Prova Prática - Laboratório de Processadores

Generated by Doxygen 1.9.1

Chapter 1

report

Chapter 2

STM32 Project Template

Template para projetos usando microcontroladores da ST e o STM32CubeMX. Consiste numa estrutura especifica de pastas, um Makefile e alguns arquivos de configuração.

2.1 Requisitos

- STM32CubeMX

  É necessário colocar o local de instalação na varíavel de ambiente `CUBE_PATH`

- make

  Linux: `sudo apt install make`
  Windows: `msys2> pacman -S make`

- CMake

  Linux: `sudo apt install cmake`
  Windows: Baixe o zip ou o instalador no Installing CMake
  É necessário que a pasta `bin` dessa instalação esteja no `PATH`. No instalador do Windows, isso é feito automaticamente

- GNU Arm Embedded Toolchain

  É necessário que a pasta `bin` dessa instalação esteja no `PATH`

- uncrustify

  Linux: `sudo apt install uncrustify`
  Windows: Baixe o .zip no SourceForge. Adicione o local do executável na variável de ambiente `PATH`.

- Visual Studio Code
  - EditorConfig
  - C/C++
  - Cortex-Debug

- STM32 Cube Programmer ou J-Link

  É necessário que o executável também esteja no `PATH`

2.2 Preparando

2.2.1 Projeto

Primeiro é necessário criar um projeto do Cube na pasta `cube/` com o nome desejado, que deve ter as seguintes opções de projeto:

Project:

- Application Structure: *Basic*
- [x] Do not generate the main()
- Toolchain / IDE: *Makefile*

Code Generator:

- STM32Cube Firmware Library Package: *Copy all used libraries into the project folder*
- Generated files:
  - *Generate peripheral initialization as a pair of .c/.h files per peripheral*
  - *Delete previously generated files when not re-generated*

Um arquivo de exemplo se encontra em `cube/stm32_project_template.ioc` com todas as configurações necessárias.

Para projetos existentes, basta mover o arquivo `.ioc` para a pasta `cube/`.

2.2.2 Compilação

O arquivo `CMakeLists.txt` deve ser editado de acordo com o projeto.

Para isso é necessário mudar o nome do projeto, o qual deve ter o mesmo do arquivo do Cube (por exemplo, `stm32_project_template.ioc`), porém sem a extensão `.ioc`.

```
# Cube file name without .ioc extension
project(stm32_project_template C ASM)
```

> Os argumentos `C` e `ASM` estão relacionados ao tipo de linguagem que o projeto utiliza (C e Assembly).

Também é necessário alterar as seguintes configurações:

```
# Device Configuration
set(DEVICE_CORTEX F3)
set(DEVICE_FAMILY STM32F3xx)
set(DEVICE_TYPE STM32F303xx)
set(DEVICE_DEF STM32F303xE)
set(DEVICE STM32F303RE)
```

Basta pegar o nome completo do microcontrolador e colocar nessas configurações, seguindo o padrão, fazendo as substituições que forem precisas por x.

> Em caso de dúvida, veja o nome do arquivo .ld gerado na pasta cube, ele contém o nome completo do microcontrolador.

2.3 Gerando arquivos

Com as configurações realizadas corretamente, você deve se direcionar para a pasta `build`. Estando lá, basta rodar o seguinte comando:

```
cmake ..
```

Esse comando é de extrema importância, pois nenhum dos outros comandos de compilação funcionarão sem ele ter sido rodado antes.

> Todos os comandos que envolvam `make` devem ser rodados dentro da pasta `build`, após o comando `cmake ..` ter sido feito.

Basicamente, ele configura o ambiente do CMake e gera os arquivos do cube, caso a pasta `cube` esteja vazia. Todavia, caso você queira apenas gerar os arquivos do cube, também é possível rodar o comando

```
make cube
```

2.4 Compilando

Para compilar os arquivos, após ter rodado `cmake ..`, ainda dentro da pasta `build`, rode:

```
make
```

O comando `make` apenas compilará o código principal, não compilando nenhum teste. Para compilar um teste, cujo arquivo se chama **nome_do_teste.c**, rode:

```
make nome_do_teste
```

2.5 Limpando Arquivos compilados

Se acontecer algum erro, pode ser necessário limpar os arquivos já compilados. Para isso, dentro da pasta `build`, faça:

```
make clean
```

Isso apaga todos os arquivos de compilação gerados, exceto aqueles que vêm das bibliotecas da ST geradas pelo Cube. Isso é feito para agilizar um novo build, já que raramente será necessário recompilar esses arquivos. Todavia, caso seja necessário, é possível limpá-los com o comando:

```
make clean_cube
```

Além disso, caso seja necessário limpar todos os arquivos de compilação, você pode rodar o comando:

```
make clean_all
```

2.6 Recompilando

Caso você queira apagar os arquivos compilados e recompilá-los, é possível fazer isso com um comando só, rodando, dentro da pasta `build`, o comando:

```
make rebuild
```

E, caso você queira apagar e recompilar todos os arquivos compilados, incluindo os do cube, rode o comando:

```
make rebuild_all
```

2.7 Gravando

Para gravar os arquivos na placa, rode
```
make flash
```

Ou, caso use um gravador com J-Link:
```
make jflash
```

Além disso, para gravar um teste, cujo nome do arquivo é **nome_do_teste.c**, deve-se rodar:
```
make flash_nome_do_teste
```

Ou, caso use um gravador com J-Link:
```
make jflash_nome_do_teste
```

2.8 Formatando

Para garantir que o código está formatado, utilize o atalho `CTRL+S` para salvar e formatar o arquivo em que se está mexendo ou, para formatar todos os arquivos do repositório de uma vez, rode:
```
make format
```

2.9 Submódulos

2.9.1 Adicionando um submódulo

Crie um diretório chamado `lib` e adicione o submódulo nele.

Exemplo:
```
mkdir lib
git submodule add --name STMSensors git@github.com:ThundeRatz/STMSensors.git lib/STMSensors
```

2.9.2 Inicializando um submódulo já existente

Ao clonar um repositório que já tem submódulos, é necessário clonar os repositórios desse submódulo. Isso pode ser feito de duas formas, clonando junto com o repositório do projeto ou depois de já ter clonado.

Exemplo:

Para se clonar junto, deve-se fazer:
```
git clone --recurse-submodules git@github.com:ThundeRatz/STM32ProjectTemplate.git
```

Para se clonar depois de já ter clonado o repositório do projeto:
```
git submodule update --init
```

2.10 Diretório de testes

O diretório `test` contém arquivos para testes de partes específicas do projeto, separando isso do código do projeto em si. Esses arquivos devem ser implementados de acordo com as necessidades dos desenvolvedores.

Para compilar e gravar um teste, siga as instruções na seção de compilação e na seção para gravação.

Cada arquivo de teste no diretório de testes funciona de forma independente, ou seja, cada um deve ter uma função `main()`, sendo cada um compilado, gravado e executado separadamente.

2.11 Debug

Em breve

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Class Documentation

5.1 ButterworthFilter Class Reference

Implementation of Butterworth second order low-pass filter A generic digital filter follows the relation a0 ∗ y[k] = sum(bi ∗ x[k - i]) - sum(aj ∗ y[k - j]) Where x[k] - measurement at instant k y[k] - filtered signal at instant k The Butterworth filter have the special property of being a maximally flat magnitude filter, in other words, is the best filter that doesn't present distortions around the cutoff frequency The formula for the continuos coefficients of the Butterworth filter is available here: https://en.wikipedia.org/wiki/Butterworth_filter The discrete version were computed with the Tustin method: https://en.wikipedia.org/wiki/↩ Bilinear_transform.

`#include <butterworth_filter.hpp>`

Public Member Functions

- ButterworthFilter (float cutoff_frequency, float sampling_frequency=1.0)

  Construct a new Butterworth Second Order filter object.
- float update (float x0)

  Produces a new value from measured data.

5.1.1 Detailed Description

Implementation of Butterworth second order low-pass filter A generic digital filter follows the relation a0 ∗ y[k] = sum(bi ∗ x[k - i]) - sum(aj ∗ y[k - j]) Where x[k] - measurement at instant k y[k] - filtered signal at instant k The Butterworth filter have the special property of being a maximally flat magnitude filter, in other words, is the best filter that doesn't present distortions around the cutoff frequency The formula for the continuos coefficients of the Butterworth filter is available here: https://en.wikipedia.org/wiki/Butterworth_filter The discrete version were computed with the Tustin method: https://en.wikipedia.org/wiki/↩ Bilinear_transform.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 ButterworthFilter()

```
ButterworthFilter::ButterworthFilter (
            float cutoff_frequency,
            float sampling_frequency = 1.0 )
```

Construct a new Butterworth Second Order filter object.

Parameters

cutoff_frequency	Low-pass cutoff frequency in Hz
sampling_frequency	Sampling frequency in Hz.

5.1.3 Member Function Documentation

5.1.3.1 update()

```
float ButterworthFilter::update (
            float x0 )
```

Produces a new value from measured data.

Parameters

x0	Last measure

Returns

Filtered value

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

- inc/butterworth_filter.hpp
- src/butterworth_filter.cpp

5.2 Button Class Reference

```
#include <button.hpp>
```

Public Member Functions

- Button (GPIO_TypeDef *port, uint16_t pin, button_pull_resistor_t pull_resistor)

  Construct a new Button object.
- button_status_t get_status ()

  Provides the status of the chosen button.

5.2.1 Constructor & Destructor Documentation

5.2.1.1 Button()

```
Button::Button (
            GPIO_TypeDef * port,
            uint16_t pin,
            button_pull_resistor_t pull_resistor )
```

Construct a new Button object.

Parameters

port	pointer to the GPIO port
pin	number of the GPIO pin
pull_resistor	pull resistor configuration

5.2.2 Member Function Documentation

5.2.2.1 get_status()

```
button_status_t Button::get_status ( )
```

Provides the status of the chosen button.

Returns

Status of the button.

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

- inc/proxy/button.hpp
- src/proxy/button.cpp

5.3 HalAdc< number_of_channels, reading_per_channel > Class Template Reference

```
#include <hal_adc.hpp>
```

Public Member Functions

- HalAdc (ADC_HandleTypeDef *adc_handle)
  Construct a new Hal Adc object.
- void update_reading (void)
  Update the ADC reading.
- uint32_t get_adc_reading (uint8_t channel) const
  Get the reading of the ADC.

Static Public Member Functions

- static void set_reading_done (void)

  Set the reading done object.

5.3.1 Constructor & Destructor Documentation

5.3.1.1 HalAdc()

```
template<uint8_t number_of_channels, uint16_t reading_per_channel>
HalAdc< number_of_channels, reading_per_channel >::HalAdc (
            ADC_HandleTypeDef * adc_handle )
```

Construct a new Hal Adc object.

Parameters

adc_handle	pointer to the ADC handle

5.3.2 Member Function Documentation

5.3.2.1 get_adc_reading()

```
template<uint8_t number_of_channels, uint16_t reading_per_channel>
uint32_t HalAdc< number_of_channels, reading_per_channel >::get_adc_reading (
            uint8_t channel ) const
```

Get the reading of the ADC.

Parameters

channel	channel of the ADC

Returns

uint32_t reading of the ADC channel

5.3.2.2 set_reading_done()

```
template<uint8_t number_of_channels, uint16_t reading_per_channel>
void HalAdc< number_of_channels, reading_per_channel >::set_reading_done (
            void  )  [static]
```

Set the reading done object.

5.3.2.3 update_reading()

```
template<uint8_t number_of_channels, uint16_t reading_per_channel>
void HalAdc< number_of_channels, reading_per_channel >::update_reading (
            void  )
```

Update the ADC reading.

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

- inc/hal/hal_adc.hpp
- src/hal/hal_adc.cpp

5.4 HalGpio Class Reference

```
#include <hal_gpio.hpp>
```

Public Member Functions

- HalGpio (GPIO_TypeDef *gpio_port, uint16_t gpio_pin)
  Construct a new Hal GPIO object.
- bool read (void) const
  Read the GPIO pin.
- void write (GPIO_PinState pin_state)
  Write to the GPIO pin.
- void toggle (void)
  Toggle the GPIO pin.

5.4.1 Constructor & Destructor Documentation

5.4.1.1 HalGpio()

```
HalGpio::HalGpio (
            GPIO_TypeDef * gpio_port,
            uint16_t gpio_pin )
```

Construct a new Hal GPIO object.

Parameters

gpio_port	pointer to the GPIO port
gpio_pin	number of the GPIO pin

5.4.2 Member Function Documentation

5.4.2.1 read()

```
bool HalGpio::read (
            void  ) const
```

Read the GPIO pin.

Returns

true if the pin is high, false otherwise

5.4.2.2 toggle()

```
void HalGpio::toggle (
            void  )
```

Toggle the GPIO pin.

5.4.2.3 write()

```
void HalGpio::write (
            GPIO_PinState pin_state )
```

Write to the GPIO pin.

Parameters

pin_state	state of the GPIO pin

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

- inc/hal/hal_gpio.hpp
- src/hal/hal_gpio.cpp

5.5 HalPwm Class Reference

`#include <hal_pwm.hpp>`

Public Member Functions

- HalPwm (TIM_HandleTypeDef *timer_handle, uint32_t channel)

  Construct a new Hal Pwm object.
- void start (void)

  Start the PWM.
- void set_compare (uint32_t compare)

  Set the PWM duty cycle.

5.5.1 Constructor & Destructor Documentation

5.5.1.1 HalPwm()

```
HalPwm::HalPwm (
            TIM_HandleTypeDef * timer_handle,
            uint32_t channel )
```

Construct a new Hal Pwm object.

Parameters

timer_handle	Timer handle
channel	Timer channel

5.5.2 Member Function Documentation

5.5.2.1 set_compare()

```
void HalPwm::set_compare (
            uint32_t compare )
```

Set the PWM duty cycle.

Parameters

compare	value to set the duty cycle

5.5.2.2 start()

```
void HalPwm::start (
            void  )
```

Start the PWM.

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

- inc/hal/hal_pwm.hpp
- src/hal/hal_pwm.cpp

5.6 HalTimer Class Reference

```
#include <hal_timer.hpp>
```

Public Member Functions

- HalTimer ()
  Construct a new Hal Timer object.
- void reset (void)
  Reset the timer.
- float get_time (void) const
  Get elapsed time since last reset.

5.6.1 Constructor & Destructor Documentation

5.6.1.1 HalTimer()

```
HalTimer::HalTimer ( )
```

Construct a new Hal Timer object.

5.6.2 Member Function Documentation

5.6.2.1 get_time()

```
float HalTimer::get_time (
            void  ) const
```

Get elapsed time since last reset.

Returns

float elapsed time in seconds

5.6.2.2 reset()

```
void HalTimer::reset (
            void  )
```

Reset the timer.

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

- inc/hal/hal_timer.hpp
- src/hal/hal_timer.cpp

5.7 LineSensors< number_of_sensors, reading_per_sensor > Class Template Reference

```
#include <line_sensors.hpp>
```

Public Member Functions

- LineSensors (ADC_HandleTypeDef *adc_handle)
  Construct a new Line Sensors object.
- float get_position ()
  Gets the line position.
- void calibrate_white ()
  Calibrates the line sensors for the white line.
- void calibrate_black ()
  Calibrates the line sensors for the black background.

5.7.1 Constructor & Destructor Documentation

5.7.1.1 LineSensors()

```
template<uint8_t number_of_sensors, uint16_t reading_per_sensor>
LineSensors< number_of_sensors, reading_per_sensor >::LineSensors (
            ADC_HandleTypeDef * adc_handle )
```

Construct a new Line Sensors object.

Parameters

adc_handle	Handle to the ADC

5.7.2 Member Function Documentation

5.7.2.1 calibrate_black()

```
template<uint8_t number_of_sensors, uint16_t reading_per_sensor>
void LineSensors< number_of_sensors, reading_per_sensor >::calibrate_black
```

Calibrates the line sensors for the black background.

5.7.2.2 calibrate_white()

```
template<uint8_t number_of_sensors, uint16_t reading_per_sensor>
void LineSensors< number_of_sensors, reading_per_sensor >::calibrate_white
```

Calibrates the line sensors for the white line.

5.7.2.3 get_position()

```
template<uint8_t number_of_sensors, uint16_t reading_per_sensor>
float LineSensors< number_of_sensors, reading_per_sensor >::get_position
```

Gets the line position.

Returns

Position of the line.

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

- inc/proxy/line_sensors.hpp
- src/proxy/line_sensors.cpp

5.8 Locomotion Class Reference

```
#include <locomotion.hpp>
```

Public Member Functions

- Locomotion (TIM_HandleTypeDef *left_motor_timer_handle, TIM_HandleTypeDef *right_motor_timer_handle, uint32_t forward_timer_channel, uint32_t backward_timer_channel, float left_deadzone=0, float right_deadzone=0)

  Construct a new Locomotion object.
- void set_speeds (int16_t linear, int16_t angular)

  Set the speeds of the motors.

Static Public Member Functions

- static float linear_decay (float angular_error, float dependency)

  Compute the linear decay of the angular error.

5.8.1 Constructor & Destructor Documentation

5.8.1.1 Locomotion()

```
Locomotion::Locomotion (
            TIM_HandleTypeDef * left_motor_timer_handle,
            TIM_HandleTypeDef * right_motor_timer_handle,
            uint32_t forward_timer_channel,
            uint32_t backward_timer_channel,
            float left_deadzone = 0,
            float right_deadzone = 0 )
```

Construct a new Locomotion object.

Parameters

left_motor_timer_handle	pointer to the left motor timer handle
right_motor_timer_handle	pointer to the right motor timer handle
forward_timer_channel	channel of the forward pwm
backward_timer_channel	channel of the backward pwm
left_deadzone	deadzone of the left motor
right_deadzone	deadzone of the right motor

5.8.2 Member Function Documentation

5.8.2.1 linear_decay()

```
float Locomotion::linear_decay (
            float angular_error,
            float dependency )  [static]
```

Compute the linear decay of the angular error.

Parameters

angular_error	Angular error
dependency	Dependency of the linear decay

Returns

float Linear decay

5.8.2.2 set_speeds()

```
void Locomotion::set_speeds (
            int16_t linear,
            int16_t angular )
```

Set the speeds of the motors.

Parameters

linear	Linear speed
angular	Angular speed

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

- inc/proxy/locomotion.hpp
- src/proxy/locomotion.cpp

5.9 Motor Class Reference

```
#include <motor.hpp>
```

Public Member Functions

- Motor (TIM_HandleTypeDef *htim, uint32_t forward_timer_channel, uint32_t backward_timer_channel, float deadzone=0)

  Construct a new Motor object.
- void set_speed (int16_t speed)

  Set the speed object.

5.9.1 Constructor & Destructor Documentation

5.9.1.1 Motor()

```
Motor::Motor (
            TIM_HandleTypeDef * htim,
            uint32_t forward_timer_channel,
            uint32_t backward_timer_channel,
            float deadzone = 0 )
```

Construct a new Motor object.

Parameters

htim	pointer to the timer handle
forward_timer_channel	channel of the forward pwm
backward_timer_channel	channel of the backward pwm
deadzone	minimum value of the pwm to start the motor

5.9.2 Member Function Documentation

5.9.2.1 set_speed()

```
void Motor::set_speed (
            int16_t speed )
```

Set the speed object.

Parameters

speed	speed of the motor

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

- inc/proxy/motor.hpp
- src/proxy/motor.cpp

5.10 PidController Class Reference

Implementation of simple PID controller Response = Kp(error + Ki ∗ integral(error) Kd ∗ d/dt(error))

```
#include <pid_controller.hpp>
```

Public Member Functions

- PidController (float kp, float ki, float kd, float setpoint=0.0, float saturation=-1.0, float max_integral=-1.0)

  Construct a new Pid Controller object.
- void set_setpoint (float setpoint)

  Set the setpoint object.
- void set_parameters (float kp, float ki, float kd, float saturation=-1.0, float max_integral=-1.0)

  Set the controller parameters.
- void reset ()

  Reset prev_error and error_acc objects.
- float update (float state)

  Update PID with new state and return response.
- float update (float state, float state_change)

  Update PID with new state and return response.

5.10.1 Detailed Description

Implementation of simple PID controller Response = Kp(error + Ki ∗ integral(error) Kd ∗ d/dt(error))

5.10.2 Constructor & Destructor Documentation

5.10.2.1 PidController()

```
PidController::PidController (
            float kp,
            float ki,
            float kd,
            float setpoint = 0.0,
            float saturation = -1.0,
            float max_integral = -1.0 )
```

Construct a new Pid Controller object.

Parameters

kp	Proportinal constant
ki	Integrative constant
kd	Derivative constant
setpoint	Desired state
saturation	Maximum response returned by the controller
max_integral	Maximum integrative response

5.10.3 Member Function Documentation

5.10.3.1 reset()

```
void PidController::reset (
            void  )
```

Reset prev_error and error_acc objects.

5.10.3.2 set_parameters()

```
void PidController::set_parameters (
            float kp,
            float ki,
            float kd,
            float saturation = -1.0,
            float max_integral = -1.0 )
```

Set the controller parameters.

Parameters

kp	Proportional constant
ki	Integrative constant
kd	Derivative constant
saturation	Maximum response returned by the controller
max_integral	Maximum integrative response

5.10.3.3 set_setpoint()

```
void PidController::set_setpoint (
            float setpoint )
```

Set the setpoint object.

Parameters

setpoint	Desired state

5.10.3.4 update() [1/2]

```
float PidController::update (
            float state )
```

Update PID with new state and return response.

Parameters

state	Current value of the controlled variable

Returns

Response

5.10.3.5 update() [2/2]

```
float PidController::update (
            float state,
            float state_change )
```

Update PID with new state and return response.

Parameters

state	Current value of the controlled variable
state_change	Derivative of the controlled variable

Returns

Response

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

- inc/pid_controller.hpp
- src/pid_controller.cpp

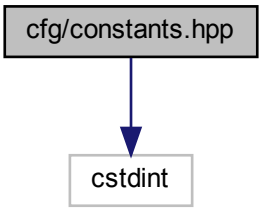
Chapter 6

File Documentation

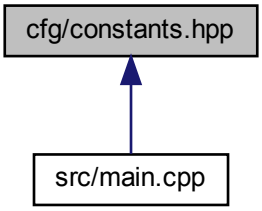
6.1 cfg/constants.hpp File Reference

```
#include <cstdint>
```

Include dependency graph for constants.hpp:



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



Variables

- constexpr float left_deadzone = 0.10
- constexpr float right_deadzone = 0.10
- constexpr float kp = 15.0
- constexpr float ki = 0.0
- constexpr float kd = 0.0
- constexpr float saturation = 100.0
- constexpr float max_integral = 40.0
- constexpr float filter_frequency = 0.5
- constexpr uint16_t linear_base_speed = 20
- constexpr float linear_decay = 15.0

6.1.1 Variable Documentation

6.1.1.1 filter_frequency

```
constexpr float filter_frequency = 0.5  [constexpr]
```

6.1.1.2 kd

```
constexpr float kd = 0.0  [constexpr]
```

6.1.1.3 ki

```
constexpr float ki = 0.0  [constexpr]
```

6.1.1.4 kp

```
constexpr float kp = 15.0  [constexpr]
```

6.1.1.5 left_deadzone

```
constexpr float left_deadzone = 0.10  [constexpr]
```

6.1.1.6 linear_base_speed

```
constexpr uint16_t linear_base_speed = 20  [constexpr]
```

6.1.1.7 linear_decay

```
constexpr float linear_decay = 15.0  [constexpr]
```

6.1.1.8 max_integral

```
constexpr float max_integral = 40.0  [constexpr]
```

6.1.1.9 right_deadzone

```
constexpr float right_deadzone = 0.10  [constexpr]
```

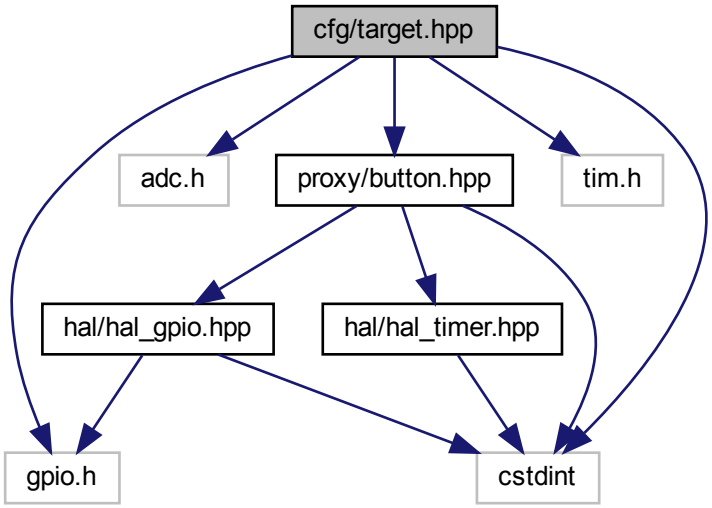
6.1.1.10 saturation

```
constexpr float saturation = 100.0  [constexpr]
```

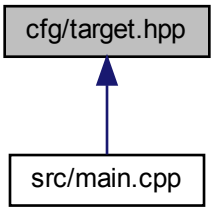
6.2 cfg/target.hpp File Reference

```
#include <cstdint>
#include "adc.h"
#include "gpio.h"
#include "tim.h"
```

```
#include "proxy/button.hpp"
```
Include dependency graph for target.hpp:



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



Variables

- GPIO_TypeDef * button_gpio_port = GPIOB
- constexpr uint16_t button_pin = GPIO_PIN_10
- const button_pull_resistor_t button_pull_resistor = BUTTON_PULL_UP
- GPIO_TypeDef * led_gpio_port = GPIOB
- constexpr uint16_t led_pin = GPIO_PIN_15
- TIM_HandleTypeDef * left_motor_timer_handle = &htim2
- TIM_HandleTypeDef * right_motor_timer_handle = &htim1
- constexpr uint16_t motor_forward_timer_channel = TIM_CHANNEL_1
- constexpr uint16_t motor_backward_timer_channel = TIM_CHANNEL_2
- ADC_HandleTypeDef * line_sensor_adc_handle = &hadc1
- constexpr uint8_t adc_num_channels = 8
- constexpr uint16_t adc_readings_per_channel = 50

6.2.1 Variable Documentation

6.2.1.1 adc_num_channels

```
constexpr uint8_t adc_num_channels = 8  [constexpr]
```

6.2.1.2 adc_readings_per_channel

```
constexpr uint16_t adc_readings_per_channel = 50  [constexpr]
```

6.2.1.3 button_gpio_port

```
GPIO_TypeDef* button_gpio_port = GPIOB
```

6.2.1.4 button_pin

```
constexpr uint16_t button_pin = GPIO_PIN_10  [constexpr]
```

6.2.1.5 button_pull_resistor

```
const button_pull_resistor_t button_pull_resistor = BUTTON_PULL_UP
```

6.2.1.6 led_gpio_port

```
GPIO_TypeDef* led_gpio_port = GPIOB
```

6.2.1.7 led_pin

```
constexpr uint16_t led_pin = GPIO_PIN_15  [constexpr]
```

6.2.1.8 left_motor_timer_handle

```
TIM_HandleTypeDef* left_motor_timer_handle = &htim2
```

6.2.1.9 line_sensor_adc_handle

```
ADC_HandleTypeDef* line_sensor_adc_handle = &hadc1
```

6.2.1.10 motor_backward_timer_channel

```
constexpr uint16_t motor_backward_timer_channel = TIM_CHANNEL_2  [constexpr]
```

6.2.1.11 motor_forward_timer_channel

```
constexpr uint16_t motor_forward_timer_channel = TIM_CHANNEL_1  [constexpr]
```

6.2.1.12 right_motor_timer_handle

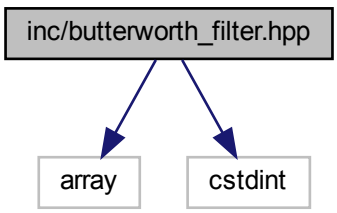
```
TIM_HandleTypeDef* right_motor_timer_handle = &htim1
```

6.3 doc/report.md File Reference

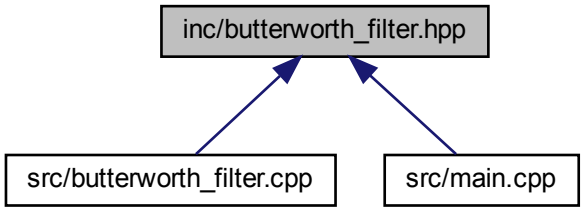
6.4 inc/butterworth_filter.hpp File Reference

```
#include <array>
#include <cstdint>
```

Include dependency graph for butterworth_filter.hpp:



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



Classes

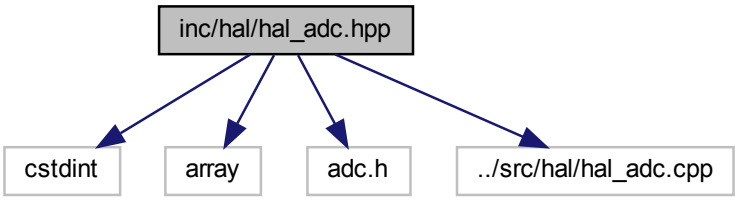
- class ButterworthFilter

  Implementation of Butterworth second order low-pass filter A generic digital filter follows the relation a0 ∗ y[k] = sum(bi ∗ x[k - i]) - sum(aj ∗ y[k - j]) Where x[k] - measurement at instant k y[k] - filtered signal at instant k The Butterworth filter have the special property of being a maximally flat magnitude filter, in other words, is the best filter that doesn't present distortions around the cutoff frequency The formula for the continuos coefficients of the Butterworth filter is available here: https://en.wikipedia.org/wiki/Butterworth_filter The discrete version were computed with the Tustin method: https://en.wikipedia.org/wiki/Bilinear_transform.

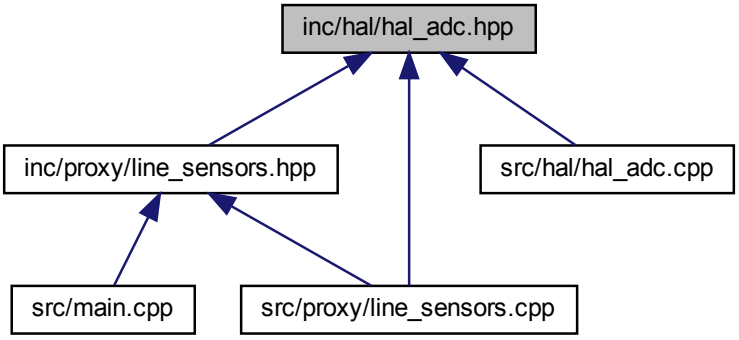
6.5 inc/hal/hal_adc.hpp File Reference

```
#include <cstdint>
#include <array>
#include "adc.h"
#include "../src/hal/hal_adc.cpp"
```

Include dependency graph for hal_adc.hpp:



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



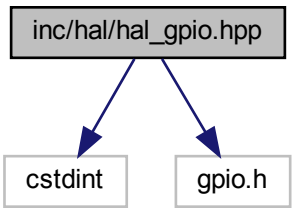
Classes

- class HalAdc< number_of_channels, reading_per_channel >

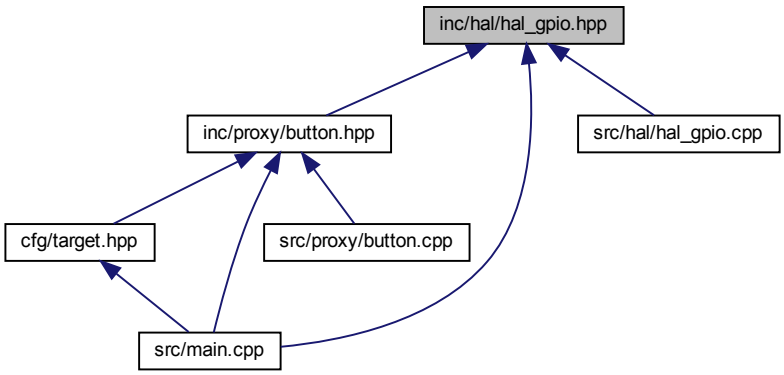
6.6 inc/hal/hal_gpio.hpp File Reference

```
#include <cstdint>
#include "gpio.h"
```

Include dependency graph for hal_gpio.hpp:



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



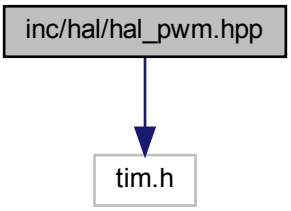
Classes

- class HalGpio

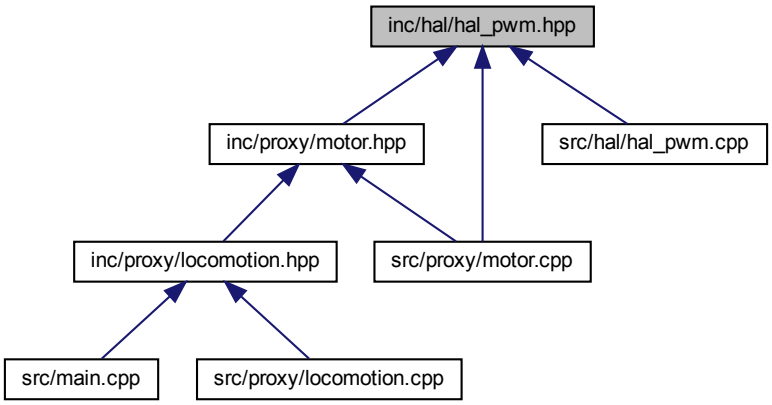
6.7 inc/hal/hal_pwm.hpp File Reference

```
#include "tim.h"
```

Include dependency graph for hal_pwm.hpp:



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

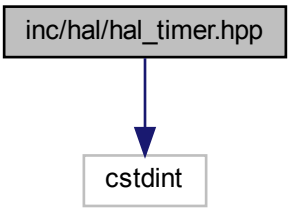


Classes

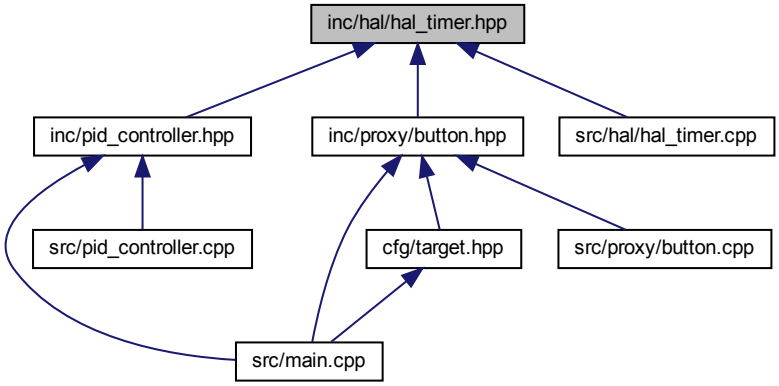
- class HalPwm

6.8 inc/hal/hal_timer.hpp File Reference

`#include <cstdint>`
Include dependency graph for hal_timer.hpp:



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

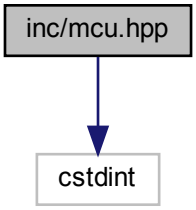


Classes

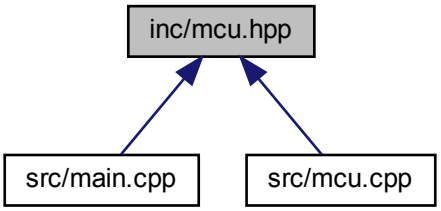
- class HalTimer

6.9 inc/mcu.hpp File Reference

`#include <cstdint>`
Include dependency graph for mcu.hpp:



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



Functions

- void mcu_init (void)

  Initializes MCU and some peripherals.
- void SystemClock_Config (void)

  Initializes System Clock.
- void mcu_sleep (uint32_t ms)

  Put the MCU to sleep.

6.9.1 Function Documentation

6.9.1.1 mcu_init()

```
void mcu_init (
            void  )
```

Initializes MCU and some peripherals.

6.9.1.2 mcu_sleep()

```
void mcu_sleep (
            uint32_t ms )
```

Put the MCU to sleep.

Parameters

ms	Sleep time in milliseconds

6.9.1.3 SystemClock_Config()

```
void SystemClock_Config (
            void  )
```

Initializes System Clock.

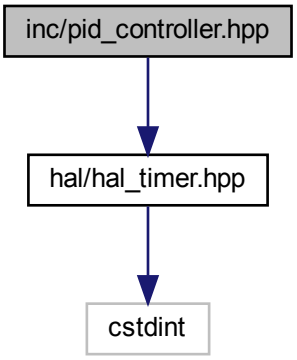
Note

Defined by cube.

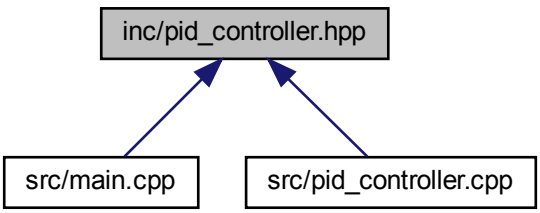
6.10 inc/pid_controller.hpp File Reference

```
#include <hal/hal_timer.hpp>
```

Include dependency graph for pid_controller.hpp:



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



Classes

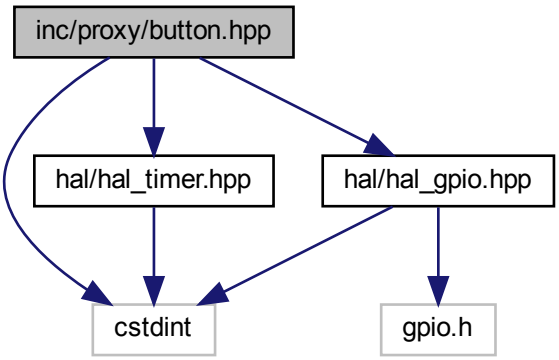
- class PidController

  Implementation of simple PID controller Response = Kp(error + Ki ∗ integral(error) Kd ∗ d/dt(error))

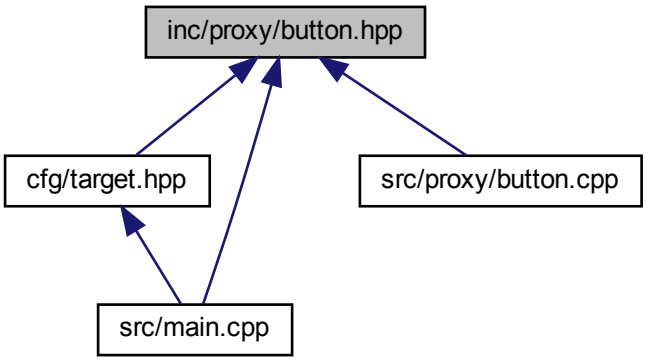
6.11 inc/proxy/button.hpp File Reference

```
#include <cstdint>
#include "hal/hal_gpio.hpp"
#include "hal/hal_timer.hpp"
```

Include dependency graph for button.hpp:



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



Classes

- class Button

Enumerations

- enum button_status_t { BUTTON_NO_PRESS , BUTTON_SHORT_PRESS , BUTTON_LONG_PRESS , BUTTON_EXTRA_LONG_PRESS }

  Button status type.
- enum button_pull_resistor_t { BUTTON_PULL_UP , BUTTON_PULL_DOWN }

6.11.1 Enumeration Type Documentation

6.11.1.1 button_pull_resistor_t

```
enum button_pull_resistor_t
```

Enumerator

BUTTON_PULL_UP	
BUTTON_PULL_DOWN	

6.11.1.2 button_status_t

```
enum button_status_t
```

Button status type.

Enumerator

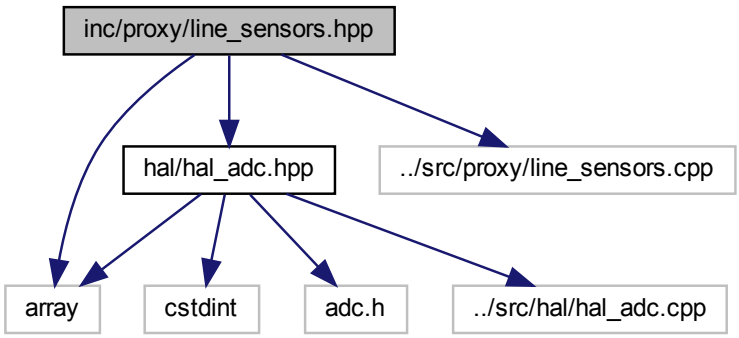
BUTTON_NO_PRESS	
BUTTON_SHORT_PRESS	
BUTTON_LONG_PRESS	
BUTTON_EXTRA_LONG_PRESS	

6.12 inc/proxy/line_sensors.hpp File Reference

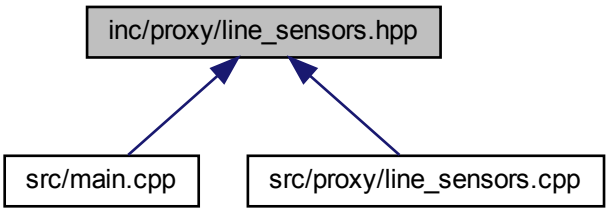
```
#include <array>
#include "hal/hal_adc.hpp"
```

```
#include "../src/proxy/line_sensors.cpp"
```

Include dependency graph for line_sensors.hpp:



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



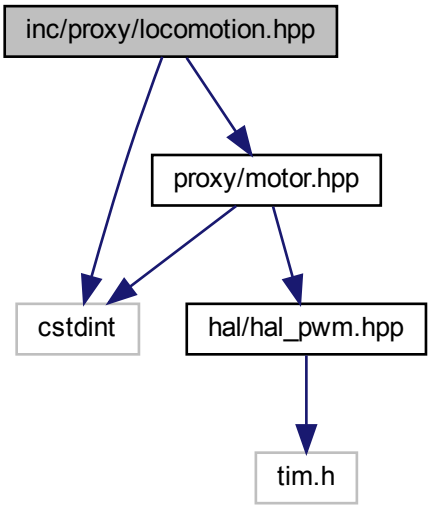
Classes

- class LineSensors< number_of_sensors, reading_per_sensor >

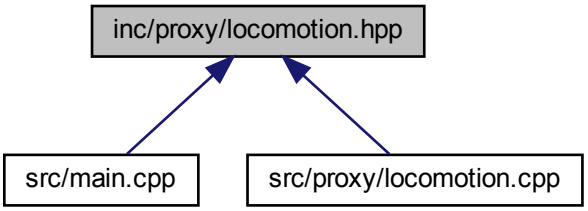
6.13 inc/proxy/locomotion.hpp File Reference

```
#include <cstdint>
#include "proxy/motor.hpp"
```

Include dependency graph for locomotion.hpp:



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



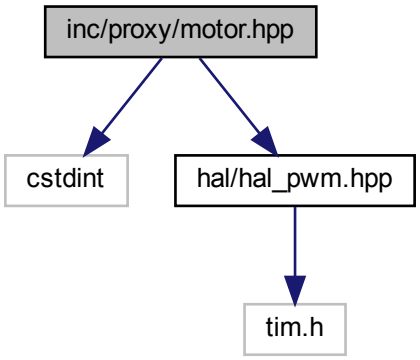
Classes

- class Locomotion

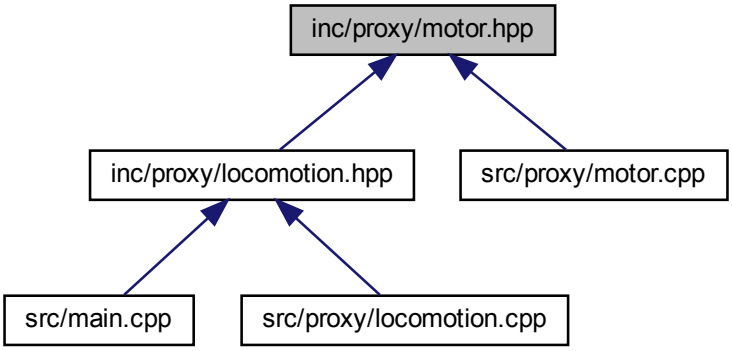
6.14 inc/proxy/motor.hpp File Reference

```
#include <cstdint>
#include "hal/hal_pwm.hpp"
```

Include dependency graph for motor.hpp:



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



Classes

- class Motor

Variables

- constexpr int16_t max_motors_speed = 100
- constexpr int16_t min_motors_speed = -max_motors_speed

6.14.1 Variable Documentation

6.14.1.1 max_motors_speed

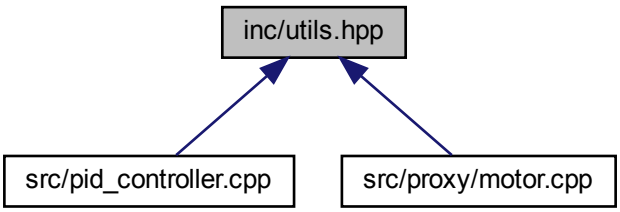
```
constexpr int16_t max_motors_speed = 100  [constexpr]
```

6.14.1.2 min_motors_speed

```
constexpr int16_t min_motors_speed = -max_motors_speed  [constexpr]
```

6.15 inc/utils.hpp File Reference

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



Macros

- #define constrain(value, min, max) (value < min ? min : (value > max ? max : value))
- #define map(value, from_min, from_max, to_min, to_max) (to_min + (to_max - to_min) * (value - from_min) / (from_max - from_min))

6.15.1 Macro Definition Documentation

6.15.1.1 constrain

```
#define constrain(
              value,
              min,
              max ) (value < min ?  min :  (value > max ?  max :  value))
```

6.15.1.2 map

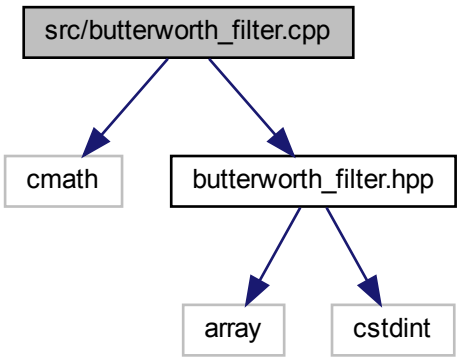
```
#define map(
              value,
              from_min,
              from_max,
              to_min,
              to_max )  (to_min + (to_max - to_min) * (value - from_min) / (from_max - from_↩
min))
```

6.16 README.md File Reference

6.17 src/butterworth_filter.cpp File Reference

```
#include <cmath>
#include "butterworth_filter.hpp"
```

Include dependency graph for butterworth_filter.cpp:

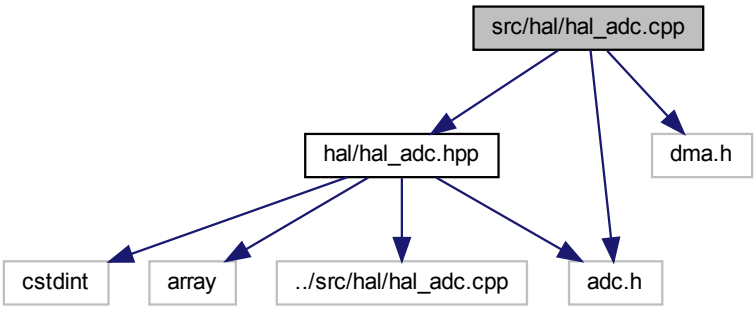


6.18 src/hal/hal_adc.cpp File Reference

```
#include "hal/hal_adc.hpp"
#include "adc.h"
```

```
#include "dma.h"
```

Include dependency graph for hal_adc.cpp:



Macros

- #define __HAL_ADC_CPP__

6.18.1 Macro Definition Documentation

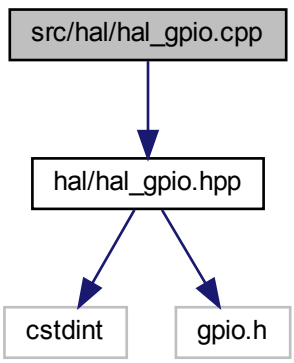
6.18.1.1 __HAL_ADC_CPP__

```
#define __HAL_ADC_CPP__
```

6.19 src/hal/hal_gpio.cpp File Reference

```
#include "hal/hal_gpio.hpp"
```

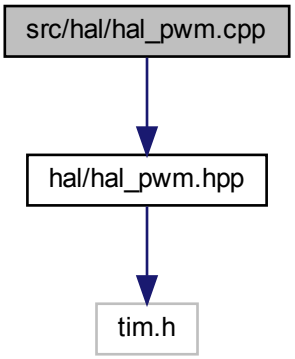
Include dependency graph for hal_gpio.cpp:



6.20 src/hal/hal_pwm.cpp File Reference

```
#include "hal/hal_pwm.hpp"
```

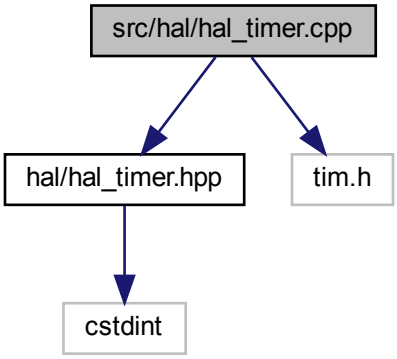
Include dependency graph for hal_pwm.cpp:



6.21 src/hal/hal_timer.cpp File Reference

```
#include "hal/hal_timer.hpp"
#include "tim.h"
```

Include dependency graph for hal_timer.cpp:

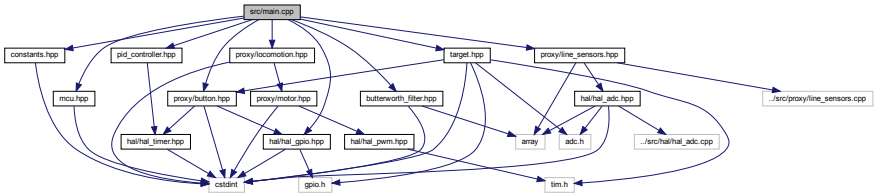


6.22 src/main.cpp File Reference

```
#include "mcu.hpp"
#include "constants.hpp"
```

```
#include "target.hpp"
#include "butterworth_filter.hpp"
#include "pid_controller.hpp"
#include "hal/hal_gpio.hpp"
#include "proxy/button.hpp"
#include "proxy/line_sensors.hpp"
#include "proxy/locomotion.hpp"
```

Include dependency graph for main.cpp:



Functions

- int main (void)
- void HAL_ADC_ConvCpltCallback (ADC_HandleTypeDef *hadc)

6.22.1 Function Documentation

6.22.1.1 HAL_ADC_ConvCpltCallback()

```
void HAL_ADC_ConvCpltCallback (
            ADC_HandleTypeDef * hadc )
```

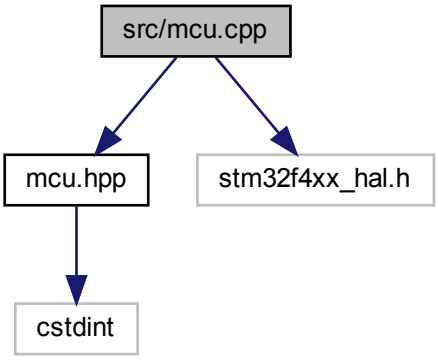
6.22.1.2 main()

```
int main (
            void  )
```

6.23 src/mcu.cpp File Reference

```
#include "mcu.hpp"
#include "stm32f4xx_hal.h"
```

Include dependency graph for mcu.cpp:



Functions

- void mcu_init (void)

  Initializes MCU and some peripherals.
- void mcu_sleep (uint32_t ms)

  Put the MCU to sleep.

6.23.1 Function Documentation

6.23.1.1 mcu_init()

```
void mcu_init (
            void  )
```

Initializes MCU and some peripherals.

6.23.1.2 mcu_sleep()

```
void mcu_sleep (
            uint32_t ms )
```

Put the MCU to sleep.

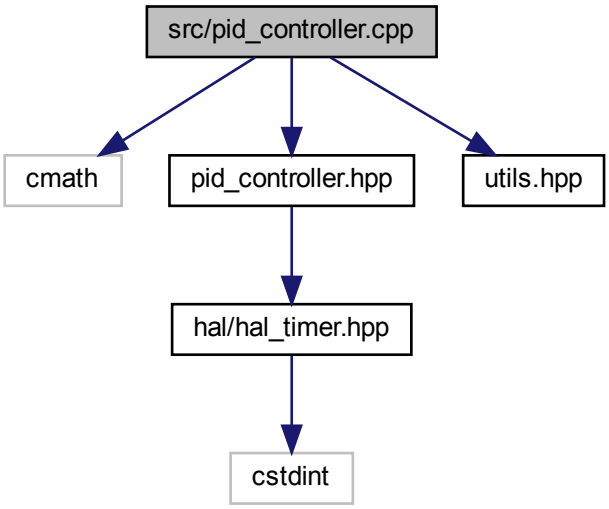
Parameters

ms	Sleep time in milliseconds

6.24 src/pid_controller.cpp File Reference

```
#include <cmath>
#include "pid_controller.hpp"
#include "utils.hpp"
```

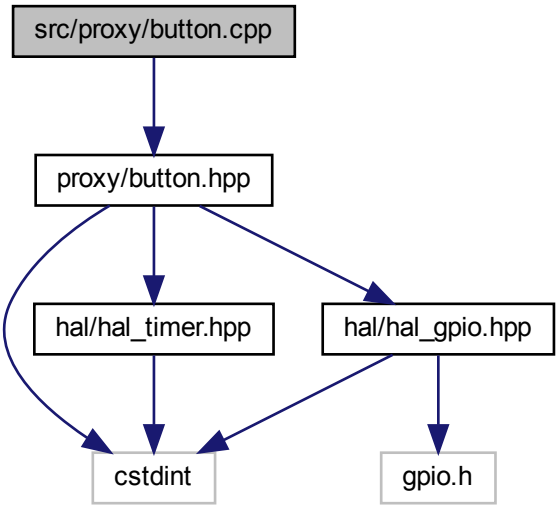
Include dependency graph for pid_controller.cpp:



6.25 src/proxy/button.cpp File Reference

```
#include "proxy/button.hpp"
```

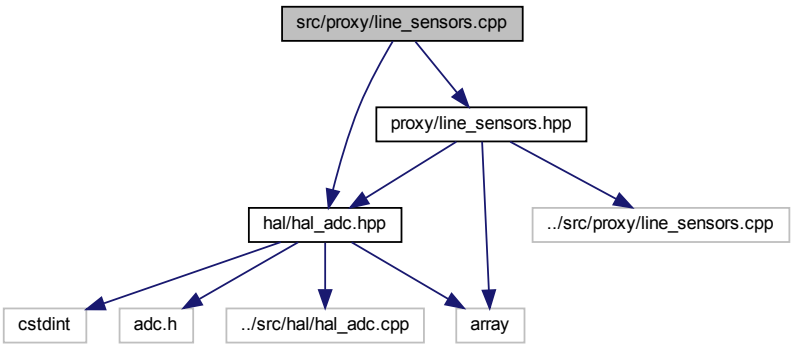
Include dependency graph for button.cpp:



6.26 src/proxy/line_sensors.cpp File Reference

```
#include "hal/hal_adc.hpp"
#include "proxy/line_sensors.hpp"
```

Include dependency graph for line_sensors.cpp:



Macros

- #define __LINE_SENSORS_CPP__

Variables

- constexpr uint32_t default_white_value = 4000
- constexpr uint32_t default_black_value = 3850

6.26.1 Macro Definition Documentation

6.26.1.1 __LINE_SENSORS_CPP__

```
#define __LINE_SENSORS_CPP__
```

6.26.2 Variable Documentation

6.26.2.1 default_black_value

```
constexpr uint32_t default_black_value = 3850  [constexpr]
```

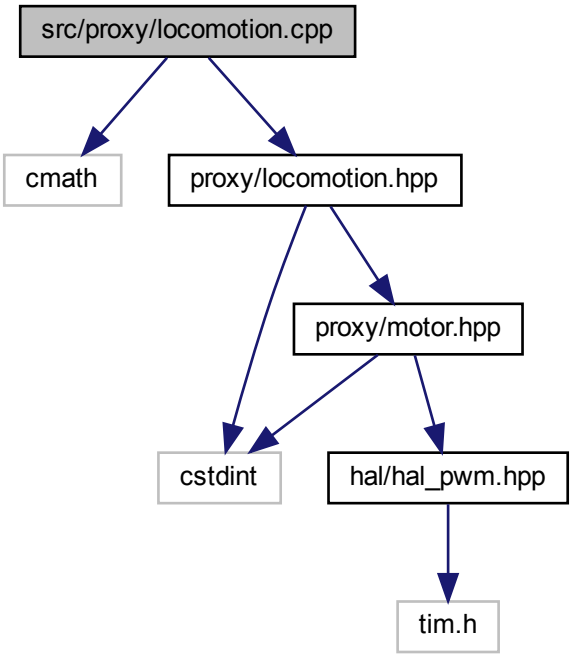
6.26.2.2 default_white_value

```
constexpr uint32_t default_white_value = 4000  [constexpr]
```

6.27 src/proxy/locomotion.cpp File Reference

```
#include <cmath>
#include "proxy/locomotion.hpp"
```

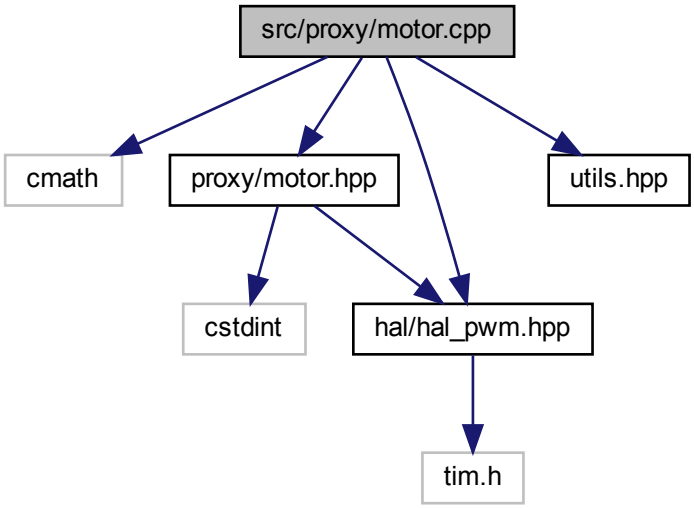
Include dependency graph for locomotion.cpp:



6.28 src/proxy/motor.cpp File Reference

```
#include <cmath>
#include "proxy/motor.hpp"
#include "hal/hal_pwm.hpp"
#include "utils.hpp"
```

Include dependency graph for motor.cpp:



Index