position (
            joint_controller_descriptor * joint,
            float position )
```

4.5.2.101 joint_controller_set_power()

```
void joint_controller_set_power (
            joint_controller_descriptor * joint,
            float power )
```

4.5.2.102 joint_controller_set_setpoint()

```
void joint_controller_set_setpoint (
            joint_controller_descriptor * joint,
            float setpoint )
```

4.5.2.103 joint_controller_update_error()

```
void joint_controller_update_error (
            joint_controller_descriptor * joint )
```

4.5.2.104 joint_controller_update_position()

```
void joint_controller_update_position (
            joint_controller_descriptor * joint )
```

4.5.2.105 joint_controller_update_power()

```
void joint_controller_update_power (
            joint_controller_descriptor * joint )
```

4.6 motor_driver.h File Reference

This file contains all the function prototypes and struct definitions for the motor_driver.c file.

```
#include "stdint.h"
#include "math.h"
#include "tim.h"
#include "unit_config.h"
#include "uart_driver.h"
#include "adc_driver.h"
```

Classes

- struct motor_descriptor

  Contains static and state information relevant to the operation of a motor driver.

Functions

- void motor_interface_zero (uint8_t motorSelect)

  Set to zero the motor encoder counters. This function is absolutely cursed, for whatever reason. Avoid, debug and/or reimplement.
- void motor_interface_update_power (uint8_t motorSelect)

  Update the power setting of the selected motor based on the relevant controller (P/I/D)
- void motor_interface_update_tot_cnt (uint8_t motorSelect)

  Update the total number of registered encoder counts since init.
- int32_t motor_interface_get_setpoint (uint8_t motorSelect)

  Returns the setpoint of the selected motors.
- int32_t motor_interface_get_total_count (uint8_t motorSelect)

  Returns the number of total registered encoder counts since init.
- uint16_t motor_interface_get_encoder_count (uint8_t motorSelect)

  Returns the current encoder hardware count.
- uint8_t motor_interface_get_id (uint8_t motorSelect)

  Returns the numerical ID of the motor.
- int32_t motor_interface_get_resolution (uint8_t motorSelect)

  Returns the resolution of the motor in encoder clicks per rad or mm.

- int32_t motor_interface_get_delta (uint8_t motorSelect)

  Returns the most recently registered change in encoder increment.
- uint8_t motor_interface_get_moving (uint8_t motorSelect)

  Returns the isMoving flag.
- void motor_interface_set_power (uint8_t motorSelect, uint8_t direction, double power)

  Lets the user set the motor power setting directly.
- void motor_interface_set_setpoint (uint8_t motorSelect, int32_t setpoint)

  Lets the user set the motor setpoint directly.
- void motor_interface_set_total_count (uint8_t motorSelect, int32_t count)

  Lets the user override the registered totalCount.
- void motor_interface_delta_setpoint (uint8_t motorSelect, int32_t delta)

  Lets the user increment or decrement the motor encoder setpoint.
- float motor_interface_calculate_torque (uint8_t motorSelect)

  Uses the motor's adc and torque constant.
- void motor_driver_zero (motor_descriptor ∗motor)

  Zero counters.
- void motor_driver_update_power (motor_descriptor ∗motor)

  Sets the power of the selected motor based on a controller heuristic.
- void motor_driver_update_tot_cnt (motor_descriptor ∗motor)

  Updates the encoder total count based on the relevant heuristic.
- int32_t motor_driver_get_setpoint (motor_descriptor ∗motor)

  Gets the total encoder count setpoint of the selected motor.
- int32_t motor_driver_get_total_cnt (motor_descriptor ∗motor)

  Gets the total encoder count since init.
- uint16_t motor_driver_get_encoder_cnt (motor_descriptor ∗motor)

  Gets the current hardware encoder count.
- uint8_t motor_driver_get_id (motor_descriptor ∗motor)
- int32_t motor_driver_get_resolution (motor_descriptor ∗motor)
- uint8_t motor_driver_get_moving (motor_descriptor ∗motor)
- int32_t motor_driver_get_delta (motor_descriptor ∗motor)
- void motor_driver_set_power (motor_descriptor ∗motor, uint8_t direction, double power)

  Set the power of a motor, limited by the motor's safety cap.
- void motor_driver_set_setpoint (motor_descriptor ∗motor, int32_t setpoint)

  Sets the total encoder count setpoint of the selected motor.
- void motor_driver_set_total_count (motor_descriptor ∗motor, int32_t count)

  Sets the total encoder count of the selected motor.
- void motor_driver_delta_setpoint (motor_descriptor ∗motor, int32_t delta)

  Changes the encoder setpoint of the relevant motor.
- void motor_driver_set_pwm_dc (uint32_t ∗timerCounter, double pct)

  Sets the duty cycle of the selected PWM timer as a percentage of max.
- void motor_driver_calc_safe_vlt (motor_descriptor ∗motor)

  Calculates the safe power percentage limit based on the descriptor struct.
- float motor_driver_calculate_torqe (motor_descriptor ∗motor, uint8_t adcSelect)

  Calculates torque from adc current and KT.
- void motor_driver_go_forward (double pct, TIM_TypeDef ∗mtr, int8_t polarity)

  Forward is the direction of increasing encoder count.
- void motor_driver_go_backward (double pct, TIM_TypeDef ∗mtr, int8_t polarity)

  Backward is the direction of decreasing encoder count.

4.6.1 Detailed Description

This file contains all the function prototypes and struct definitions for the motor_driver.c file.

Attention

Motor driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver makes use of the STM32's timer peripheral to generate PWM signals driving the DRV8251A H-bridge motor drivers. The motor descriptor structs hold information relevant to the control of each motor, most importantly the safe voltage limit for each motor embedded in the robotic arm. Additionally, "trip" information such as the total number of encoder counts registered since startup, critical to the state estimation of the arm.

As all STM32s are responsible for the driving of two motors each, the .c file defines two instances of the descriptor struct, motor1 and motor2.

4.6.2 Function Documentation

4.6.2.1 motor_driver_calc_safe_vlt()

```
void motor_driver_calc_safe_vlt (
            motor_descriptor * motor )
```

Calculates the safe power percentage limit based on the descriptor struct.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

4.6.2.2 motor_driver_calculate_torqe()

```
float motor_driver_calculate_torqe (
            motor_descriptor * motor,
            uint8_t adcSelect )
```

Calculates torque from adc current and KT.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

Returns

Nm torque

4.6.2.3 motor_driver_delta_setpoint()

```
void motor_driver_delta_setpoint (
            motor_descriptor * motor,
            int32_t delta )
```

Changes the encoder setpoint of the relevant motor.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2
delta	Change to encoder count setpoint

4.6.2.4 motor_driver_get_delta()

```
int32_t motor_driver_get_delta (
            motor_descriptor * motor )
```

Parameters

motor	

Returns

4.6.2.5 motor_driver_get_encoder_cnt()

```
uint16_t motor_driver_get_encoder_cnt (
            motor_descriptor * motor )
```

Gets the current hardware encoder count.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

Returns

TIMx->CNT

4.6.2.6 motor_driver_get_id()

```
uint8_t motor_driver_get_id (
            motor_descriptor * motor )
```

Parameters

motorSelect	

Returns

4.6.2.7 motor_driver_get_moving()

```
uint8_t motor_driver_get_moving (
            motor_descriptor * motor )
```

Parameters

motor	

Returns

4.6.2.8 motor_driver_get_resolution()

```
int32_t motor_driver_get_resolution (
            motor_descriptor * motor )
```

Parameters

motor	

Returns

4.6.2.9 motor_driver_get_setpoint()

```
int32_t motor_driver_get_setpoint (
            motor_descriptor * motor )
```

Gets the total encoder count setpoint of the selected motor.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

Returns

encoderTotalSetpoint

4.6.2.10 motor_driver_get_total_cnt()

```
int32_t motor_driver_get_total_cnt (
            motor_descriptor * motor )
```

Gets the total encoder count since init.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

Returns

encoderTotalCount

4.6.2.11 motor_driver_go_backward()

```
void motor_driver_go_backward (
            double pct,
            TIM_TypeDef * mtr,
            int8_t polarity )
```

Backward is the direction of decreasing encoder count.

Parameters

pct	Percentage of input voltage
mtr	Pointer to timer

4.6.2.12 motor_driver_go_forward()

```
void motor_driver_go_forward (
            double pct,
            TIM_TypeDef * mtr,
            int8_t polarity )
```

Forward is the direction of increasing encoder count.

Parameters

pct	Percentage of input voltage
mtr	Pointer to timer

4.6.2.13 motor_driver_set_power()

```
void motor_driver_set_power (
            motor_descriptor * motor,
            uint8_t direction,
            double power )
```

Set the power of a motor, limited by the motor's safety cap.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2
direction	0 or 1 for forwards or backwards, respectively
power	percentage of input voltage

4.6.2.14 motor_driver_set_pwm_dc()

```
void motor_driver_set_pwm_dc (
            uint32_t * timerCounter,
            double pct )
```

Sets the duty cycle of the selected PWM timer as a percentage of max.

Parameters

timerCounter	Pointer to the relevant TIMx->CNT register
pct	Percentage of maximum duty cycle

4.6.2.15 motor_driver_set_setpoint()

```
void motor_driver_set_setpoint (
            motor_descriptor * motor,
            int32_t setpoint )
```

Sets the total encoder count setpoint of the selected motor.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2
setpoint	Total encoder count

4.6.2.16 motor_driver_set_total_count()

```
void motor_driver_set_total_count (
            motor_descriptor * motor,
            int32_t count )
```

Sets the total encoder count of the selected motor.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2
count	Count to set

4.6.2.17 motor_driver_update_power()

```
void motor_driver_update_power (
            motor_descriptor * motor )
```

Sets the power of the selected motor based on a controller heuristic.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

4.6.2.18 motor_driver_update_tot_cnt()

```
void motor_driver_update_tot_cnt (
            motor_descriptor * motor )
```

Updates the encoder total count based on the relevant heuristic.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

4.6.2.19 motor_driver_zero()

```
void motor_driver_zero (
            motor_descriptor * motor )
```

Zero counters.

Parameters

motor	Pointer to the relevant motor struct, motor1 or motor2

4.6.2.20 motor_interface_calculate_torque()

```
float motor_interface_calculate_torque (
            uint8_t motorSelect )
```

Uses the motor's adc and torque constant.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

float, Nm torque

4.6.2.21 motor_interface_delta_setpoint()

```
void motor_interface_delta_setpoint (
            uint8_t motorSelect,
            int32_t delta )
```

Lets the user increment or decrement the motor encoder setpoint.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively
delta	int32_t, number of encoder clicks by which the setpoint is changed

4.6.2.22 motor_interface_get_delta()

```
int32_t motor_interface_get_delta (
            uint8_t motorSelect )
```

Returns the most recently registered change in encoder increment.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

mostRecentDelta

4.6.2.23 motor_interface_get_encoder_count()

```
uint16_t motor_interface_get_encoder_count (
            uint8_t motorSelect )
```

Returns the current encoder hardware count.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

TIMx->CNT

4.6.2.24 motor_interface_get_id()

```
uint8_t motor_interface_get_id (
            uint8_t motorSelect )
```

Returns the numerical ID of the motor.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

motorID

4.6.2.25 motor_interface_get_moving()

```
uint8_t motor_interface_get_moving (
            uint8_t motorSelect )
```

Returns the isMoving flag.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

isMoving

4.6.2.26 motor_interface_get_resolution()

```
int32_t motor_interface_get_resolution (
            uint8_t motorSelect )
```

Returns the resolution of the motor in encoder clicks per rad or mm.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

resolution

4.6.2.27 motor_interface_get_setpoint()

```
int32_t motor_interface_get_setpoint (
            uint8_t motorSelect )
```

Returns the setpoint of the selected motors.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

encoderTotalSetpoint

4.6.2.28 motor_interface_get_total_count()

```
int32_t motor_interface_get_total_count (
            uint8_t motorSelect )
```

Returns the number of total registered encoder counts since init.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

Returns

encoderTotalCount

4.6.2.29 motor_interface_set_power()

```
void motor_interface_set_power (
            uint8_t motorSelect,
            uint8_t direction,
            double power )
```

Lets the user set the motor power setting directly.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively
direction	0 or 1 for backwards or forwards, respectively
power	percentage of input voltage

4.6.2.30 motor_interface_set_setpoint()

```
void motor_interface_set_setpoint (
            uint8_t motorSelect,
            int32_t setpoint )
```

Lets the user set the motor setpoint directly.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively
setpoint	int32_t

4.6.2.31 motor_interface_set_total_count()

```
void motor_interface_set_total_count (
            uint8_t motorSelect,
            int32_t count )
```

Lets the user override the registered totalCount.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively
count	int32_t

4.6.2.32 motor_interface_update_power()

```
void motor_interface_update_power (
            uint8_t motorSelect )
```

Update the power setting of the selected motor based on the relevant controller (P/I/D)

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

4.6.2.33 motor_interface_update_tot_cnt()

```
void motor_interface_update_tot_cnt (
            uint8_t motorSelect )
```

Update the total number of registered encoder counts since init.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

4.6.2.34 motor_interface_zero()

```
void motor_interface_zero (
            uint8_t motorSelect )
```

Set to zero the motor encoder counters. This function is absolutely cursed, for whatever reason. Avoid, debug and/or reimplement.

Parameters

motorSelect	0 or 1 for motor1 or motor2 respectively

4.7 ros_uart_parser.h File Reference

This file contains all the function prototypes for the ros_uart_parser.c file.

```
#include "string.h"
#include <stdio.h>
#include "stdint.h"
#include "unit_config.h"
#include "joint_controller.h"
#include "state_machine.h"
#include "can_driver.h"
#include "string_cmd_parser.h"
```

Functions

- void ros_interface_set_rxBuffer (uint8_t ∗input)
- void ros_interface_get_rxBuffer (uint8_t ∗target)
- void ros_interface_clear_rxBuffer ()
- void ros_interface_set_newMsgFlag ()
- uint8_t ros_interface_get_newMsgFlag ()
- void ros_interface_clear_newMsgFlag ()
- void ros_interface_parse_input ()
- void ros_interface_set_pinchPos (float pos)

4.7.1 Detailed Description

This file contains all the function prototypes for the ros_uart_parser.c file.

Attention

ROS/UART serial interface driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver makes use of the STM32's UART peripheral, as well as the string command parser module and C string libraries to do basic processing of incoming and outgoing UART data.

The parser/handler functions are triggered by the HAL_UART_RxCpltCallback

4.7.2 Function Documentation

4.7.2.1 ros_interface_clear_newMsgFlag()

```
void ros_interface_clear_newMsgFlag ( )
```

4.7.2.2 ros_interface_clear_rxBuffer()

```
void ros_interface_clear_rxBuffer ( )
```

4.7.2.3 ros_interface_get_newMsgFlag()

```
uint8_t ros_interface_get_newMsgFlag ( )
```

4.7.2.4 ros_interface_get_rxBuffer()

```
void ros_interface_get_rxBuffer (
            uint8_t * target )
```

4.7.2.5 ros_interface_parse_input()

```
void ros_interface_parse_input ( )
```

4.7.2.6 ros_interface_set_newMsgFlag()

```
void ros_interface_set_newMsgFlag ( )
```

4.7.2.7 ros_interface_set_pinchPos()

```
void ros_interface_set_pinchPos (
            float pos )
```

4.7.2.8 ros_interface_set_rxBuffer()

```
void ros_interface_set_rxBuffer (
            uint8_t * input )
```

4.8 state_machine.h File Reference

This file contains all the function prototypes and struct definitions for the state_machine.c file.

```
#include "stdint.h"
#include "tim.h"
#include "unit_config.h"
#include "uart_driver.h"
#include "joint_controller.h"
#include "can_driver.h"
```

Enumerations

- enum global_state {
  GS_ERROR , GS_IDLE , GS_CALIBRATING , GS_OPERATING ,
  num_gStates }
- enum calibration_state {
  CS_ERROR , CS_RAIL , CS_SHOULDER , CS_ELBOW ,
  CS_WRIST , CS_TWIST , CS_PINCH , num_cStates }

Functions

- uint8_t state_interface_get_global_state ()
- void state_interface_set_global_state (uint8_t inState)
- void state_interface_broadcast_global_state ()
- uint8_t state_interface_get_calibration_state ()
- void state_interface_set_calibration_state (uint8_t inState)
- void state_interface_broadcast_calibration_state ()
- void state_interface_set_es_flag ()
- uint8_t state_interface_get_es_flag ()
- void state_interface_clear_es_flag ()
- void state_interface_set_tw_flag ()
- uint8_t state_interface_get_tw_flag ()
- void state_interface_clear_tw_flag ()
- void state_calibrate_rail ()
- void state_calibrate_shoulder ()
- void state_calibrate_elbow ()
- void state_calibrate_wrist ()
- void state_calibrate_twist ()
- void state_calibrate_pinch ()

4.8.1 Detailed Description

This file contains all the function prototypes and struct definitions for the state_machine.c file.

Attention

Motor driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver makes use of the STM32's timer peripheral to generate PWM signals driving the DRV8251A H-bridge motor drivers. The motor descriptor structs hold information relevant to the control of each motor, most importantly the safe voltage limit for each motor embedded in the robotic arm. Additionally, "trip" information such as the total number of encoder counts registered since startup, critical to the state estimation of the arm.

4.8.2 Enumeration Type Documentation

4.8.2.1 calibration_state

```
enum calibration_state
```

Enumerator

CS_ERROR	
CS_RAIL	
CS_SHOULDER	
CS_ELBOW	
CS_WRIST	
CS_TWIST	
CS_PINCH	
num_cStates	

4.8.2.2 global_state

```
enum global_state
```

Enumerator

GS_ERROR	
GS_IDLE	
GS_CALIBRATING	
GS_OPERATING	
num_gStates	

4.8.3 Function Documentation

4.8.3.1 state_calibrate_elbow()

```
void state_calibrate_elbow ( )
```

4.8.3.2 state_calibrate_pinch()

```
void state_calibrate_pinch ( )
```

4.8.3.3 state_calibrate_rail()

```
void state_calibrate_rail ( )
```

4.8.3.4 state_calibrate_shoulder()

```
void state_calibrate_shoulder ( )
```

4.8.3.5 state_calibrate_twist()

```
void state_calibrate_twist ( )
```

4.8.3.6 state_calibrate_wrist()

```
void state_calibrate_wrist ( )
```

4.8.3.7 state_interface_broadcast_calibration_state()

```
void state_interface_broadcast_calibration_state ( )
```

4.8.3.8 state_interface_broadcast_global_state()

```
void state_interface_broadcast_global_state ( )
```

4.8.3.9 state_interface_clear_es_flag()

```
void state_interface_clear_es_flag ( )
```

4.8.3.10 state_interface_clear_tw_flag()

```
void state_interface_clear_tw_flag ( )
```

4.8.3.11 state_interface_get_calibration_state()

```
uint8_t state_interface_get_calibration_state ( )
```

4.8.3.12 state_interface_get_es_flag()

```
uint8_t state_interface_get_es_flag ( )
```

4.8.3.13 state_interface_get_global_state()

```
uint8_t state_interface_get_global_state ( )
```

4.8.3.14 state_interface_get_tw_flag()

```
uint8_t state_interface_get_tw_flag ( )
```

4.8.3.15 state_interface_set_calibration_state()

```
void state_interface_set_calibration_state (
             uint8_t inState )
```

4.8.3.16 state_interface_set_es_flag()

```
void state_interface_set_es_flag ( )
```

4.8.3.17 state_interface_set_global_state()

```
void state_interface_set_global_state (
             uint8_t inState )
```

4.8.3.18 state_interface_set_tw_flag()

```
void state_interface_set_tw_flag ( )
```

4.9 string_cmd_parser.h File Reference

This file contains all the function prototypes and struct definitions for the string_cmd_parser.c file.

```
#include "string.h"
#include <stdio.h>
#include <stdlib.h>
#include "stdint.h"
#include "unit_config.h"
#include "motor_driver.h"
#include "can_driver.h"
#include "joint_controller.h"
#include "state_machine.h"
#include "ros_uart_parser.h"
```

Classes

- struct string_cmd_processor_args

  Wrapper struct to enable a variable number of arguments to the string processor.
- struct string_cmd_pair

  Pairs a command string token with a function pointer.

Macros

- #define string_cmd_processor(...) string_cmd_processor_wrp((string_cmd_processor_args∗){__VA_↩ ARGS__})
- #define NUM_STRING_COMMANDS 0xC

Functions

- void string_cmd_processor_wrp (string_cmd_processor_args ∗input)

  Wrapper function to enable a variable number of arguments to the string processor.
- void string_cmd_processor_base (char ∗inputString, uint8_t stringLength)

  Starting point for string processing, splits and incoming string into tokens delimited by " ".
- void string_cmd_category_local_motor (uint8_t motor, char(∗inputTokens)[64])

  Handles commands to a motor/joint controller local to the STM32 connected to the serial interface.
- void string_cmd_category_remote_motor (uint8_t motor, char(∗inputTokens)[64])

  Handles commands to a motor/joint controller accessible via CAN from the STM32 connected to the serial interface.
- void string_cmd_category_adc ()

  Handles commands to an ADC (NOTE: Not implemented)
- void string_cmd_category_accelerometer ()

  Handles commands to an accelerometer (NOTE: Not implemented)
- void string_cmd_rail (char(∗inputTokens)[64])

  Called when "rail" token is registered; concerns the rail linear joint.
- void string_cmd_shoulder (char(∗inputTokens)[64])

  Called when "shoulder" token is registered; concerns the shoulder joint.
- void string_cmd_elbow (char(∗inputTokens)[64])

  Called when "elbow" token is registered; concerns the elbow joint.
- void string_cmd_wrist (char(∗inputTokens)[64])

Called when "wrist" token is registered; concerns the wrist joint.

- void string_cmd_twist (char(∗inputTokens)[64])

  Called when "twist" token is registered; concerns the twist joint.
- void string_cmd_pinch (char(∗inputTokens)[64])

  Called when "pinch" token is registered; concerns the pinch joint.
- void string_cmd_can (char(∗inputTokens)[64])

  Called when "can" token is registered; concerns the CAN bus.
- void string_cmd_stop (char(∗inputTokens)[64])

  Called when "S" token is registered; soft emergency stop for all joints.
- void string_cmd_acc1 (char(∗inputTokens)[64])

  Called when "acc1" token is registered; concerns the shoulder accelerometer.
- void string_cmd_rly (char(∗inputTokens)[64])

  Called when "relay" token is registered; concerns the motor driver enable relays.
- void string_cmd_home (char(∗inputTokens)[64])

  Called when "home" token is registered; starts the home zero calibration.
- void string_cmd_state (char(∗inputTokens)[64])

  Called when "state" token is registered; lets the user set a global state.

Variables

- static string_cmd_pair stringCmdList [NUM_STRING_COMMANDS]

  Pairing of string commands and handler functions.

4.9.1 Detailed Description

This file contains all the function prototypes and struct definitions for the string_cmd_parser.c file.

Attention

Keyboard input string parser for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The parser makes use of the joint controller, motor driver, and the STM32's UART peripheral as well as C string libraries to enable debugging of the arm during development. It lets the user write specified commands to a serial interface such as PuTTy to control joints directly, as well as make sensor readouts on demand.

The string_cmd_pair list defines valid commands available to the user, which may be expanded at will. When using the arm with a serial interface, an input string beginning with one of the defined words will be handled by the corresponding function. In the interest of standardisation, all input strings limited to 64 characters.

4.9.2 Macro Definition Documentation

4.9.2.1 NUM_STRING_COMMANDS

```
#define NUM_STRING_COMMANDS 0xC
```

4.9.2.2 string_cmd_processor

```
#define string_cmd_processor(
              ... ) string_cmd_processor_wrp((string_cmd_processor_args*){__VA_ARGS__})
```

4.9.3 Function Documentation

4.9.3.1 string_cmd_acc1()

```
void string_cmd_acc1 (
             char(*) inputTokens[64] )
```

Called when "acc1" token is registered; concerns the shoulder accelerometer.

Parameters

inputTokens	Arguments to parse

4.9.3.2 string_cmd_can()

```
void string_cmd_can (
             char(*) inputTokens[64] )
```

Called when "can" token is registered; concerns the CAN bus.

Parameters

inputTokens	Arguments to parse

4.9.3.3 string_cmd_category_accelerometer()

```
void string_cmd_category_accelerometer ( )
```

Handles commands to an accelerometer (NOTE: Not implemented)

4.9.3.4 string_cmd_category_adc()

```
void string_cmd_category_adc ( )
```

Handles commands to an ADC (NOTE: Not implemented)

4.9.3.5 string_cmd_category_local_motor()

```
void string_cmd_category_local_motor (
            uint8_t motor,
            char(*) inputTokens[64] )
```

Handles commands to a motor/joint controller local to the STM32 connected to the serial interface.

Parameters

motor	One of two motors, 0 or 1
inputTokens	Tokens received from the string processor

4.9.3.6 string_cmd_category_remote_motor()

```
void string_cmd_category_remote_motor (
            uint8_t motor,
            char(*) inputTokens[64] )
```

Handles commands to a motor/joint controller accessible via CAN from the STM32 connected to the serial interface.

Parameters

motor	One of two motors, 0 or 1
inputTokens	Tokens received from the string processor

4.9.3.7 string_cmd_elbow()

```
void string_cmd_elbow (
            char(*) inputTokens[64] )
```

Called when "elbow" token is registered; concerns the elbow joint.

Parameters

inputTokens	Arguments to parse

4.9.3.8 string_cmd_home()

```
void string_cmd_home (
            char(*) inputTokens[64] )
```

Called when "home" token is registered; starts the home zero calibration.

Parameters

inputTokens	Arguments to parse

4.9.3.9 string_cmd_pinch()

```
void string_cmd_pinch (
            char(*) inputTokens[64] )
```

Called when "pinch" token is registered; concerns the pinch joint.

Parameters

inputTokens	Arguments to parse

4.9.3.10 string_cmd_processor_base()

```
void string_cmd_processor_base (
            char * inputString,
            uint8_t stringLength )
```

Starting point for string processing, splits and incoming string into tokens delimited by " ".

Parameters

inputString	The string to be processed
stringLength	Length of the string to be processed

4.9.3.11 string_cmd_processor_wrp()

```
void string_cmd_processor_wrp (
            string_cmd_processor_args * input )
```

Wrapper function to enable a variable number of arguments to the string processor.

Parameters

input	Pointer to arguments

4.9.3.12 string_cmd_rail()

```
void string_cmd_rail (
             char(*) inputTokens[64] )
```

Called when "rail" token is registered; concerns the rail linear joint.

Parameters

inputTokens	Arguments to parse

4.9.3.13 string_cmd_rly()

```
void string_cmd_rly (
             char(*) inputTokens[64] )
```

Called when "relay" token is registered; concerns the motor driver enable relays.

Parameters

inputTokens	Arguments to parse

4.9.3.14 string_cmd_shoulder()

```
void string_cmd_shoulder (
             char(*) inputTokens[64] )
```

Called when "shoulder" token is registered; concerns the shoulder joint.

Parameters

inputTokens	Arguments to parse

4.9.3.15 string_cmd_state()

```
void string_cmd_state (
             char(*) inputTokens[64] )
```

Called when "state" token is registered; lets the user set a global state.

Parameters

inputTokens	

4.9.3.16 string_cmd_stop()

```
void string_cmd_stop (
            char(*) inputTokens[64] )
```

Called when "S" token is registered; soft emergency stop for all joints.

Parameters

inputTokens	Arguments to parse (none)

4.9.3.17 string_cmd_twist()

```
void string_cmd_twist (
            char(*) inputTokens[64] )
```

Called when "twist" token is registered; concerns the twist joint.

Parameters

inputTokens	Arguments to parse

4.9.3.18 string_cmd_wrist()

```
void string_cmd_wrist (
            char(*) inputTokens[64] )
```

Called when "wrist" token is registered; concerns the wrist joint.

Parameters

inputTokens	Arguments to parse

4.9.4 Variable Documentation

4.9.4.1 stringCmdList

```
string_cmd_pair stringCmdList[NUM_STRING_COMMANDS]  [static]
```

Initial value:

```
=
  {
  {"rail", string_cmd_rail},
  {"shoulder", string_cmd_shoulder},
  {"elbow", string_cmd_elbow},
  {"wrist", string_cmd_wrist},
  {"twist", string_cmd_twist},
  {"pinch", string_cmd_pinch},
  {"can", string_cmd_can},
  {"s", string_cmd_stop},
  {"accl", string_cmd_accl},
  {"relay", string_cmd_rly},
  {"home", string_cmd_home},
  {"state", string_cmd_state},
  }
```

Pairing of string commands and handler functions.

4.10 uart_driver.h File Reference

This file contains all the function prototypes for the string_cmd_parser.c file.

```
#include "string.h"
#include <stdio.h>
#include "usart.h"
#include "unit_config.h"
#include "string_cmd_parser.h"
#include "ros_uart_parser.h"
#include "joint_controller.h"
#include "state_machine.h"
```

Functions

- void uart_send_string (char ∗str)

  Sends a string over the UART peripheral interface.
- void uart_parse_hmi_input (char ∗input, uint8_t ∗buffer, uint8_t bufferLength, uint8_t ∗bufferPos)

  Reads incoming UART data and enables a keyboard based HMI to the STM32.
- void uart_parse_ros_input (char ∗input)

  DEPRECATED, replaced by ROS interface.
- void uart_hmi_rx_handler ()

  Handles incoming UART data when the peripheral is used as HMI.
- void uart_hmi_init ()

  Initializes the UART peripheral as HMI.
- void uart_ros_rx_handler ()

  Handles incoming UART data when the peripheral is used for ROS.
- void uart_ros_init ()

  Initializes the UART peripheral as ROS interface.

4.10.1 Detailed Description

This file contains all the function prototypes for the string_cmd_parser.c file.

Attention

UART serial interface driver for the TTK4900 Master project of Kristian Blom, spring semester of 2024. The driver makes use of the STM32's UART peripheral, as well as the string command parser module and C string libraries to do basic processing of incoming and outgoing UART data.

The parser/handler functions are triggered by the HAL_UART_RxCpltCallback

4.10.2 Function Documentation

4.10.2.1 uart_hmi_init()

```
void uart_hmi_init ( )
```

Initializes the UART peripheral as HMI.

4.10.2.2 uart_hmi_rx_handler()

```
void uart_hmi_rx_handler ( )
```

Handles incoming UART data when the peripheral is used as HMI.

4.10.2.3 uart_parse_hmi_input()

```
void uart_parse_hmi_input (
             char * input,
             uint8_t * buffer,
             uint8_t bufferLength,
             uint8_t * bufferPos )
```

Reads incoming UART data and enables a keyboard based HMI to the STM32.

Parameters

input	Last incoming byte
buffer	String buffer holding the currently relevant strings
bufferLength	Length of buffer
bufferPos	Keyboard cursor position in buffer

4.10.2.4 uart_parse_ros_input()

```
void uart_parse_ros_input (
             char * input )
```

DEPRECATED, replaced by ROS interface.

Parameters

input	

4.10.2.5 uart_ros_init()

```
void uart_ros_init ( )
```

Initializes the UART peripheral as ROS interface.

4.10.2.6 uart_ros_rx_handler()

```
void uart_ros_rx_handler ( )
```

Handles incoming UART data when the peripheral is used for ROS.

4.10.2.7 uart_send_string()

```
void uart_send_string (
             char * str )
```

Sends a string over the UART peripheral interface.

Parameters

str	string to send

4.11 unit_config.h File Reference

This file contains global information relevant to building the project for a specific STM32 in the robotic arm.

Macros

- #define MTR1 TIM15
- #define MTR2 TIM1
- #define ENC1 TIM3
- #define ENC2 TIM8
- #define PWM_CTR_PRD 2880
- #define GLOBAL_DEBUG 1
- #define VOLTAGE_IN 20
- #define TORSO 0
- #define SHOULDER 1
- #define HAND 2
- #define ACTIVE_UNIT SHOULDER
- #define UART_INTERFACE 0
- #define USB_INTERFACE 1
- #define HW_INTERFACE UART_INTERFACE
- #define CMD_MODE_TERMINAL 0
- #define CMD_MODE_ROS 1
- #define SW_INTERFACE CMD_MODE_ROS
- #define CAN_FILTER_M 0x040
- #define CAN_FILTERMASK_M 0x3C0
- #define CAN_FILTER_A 0x000
- #define CAN_FILTERMASK_A 0x2E0
- #define CAN_FILTER_G 0x400
- #define CAN_FILTERMASK_G 0x7E0
- #define UART_INPUT 1

Functions

- void activate_peripherals ()

4.11.1 Detailed Description

This file contains global information relevant to building the project for a specific STM32 in the robotic arm.

Attention

STM32 project configuration for the TTK4900 Master project of Kristian Blom, spring semester of 2024. This file specifies information unique to each STM32, i.e. the torso, shoulder and hand unit, respectively, as well as some convenience definitions common to all three.

The most important parameter is ACTIVE_UNIT, as this flag specifies which modules will be compiled, CAN message ID filters, and safe power levels for each motor. CAUTION: Flashing an STM32 unit with the wrong ACTIVE↩ _UNIT flag may cause harm to the motors, as an inappropriate safe power level may be calculated for that motor.

4.11.2 Macro Definition Documentation

4.11.2.1 ACTIVE_UNIT

```
#define ACTIVE_UNIT SHOULDER
```

4.11.2.2 CAN_FILTER_A

```
#define CAN_FILTER_A 0x000
```

4.11.2.3 CAN_FILTER_G

```
#define CAN_FILTER_G 0x400
```

4.11.2.4 CAN_FILTER_M

```
#define CAN_FILTER_M 0x040
```

4.11.2.5 CAN_FILTERMASK_A

```
#define CAN_FILTERMASK_A 0x2E0
```

4.11.2.6 CAN_FILTERMASK_G

```
#define CAN_FILTERMASK_G 0x7E0
```

4.11.2.7 CAN_FILTERMASK_M

```
#define CAN_FILTERMASK_M 0x3C0
```

4.11.2.8 CMD_MODE_ROS

```
#define CMD_MODE_ROS 1
```

4.11.2.9 CMD_MODE_TERMINAL

```
#define CMD_MODE_TERMINAL 0
```

4.11.2.10 ENC1

```
#define ENC1 TIM3
```

4.11.2.11 ENC2

```
#define ENC2 TIM8
```

4.11.2.12 GLOBAL_DEBUG

```
#define GLOBAL_DEBUG 1
```

4.11.2.13 HAND

```
#define HAND 2
```

4.11.2.14 HW_INTERFACE

```
#define HW_INTERFACE UART_INTERFACE
```

4.11.2.15 MTR1

```
#define MTR1 TIM15
```

4.11.2.16 MTR2

```
#define MTR2 TIM1
```

4.11.2.17 PWM_CTR_PRD

```
#define PWM_CTR_PRD 2880
```

4.11.2.18 SHOULDER

```
#define SHOULDER 1
```

4.11.2.19 SW_INTERFACE

```
#define SW_INTERFACE CMD_MODE_ROS
```

4.11.2.20 TORSO

```
#define TORSO 0
```

4.11.2.21 UART_INPUT

```
#define UART_INPUT 1
```

4.11.2.22 UART_INTERFACE

```
#define UART_INTERFACE 0
```

4.11.2.23 USB_INTERFACE

```
#define USB_INTERFACE 1
```

4.11.2.24 VOLTAGE_IN

```
#define VOLTAGE_IN 20
```

4.11.3 Function Documentation

4.11.3.1 activate_peripherals()

```
void activate_peripherals ( )
```

Index