

## Prova Prática - Laboratório de Processadores

Generated by Doxygen 1.9.1



<b>1 report</b>	<b>1</b>
<b>2 STM32 Project Template</b>	<b>3</b>
2.1 Requisitos	3
2.2 Preparando	4
2.2.1 Projeto	4
2.2.2 Compilação	4
2.3 Gerando arquivos	5
2.4 Compilando	5
2.5 Limpando Arquivos compilados	5
2.6 Recompilando	5
2.7 Gravando	6
2.8 Formatando	6
2.9 Submódulos	6
2.9.1 Adicionando um submódulo	6
2.9.2 Inicializando um submódulo já existente	6
2.10 Diretório de testes	6
2.11 Debug	6
<b>3 Class Index</b>	<b>7</b>
3.1 Class List	7
<b>4 File Index</b>	<b>9</b>
4.1 File List	9
<b>5 Class Documentation</b>	<b>11</b>
5.1 ButterworthFilter Class Reference	11
5.1.1 Detailed Description	11
5.1.2 Constructor & Destructor Documentation	11
5.1.2.1 ButterworthFilter()	11
5.1.3 Member Function Documentation	12
5.1.3.1 update()	12
5.2 Button Class Reference	12
5.2.1 Constructor & Destructor Documentation	12
5.2.1.1 Button()	13
5.2.2 Member Function Documentation	13
5.2.2.1 get_status()	13
5.3 HalAdc< number_of_channels, reading_per_channel > Class Template Reference	13
5.3.1 Constructor & Destructor Documentation	14
5.3.1.1 HalAdc()	14
5.3.2 Member Function Documentation	14
5.3.2.1 get_adc_reading()	14
5.3.2.2 set_reading_done()	14
5.3.2.3 update_reading()	15

5.4 HalGpio Class Reference	15
5.4.1 Constructor & Destructor Documentation	15
5.4.1.1 HalGpio()	15
5.4.2 Member Function Documentation	16
5.4.2.1 read()	16
5.4.2.2 toggle()	16
5.4.2.3 write()	16
5.5 HalPwm Class Reference	17
5.5.1 Constructor & Destructor Documentation	17
5.5.1.1 HalPwm()	17
5.5.2 Member Function Documentation	17
5.5.2.1 set_compare()	17
5.5.2.2 start()	18
5.6 HalTimer Class Reference	18
5.6.1 Constructor & Destructor Documentation	18
5.6.1.1 HalTimer()	18
5.6.2 Member Function Documentation	18
5.6.2.1 get_time()	19
5.6.2.2 reset()	19
5.7 LineSensors< number_of_sensors, reading_per_sensor > Class Template Reference	19
5.7.1 Constructor & Destructor Documentation	19
5.7.1.1 LineSensors()	19
5.7.2 Member Function Documentation	20
5.7.2.1 calibrate_black()	20
5.7.2.2 calibrate_white()	20
5.7.2.3 get_position()	20
5.8 Locomotion Class Reference	20
5.8.1 Constructor & Destructor Documentation	21
5.8.1.1 Locomotion()	21
5.8.2 Member Function Documentation	21
5.8.2.1 linear_decay()	21
5.8.2.2 set_speeds()	23
5.9 Motor Class Reference	23
5.9.1 Constructor & Destructor Documentation	23
5.9.1.1 Motor()	24
5.9.2 Member Function Documentation	24
5.9.2.1 set_speed()	24
5.10 PidController Class Reference	24
5.10.1 Detailed Description	25
5.10.2 Constructor & Destructor Documentation	25
5.10.2.1 PidController()	25
5.10.3 Member Function Documentation	25

5.10.3.1 reset()	26
5.10.3.2 set_parameters()	26
5.10.3.3 set_setpoint()	26
5.10.3.4 update() [1/2]	26
5.10.3.5 update() [2/2]	27
<b>6 File Documentation</b>	<b>29</b>
6.1 cfg/constants.hpp File Reference	29
6.1.1 Variable Documentation	30
6.1.1.1 filter_frequency	30
6.1.1.2 kd	30
6.1.1.3 ki	30
6.1.1.4 kp	30
6.1.1.5 left_deadzone	30
6.1.1.6 linear_base_speed	31
6.1.1.7 linear_decay	31
6.1.1.8 max_integral	31
6.1.1.9 right_deadzone	31
6.1.1.10 saturation	31
6.2 cfg/target.hpp File Reference	31
6.2.1 Variable Documentation	33
6.2.1.1 adc_num_channels	33
6.2.1.2 adc_readings_per_channel	33
6.2.1.3 button_gpio_port	33
6.2.1.4 button_pin	33
6.2.1.5 button_pull_resistor	33
6.2.1.6 led_gpio_port	33
6.2.1.7 led_pin	33
6.2.1.8 left_motor_timer_handle	34
6.2.1.9 line_sensor_adc_handle	34
6.2.1.10 motor_backward_timer_channel	34
6.2.1.11 motor_forward_timer_channel	34
6.2.1.12 right_motor_timer_handle	34
6.3 doc/report.md File Reference	34
6.4 inc/butterworth_filter.hpp File Reference	34
6.5 inc/hal/hal_adc.hpp File Reference	35
6.6 inc/hal/hal_gpio.hpp File Reference	36
6.7 inc/hal/hal_pwm.hpp File Reference	37
6.8 inc/hal/hal_timer.hpp File Reference	38
6.9 inc/mcu.hpp File Reference	39
6.9.1 Function Documentation	40
6.9.1.1 mcu_init()	40

6.9.1.2 mcu_sleep()	40
6.9.1.3 SystemClock_Config()	41
6.10 inc/pid_controller.hpp File Reference	41
6.11 inc/proxy/button.hpp File Reference	42
6.11.1 Enumeration Type Documentation	43
6.11.1.1 button_pull_resistor_t	43
6.11.1.2 button_status_t	43
6.12 inc/proxy/line_sensors.hpp File Reference	43
6.13 inc/proxy/locomotion.hpp File Reference	44
6.14 inc/proxy/motor.hpp File Reference	45
6.14.1 Variable Documentation	46
6.14.1.1 max_motors_speed	47
6.14.1.2 min_motors_speed	47
6.15 inc/utils.hpp File Reference	47
6.15.1 Macro Definition Documentation	47
6.15.1.1 constrain	47
6.15.1.2 map	48
6.16 README.md File Reference	48
6.17 src/butterworth_filter.cpp File Reference	48
6.18 src/hal/hal_adc.cpp File Reference	48
6.18.1 Macro Definition Documentation	49
6.18.1.1 __HAL_ADC_CPP__	49
6.19 src/hal/hal_gpio.cpp File Reference	49
6.20 src/hal/hal_pwm.cpp File Reference	50
6.21 src/hal/hal_timer.cpp File Reference	50
6.22 src/main.cpp File Reference	50
6.22.1 Function Documentation	51
6.22.1.1 HAL_ADC_ConvCpltCallback()	51
6.22.1.2 main()	51
6.23 src/mcu.cpp File Reference	52
6.23.1 Function Documentation	52
6.23.1.1 mcu_init()	52
6.23.1.2 mcu_sleep()	52
6.24 src/pid_controller.cpp File Reference	53
6.25 src/proxy/button.cpp File Reference	53
6.26 src/proxy/line_sensors.cpp File Reference	54
6.26.1 Macro Definition Documentation	55
6.26.1.1 __LINE_SENSORS_CPP__	55
6.26.2 Variable Documentation	55
6.26.2.1 default_black_value	55
6.26.2.2 default_white_value	55
6.27 src/proxy/locomotion.cpp File Reference	55

---

6.28 src/proxy/motor.cpp File Reference . . . . .	56
<b>Index</b>	<b>59</b>





# Chapter 1

## report



## Chapter 2

# STM32 Project Template

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

### 2.1 Requisitos

- [STM32CubeMX](#)

É necessário colocar o local de instalação na variável 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:

#### ButterworthFilter

Implementation of Butterworth second order low-pass filter A generic digital filter follows the relation  $a_0 * y[k] = \sum(b_i * x[k - i]) - \sum(a_j * y[k - j])$  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 continuous coefficients of the Butterworth filter is available here: [https://en.wikipedia.org/wiki/Butterworth\\_filter](https://en.wikipedia.org/wiki/Butterworth_filter) The discrete version were computed with the Tustin method: [https://en.wikipedia.org/wiki/Bilinear\\_transform](https://en.wikipedia.org/wiki/Bilinear_transform) . . . . . 11

Button . . . . . 12

HalAdc< number\_of\_channels, reading\_per\_channel > . . . . . 13

HalGpio . . . . . 15

HalPwm . . . . . 17

HalTimer . . . . . 18

LineSensors< number\_of\_sensors, reading\_per\_sensor > . . . . . 19

Locomotion . . . . . 20

Motor . . . . . 23

#### PidController

Implementation of simple PID controller Response =  $K_p(\text{error} + K_i * \text{integral}(\text{error}) K_d * \frac{d}{dt}(\text{error}))$  . . . . . 24





## Chapter 4

# File Index

### 4.1 File List

Here is a list of all files with brief descriptions:

cfg/	constants.hpp	29
cfg/	target.hpp	31
inc/	butterworth_filter.hpp	34
inc/	mcu.hpp	39
inc/	pid_controller.hpp	41
inc/	utils.hpp	47
inc/hal/	hal_adc.hpp	35
inc/hal/	hal_gpio.hpp	36
inc/hal/	hal_pwm.hpp	37
inc/hal/	hal_timer.hpp	38
inc/proxy/	button.hpp	42
inc/proxy/	line_sensors.hpp	43
inc/proxy/	locomotion.hpp	44
inc/proxy/	motor.hpp	45
src/	butterworth_filter.cpp	48
src/	main.cpp	50
src/	mcu.cpp	52
src/	pid_controller.cpp	53
src/hal/	hal_adc.cpp	48
src/hal/	hal_gpio.cpp	49
src/hal/	hal_pwm.cpp	50
src/hal/	hal_timer.cpp	50
src/proxy/	button.cpp	53
src/proxy/	line_sensors.cpp	54
src/proxy/	locomotion.cpp	55
src/proxy/	motor.cpp	56



## Chapter 5

# Class Documentation

### 5.1 ButterworthFilter Class Reference

Implementation of Butterworth second order low-pass filter A generic digital filter follows the relation  $a_0 * y[k] = \sum(b_i * x[k - i]) - \sum(a_j * y[k - j])$  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 continuous coefficients of the Butterworth filter is available here: [https://en.wikipedia.org/wiki/Butterworth\\_filter](https://en.wikipedia.org/wiki/Butterworth_filter) The discrete version were computed with the Tustin method: [https://en.wikipedia.org/wiki/Bilinear\\_transform](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  $a_0 * y[k] = \sum(b_i * x[k - i]) - \sum(a_j * y[k - j])$  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 continuous coefficients of the Butterworth filter is available here: [https://en.wikipedia.org/wiki/Butterworth\\_filter](https://en.wikipedia.org/wiki/Butterworth_filter) The discrete version were computed with the Tustin method: [https://en.wikipedia.org/wiki/Bilinear\\_transform](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**

<i>cutoff_frequency</i>	Low-pass cutoff frequency in Hz
<i>sampling_frequency</i>	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**

<i>x0</i>	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

<i>port</i>	pointer to the GPIO port
<i>pin</i>	number of the GPIO pin
<i>pull_resistor</i>	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

<i>adc_handle</i>	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

<i>channel</i>	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**

<i>gpio_port</i>	pointer to the GPIO port
<i>gpio_pin</i>	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**

<i>pin_state</i>	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

<i>timer_handle</i>	Timer handle
<i>channel</i>	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

<i>compare</i>	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**

<i>adc_handle</i>	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

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

<i>angular_error</i>	Angular error
<i>dependency</i>	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

<i>linear</i>	Linear speed
<i>angular</i>	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

<i>htim</i>	pointer to the timer handle
<i>forward_timer_channel</i>	channel of the forward pwm
<i>backward_timer_channel</i>	channel of the backward pwm
<i>deadzone</i>	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

<i>speed</i>	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 =  $K_p(\text{error} + K_i * \text{integral}(\text{error}) + K_d * \text{d/dt}(\text{error}))$

```
#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  $\text{Response} = K_p(\text{error} + K_i * \text{integral}(\text{error}) + K_d * d/dt(\text{error}))$

### 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

<i>kp</i>	Proportional constant
<i>ki</i>	Integrative constant
<i>kd</i>	Derivative constant
<i>setpoint</i>	Desired state
<i>saturation</i>	Maximum response returned by the controller
<i>max_integral</i>	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

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

### 5.10.3.3 set\_setpoint()

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

Set the setpoint object.

#### Parameters

<i>setpoint</i>	Desired state
-----------------	---------------

### 5.10.3.4 update() [1/2]

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

Update PID with new state and return response.

## Parameters

<i>state</i>	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

<i>state</i>	Current value of the controlled variable
<i>state_change</i>	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](#)



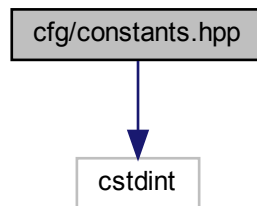
## Chapter 6

# File Documentation

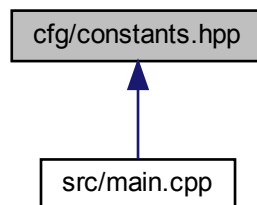
### 6.1 cfg/constants.hpp File Reference

```
#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

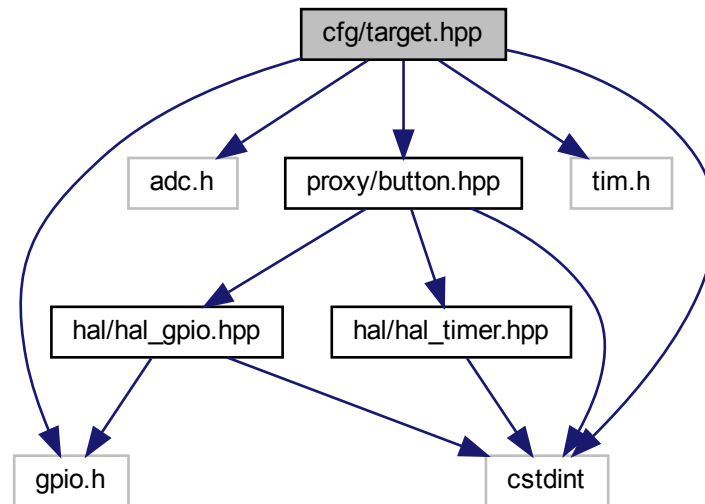
```
constexpr float saturation = 100.0 [constexpr]
```

## 6.2 cfg/target.hpp File Reference

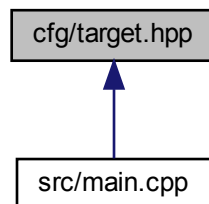
```
#include <cstdint>
#include "adc.h"
#include "gpio.h"
#include "tim.h"
```

```
#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
```

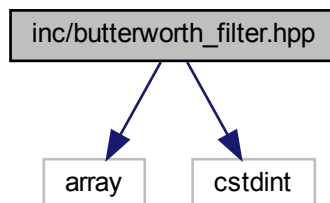
### 6.3 doc/report.md File Reference

### 6.4 inc/butterworth\_filter.hpp File Reference

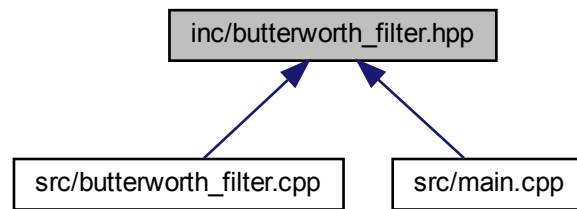
```
#include <array>
```

```
#include <cstdint>
```

Include dependency graph for butterworth\_filter.hpp:



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



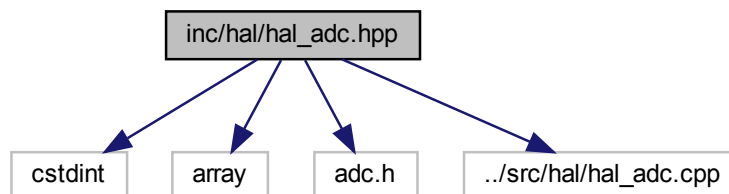
## Classes

- class [ButterworthFilter](#)

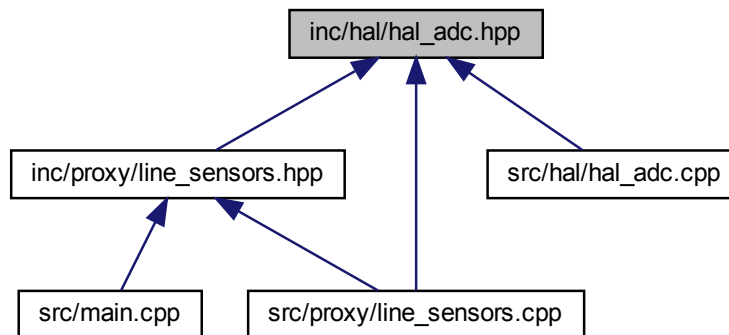
Implementation of Butterworth second order low-pass filter A generic digital filter follows the relation  $a_0 * y[k] = \sum(b_i * x[k - i]) - \sum(a_j * y[k - j])$  Where  $x[k]$  - measurement at instant  $k$   $y[k]$  - filtered signal at instant  $k$  The Butterworth filter have the special property 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 continuous coefficients of the Butterworth filter is available here: [https://en.wikipedia.org/wiki/Butterworth\\_filter](https://en.wikipedia.org/wiki/Butterworth_filter) The discrete version were computed with the Tustin method: [https://en.wikipedia.org/wiki/Bilinear\\_transform](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

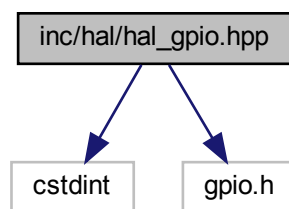
- class [HalAdc< number\\_of\\_channels, reading\\_per\\_channel >](#)

## 6.6 inc/hal/hal\_gpio.hpp File Reference

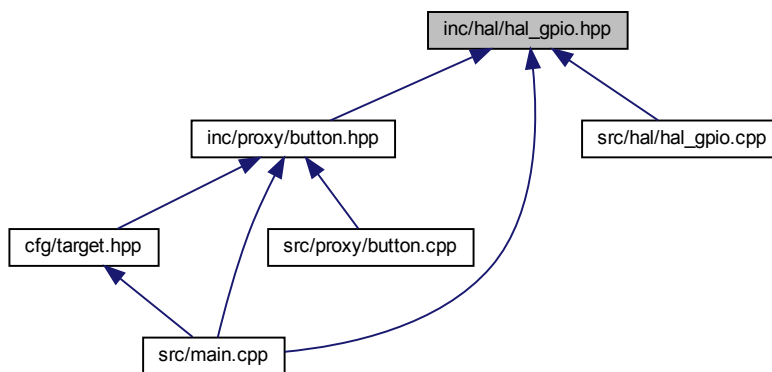
```
#include <stdint>
```

```
#include "gpio.h"
```

Include dependency graph for `hal_gpio.hpp`:



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



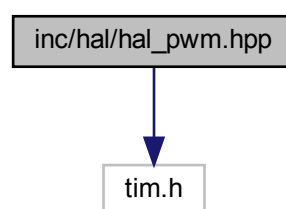
## Classes

- class [HalGpio](#)

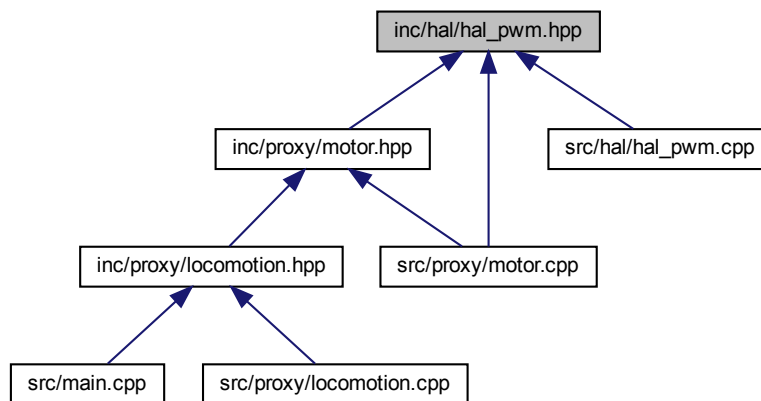
## 6.7 inc/hal/hal\_pwm.hpp File Reference

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

Include dependency graph for `hal_pwm.hpp`:



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



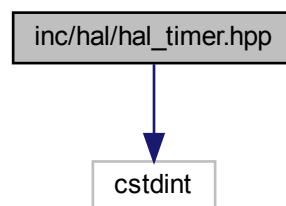
## Classes

- class [HalPwm](#)

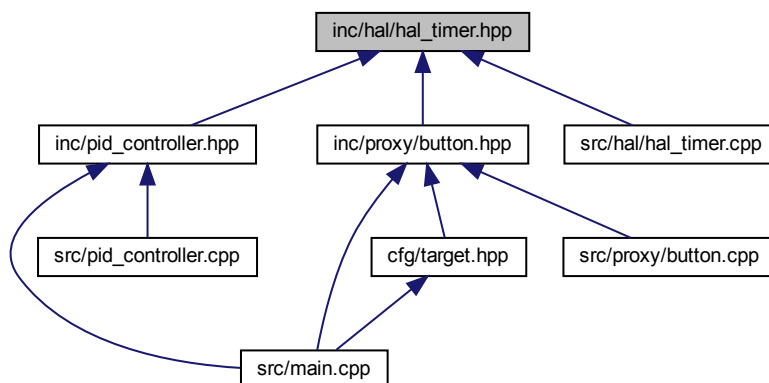
## 6.8 inc/hal/hal\_timer.hpp File Reference

```
#include <cstdint>
```

Include dependency graph for `hal_timer.hpp`:



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



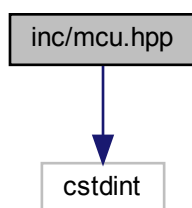
## Classes

- class [HalTimer](#)

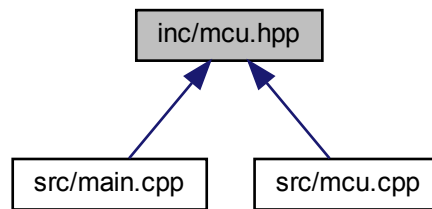
## 6.9 inc/mcu.hpp File Reference

```
#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

<i>ms</i>	Sleep time in milliseconds
-----------	----------------------------



### 6.9.1.3 SystemClock\_Config()

```
void SystemClock_Config (  
    void )
```

Initializes System Clock.

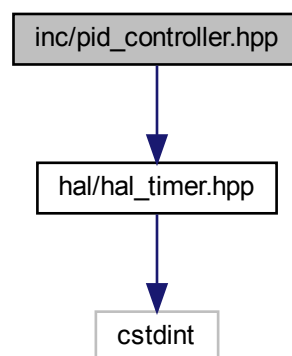
#### Note

Defined by cube.

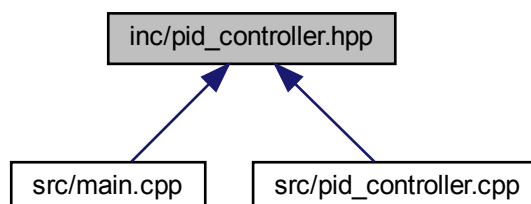
## 6.10 inc/pid\_controller.hpp File Reference

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

Include dependency graph for pid\_controller.hpp:



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



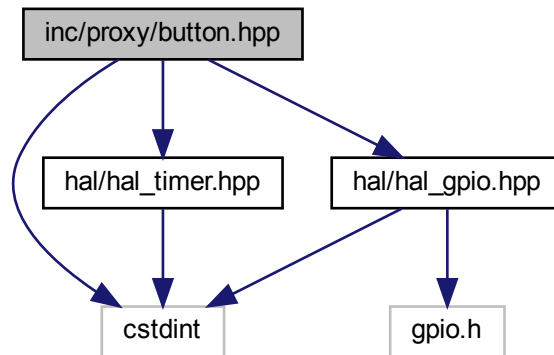
## Classes

- class [PidController](#)

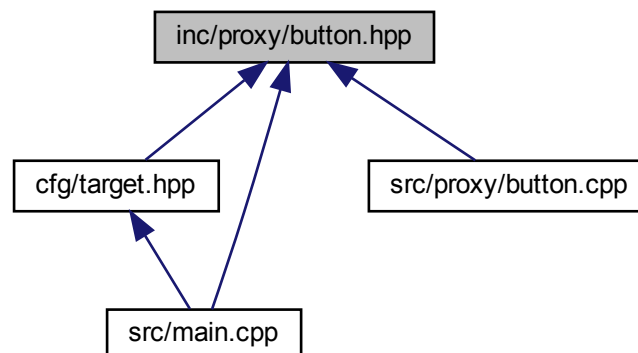
*Implementation of simple PID controller  $\text{Response} = K_p(\text{error} + K_i * \text{integral}(\text{error}) + K_d * d/dt(\text{error}))$*

## 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

<a href="#">BUTTON_PULL_UP</a>	
<a href="#">BUTTON_PULL_DOWN</a>	

#### 6.11.1.2 [button\\_status\\_t](#)

enum [button\\_status\\_t](#)

[Button](#) status type.

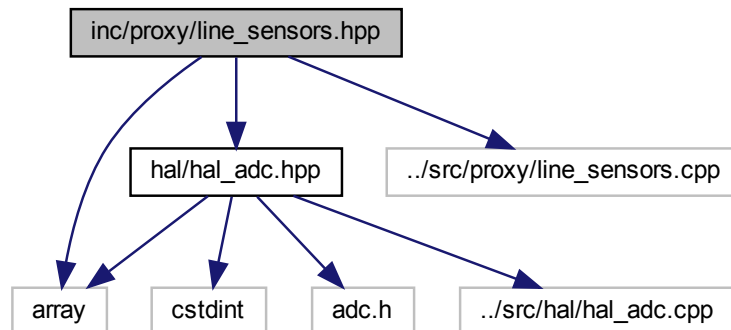
Enumerator

<a href="#">BUTTON_NO_PRESS</a>	
<a href="#">BUTTON_SHORT_PRESS</a>	
<a href="#">BUTTON_LONG_PRESS</a>	
<a href="#">BUTTON_EXTRA_LONG_PRESS</a>	

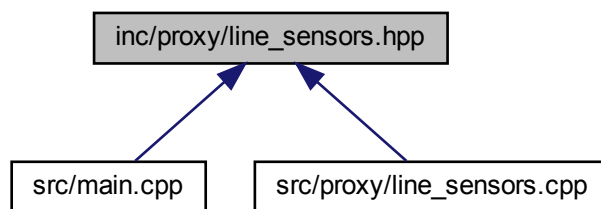
## 6.12 inc/proxy/line\_sensors.hpp File Reference

```
#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:



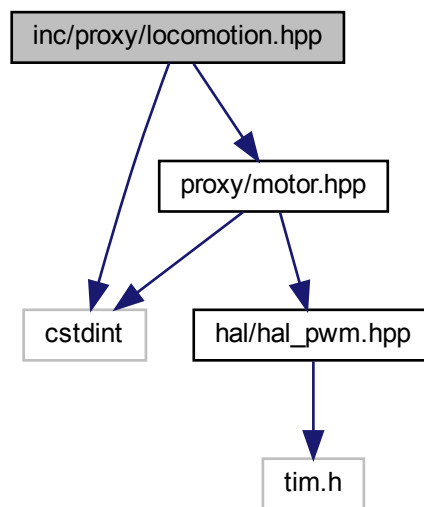
## Classes

- class [LineSensors< number\\_of\\_sensors, reading\\_per\\_sensor >](#)

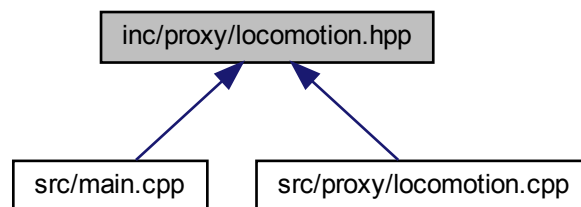
## 6.13 inc/proxy/locomotion.hpp File Reference

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

Include dependency graph for locomotion.hpp:



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



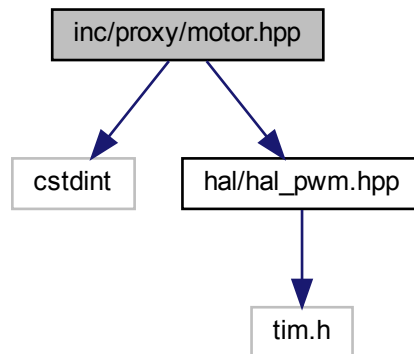
## Classes

- class [Locomotion](#)

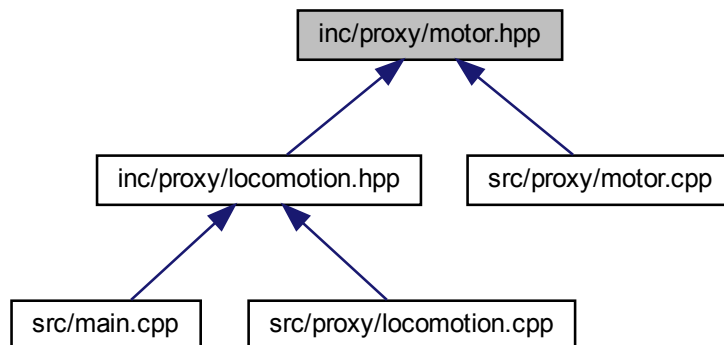
## 6.14 inc/proxy/motor.hpp File Reference

```
#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

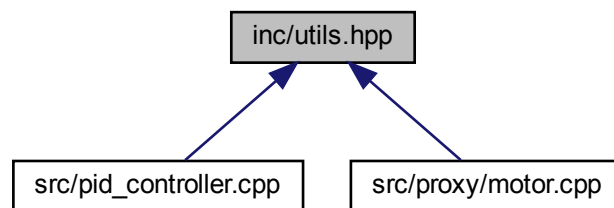
```
constexpr int16_t max_motors_speed = 100 [constexpr]
```

#### 6.14.1.2 min\_motors\_speed

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

## 6.15 inc/utls.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))
```

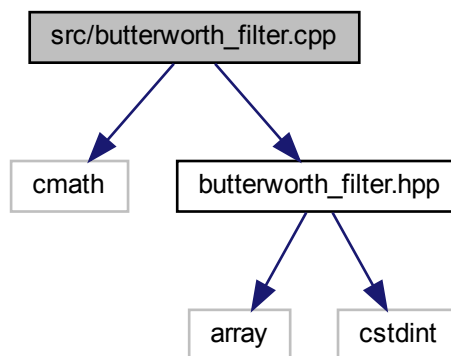
### 6.15.1.2 map

```
#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:
```



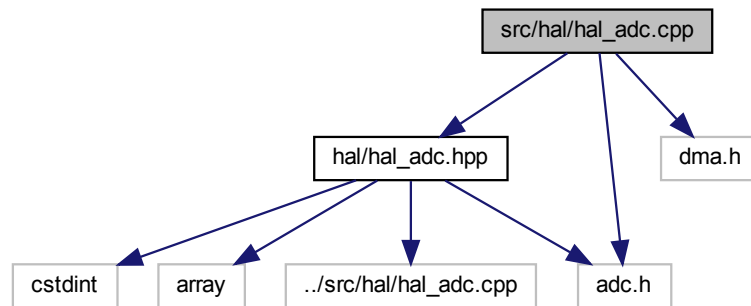
## 6.18 src/hal/hal\_adc.cpp File Reference

```
#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

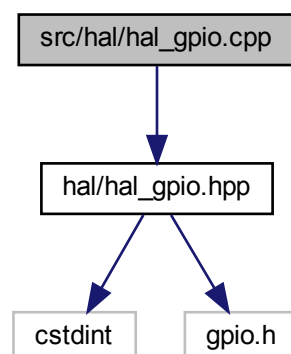
#### 6.18.1.1 \_\_HAL\_ADC\_CPP\_\_

```
#define __HAL_ADC_CPP__
```

## 6.19 src/hal/hal\_gpio.cpp File Reference

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

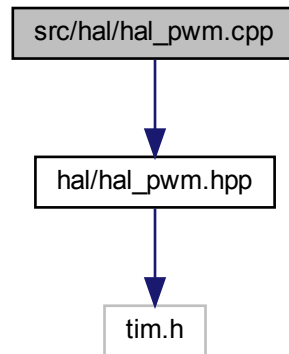
Include dependency graph for hal\_gpio.cpp:



## 6.20 src/hal/hal\_pwm.cpp File Reference

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

Include dependency graph for hal\_pwm.cpp:

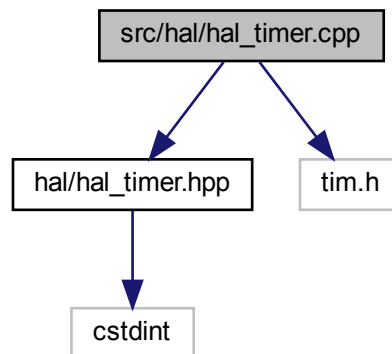


## 6.21 src/hal/hal\_timer.cpp File Reference

```
#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"
```

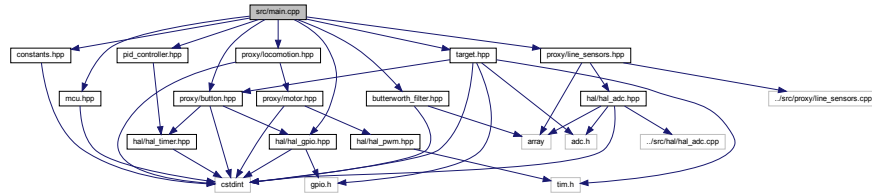
```
#include "constants.hpp"
```

```

#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()

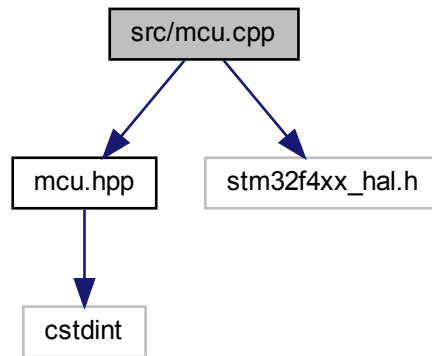
```

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 )
```

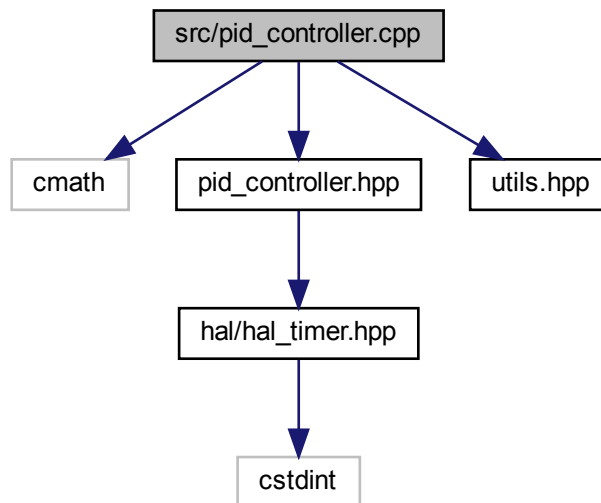
Put the MCU to sleep.

## Parameters

<i>ms</i>	Sleep time in milliseconds
-----------	----------------------------

## 6.24 src/pid\_controller.cpp File Reference

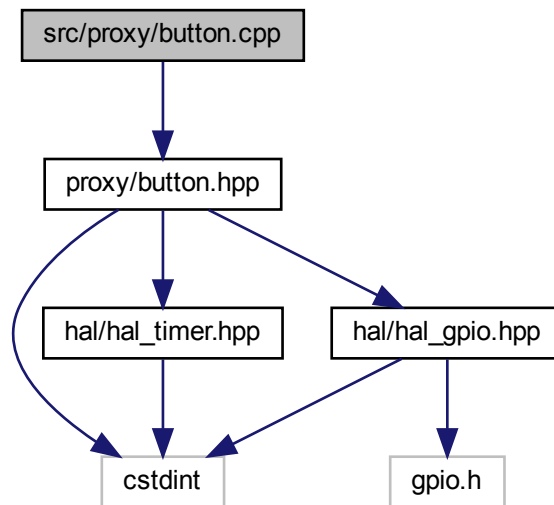
```
#include <cmath>
#include "pid_controller.hpp"
#include "utils.hpp"
Include dependency graph for pid_controller.cpp:
```



## 6.25 src/proxy/button.cpp File Reference

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

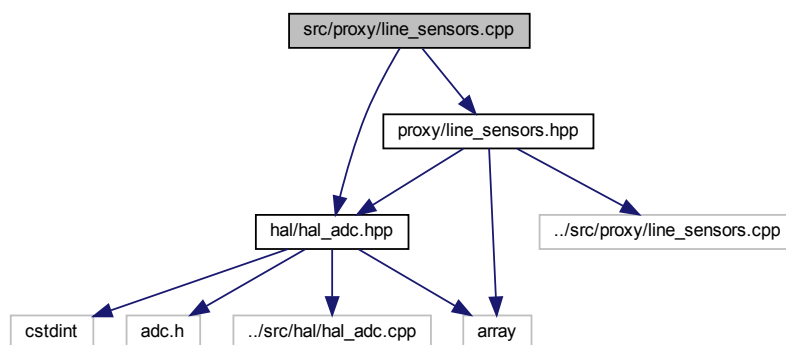
Include dependency graph for button.cpp:



## 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]
```

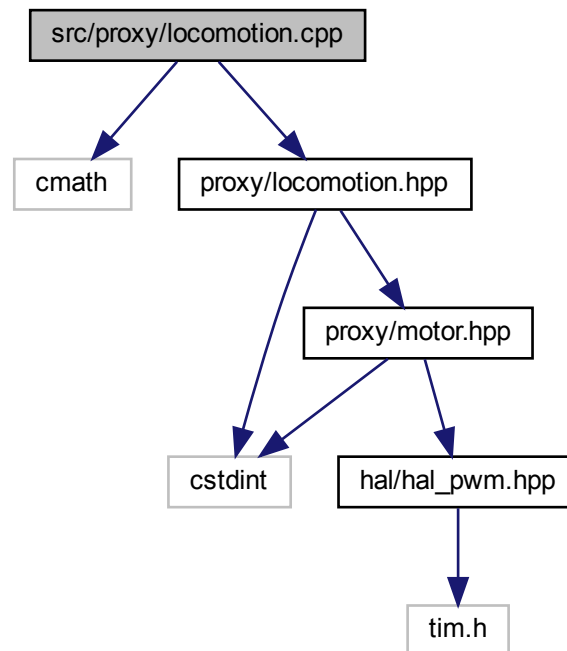
#### 6.26.2.2 `default_white_value`

```
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:

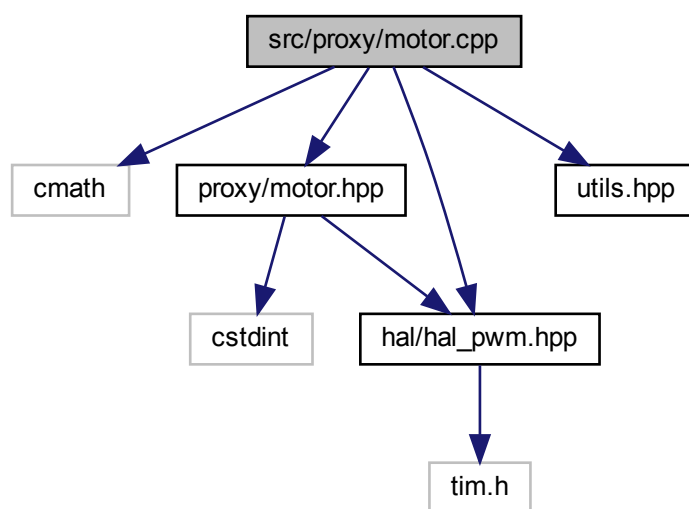


## 6.28 src/proxy/motor.cpp File Reference

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



Include dependency graph for motor.cpp:





# Index

- `__HAL_ADC_CPP__`
  - `hal_adc.cpp`, [49](#)
- `__LINE_SENSORS_CPP__`
  - `line_sensors.cpp`, [55](#)
- `adc_num_channels`
  - `target.hpp`, [33](#)
- `adc_readings_per_channel`
  - `target.hpp`, [33](#)
- `ButterworthFilter`, [11](#)
  - `ButterworthFilter`, [11](#)
  - `update`, [12](#)
- `Button`, [12](#)
  - `Button`, [12](#)
  - `get_status`, [13](#)
- `button.hpp`
  - `BUTTON_EXTRA_LONG_PRESS`, [43](#)
  - `BUTTON_LONG_PRESS`, [43](#)
  - `BUTTON_NO_PRESS`, [43](#)
  - `BUTTON_PULL_DOWN`, [43](#)
  - `button_pull_resistor_t`, [43](#)
  - `BUTTON_PULL_UP`, [43](#)
  - `BUTTON_SHORT_PRESS`, [43](#)
  - `button_status_t`, [43](#)
- `BUTTON_EXTRA_LONG_PRESS`
  - `button.hpp`, [43](#)
- `button_gpio_port`
  - `target.hpp`, [33](#)
- `BUTTON_LONG_PRESS`
  - `button.hpp`, [43](#)
- `BUTTON_NO_PRESS`
  - `button.hpp`, [43](#)
- `button_pin`
  - `target.hpp`, [33](#)
- `BUTTON_PULL_DOWN`
  - `button.hpp`, [43](#)
- `button_pull_resistor`
  - `target.hpp`, [33](#)
- `button_pull_resistor_t`
  - `button.hpp`, [43](#)
- `BUTTON_PULL_UP`
  - `button.hpp`, [43](#)
- `BUTTON_SHORT_PRESS`
  - `button.hpp`, [43](#)
- `button_status_t`
  - `button.hpp`, [43](#)
- `calibrate_black`
  - `LineSensors< number_of_sensors, reading_per_sensor >`, [20](#)
- `calibrate_white`
  - `LineSensors< number_of_sensors, reading_per_sensor >`, [20](#)
- `cfg/constants.hpp`, [29](#)
- `cfg/target.hpp`, [31](#)
- `constants.hpp`
  - `filter_frequency`, [30](#)
  - `kd`, [30](#)
  - `ki`, [30](#)
  - `kp`, [30](#)
  - `left_deadzone`, [30](#)
  - `linear_base_speed`, [30](#)
  - `linear_decay`, [31](#)
  - `max_integral`, [31](#)
  - `right_deadzone`, [31](#)
  - `saturation`, [31](#)
- `constrain`
  - `utils.hpp`, [47](#)
- `default_black_value`
  - `line_sensors.cpp`, [55](#)
- `default_white_value`
  - `line_sensors.cpp`, [55](#)
- `doc/report.md`, [34](#)
- `filter_frequency`
  - `constants.hpp`, [30](#)
- `get_adc_reading`
  - `HalAdc< number_of_channels, reading_per_channel >`, [14](#)
- `get_position`
  - `LineSensors< number_of_sensors, reading_per_sensor >`, [20](#)
- `get_status`
  - `Button`, [13](#)
- `get_time`
  - `HalTimer`, [18](#)
- `hal_adc.cpp`
  - `__HAL_ADC_CPP__`, [49](#)
- `HAL_ADC_ConvCpltCallback`
  - `main.cpp`, [51](#)
- `HalAdc`
  - `HalAdc< number_of_channels, reading_per_channel >`, [14](#)
- `HalAdc< number_of_channels, reading_per_channel >`, [13](#)

- get\_adc\_reading, 14
  - HalAdc, 14
  - set\_reading\_done, 14
  - update\_reading, 15
- HalGpio, 15
  - HalGpio, 15
  - read, 16
  - toggle, 16
  - write, 16
- HalPwm, 17
  - HalPwm, 17
  - set\_compare, 17
  - start, 18
- HalTimer, 18
  - get\_time, 18
  - HalTimer, 18
  - reset, 19
- inc/butterworth\_filter.hpp, 34
- inc/hal/hal\_adc.hpp, 35
- inc/hal/hal\_gpio.hpp, 36
- inc/hal/hal\_pwm.hpp, 37
- inc/hal/hal\_timer.hpp, 38
- inc/mcu.hpp, 39
- inc/pid\_controller.hpp, 41
- inc/proxy/button.hpp, 42
- inc/proxy/line\_sensors.hpp, 43
- inc/proxy/locomotion.hpp, 44
- inc/proxy/motor.hpp, 45
- inc/utils.hpp, 47
- kd
  - constants.hpp, 30
- ki
  - constants.hpp, 30
- kp
  - constants.hpp, 30
- led\_gpio\_port
  - target.hpp, 33
- led\_pin
  - target.hpp, 33
- left\_deadzone
  - constants.hpp, 30
- left\_motor\_timer\_handle
  - target.hpp, 33
- line\_sensor\_adc\_handle
  - target.hpp, 34
- line\_sensors.cpp
  - \_\_LINE\_SENSORS\_CPP\_\_, 55
  - default\_black\_value, 55
  - default\_white\_value, 55
- linear\_base\_speed
  - constants.hpp, 30
- linear\_decay
  - constants.hpp, 31
  - Locomotion, 21
- LineSensors
  - LineSensors< number\_of\_sensors, reading\_per\_sensor >, 19
  - LineSensors< number\_of\_sensors, reading\_per\_sensor >, 19
    - calibrate\_black, 20
    - calibrate\_white, 20
    - get\_position, 20
    - LineSensors, 19
  - Locomotion, 20
    - linear\_decay, 21
    - Locomotion, 21
    - set\_speeds, 23
- main
  - main.cpp, 51
- main.cpp
  - HAL\_ADC\_ConvCpltCallback, 51
  - main, 51
- map
  - utils.hpp, 47
- max\_integral
  - constants.hpp, 31
- max\_motors\_speed
  - motor.hpp, 46
- mcu.cpp
  - mcu\_init, 52
  - mcu\_sleep, 52
- mcu.hpp
  - mcu\_init, 40
  - mcu\_sleep, 40
  - SystemClock\_Config, 41
- mcu\_init
  - mcu.cpp, 52
  - mcu.hpp, 40
- mcu\_sleep
  - mcu.cpp, 52
  - mcu.hpp, 40
- min\_motors\_speed
  - motor.hpp, 47
- Motor, 23
  - Motor, 23
  - set\_speed, 24
- motor.hpp
  - max\_motors\_speed, 46
  - min\_motors\_speed, 47
- motor\_backward\_timer\_channel
  - target.hpp, 34
- motor\_forward\_timer\_channel
  - target.hpp, 34
- PidController, 24
  - PidController, 25
  - reset, 25
  - set\_parameters, 26
  - set\_setpoint, 26
  - update, 26, 27
- read
  - HalGpio, 16

- README.md, [48](#)
- reset
  - HalTimer, [19](#)
  - PidController, [25](#)
- right\_deadzone
  - constants.hpp, [31](#)
- right\_motor\_timer\_handle
  - target.hpp, [34](#)
- saturation
  - constants.hpp, [31](#)
- set\_compare
  - HalPwm, [17](#)
- set\_parameters
  - PidController, [26](#)
- set\_reading\_done
  - HalAdc< number\_of\_channels, reading\_per\_channel  
>, [14](#)
- set\_setpoint
  - PidController, [26](#)
- set\_speed
  - Motor, [24](#)
- set\_speeds
  - Locomotion, [23](#)
- src/butterworth\_filter.cpp, [48](#)
- src/hal/hal\_adc.cpp, [48](#)
- src/hal/hal\_gpio.cpp, [49](#)
- src/hal/hal\_pwm.cpp, [50](#)
- src/hal/hal\_timer.cpp, [50](#)
- src/main.cpp, [50](#)
- src/mcu.cpp, [52](#)
- src/pid\_controller.cpp, [53](#)
- src/proxy/button.cpp, [53](#)
- src/proxy/line\_sensors.cpp, [54](#)
- src/proxy/locomotion.cpp, [55](#)
- src/proxy/motor.cpp, [56](#)
- start
  - HalPwm, [18](#)
- SystemClock\_Config
  - mcu.hpp, [41](#)
- target.hpp
  - adc\_num\_channels, [33](#)
  - adc\_readings\_per\_channel, [33](#)
  - button\_gpio\_port, [33](#)
  - button\_pin, [33](#)
  - button\_pull\_resistor, [33](#)
  - led\_gpio\_port, [33](#)
  - led\_pin, [33](#)
  - left\_motor\_timer\_handle, [33](#)
  - line\_sensor\_adc\_handle, [34](#)
  - motor\_backward\_timer\_channel, [34](#)
  - motor\_forward\_timer\_channel, [34](#)
  - right\_motor\_timer\_handle, [34](#)
- toggle
  - HalGpio, [16](#)
- update
  - ButterworthFilter, [12](#)
  - PidController, [26](#), [27](#)
  - update\_reading
    - HalAdc< number\_of\_channels, reading\_per\_channel  
>, [15](#)
  - utils.hpp
    - constrain, [47](#)
    - map, [47](#)
  - write
    - HalGpio, [16](#)