

ME 507 Strobe Tuner

1

Generated by Doxygen 1.10.0

1 ME 507 Strobe Tuner Documentation	1
1.0.1 Overview and Theory	1
1.0.2 Features	1
1.0.3 Getting Started in Software	2
1.0.4 Software Implementation - Classes	3
1.0.5 Software Implementation - Cooperative Multitasking	3
1.0.6 Conclusion	4
2 Data Structure Index	5
2.1 Data Structures	5
3 File Index	7
3.1 File List	7
4 Data Structure Documentation	9
4.1 CLController Class Reference	9
4.1.1 Detailed Description	9
4.1.2 Field Documentation	10
4.1.2.1 curr	10
4.1.2.2 curr_time	10
4.1.2.3 eff	10
4.1.2.4 err	11
4.1.2.5 err_acc	11
4.1.2.6 initial_time	11
4.1.2.7 kd	11
4.1.2.8 kf	11
4.1.2.9 ki	11
4.1.2.10 kp	11
4.1.2.11 prev_err_index	11
4.1.2.12 prev_err_list	12
4.1.2.13 prev_err_list_length	12
4.1.2.14 setpoint	12
4.1.2.15 slope	12
4.2 Display Class Reference	12
4.2.1 Detailed Description	12
4.2.2 Field Documentation	13
4.2.2.1 curr_note	13
4.2.2.2 hi2c	13
4.2.2.3 huart	13
4.3 Encoder Class Reference	13
4.3.1 Detailed Description	14
4.3.2 Field Documentation	14
4.3.2.1 curr_count	14

4.3.2.2 curr_time	15
4.3.2.3 dt	15
4.3.2.4 dx	15
4.3.2.5 pos	15
4.3.2.6 prev_count	15
4.3.2.7 prev_time	15
4.3.2.8 speed	15
4.3.2.9 timer	15
4.3.2.10 timing_timer	16
4.4 Motor Class Reference	16
4.4.1 Detailed Description	16
4.4.2 Field Documentation	16
4.4.2.1 channels	16
4.4.2.2 duty_cycle	17
4.4.2.3 enable_flag	17
4.4.2.4 timer	17
4.5 PitchEncoder Class Reference	17
4.5.1 Detailed Description	17
4.5.2 Field Documentation	18
4.5.2.1 delta	18
4.5.2.2 encoder	18
4.5.2.3 pitch	18
4.6 RC_Controller Class Reference	18
4.6.1 Detailed Description	18
4.6.2 Field Documentation	19
4.6.2.1 fe1	19
4.6.2.2 fe2	19
4.6.2.3 fe_flag1	19
4.6.2.4 fe_flag2	19
4.6.2.5 period1	20
4.6.2.6 period2	20
4.6.2.7 re1	20
4.6.2.8 re2	20
4.6.2.9 timer	20
5 File Documentation	21
5.1 doxy_core/Inc/CLController.h File Reference	21
5.1.1 Detailed Description	21
5.1.2 Function Documentation	21
5.1.2.1 reset_controller()	21
5.1.2.2 run()	22
5.2 CLController.h	23

5.3 doxy_core/Inc/controller_driver.h File Reference	23
5.3.1 Detailed Description	24
5.3.2 Function Documentation	24
5.3.2.1 controller_driver_calc_per1()	24
5.3.2.2 controller_driver_calc_per2()	25
5.4 controller_driver.h	25
5.5 doxy_core/Inc/display_driver.h File Reference	26
5.5.1 Detailed Description	26
5.5.2 Function Documentation	26
5.5.2.1 display_note()	26
5.6 display_driver.h	28
5.7 doxy_core/Inc/encoder_handler.h File Reference	28
5.7.1 Detailed Description	28
5.7.2 Function Documentation	29
5.7.2.1 delta()	29
5.7.2.2 encoder_calc_speed()	30
5.7.2.3 encoder_read_curr_state()	31
5.7.2.4 zero()	31
5.8 encoder_handler.h	32
5.9 doxy_core/Inc/main.h File Reference	32
5.9.1 Detailed Description	33
5.9.2 Macro Definition Documentation	33
5.9.2.1 TCK_GPIO_Port	33
5.9.2.2 TCK_Pin	33
5.9.2.3 TMS_GPIO_Port	33
5.9.2.4 TMS_Pin	33
5.9.2.5 USART_RX_GPIO_Port	34
5.9.2.6 USART_RX_Pin	34
5.9.2.7 USART_TX_GPIO_Port	34
5.9.2.8 USART_TX_Pin	34
5.9.3 Function Documentation	34
5.9.3.1 Error_Handler()	34
5.9.3.2 HAL_TIM_MspPostInit()	34
5.10 main.h	35
5.11 doxy_core/Inc/motor_driver.h File Reference	35
5.11.1 Detailed Description	36
5.11.2 Function Documentation	36
5.11.2.1 motor_enable_disable()	36
5.11.2.2 motor_set_duty_cycle()	37
5.12 motor_driver.h	38
5.13 doxy_core/Inc/pitch_encoder_handler.h File Reference	38
5.13.1 Detailed Description	39

5.13.2 Function Documentation	39
5.13.2.1 get_pitch()	39
5.14 pitch_encoder_handler.h	40
5.15 doxy_core/README.md File Reference	40
5.16 doxy_core/Src/CLController.c File Reference	40
5.16.1 Detailed Description	40
5.16.2 Function Documentation	40
5.16.2.1 reset_controller()	40
5.16.2.2 run()	41
5.17 CLController.c	42
5.18 doxy_core/Src/controller_driver.c File Reference	42
5.18.1 Detailed Description	43
5.18.2 Function Documentation	43
5.18.2.1 controller_driver_calc_per1()	43
5.18.2.2 controller_driver_calc_per2()	43
5.19 controller_driver.c	44
5.20 doxy_core/Src/display_driver.c File Reference	44
5.20.1 Detailed Description	44
5.20.2 Function Documentation	44
5.20.2.1 display_note()	44
5.20.3 Variable Documentation	45
5.20.3.1 disp_addr	45
5.20.3.2 note_addresses	46
5.20.3.3 Pitch_Buffer	46
5.20.3.4 Pitch_Message	46
5.21 display_driver.c	46
5.22 doxy_core/Src/encoder_handler.c File Reference	47
5.22.1 Detailed Description	47
5.22.2 Function Documentation	47
5.22.2.1 delta()	47
5.22.2.2 encoder_calc_speed()	48
5.22.2.3 encoder_read_curr_state()	49
5.22.2.4 zero()	49
5.23 encoder_handler.c	50
5.24 doxy_core/Src/main.c File Reference	50
5.24.1 Detailed Description	52
5.24.2 Function Documentation	52
5.24.2.1 display_task()	52
5.24.2.2 Error_Handler()	52
5.24.2.3 main()	53
5.24.2.4 motor_task()	55
5.24.2.5 SystemClock_Config()	56

5.24.3 Variable Documentation	57
5.24.3.1 Buffer	57
5.24.3.2 Eff_Buffer	57
5.24.3.3 EndMSG	57
5.24.3.4 hadc1	57
5.24.3.5 hi2c2	57
5.24.3.6 htim1	57
5.24.3.7 htim3	57
5.24.3.8 htim4	58
5.24.3.9 htim5	58
5.24.3.10 htim6	58
5.24.3.11 htim8	58
5.24.3.12 huart2	58
5.24.3.13 led_buff	58
5.24.3.14 Pos_Buffer	58
5.24.3.15 Space	59
5.24.3.16 Speed_Buffer	59
5.24.3.17 StartMSG	59
5.24.3.18 t1state	59
5.24.3.19 t2state	59
5.25 main.c	60
5.26 doxy_core/Src/motor_driver.c File Reference	70
5.26.1 Detailed Description	70
5.26.2 Function Documentation	71
5.26.2.1 motor_enable_disable()	71
5.26.2.2 motor_set_duty_cycle()	71
5.27 motor_driver.c	72
5.28 doxy_core/Src/pitch_encoder_handler.c File Reference	74
5.28.1 Detailed Description	74
5.28.2 Function Documentation	74
5.28.2.1 get_pitch()	74
5.29 pitch_encoder_handler.c	75
Index	77

Chapter 1

ME 507 Strobe Tuner Documentation

Project Members: Jojo Penrose, Jared Sinasohn

Welcome to the ME 507 Strobe Tuner Documentation!

This manual provides an overview of the software side of our ME 507 Strobe Tuner project, including its features, setup, and usage instructions.

1.0.1 Overview and Theory

A strobe tuner is a highly accurate device used for tuning musical instruments by measuring the frequency of sound waves. Unlike conventional tuners, which often use needle or LED indicators to show pitch deviation, a strobe tuner operates by comparing the sound wave of the instrument to a reference frequency. It uses a rotating disk with accurately patterned slits cut into it, illuminated by a strobe light, to visually represent the difference between the played note and the desired pitch. As the disk spins, patterns of light and dark bands appear to move. When the frequency of the played note matches the reference frequency, the bands appear stationary, indicating that the instrument is in tune. If the instrument is out of tune, the bands will appear to move, depending on whether the pitch is flat or sharp.

However, in our research, we found that most strobe tuners can be thousands of dollars, which is an amount of money many musicians are unable to afford. So, our team set out to produce a strobe tuner running off a low-cost microcontroller with low cost parts to eventually make a low cost tuner that could one day be turned into a full product. We designed custom PCB's, created a full 3d-printed body, and coded in C to create an initial prototype. All in all, even though the initial prototype had many bugs, we were able to create a proof of concept showing our project is viable. If you would like to see more about the project, including information about the PCB Design, Analog filtering, mechanical design, and the trials and tribulations, visit [this link](#) which will take you to a Google Doc report about the project.

1.0.2 Features

Here are some of the features of our project:

- Aesthetically pleasing exterior
- DC motor with full PIDF speed control
- I2C driven 16-segment display with sharp/flat LED's for display of the current tuning pitch
- Rotary knob encoder for custom pitch selection Here are some things we couldn't quite implement right now but will implement in the future:
- Input capture on the Microphone to auto-select tuning pitch
- Current sense on the motor through an ADC for over-current shutdown
- Making the I2C driver work

1.0.3 Getting Started in Software

All code for this project was completed in C using STM32CubeID (Cube). Our project runs on the STM32L476RGT6 chip, and before the code can be run it must be configured correctly using Cube's pin configuration GUI. The a table of the correct pin configurations for this project can be found below:

Category	Pin	Function	Description of Purpose	Notes
Serial Wire Debugging	PA13	SWD I/O	Serial wire Debugging via ST Link	N/A
	PA14	SWD Clock		
	PB3	SWD SWO		
GPIO	PA10	Output	Motor Enable Pin	N/A
	PB2	Output	Display Enable Pin	N/A
	PC10	Input	Auto/Manual Switch input	N/A
I2C	PB10	I2C2 SCL	I2C2 Bus for running the 16 segment display driver	20 KHz clock frequency
	PB11	I2C2 SDA		
UART	PA2	UART2 TX	UART for communicating to computer for debugging purposes	N/A
	PA3	UART2 RX		
ADC	PC5	ADC1 IN14	ADC for current sense of the motor	Single-ended
Timers	PA8	TIM1 CH2	Motor PWM Channels	CH1 CH2 set to PWM output, PSC = 0, ARR = 999
	PA9	TIM1 CH1		
	PA7	TIM3 CH1/CH2	Input Capture for the trigger of the RC Controller (Used for E-Stop)	CH1 Indirect Capture, rising edge detection, CH2 Direct Capture, falling edge detection, Enable NVIC Global Interrupts, PSC = 79, ARR = 65535
	PB0	TIM3 CH3/CH4	Input Capture for the wheel of the RC Controller	CH3 direct Capture, rising edge detection, CH4 indirect Capture, falling edge detection, Enable NVIC Global Interrupts, PSC = 79, ARR = 65536
	PB6	TIM4 CH1	Motor Encoder Quadrature Channels	Set timer to Encoder Mode, Capture on T1 and T2 PSC = 0, ARR = 65535
	PB7	TIM4 CH2		
	PA0	TIM5 CH1	Pitch Knob Rotary encoder Quadrature Channels	Capture on T1 and T2 PSC = 0, ARR = 1073741824
	PA1	TIM5 CH2		
	N/A	TIM6	Internal timer for speed calculations	N/A
	PC6	TIM8 CH1	Input Capture for microphone	Input Capture Direct Mode, PSC = 79, ARR = 65535, Rising Edge Detection, Enable Global NVIC interrupts
Crystal Oscillator	PH0	RCC OSC IN	Crystal Oscillator pins	80 MHZ internal clock
	PH1	RCC OSC OUT		

1.0.4 Software Implementation - Classes

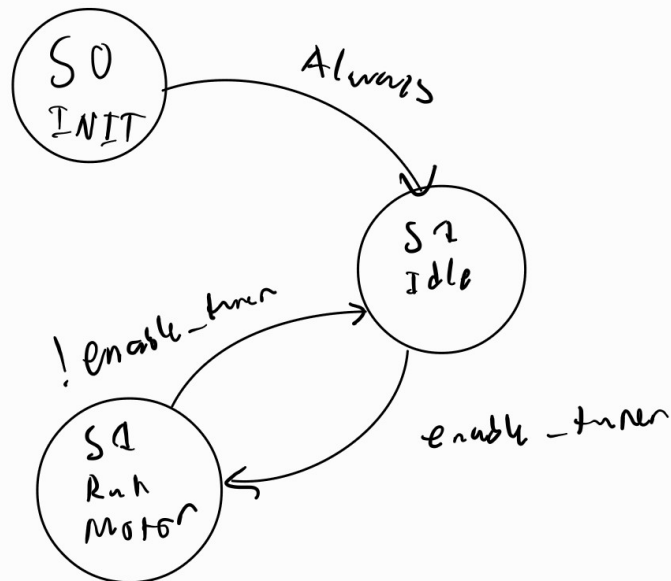
Even though C does not support object oriented programming, an object oriented approach was taken when coding this project to streamline the coding process. Classes for a DC motor, RC_controller (not used but would have been used for E-stop), a [Display](#), a Generic [Encoder](#), a Pitch Selection encoder, an a Closed Loop Controller with full PIDF were written for the purposes of this project, and can be found below.

- [CLController](#)
- [Display](#)
- [Encoder](#)
- [Motor](#)
- [PitchEncoder](#)
- [RC_Controller](#)

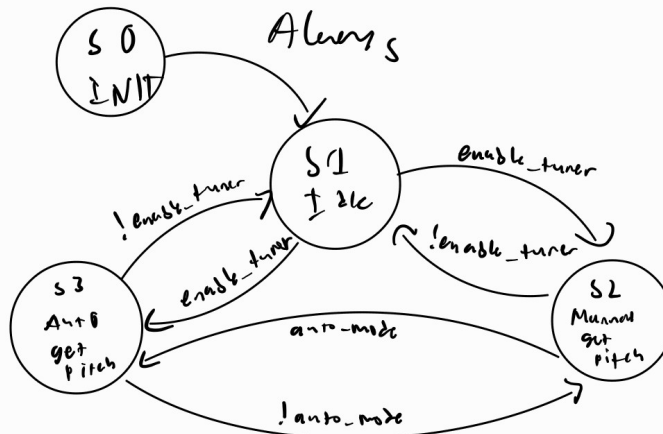
1.0.5 Software Implementation - Cooperative Multitasking

In order to run all peripherals at once, a Cooperative Multitasking approach was taken. For this, different tasks for the microcontroller to execute were designed with each task handling a different function. The two tasks that were written are for handling pitch selection and the LED display, and the handling of the motor. Finite State machines were drawn for each task to explain how each task would operate, and images of these finite state machines can be found below.

Task 1 - Motor handler



Task 2 - Sound and display task



Each task was run in a round-robin style, where each task assumes none of the other tasks will take a significant amount of time, allowing for each task to run simultaneously. This was deemed satisfactory for this project since there are only two tasks and each task is not all that complicated.

1.0.6 Conclusion

In this project, our team was able to create a proof of concept prototype for a mechanical strobe tuner. We are excited to continue this project in the future to make a low-cost strobe tuner that we can be proud of. Feel free to look through the documentation in this manual.

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

CLController

An implementation of a PIDF (F = Feedforward constant) closed loop controller. It is a generic controller that takes in a measurement, controller constants, and a setpoint and outputs an effort to be sent to the actuator 9

Display

The class that implements the control of the TLC59116 16-Channel FM+ I2C-Bus Constant-Current LED Sink Driver to display the current note on the tuner 12

Encoder

A generic implementation of a quadrature encoder that tracks the position of the encoder in counts and the speed the encoder is traveling at in counts per second 13

Motor

An implementation of a motor driver using a struct to emulate Object Oriented Programming. The motor has 4 parameters, timer which indicates the timer to be used to run the motor, channels, which indicates the channels to be used to run the motor, duty_cycle, the duty cycle to run the motor at, and enable_flag, which determines if the motor is allowed to run 16

PitchEncoder

This class, in an object oriented sense, inherits the [Encoder](#) class but specifically reads values from the pitch encoder to map them to a specific pitch 17

RC_Controller

An implementation of an RC controller that sends pulse widths from its two control axes—a trigger and a wheel—through infrared. The motor has parameters for the timer used for the input capture of the pulse widths, the pulse widths sent by the controller for each of the axes, the values of time of the rising edges and falling axes for each axis, and flags that indicate a falling edge has been detected and therefore a period should be calculated 18

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

doxy_core/Inc/ CLController.h	
This file defines the abstract class and methods for a closed-loop PIDF controller	21
doxy_core/Inc/ controller_driver.h	
A pseudo-object-oriented structure for implementing the calculation of a pulse-width read from a two-channel rc controller	23
doxy_core/Inc/ display_driver.h	
The header file for the i2c driven 16 segment display class	26
doxy_core/Inc/ encoder_handler.h	
The header file for a class that implements a generic quadrature encoder	28
doxy_core/Inc/ main.h	
: Header for main.c file. This file contains the common defines of the application	32
doxy_core/Inc/ motor_driver.h	
The header file for a generic DC motor driven from a standard h-bridge motor driver	35
doxy_core/Inc/ pitch_encoder_handler.h	
.	38
doxy_core/Src/ CLController.c	
This file implements the methods for the CLController class	40
doxy_core/Src/ controller_driver.c	
This file implements the mehtods to calculate the period for the RC_Controller class	42
doxy_core/Src/ display_driver.c	
Implements the methods from the Display class	44
doxy_core/Src/ encoder_handler.c	
This file implements the methods in the Encoder class	47
doxy_core/Src/ main.c	
: Main program body	50
doxy_core/Src/ motor_driver.c	
This file implements the Motor struct to allow for pseudo object oriented programming motor control	70
doxy_core/Src/ pitch_encoder_handler.c	
Implements the methods of the PitchEncoder class	74

Chapter 4

Data Structure Documentation

4.1 CLController Class Reference

An implementation of a PIDF (F = Feedforward constant) closed loop controller. It is a generic controller that takes in a measurement, controller constants, and a setpoint and outputs an effort to be sent to the actuator.

```
#include <CLController.h>
```

Data Fields

- uint32_t [kp](#)
- uint32_t [ki](#)
- uint32_t [kd](#)
- uint32_t [kf](#)
- int32_t [setpoint](#)
- int32_t [eff](#)
- int32_t [curr](#)
- int32_t [err](#)
- int32_t [err_acc](#)
- uint8_t [prev_err_index](#)
- uint32_t [initial_time](#)
- uint32_t [curr_time](#)
- int32_t [slope](#)
- uint32_t [prev_err_list_length](#)
- int32_t [prev_err_list](#) []

4.1.1 Detailed Description

An implementation of a PIDF (F = Feedforward constant) closed loop controller. It is a generic controller that takes in a measurement, controller constants, and a setpoint and outputs an effort to be sent to the actuator.

Parameters

<i>kp</i>	The propotional control constant multiplied by 10 ⁶ for accuracy reasons
<i>ki</i>	The integral control constant multiplied by 10 ⁶ for accuracy reasons
<i>kd</i>	The derivative control constant multiplied by 10 ⁶ for accuracy reasons

Parameters

<i>kf</i>	The Feedforward constant multiplied by 10^6 for accuracy reasons. The feedforward constant allows for simpler speed control in this project.
<i>setpoint</i>	The reference input value of the controller
<i>eff</i>	The output effort calculated by the closed loop controller
<i>curr</i>	The currently measured value of the sensor in the loop
<i>err</i>	The error between the setpoint and measured values, used for proportional control
<i>err_acc</i>	How much error has been accumulating over time, used to estimate an integral control

Note

the following parameters all have to do with derivative control. Due to the nasty things noise can do to derivative control, a simple impulse filter has been implemented. It works by accumulating a list of error values and taking the slope vs time of that list once the list is full, and using that derivative list until the list fills again. This effectively filters out impulses due to the averaging, and thus creates a smoother, noise-free derivative control.

Parameters

<i>prev_err_index</i>	the index for the list of previous errors
<i>initial_time</i>	The initial time when the controller began calculating derivative error data
<i>curr_time</i>	The current time at which derivative error has been calculated
<i>slope</i>	The overall slope of the list of derivative errors
<i>prev_err_list_length</i>	The number of error samples to be taken for the list, as chosen by the control programmer
<i>prev_err_list[]</i>	The list previous error values are stored in. @Attention this must be the same length as the previous_err_list_length

Definition at line 38 of file [CLController.h](#).

4.1.2 Field Documentation

4.1.2.1 curr

```
int32_t curr
```

Definition at line 45 of file [CLController.h](#).

4.1.2.2 curr_time

```
uint32_t curr_time
```

Definition at line 50 of file [CLController.h](#).

4.1.2.3 eff

```
int32_t eff
```

Definition at line 44 of file [CLController.h](#).

4.1.2.4 err

```
int32_t err
```

Definition at line 46 of file [CLController.h](#).

4.1.2.5 err_acc

```
int32_t err_acc
```

Definition at line 47 of file [CLController.h](#).

4.1.2.6 initial_time

```
uint32_t initial_time
```

Definition at line 49 of file [CLController.h](#).

4.1.2.7 kd

```
uint32_t kd
```

Definition at line 41 of file [CLController.h](#).

4.1.2.8 kf

```
uint32_t kf
```

Definition at line 42 of file [CLController.h](#).

4.1.2.9 ki

```
uint32_t ki
```

Definition at line 40 of file [CLController.h](#).

4.1.2.10 kp

```
uint32_t kp
```

Definition at line 39 of file [CLController.h](#).

4.1.2.11 prev_err_index

```
uint8_t prev_err_index
```

Definition at line 48 of file [CLController.h](#).

4.1.2.12 prev_err_list

```
int32_t prev_err_list[]
```

Definition at line 53 of file [CLController.h](#).

4.1.2.13 prev_err_list_length

```
uint32_t prev_err_list_length
```

Definition at line 52 of file [CLController.h](#).

4.1.2.14 setpoint

```
int32_t setpoint
```

Definition at line 43 of file [CLController.h](#).

4.1.2.15 slope

```
int32_t slope
```

Definition at line 51 of file [CLController.h](#).

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

- doxy_core/inc/[CLController.h](#)

4.2 Display Class Reference

The class that implements the control of the TLC59116 16-Channel FM+ I2C-Bus Constant-Current LED Sink Driver to display the current note on the tuner.

```
#include <display_driver.h>
```

Data Fields

- I2C_HandleTypeDef * [hi2c](#)
- uint8_t [curr_note](#)
- USART_TypeDef * [huart](#)

4.2.1 Detailed Description

The class that implements the control of the TLC59116 16-Channel FM+ I2C-Bus Constant-Current LED Sink Driver to display the current note on the tuner.

Parameters

<i>hi2c</i>	The i2c bus the display driver is on
<i>curr_note</i>	The current note to display, a value from 0-11 mapping from A-Ab
<i>huart</i>	The uart bus to provide serial communication to a computer console for displaying the current note in debug note. In the final version, this parameter will be removed as it is unnecessary to run the actual display, but for now we need it to see what notes are being displayed to the console.

Definition at line 24 of file [display_driver.h](#).

4.2.2 Field Documentation

4.2.2.1 curr_note

```
uint8_t curr_note
```

Definition at line 26 of file [display_driver.h](#).

4.2.2.2 hi2c

```
I2C_HandleTypeDef* hi2c
```

Definition at line 25 of file [display_driver.h](#).

4.2.2.3 huart

```
USART_TypeDef* huart
```

Definition at line 27 of file [display_driver.h](#).

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

- [doxy_core/Inc/display_driver.h](#)

4.3 Encoder Class Reference

A generic implementation of a quadrature encoder that tracks the position of the encoder in counts and the speed the encoder is traveling at in counts per second.

```
#include <encoder_handler.h>
```

Data Fields

- TIM_HandleTypeDef * [timer](#)
- TIM_HandleTypeDef * [timing_timer](#)
- uint32_t [prev_count](#)
- uint32_t [curr_count](#)
- uint32_t [prev_time](#)
- uint32_t [curr_time](#)
- int32_t [pos](#)
- int32_t [speed](#)
- int32_t [dx](#)
- int32_t [dt](#)

4.3.1 Detailed Description

A generic implementation of a quadrature encoder that tracks the position of the encoder in counts and the speed the encoder is traveling at in counts per second.

Parameters

<i>timer</i>	The timer the encoder channels are on.
<i>timing_timer</i>	The timer that is used to track time in order to calculate speed accurately.

Attention

The `timing_timer` must be configured such that it counts values in microseconds

Parameters

<i>prev_count</i>	The previous count timer read.
<i>curr_count</i>	The current count timer reads.
<i>prev_time</i>	The previous count <code>timing_timer</code> read.
<i>curr_time</i>	The current count <code>timing_timer</code> reads.
<i>pos</i>	The position of the encoder in counts.
<i>speed</i>	The speed the motor is traveling at in counts per second
<i>dx</i>	The difference in position between the previous encoder reading and the current encoder reading in counts
<i>dx</i>	The difference in time between the previous encoder reading and the current encoder reading in microseconds

Definition at line 32 of file [encoder_handler.h](#).

4.3.2 Field Documentation

4.3.2.1 curr_count

```
uint32_t curr_count
```

Definition at line 36 of file [encoder_handler.h](#).

4.3.2.2 curr_time

```
uint32_t curr_time
```

Definition at line 38 of file [encoder_handler.h](#).

4.3.2.3 dt

```
int32_t dt
```

Definition at line 42 of file [encoder_handler.h](#).

4.3.2.4 dx

```
int32_t dx
```

Definition at line 41 of file [encoder_handler.h](#).

4.3.2.5 pos

```
int32_t pos
```

Definition at line 39 of file [encoder_handler.h](#).

4.3.2.6 prev_count

```
uint32_t prev_count
```

Definition at line 35 of file [encoder_handler.h](#).

4.3.2.7 prev_time

```
uint32_t prev_time
```

Definition at line 37 of file [encoder_handler.h](#).

4.3.2.8 speed

```
int32_t speed
```

Definition at line 40 of file [encoder_handler.h](#).

4.3.2.9 timer

```
TIM_HandleTypeDef* timer
```

Definition at line 33 of file [encoder_handler.h](#).

4.3.2.10 timing_timer

```
TIM_HandleTypeDef* timing_timer
```

Definition at line 34 of file [encoder_handler.h](#).

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

- doxy_core/Inc/[encoder_handler.h](#)

4.4 Motor Class Reference

An implementation of a motor driver using a struct to emulate Object Oriented Programming. The motor has 4 parameters, timer which indicates the timer to be used to run the motor, channels, which indicates the channels to be used to run the motor, duty_cycle, the duty cycle to run the motor at, and enable_flag, which determines if the motor is allowed to run.

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

Data Fields

- TIM_HandleTypeDef * [timer](#)
- uint8_t [channels](#)
- int32_t [duty_cycle](#)
- uint8_t [enable_flag](#)

4.4.1 Detailed Description

An implementation of a motor driver using a struct to emulate Object Oriented Programming. The motor has 4 parameters, timer which indicates the timer to be used to run the motor, channels, which indicates the channels to be used to run the motor, duty_cycle, the duty cycle to run the motor at, and enable_flag, which determines if the motor is allowed to run.

Parameters

<i>timer</i>	The microcontroller timer to use to control the pwm in the motor. This is a timer pointer.
<i>channels</i>	The channels the PWM signal should run on. 1 corresponds to 1 and 2, and 2 corresponds to 3 and 4.
<i>duty_cycle</i>	The duty cycle of the motor from -ARR to ARR, which is 1000
<i>enable_flag</i>	The flag which indicates whether or not the motor should be allowed to run. A zero means the motor is disabled and a one means the motor is enabled.

Definition at line 29 of file [motor_driver.h](#).

4.4.2 Field Documentation

4.4.2.1 channels

```
uint8_t channels
```


Definition at line 31 of file [motor_driver.h](#).

4.4.2.2 duty_cycle

```
int32_t duty_cycle
```

Definition at line 32 of file [motor_driver.h](#).

4.4.2.3 enable_flag

```
uint8_t enable_flag
```

Definition at line 33 of file [motor_driver.h](#).

4.4.2.4 timer

```
TIM_HandleTypeDef* timer
```

Definition at line 30 of file [motor_driver.h](#).

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

- [doxy_core/Inc/motor_driver.h](#)

4.5 PitchEncoder Class Reference

This class, in an object oriented sense, inherits the [Encoder](#) class but specifically reads values from the pitch encoder to map them to a specific pitch.

```
#include <pitch_encoder_handler.h>
```

Data Fields

- int16_t [pitch](#)
- [Encoder](#) * [encoder](#)
- int16_t [delta](#)

4.5.1 Detailed Description

This class, in an object oriented sense, inherits the [Encoder](#) class but specifically reads values from the pitch encoder to map them to a specific pitch.

Parameters

<i>pitch</i>	The current pitch the PitchEncoder read
<i>encoder</i>	The base encoder the PitchEncoder uses (This is equivalent to inheriting the encoder class in an object-oriented language)
<i>delta</i>	The difference between the previous reading of the encoder and the current reading of the encoder.

Definition at line 23 of file [pitch_encoder_handler.h](#).

4.5.2 Field Documentation

4.5.2.1 delta

```
int16_t delta
```

Definition at line 26 of file [pitch_encoder_handler.h](#).

4.5.2.2 encoder

```
Encoder* encoder
```

Definition at line 25 of file [pitch_encoder_handler.h](#).

4.5.2.3 pitch

```
int16_t pitch
```

Definition at line 24 of file [pitch_encoder_handler.h](#).

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

- [doxy_core/Inc/pitch_encoder_handler.h](#)

4.6 RC_Controller Class Reference

An implementation of an RC controller that sends pulse widths from its two control axes—a trigger and a wheel—through infrared. The motor has parameters for the timer used for the input capture of the pulse widths, the pulse widths sent by the controller for each of the axes, the values of time of the rising edges and falling axes for each axis, and flags that indicate a falling edge has been detected and therefore a period should be calculated.

```
#include <controller_driver.h>
```

Data Fields

- TIM_HandleTypeDef * [timer](#)
- uint32_t [period1](#)
- uint32_t [period2](#)
- uint32_t [re1](#)
- uint32_t [re2](#)
- uint32_t [fe1](#)
- uint32_t [fe2](#)
- uint8_t [fe_flag1](#)
- uint8_t [fe_flag2](#)

4.6.1 Detailed Description

An implementation of an RC controller that sends pulse widths from its two control axes—a trigger and a wheel—through infrared. The motor has parameters for the timer used for the input capture of the pulse widths, the pulse widths sent by the controller for each of the axes, the values of time of the rising edges and falling axes for each axis, and flags that indicate a falling edge has been detected and therefore a period should be calculated.

Parameters

<i>timer</i>	The microcontroller timer to use to run the input capture of the RC controller. This is a timer pointer. The timer should be configured such that each count of the timer is read in microseconds, which in this case requires a prescaler of 79 and an auto-reload of 65535.
<i>period1</i>	The pulse width of the first axis
<i>period2</i>	The pulse width of the second axis
<i>re1</i>	Timer value on the rising edge captured on the first axis
<i>re2</i>	Timer value on the rising edge captured on the second axis
<i>fe1</i>	Timer value on the falling edge captured on the first axis
<i>fe2</i>	Timer value on the falling edge captured on the second axis
<i>fe_flag1</i>	Flag that is set by the input capture callback function to indicate a falling edge has been detected for the first axis.
<i>fe_flag2</i>	Flag that is set by the input capture callback function to indicate a falling edge has been detected for the first axis.

Definition at line 33 of file [controller_driver.h](#).

4.6.2 Field Documentation

4.6.2.1 fe1

```
uint32_t fe1
```

Definition at line 39 of file [controller_driver.h](#).

4.6.2.2 fe2

```
uint32_t fe2
```

Definition at line 40 of file [controller_driver.h](#).

4.6.2.3 fe_flag1

```
uint8_t fe_flag1
```

Definition at line 41 of file [controller_driver.h](#).

4.6.2.4 fe_flag2

```
uint8_t fe_flag2
```

Definition at line 42 of file [controller_driver.h](#).

4.6.2.5 period1

```
uint32_t period1
```

Definition at line 35 of file [controller_driver.h](#).

4.6.2.6 period2

```
uint32_t period2
```

Definition at line 36 of file [controller_driver.h](#).

4.6.2.7 re1

```
uint32_t re1
```

Definition at line 37 of file [controller_driver.h](#).

4.6.2.8 re2

```
uint32_t re2
```

Definition at line 38 of file [controller_driver.h](#).

4.6.2.9 timer

```
TIM_HandleTypeDef* timer
```

Definition at line 34 of file [controller_driver.h](#).

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

- [doxy_core/Inc/controller_driver.h](#)

Chapter 5

File Documentation

5.1 doxy_core/inc/CLController.h File Reference

This file defines the abstract class and methods for a closed-loop PIDF controller.

```
#include "stm32L4xx_hal.h"
```

Data Structures

- class [CLController](#)

An implementation of a PIDF (F = Feedforward constant) closed loop controller. It is a generic controller that takes in a measurement, controller constants, and a setpoint and outputs an effort to be sent to the actuator.

Functions

- uint32_t [run](#) ([CLController](#) *con, int32_t measured)
- void [reset_controller](#) ([CLController](#) *con)

5.1.1 Detailed Description

This file defines the abstract class and methods for a closed-loop PIDF controller.

Date

May 29, 2024

Author

Jared Sinasohn

Definition in file [CLController.h](#).

5.1.2 Function Documentation

5.1.2.1 reset_controller()

```
void reset_controller (  
    CLController * con )
```

This function hard resets the controller (keeping controller gains the same)

Parameters

<i>con</i>	The CLController instance to run the plant with
------------	---

Definition at line 50 of file [CLController.c](#).

```

00050                                     {
00051     con->err = 0;
00052     con->eff = 0;
00053     con->err_acc = 0;
00054     con->slope = 0;
00055     con->curr = 0;
00056     con->initial_time = HAL_GetTick();
00057     con->curr_time = HAL_GetTick();
00058 }
```

5.1.2.2 run()

```

uint32_t run (
    CLController * con,
    int32_t measured )
```

This function runs the Closed Loop Controller based on the parameters set up initially by the programmer

Parameters

<i>con</i>	The CLController instance to run the plant with
<i>measured</i>	The value measured by the sensor to be fed back into the controller

Returns

eff The effort/output calculated by the Controller to send to the actuator

This function runs the Closed Loop Controller based on the parameters set up initially by the programmer

Parameters

<i>con</i>	The CLController instance to run the plant with
<i>measured</i>	The value measured by the sensor to be fed back into the controller

Returns

eff The effort/output calculated by the Controller to send to the actuator

Attention

This function should be run at a rate slower than 1ms so as to not have zero time differential (if you are not using derivative control you can run at whatever speed you desire.)

store the sensor value as the current value

get the current time of simulation.

calculate the error between the setpoint and the measured value

add this error to the accumulated error, but only if integral control has been implemented.

set the effort to be the proportional control plus integral control plus feed-forward control

if derivative control is enabled, add the current error to the list of errors and increment the index

if the list of errors is full, calculate the slope of the list and multiply it by the derivative constant and add it to the effort

Definition at line 18 of file [CLController.c](#).

```
00018 {
00020     con->curr = measured;
00022     con->curr_time = HAL_GetTick();
00024     con->err = con->setpoint - con->curr;
00026     if(con->ki > 0){
00027         con->err_acc = con->err_acc + con->err;
00028     }
00030     con->eff = (con->kf * con->setpoint)/1000000 + (con->kp * con->err)/1000000 + (con->ki *
con->err_acc)/1000000;
00032     if(con->kd > 0){
00033         con->prev_err_list[con->prev_err_index] = con->err;
00034         con->prev_err_index += 1;
00035     }
00037     if(con->kd > 0 && con->prev_err_index >= con->prev_err_list_length){
00038         con->slope =
((con->prev_err_list[con->prev_err_index-1]-con->prev_err_list[0]))*1000/(con->curr_time -
con->initial_time);
00039         con->eff += con->kd * con->slope;
00040         con->prev_err_index = 0;
00041         con->initial_time = con->curr_time;
00042     }
00043     return con->eff;
00044 }
```

5.2 CLController.h

[Go to the documentation of this file.](#)

```
00001
00010 #ifndef SRC_CLCONTROLLER_H_
00011 #define SRC_CLCONTROLLER_H_
00012
00013 #include "stm32l4xx_hal.h"
00014
00038 typedef struct{
00039     uint32_t kp;
00040     uint32_t ki;
00041     uint32_t kd;
00042     uint32_t kf;
00043     int32_t setpoint;
00044     int32_t eff;
00045     int32_t curr;
00046     int32_t err;
00047     int32_t err_acc;
00048     uint8_t prev_err_index;
00049     uint32_t initial_time;
00050     uint32_t curr_time;
00051     int32_t slope;
00052     uint32_t prev_err_list_length;
00053     int32_t prev_err_list[];
00054 }CLController;
00055
00056
00066 uint32_t run(CLController* con, int32_t measured);
00072 void reset_controller(CLController* con);
00073
00074
00075
00076 #endif /* SRC_CLCONTROLLER_H_ */
```

5.3 doxy_core/Inc/controller_driver.h File Reference

A pseudo-object-oriented structure for implementing the calculation of a pulse-width read from a two-channel rc controller.

```
#include "stm32l4xx_hal.h"
```

Data Structures

- class [RC_Controller](#)

An implementation of an RC controller that sends pulse widths from its two control axes—a trigger and a wheel—through infrared. The motor has parameters for the timer used for the input capture of the pulse widths, the pulse widths sent by the controller for each of the axes, the values of time of the rising edges and falling axes for each axis, and flags that indicate a falling edge has been detected and therefore a period should be calculated.

Functions

- void [controller_driver_calc_per1](#) ([RC_Controller](#) *controller)
- void [controller_driver_calc_per2](#) ([RC_Controller](#) *controller)

5.3.1 Detailed Description

A pseudo-object-oriented structure for implementing the calculation of a pulse-width read from a two-channel rc controller.

Date

May 9, 2024

Author

Jared Sinasohn

Definition in file [controller_driver.h](#).

5.3.2 Function Documentation

5.3.2.1 controller_driver_calc_per1()

```
void controller_driver_calc_per1 (
    RC\_Controller * controller )
```

Calculates the pulse width sent by the first axis

Parameters

<i>controller,the</i>	RC Controller instance to caculate and operate on
-----------------------	---

Calculates the pulse width sent by the first axis. This number, if the timer is set up correctly, should be a number

Parameters

<i>controller,the</i>	RC Controller instance to caculate and operate on
-----------------------	---

Definition at line 15 of file [controller_driver.c](#).


```

00015                                     {
00016     uint32_t per1 = controller->fe1-controller->re1;
00017     if(per1 < 2050 && per1 > 950){
00018         controller->period1 = per1;
00019     }
00020 }

```

5.3.2.2 controller_driver_calc_per2()

```

void controller_driver_calc_per2 (
    RC_Controller * controller )

```

Calculates the pulse width sent by the second axis

Parameters

<i>controller,the</i>	RC Controller instance to caculate and operate on
-----------------------	---

Calculates the pulse width sent by the first axis. This number, if the timer is set up correctly, should be a number

Parameters

<i>controller,the</i>	RC Controller instance to caculate and operate on
-----------------------	---

Definition at line 26 of file [controller_driver.c](#).

```

00026                                     {
00027     uint32_t per2 = controller->fe2-controller->re2;
00028     if(per2 < 2050 && per2 > 950){
00029         controller->period2 = per2;
00030     }
00031 }

```

5.4 controller_driver.h

[Go to the documentation of this file.](#)

```

00001
00009 #ifndef SRC_CONTROLLER_DRIVER_H_
00010 #define SRC_CONTROLLER_DRIVER_H_
00011
00012
00013 #include "stm32l4xx_hal.h"
00014
00033 typedef struct {
00034     TIM_HandleTypeDef* timer; // Handle to the HAL timer object
00035     uint32_t period1; // pulse width of the first axis
00036     uint32_t period2; // pulse width of the second axis
00037     uint32_t re1; // Timer value on the rising edge captured on the first axis
00038     uint32_t re2; // Timer value on the rising edge captured on the second axis
00039     uint32_t fe1; // Timer value on the falling edge captured on the first axis
00040     uint32_t fe2; // Timer value on the falling edge captured on the second axis
00041     uint8_t fe_flag1; // flag that is set by the input capture callback function to indicate a
    falling edge has been detected for the first axis.
00042     uint8_t fe_flag2; // flag that is set by the input capture callback function to indicate a
    falling edge has been detected for the first axis.
00043 } RC_Controller;
00044
00050 void controller_driver_calc_per1(RC_Controller* controller);
00051
00057 void controller_driver_calc_per2(RC_Controller* controller);
00058
00059 #endif /* SRC_CONTROLLER_DRIVER_H_ */

```

5.5 doxy_core/inc/display_driver.h File Reference

The header file for the i2c driven 16 segment display class.

```
#include "stm32l4xx_hal.h"
#include "stdio.h"
```

Data Structures

- class [Display](#)

The class that implements the control of the TLC59116 16-Channel FM+ I2C-Bus Constant-Current LED Sink Driver to display the current note on the tuner.

Functions

- `uint8_t display_note (Display *disp, uint8_t note)`

5.5.1 Detailed Description

The header file for the i2c driven 16 segment display class.

Date

May 30, 2024

Author

Jared Sinasohn

Definition in file [display_driver.h](#).

5.5.2 Function Documentation

5.5.2.1 display_note()

```
uint8_t display_note (
    Display * disp,
    uint8_t note )
```

This function displays the current note on the display.

Parameters

<i>disp</i>	The display instance
<i>note, the</i>	note to be displayed from 0-11.

Returns

`curr_note` The note that is displayed.

This function displays the current note on the display.

Parameters

<i>disp</i>	The display instance
<i>note</i>	The note to be displayed from 0-11.

Returns

`curr_note` The note that is displayed.

Attention

The Current function uses uart to display the current note, but at the end would use i2c to the driver.

Definition at line 28 of file [display_driver.c](#).

```

00028                                     {
00029     if (disp->curr_note == note) {
00030         return disp->curr_note;
00031     }
00032     disp->curr_note = note;
00033     switch (disp->curr_note) {
00034         case 0:
00035             sprintf(Pitch_Buffer, "A \r\n");
00036             break;
00037         case 1:
00038             sprintf(Pitch_Buffer, "Bb \r\n");
00039             break;
00040         case 2:
00041             sprintf(Pitch_Buffer, "B \r\n");
00042             break;
00043         case 3:
00044             sprintf(Pitch_Buffer, "C \r\n");
00045             break;
00046         case 4:
00047             sprintf(Pitch_Buffer, "Db \r\n");
00048             break;
00049         case 5:
00050             sprintf(Pitch_Buffer, "D \r\n");
00051             break;
00052         case 6:
00053             sprintf(Pitch_Buffer, "Eb \r\n");
00054             break;
00055         case 7:
00056             sprintf(Pitch_Buffer, "E \r\n");
00057             break;
00058         case 8:
00059             sprintf(Pitch_Buffer, "F \r\n");
00060             break;
00061         case 9:
00062             sprintf(Pitch_Buffer, "Gb \r\n");
00063             break;
00064         case 10:
00065             sprintf(Pitch_Buffer, "G \r\n");
00066             break;
00067         case 11:
00068             sprintf(Pitch_Buffer, "Ab \r\n");
00069             break;
00070     }
00071     HAL_UART_Transmit(disp->huart, Pitch_Message, sizeof(Pitch_Message), 10000);
00072     HAL_UART_Transmit(disp->huart, Pitch_Buffer, sizeof(Pitch_Buffer), 10000);
00073     return disp->curr_note;
00074 }
```

5.6 display_driver.h

[Go to the documentation of this file.](#)

```
00001
00009 #ifndef SRC_DISPLAY_DRIVER_H_
00010 #define SRC_DISPLAY_DRIVER_H_
00011
00012 #include "stm32l4xx_hal.h"
00013 #include "stdio.h"
00014
00024 typedef struct{
00025     I2C_HandleTypeDef* hi2c;
00026     uint8_t curr_note;
00027     USART_TypeDef * huart;
00028 }Display;
00029
00035 uint8_t display_note(Display* disp, uint8_t note);
00036 #endif /* SRC_DISPLAY_DRIVER_H_ */
```

5.7 doxy_core/Inc/encoder_handler.h File Reference

The header file for a class that implements a generic quadrature encoder.

```
#include "stm32L4xx_hal.h"
```

Data Structures

- class [Encoder](#)

A generic implementation of a quadrature encoder that tracks the position of the encoder in counts and the speed the encoder is traveling at in counts per second.

Functions

- void [encoder_read_curr_state](#) ([Encoder](#) *encoder)
- int32_t [encoder_calc_speed](#) ([Encoder](#) *encoder, int32_t dx, int32_t dt)
- void [zero](#) ([Encoder](#) *encoder)
- int32_t [delta](#) (TIM_HandleTypeDef *timer, uint32_t initial, uint32_t final)

5.7.1 Detailed Description

The header file for a class that implements a generic quadrature encoder.

Date

May 18, 2024

Author

Jared Sinasohn

Definition in file [encoder_handler.h](#).

5.7.2 Function Documentation

5.7.2.1 `delta()`

```
int32_t delta (  
    TIM_HandleTypeDef * timer,  
    uint32_t initial,  
    uint32_t final )
```

This function calculates a delta between two values from a timer, while accounting for overflow.

Parameters

<i>timer</i>	The timer the initial and final values come from
<i>initial</i>	The initial value of the timer
<i>final</i>	The final value of the timer

Returns

delta The corrected delta value to account for overflow and underflow.

get the auto reload value of the timer since it is the maximum value a timer can be

we can determine if something has overflowed or underflowed by assuming a delta will never be greater than half the auto reload value, which if the encoder is read enough is a good assumption

if the value underflows, the delta will be a positive value greater than overflow, so just subtract off ARR+1 from the underflowed delta

similarly, if the value overflows, the delta will be a negative value less than negative of overflow, so add ARR+1 to the overflowed delta

Definition at line 71 of file [encoder_handler.c](#).

```

00071                                     {
00073     uint32_t ARR = (int32_t)(timer->Init.Period);
00075     int32_t overflow = ((ARR-1)/2)+1;
00076     //1 Calculate the delta
00077     int32_t delta = final-initial;
00079     if(delta >= overflow){
00080         delta = delta - overflow*2;
00082     }else if(delta <= -1*overflow){
00083         delta = delta + overflow;
00084     }
00085     return delta;
00086 }
```

5.7.2.2 encoder_calc_speed()

```

int32_t encoder_calc_speed (
    Encoder * encoder,
    int32_t dx,
    int32_t dt )
```

This function calculates a speed based on a differential position and a differential time.

Parameters

<i>encoder</i>	The encoder instance to operate on
<i>dx</i>	The differential position term in encoder counts.
<i>dt</i>	The differential time term in microseconds

Returns

The calculated speed in counts per second

if the delta time is zero, return the previous speed to avoid divide by zero errors.

return the speed in counts/second knowing that dt is in microseconds

Definition at line 39 of file [encoder_handler.c](#).

```
00039
00041     if(dt == 0){
00042         return encoder->speed;
00043     }
00045     return ((dx)*1000000)/dt;
00046 }
```

5.7.2.3 encoder_read_curr_state()

```
void encoder_read_curr_state (
    Encoder * encoder )
```

This function reads the current encoder state and uses it to calculate the speed and position of the encoder.

Parameters

<i>encoder</i>	The encoder instance to operate on.
----------------	-------------------------------------

Attention

If the `timing_timer` is set up correctly for the inputted encoder, this function should be called no faster than every microsecond.

set the previous times and previous counts to the previously current values

get the count and time values from the two timers

calculate the difference between the current counts/times and previous counts/times using the delta function

set the encoder position to be the previous encoder position plus the delta position

set the speed of the encoder to the calculated value via [encoder_calc_speed\(\)](#)

Definition at line 15 of file [encoder_handler.c](#).

```
00015
00017     encoder->prev_count = encoder->curr_count;
00018     encoder->prev_time = encoder->curr_time;
00020     encoder->curr_count = encoder->timer->Instance->CNT;
00021     encoder->curr_time = encoder->timing_timer->Instance->CNT;
00023     encoder->dx = delta(encoder->timer, encoder->prev_count, encoder->curr_count);
00024     encoder->dt = delta(encoder->timing_timer, encoder->prev_time, encoder->curr_time);
00026     encoder->pos = encoder->pos + encoder->dx;
00028     encoder->speed = encoder_calc_speed(encoder, encoder->dx, encoder->dt);
00029
00030 }
```

5.7.2.4 zero()

```
void zero (
    Encoder * encoder )
```

This function zeros the encoder, including the `timing_timer` and timer timers.

Parameters

<i>encoder</i>	The encoder instance to operate on
----------------	------------------------------------

Definition at line 52 of file [encoder_handler.c](#).

```

00052     {
00053         encoder->timer->Instance->CNT = 0;
00054         encoder->timing_timer->Instance->CNT = 0;
00055         encoder->prev_count = 0;
00056         encoder->curr_count = 0;
00057         encoder->prev_time = 0;
00058         encoder->curr_time = 1;
00059         encoder->pos = 0;
00060         encoder->speed = 0;
00061     }

```

5.8 encoder_handler.h

[Go to the documentation of this file.](#)

```

00001
00009 #ifndef SRC_ENCODER_HANDLER_H_
00010 #define SRC_ENCODER_HANDLER_H_
00011
00012 #include "stm32L4xx_hal.h"
00013
00014
00032 typedef struct{
00033     TIM_HandleTypeDef* timer;
00034     TIM_HandleTypeDef* timing_timer;
00035     uint32_t prev_count;
00036     uint32_t curr_count;
00037     uint32_t prev_time;
00038     uint32_t curr_time;
00039     int32_t pos;
00040     int32_t speed;
00041     int32_t dx;
00042     int32_t dt;
00043 }Encoder;
00044
00050 void encoder_read_curr_state(Encoder* encoder);
00051
00059 int32_t encoder_calc_speed(Encoder* encoder,int32_t dx, int32_t dt);
00060
00065 void zero(Encoder* encoder);
00066
00075 int32_t delta(TIM_HandleTypeDef* timer, uint32_t initial, uint32_t final);
00076
00077 #endif /* SRC_ENCODER_HANDLER_H_ */

```

5.9 doxy_core/Inc/main.h File Reference

: Header for [main.c](#) file. This file contains the common defines of the application.

```
#include "stm32l4xx_hal.h"
```

Macros

- `#define USART_TX_Pin` GPIO_PIN_2
- `#define USART_TX_GPIO_Port` GPIOA
- `#define USART_RX_Pin` GPIO_PIN_3
- `#define USART_RX_GPIO_Port` GPIOA
- `#define TMS_Pin` GPIO_PIN_13
- `#define TMS_GPIO_Port` GPIOA
- `#define TCK_Pin` GPIO_PIN_14
- `#define TCK_GPIO_Port` GPIOA

Functions

- void [HAL_TIM_MspPostInit](#) (TIM_HandleTypeDef *htim)
- void [Error_Handler](#) (void)

This function is executed in case of error occurrence.

5.9.1 Detailed Description

: Header for [main.c](#) file. This file contains the common defines of the application.

!

Attention

Copyright (c) 2024 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [main.h](#).

5.9.2 Macro Definition Documentation

5.9.2.1 TCK_GPIO_Port

```
#define TCK_GPIO_Port GPIOA
```

Definition at line 69 of file [main.h](#).

5.9.2.2 TCK_Pin

```
#define TCK_Pin GPIO_PIN_14
```

Definition at line 68 of file [main.h](#).

5.9.2.3 TMS_GPIO_Port

```
#define TMS_GPIO_Port GPIOA
```

Definition at line 67 of file [main.h](#).

5.9.2.4 TMS_Pin

```
#define TMS_Pin GPIO_PIN_13
```

Definition at line 66 of file [main.h](#).

5.9.2.5 USART_RX_GPIO_Port

```
#define USART_RX_GPIO_Port GPIOA
```

Definition at line 65 of file [main.h](#).

5.9.2.6 USART_RX_Pin

```
#define USART_RX_Pin GPIO_PIN_3
```

Definition at line 64 of file [main.h](#).

5.9.2.7 USART_TX_GPIO_Port

```
#define USART_TX_GPIO_Port GPIOA
```

Definition at line 63 of file [main.h](#).

5.9.2.8 USART_TX_Pin

```
#define USART_TX_Pin GPIO_PIN_2
```

Definition at line 62 of file [main.h](#).

5.9.3 Function Documentation

5.9.3.1 Error_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

<i>None</i>	
-------------	--

Definition at line 1054 of file [main.c](#).

```
01055 {
01056     /* USER CODE BEGIN Error_Handler_Debug */
01057     /* User can add his own implementation to report the HAL error return state */
01058     __disable_irq();
01059     while (1)
01060     {
01061     }
01062     /* USER CODE END Error_Handler_Debug */
01063 }
```

5.9.3.2 HAL_TIM_MspPostInit()

```
void HAL_TIM_MspPostInit (
```

```
TIM_HandleTypeDef * htim )
```

5.10 main.h

[Go to the documentation of this file.](#)

```
00001 /* USER CODE BEGIN Header */
00019 /* USER CODE END Header */
00020
00021 /* Define to prevent recursive inclusion -----*/
00022 #ifndef __MAIN_H
00023 #define __MAIN_H
00024
00025 #ifdef __cplusplus
00026 extern "C" {
00027 #endif
00028
00029 /* Includes -----*/
00030 #include "stm32l4xx_hal.h"
00031
00032 /* Private includes -----*/
00033 /* USER CODE BEGIN Includes */
00034
00035 /* USER CODE END Includes */
00036
00037 /* Exported types -----*/
00038 /* USER CODE BEGIN ET */
00039
00040 /* USER CODE END ET */
00041
00042 /* Exported constants -----*/
00043 /* USER CODE BEGIN EC */
00044
00045 /* USER CODE END EC */
00046
00047 /* Exported macro -----*/
00048 /* USER CODE BEGIN EM */
00049
00050 /* USER CODE END EM */
00051
00052 void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
00053
00054 /* Exported functions prototypes -----*/
00055 void Error_Handler(void);
00056
00057 /* USER CODE BEGIN EFP */
00058
00059 /* USER CODE END EFP */
00060
00061 /* Private defines -----*/
00062 #define USART_TX_Pin GPIO_PIN_2
00063 #define USART_TX_GPIO_Port GPIOA
00064 #define USART_RX_Pin GPIO_PIN_3
00065 #define USART_RX_GPIO_Port GPIOA
00066 #define TMS_Pin GPIO_PIN_13
00067 #define TMS_GPIO_Port GPIOA
00068 #define TCK_Pin GPIO_PIN_14
00069 #define TCK_GPIO_Port GPIOA
00070
00071 /* USER CODE BEGIN Private defines */
00072
00073 /* USER CODE END Private defines */
00074
00075 #ifdef __cplusplus
00076 }
00077 #endif
00078
00079 #endif /* __MAIN_H */
```

5.11 doxy_core/inc/motor_driver.h File Reference

The header file for a generic DC motor driven from a standard h-bridge motor driver.

```
#include "stm32l4xx_hal.h"
```

Data Structures

- class [Motor](#)

An implementation of a motor driver using a struct to emulate Object Oriented Programming. The motor has 4 parameters, timer which indicates the timer to be used to run the motor, channels, which indicates the channels to be used to run the motor, duty_cycle, the duty cycle to run the motor at, and enable_flag, which determines if the motor is allowed to run.

Functions

- void [motor_set_duty_cycle](#) ([Motor](#) *motor, int32_t doot)
- void [motor_enable_disable](#) ([Motor](#) *motor, uint8_t enable)

5.11.1 Detailed Description

The header file for a generic DC motor driven from a standard h-bridge motor driver.

Date

Apr 25, 2024

Author

Jared Sinasohn

Definition in file [motor_driver.h](#).

5.11.2 Function Documentation

5.11.2.1 motor_enable_disable()

```
void motor_enable_disable (  
    Motor * motor,  
    uint8_t enable )
```

Enables or disables motor based on user input

Parameters

<i>motor, the</i>	Motor struct to act upon
<i>enable, the</i>	boolean of whether to enable or disable the motor with 1 being to enable and 0 being to disable.

Definition at line 87 of file [motor_driver.c](#).

```
00087                                     {  
00088     // if user wants to enable motor  
00089     if(enable == 1){  
00090         motor->enable_flag = 1;  
00091         // First retrieve ARR to set motor to brake mode  
00092         uint32_t ARR = (uint32_t)(motor->timer->Init.Period + 1);  
00093  
00094         // Now set the correct motor pair to brake mode.
```

```

00095         if(motor->channels == 1){
00096             motor->timer->Instance->CCR1 = ARR;
00097             motor->timer->Instance->CCR2 = ARR;
00098         } else if(motor->channels == 2){
00099             motor->timer->Instance->CCR3 = ARR;
00100             motor->timer->Instance->CCR4 = ARR;
00101         }else{
00102             return;
00103         }
00104
00105         // set the motor's enable flag to 1
00106         motor->enable_flag = 1;
00107
00108         // if user wants to disable motor
00109     } else if(enable == 0){
00110         motor->enable_flag = 0;
00111         if(motor->channels == 1){
00112             motor->timer->Instance->CCR1 = 0;
00113             motor->timer->Instance->CCR2 = 0;
00114         } else if(motor->channels == 2){
00115             motor->timer->Instance->CCR3 = 0;
00116             motor->timer->Instance->CCR4 = 0;
00117         }else{
00118             return;
00119         }
00120
00121         // set the motor's enable flag to 0
00122         motor->enable_flag = 0;
00123     }
00124 }

```

5.11.2.2 motor_set_duty_cycle()

```

void motor_set_duty_cycle (
    Motor * motor,
    int32_t doot )

```

This function implements the duty cycle setting of the motor. It takes in the motor struct and a duty cycle from -100 to 100 (though the function saturates values above and below these values).

Parameters

<i>motor,the</i>	Motor struct to be operated on.
<i>doot,the</i>	duty cycle to be set to.

Definition at line 21 of file [motor_driver.c](#).

```

00021     {
00022         motor->duty_cycle = doot;
00023         // First, check if the motor is disabled
00024         if(motor->enable_flag != 1){
00025             // if the enable flag isn't set exit the function and do nothing.
00026             // we are also using != 1 so if there is a stray value in memory,
00027             // the motor doesn't accidentally enable.
00028             return;
00029         }
00030
00031         // Next, saturate the duty cycle just in case.
00032         if(doot < -100){
00033             doot = -100;
00034         }
00035         if(doot > 100){
00036             doot = 100;
00037         }
00038
00039         // We need to get the auto reload value for the timer we are using
00040         // signed value so we don't run into sign issues later
00041         int32_t ARR = (int32_t)(motor->timer->Init.Period + 1);
00042
00043         // Now calculate the duty cycle in terms of the CCR value
00044         doot = doot*ARR/100; // multiply first so we don't lose data
00045
00046         // now we need to set the motors to the correct duty cycles
00047         // Forwards will be channels 1 and 3 for motors 1 and 2 respectively
00048         // Backwards will be channels 2 and 4 for motors 1 and 2 respectively

```

```

00049
00050
00051 // the below CCR's are based on the logic table of the toshiba, setting motor.
00052 // to brake mode
00053 // if duty cycle is <0
00054 if (doot < 0){
00055     // check if it is the first or second motor.
00056     if(motor->channels == 1){
00057         motor->timer->Instance->CCR1 = ARR;
00058         motor->timer->Instance->CCR2 = ARR + doot;
00059     } else if(motor->channels == 2){
00060         motor->timer->Instance->CCR3 = ARR;
00061         motor->timer->Instance->CCR4 = ARR + doot;
00062     }else{
00063         // if neither return
00064         return;
00065     }
00066 // if duty cycle >=0
00067 } else{
00068     if(motor->channels == 1){
00069         motor->timer->Instance->CCR1 = ARR - doot;
00070         motor->timer->Instance->CCR2 = ARR;
00071     } else if(motor->channels == 2){
00072         motor->timer->Instance->CCR3 = ARR - doot;
00073         motor->timer->Instance->CCR4 = ARR;
00074     }else{
00075         return;
00076     }
00077 }
00078 }

```

5.12 motor_driver.h

[Go to the documentation of this file.](#)

```

00001
00008 #ifndef SRC_MOTOR_DRIVER_H_
00009 #define SRC_MOTOR_DRIVER_H_
00010
00011 //include hal library
00012 #include "stm32l4xx_hal.h"
00013
00029 typedef struct {
00030     TIM_HandleTypeDef* timer; // Handle to the HAL timer object
00031     uint8_t channels; // which channels to use. using channels 1-4 so 1 corresponds to 1 and 2 and 2
00032     // corresponds to 3 and 4
00033     int32_t duty_cycle; // duty cycle of the motor from -ARR to ARR, which is 1000
00034     uint8_t enable_flag; // flag that enables the motor
00035 } Motor;
00044 void motor_set_duty_cycle(Motor* motor, int32_t doot);
00045
00053 void motor_enable_disable(Motor* motor, uint8_t enable);
00054
00055
00056 #endif /* SRC_MOTOR_DRIVER_H_ */

```

5.13 doxy_core/Inc/pitch_encoder_handler.h File Reference

```

#include "stm32L4xx_hal.h"
#include "encoder_handler.h"

```

Data Structures

- class [PitchEncoder](#)

This class, in an object oriented sense, inherits the [Encoder](#) class but specifically reads values from the pitch encoder to map them to a specific pitch.

Functions

- uint32_t [get_pitch](#) ([PitchEncoder](#) *p_enc)

5.13.1 Detailed Description

Date

May 30, 2024

Author

Jared Sinasohn

Definition in file [pitch_encoder_handler.h](#).

5.13.2 Function Documentation

5.13.2.1 [get_pitch\(\)](#)

```
uint32_t get_pitch (
    PitchEncoder * p_enc )
```

This function gets the current pitch based on the pitch selection knob

Parameters

<i>p_enc</i>	The pitch encoder object to read from
--------------	---------------------------------------

Returns

the current pitch, a number 0-11 mapped through the chromatic notes from A-Ab

read the current state of the encoder

store the delta of the encoder

add the delta to the pitch

if the pitch hasn't changed, just return the pitch

we can treat the pitch as a number between 0 and 11, which can underflow and overflow. We can run a similar algorithm to the [delta\(\)](#) function in the encoder class to correct for this.

Definition at line 15 of file [pitch_encoder_handler.c](#).

```
00015         {
00017     encoder_read_curr_state((p_enc->encoder));
00019     p_enc->delta = (int16_t)(p_enc->encoder->dx);
00021     p_enc->pitch += p_enc->delta;
00023     if(p_enc->delta == 0){
00024         return p_enc->pitch;
00025     }
00027     if(p_enc->pitch < 0){
00028         p_enc->pitch += 12;
00029     }
00030     if(p_enc->pitch >= 12){
00031         p_enc->pitch -= 12;
00032     }
00033     return p_enc->pitch;
00034 }
```

5.14 pitch_encoder_handler.h

[Go to the documentation of this file.](#)

```
00001
00008 #ifndef SRC_PITCH_ENCODER_HANDLER_H_
00009 #define SRC_PITCH_ENCODER_HANDLER_H_
00010
00011 #include "stm32L4xx_hal.h"
00012 #include "encoder_handler.h"
00013
00023 typedef struct{
00024     int16_t pitch;
00025     Encoder* encoder;
00026     int16_t delta;
00027 }PitchEncoder;
00028
00034 uint32_t get_pitch(PitchEncoder* p_enc);
00035
00036 #endif /* SRC_PITCH_ENCODER_HANDLER_H_ */
```

5.15 doxy_core/README.md File Reference

5.16 doxy_core/Src/CLController.c File Reference

this file implements the methods for the [CLController](#) class.

```
#include "CLController.h"
```

Functions

- uint32_t [run](#) ([CLController](#) *con, int32_t measured)
- void [reset_controller](#) ([CLController](#) *con)

5.16.1 Detailed Description

this file implements the methods for the [CLController](#) class.

Date

May 29, 2024

Author

Jared Sinasohn

Definition in file [CLController.c](#).

5.16.2 Function Documentation

5.16.2.1 reset_controller()

```
void reset_controller (
    CLController * con )
```

This function hard resets the controller (keeping controller gains the same)

Parameters

<i>con</i>	The CLController instance to run the plant with
------------	---

Definition at line 50 of file [CLController.c](#).

```

00050                                     {
00051     con->err = 0;
00052     con->eff = 0;
00053     con->err_acc = 0;
00054     con->slope = 0;
00055     con->curr = 0;
00056     con->initial_time = HAL_GetTick();
00057     con->curr_time = HAL_GetTick();
00058 }
```

5.16.2.2 run()

```

uint32_t run (
    CLController * con,
    int32_t measured )
```

This function runs the Closed Loop Controller based on the parameters set up initially by the programmer

Parameters

<i>con</i>	The CLController instance to run the plant with
<i>measured</i>	The value measured by the sensor to be fed back into the controller

Returns

eff The effort/output calculated by the Controller to send to the actuator

Attention

This function should be run at a rate slower than 1ms so as to not have zero time differential (if you are not using derivative control you can run at whatever speed you desire.)

store the sensor value as the current value

get the current time of simulation.

calculate the error between the setpoint and the measured value

add this error to the accumulated error, but only if integral control has been implemented.

set the effort to be the proportional control plus integral control plus feed-forward control

if derivative control is enabled, add the current error to the list of errors and increment the index

if the list of errors is full, calculate the slope of the list and multiply it by the derivative constant and add it to the effort

Definition at line 18 of file [CLController.c](#).

```

00018                                     {
00020     con->curr = measured;
00022     con->curr_time = HAL_GetTick();
00024     con->err = con->setpoint - con->curr;
00026     if (con->ki > 0) {
```

```

00027         con->err_acc = con->err_acc + con->err;
00028     }
00030     con->eff = (con->kf * con->setpoint)/1000000 + (con->kp * con->err)/1000000 + (con->ki *
con->err_acc)/1000000;
00032     if(con->kd > 0){
00033         con->prev_err_list[con->prev_err_index] = con->err;
00034         con->prev_err_index += 1;
00035     }
00037     if(con->kd > 0 && con->prev_err_index >= con->prev_err_list_length){
00038         con->slope =
((con->prev_err_list[con->prev_err_index-1]-con->prev_err_list[0]))*1000/(con->curr_time -
con->initial_time);
00039         con->eff += con->kd * con->slope;
00040         con->prev_err_index = 0;
00041         con->initial_time = con->curr_time;
00042     }
00043     return con->eff;
00044 }

```

5.17 CLController.c

[Go to the documentation of this file.](#)

```

00001
00008 #include "CLController.h"
00009
00018 uint32_t run(CLController* con, int32_t measured){
00020     con->curr = measured;
00022     con->curr_time = HAL_GetTick();
00024     con->err = con->setpoint - con->curr;
00026     if(con->ki > 0){
00027         con->err_acc = con->err_acc + con->err;
00028     }
00030     con->eff = (con->kf * con->setpoint)/1000000 + (con->kp * con->err)/1000000 + (con->ki *
con->err_acc)/1000000;
00032     if(con->kd > 0){
00033         con->prev_err_list[con->prev_err_index] = con->err;
00034         con->prev_err_index += 1;
00035     }
00037     if(con->kd > 0 && con->prev_err_index >= con->prev_err_list_length){
00038         con->slope =
((con->prev_err_list[con->prev_err_index-1]-con->prev_err_list[0]))*1000/(con->curr_time -
con->initial_time);
00039         con->eff += con->kd * con->slope;
00040         con->prev_err_index = 0;
00041         con->initial_time = con->curr_time;
00042     }
00043     return con->eff;
00044 }
00050 void reset_controller(CLController* con){
00051     con->err = 0;
00052     con->eff = 0;
00053     con->err_acc = 0;
00054     con->slope = 0;
00055     con->curr = 0;
00056     con->initial_time = HAL_GetTick();
00057     con->curr_time = HAL_GetTick();
00058 }

```

5.18 doxy_core/Src/controller_driver.c File Reference

This file implements the methods to calculate the period for the [RC_Controller](#) class.

```
#include "controller_driver.h"
```

Functions

- void [controller_driver_calc_per1](#) ([RC_Controller](#) *controller)
- void [controller_driver_calc_per2](#) ([RC_Controller](#) *controller)

5.18.1 Detailed Description

This file implements the methods to calculate the period for the [RC_Controller](#) class.

Date

May 9, 2024

Author

Jared Sinasohn

Definition in file [controller_driver.c](#).

5.18.2 Function Documentation

5.18.2.1 controller_driver_calc_per1()

```
void controller_driver_calc_per1 (
    RC_Controller * controller )
```

Calculates the pulse width sent by the first axis. This number, if the timer is set up correctly, should be a number

Parameters

<i>controller, the</i>	RC Controller instance to calculate and operate on
------------------------	--

Definition at line 15 of file [controller_driver.c](#).

```
00015 {
00016     uint32_t per1 = controller->fe1-controller->re1;
00017     if(per1 < 2050 && per1 > 950){
00018         controller->period1 = per1;
00019     }
00020 }
```

5.18.2.2 controller_driver_calc_per2()

```
void controller_driver_calc_per2 (
    RC_Controller * controller )
```

Calculates the pulse width sent by the first axis. This number, if the timer is set up correctly, should be a number

Parameters

<i>controller, the</i>	RC Controller instance to calculate and operate on
------------------------	--

Definition at line 26 of file [controller_driver.c](#).

```
00026 {
00027     uint32_t per2 = controller->fe2-controller->re2;
00028     if(per2 < 2050 && per2 > 950){
00029         controller->period2 = per2;
00030     }
00031 }
```

5.19 controller_driver.c

[Go to the documentation of this file.](#)

```
00001
00008 #include "controller_driver.h"
00009
00015 void controller_driver_calc_per1(RC_Controller* controller){
00016     uint32_t per1 = controller->fe1-controller->re1;
00017     if(per1 < 2050 && per1 > 950){
00018         controller->period1 = per1;
00019     }
00020 }
00026 void controller_driver_calc_per2(RC_Controller* controller){
00027     uint32_t per2 = controller->fe2-controller->re2;
00028     if(per2 < 2050 && per2 > 950){
00029         controller->period2 = per2;
00030     }
00031 }
```

5.20 doxy_core/Src/display_driver.c File Reference

Implements the methods from the [Display](#) class.

```
#include "display_driver.h"
```

Functions

- [uint8_t display_note](#) ([Display](#) *disp, [uint8_t](#) note)

Variables

- [uint16_t note_addresses](#) [12] = {0b1101100001111000,0b1101101101110101,0b1101101101110100,0b1101101100000000,0b1101101101110101,0b1101101101110100,0b1101101101110101,0b1101101101110100,0b1101101101110101,0b1101101101110100,0b1101101101110101,0b1101101101110100}
- [uint16_t disp_addr](#) = 0b11000001
- [uint8_t Pitch_Message](#) [] = "Current Pitch: "
- [uint8_t Pitch_Buffer](#) [50] = {0}

5.20.1 Detailed Description

Implements the methods from the [Display](#) class.

Date

Jun 9, 2024

Author

Jared Sinasohn

Definition in file [display_driver.c](#).

5.20.2 Function Documentation

5.20.2.1 display_note()

```
uint8_t display_note (
    Display * disp,
    uint8\_t note )
```

This function displays the current note on the display.

Parameters

<i>disp</i>	The display instance
<i>note</i>	The note to be displayed from 0-11.

Returns

curr_note The note that is displayed.

Attention

The Current function uses uart to display the current note, but at the end would use i2c to the driver.

Definition at line 28 of file [display_driver.c](#).

```

00028                                     {
00029     if (disp->curr_note == note) {
00030         return disp->curr_note;
00031     }
00032     disp->curr_note = note;
00033     switch (disp->curr_note) {
00034         case 0:
00035             sprintf(Pitch_Buffer, "A \r\n");
00036             break;
00037         case 1:
00038             sprintf(Pitch_Buffer, "Bb \r\n");
00039             break;
00040         case 2:
00041             sprintf(Pitch_Buffer, "B \r\n");
00042             break;
00043         case 3:
00044             sprintf(Pitch_Buffer, "C \r\n");
00045             break;
00046         case 4:
00047             sprintf(Pitch_Buffer, "Db \r\n");
00048             break;
00049         case 5:
00050             sprintf(Pitch_Buffer, "D \r\n");
00051             break;
00052         case 6:
00053             sprintf(Pitch_Buffer, "Eb \r\n");
00054             break;
00055         case 7:
00056             sprintf(Pitch_Buffer, "E \r\n");
00057             break;
00058         case 8:
00059             sprintf(Pitch_Buffer, "F \r\n");
00060             break;
00061         case 9:
00062             sprintf(Pitch_Buffer, "Gb \r\n");
00063             break;
00064         case 10:
00065             sprintf(Pitch_Buffer, "G \r\n");
00066             break;
00067         case 11:
00068             sprintf(Pitch_Buffer, "Ab \r\n");
00069             break;
00070     }
00071     HAL_UART_Transmit(disp->huart, Pitch_Message, sizeof(Pitch_Message), 10000);
00072     HAL_UART_Transmit(disp->huart, Pitch_Buffer, sizeof(Pitch_Buffer), 10000);
00073     return disp->curr_note;
00074 }

```

5.20.3 Variable Documentation

5.20.3.1 disp_addr

disp_addr = 0b11000001

The address of the display driver on the i2c bus

Definition at line 18 of file [display_driver.c](#).


```

00059         sprintf(Pitch_Buffer, "F \r\n");
00060         break;
00061     case 9:
00062         sprintf(Pitch_Buffer, "Gb \r\n");
00063         break;
00064     case 10:
00065         sprintf(Pitch_Buffer, "G \r\n");
00066         break;
00067     case 11:
00068         sprintf(Pitch_Buffer, "Ab \r\n");
00069         break;
00070     }
00071     HAL_UART_Transmit (disp->huart, Pitch_Message, sizeof(Pitch_Message), 10000);
00072     HAL_UART_Transmit (disp->huart, Pitch_Buffer, sizeof(Pitch_Buffer), 10000);
00073     return disp->curr_note;
00074 }
00075

```

5.22 doxy_core/Src/encoder_handler.c File Reference

This file implements the methods in the [Encoder](#) class.

```
#include "encoder_handler.h"
```

Functions

- void [encoder_read_curr_state](#) ([Encoder](#) *encoder)
- int32_t [encoder_calc_speed](#) ([Encoder](#) *encoder, int32_t dx, int32_t dt)
- void [zero](#) ([Encoder](#) *encoder)
- int32_t [delta](#) (TIM_HandleTypeDef *timer, uint32_t initial, uint32_t final)

5.22.1 Detailed Description

This file implements the methods in the [Encoder](#) class.

Date

May 23, 2024

Author

Jared Sinasohn

Definition in file [encoder_handler.c](#).

5.22.2 Function Documentation

5.22.2.1 [delta\(\)](#)

```

int32_t delta (
    TIM_HandleTypeDef * timer,
    uint32_t initial,
    uint32_t final )

```

This function calculates a delta between two values from a timer, while accounting for overflow.

Parameters

<i>timer</i>	The timer the initial and final values come from
<i>initial</i>	The initial value of the timer
<i>final</i>	The final value of the timer

Returns

delta The corrected delta value to account for overflow and underflow.

get the auto reload value of the timer since it is the maximum value a timer can be

we can determine if something has overflowed or underflowed by assuming a delta will never be greater than half the auto reload value, which if the encoder is read enough is a good assumption

if the value underflows, the delta will be a positive value greater than overflow, so just subtract off ARR+1 from the underflowed delta

similarly, if the value overflows, the delta will be a negative value less than negative of overflow, so add ARR+1 to the overflowed delta

Definition at line 71 of file [encoder_handler.c](#).

```

00071                                     {
00073     uint32_t ARR = (int32_t)(timer->Init.Period);
00075     int32_t overflow = ((ARR-1)/2)+1;
00076     //1 Calculate the delta
00077     int32_t delta = final-initial;
00079     if(delta >= overflow){
00080         delta = delta - overflow*2;
00082     }else if(delta <= -1*overflow){
00083         delta = delta + overflow;
00084     }
00085     return delta;
00086 }
```

5.22.2.2 encoder_calc_speed()

```

int32_t encoder_calc_speed (
    Encoder * encoder,
    int32_t dx,
    int32_t dt )
```

This function calculates a speed based on a differential position and a differential time.

Parameters

<i>encoder</i>	The encoder instance to operate on
<i>dx</i>	The differential position term in encoder counts.
<i>dt</i>	The differential time term in microseconds

Returns

The calculated speed in counts per second

if the delta time is zero, return the previous speed to avoid divide by zero errors.

return the speed in counts/second knowing that dt is in microseconds

Definition at line 39 of file [encoder_handler.c](#).

```
00039
00041     if(dt == 0){
00042         return encoder->speed;
00043     }
00045     return ((dx)*1000000)/dt;
00046 }
```

5.22.2.3 encoder_read_curr_state()

```
void encoder_read_curr_state (
    Encoder * encoder )
```

This function reads the current encoder state and uses it to calculate the speed and position of the encoder.

Parameters

<i>encoder</i>	The encoder instance to operate on.
----------------	-------------------------------------

Attention

If the timing_timer is set up correctly for the inputted encoder, this function should be called no faster than every microsecond.

set the previous times and previous counts to the previously current values

get the count and time values from the two timers

calculate the difference between the current counts/times and previous counts/times using the delta function

set the encoder position to be the previous encoder position plus the delta position

set the speed of the encoder to the calculated value via [encoder_calc_speed\(\)](#)

Definition at line 15 of file [encoder_handler.c](#).

```
00015
00017     encoder->prev_count = encoder->curr_count;
00018     encoder->prev_time = encoder->curr_time;
00020     encoder->curr_count = encoder->timer->Instance->CNT;
00021     encoder->curr_time = encoder->timing_timer->Instance->CNT;
00023     encoder->dx = delta(encoder->timer, encoder->prev_count, encoder->curr_count);
00024     encoder->dt = delta(encoder->timing_timer, encoder->prev_time, encoder->curr_time);
00026     encoder->pos = encoder->pos + encoder->dx;
00028     encoder->speed = encoder_calc_speed(encoder, encoder->dx, encoder->dt);
00029
00030 }
```

5.22.2.4 zero()

```
void zero (
    Encoder * encoder )
```

This function zeros the encoder, including the timing_timer and timer timers.

Parameters

<i>encoder</i>	The encoder instance to operate on
----------------	------------------------------------

Definition at line 52 of file [encoder_handler.c](#).

```

00052     {
00053         encoder->timer->Instance->CNT = 0;
00054         encoder->timing_timer->Instance->CNT = 0;
00055         encoder->prev_count = 0;
00056         encoder->curr_count = 0;
00057         encoder->prev_time = 0;
00058         encoder->curr_time = 1;
00059         encoder->pos = 0;
00060         encoder->speed = 0;
00061     }

```

5.23 encoder_handler.c

[Go to the documentation of this file.](#)

```

00001
00008 #include "encoder_handler.h"
00009
00015 void encoder_read_curr_state(Encoder* encoder){
00017     encoder->prev_count = encoder->curr_count;
00018     encoder->prev_time = encoder->curr_time;
00020     encoder->curr_count = encoder->timer->Instance->CNT;
00021     encoder->curr_time = encoder->timing_timer->Instance->CNT;
00023     encoder->dx = delta(encoder->timer, encoder->prev_count,encoder->curr_count);
00024     encoder->dt = delta(encoder->timing_timer, encoder->prev_time,encoder->curr_time);
00026     encoder->pos = encoder->pos + encoder->dx;
00028     encoder->speed = encoder_calc_speed(encoder,encoder->dx,encoder->dt);
00029
00030 }
00031
00039 int32_t encoder_calc_speed(Encoder* encoder, int32_t dx,int32_t dt){
00041     if(dt == 0){
00042         return encoder->speed;
00043     }
00045     return ((dx)*1000000)/dt;
00046 }
00047
00052 void zero(Encoder* encoder){
00053     encoder->timer->Instance->CNT = 0;
00054     encoder->timing_timer->Instance->CNT = 0;
00055     encoder->prev_count = 0;
00056     encoder->curr_count = 0;
00057     encoder->prev_time = 0;
00058     encoder->curr_time = 1;
00059     encoder->pos = 0;
00060     encoder->speed = 0;
00061 }
00062
00071 int32_t delta(TIM_HandleTypeDef* timer, uint32_t initial, uint32_t final){
00073     uint32_t ARR = (int32_t)(timer->Init.Period );
00075     int32_t overflow = ((ARR-1)/2)+1;
00076     //1 Calculate the delta
00077     int32_t delta = final-initial;
00079     if(delta >= overflow){
00080         delta = delta - overflow*2;
00082     }else if(delta <= -1*overflow){
00083         delta = delta + overflow;
00084     }
00085     return delta;
00086 }
00087
00088
00089

```

5.24 doxy_core/Src/main.c File Reference

: Main program body

```
#include "main.h"
#include "stm32l4xx_hal.h"
#include "CLController.h"
#include "controller_driver.h"
#include "display_driver.h"
#include "encoder_handler.h"
#include "motor_driver.h"
#include "pitch_encoder_handler.h"
#include <stdio.h>
```

Functions

- void [SystemClock_Config](#) (void)
System Clock Configuration.
- void [display_task](#) (uint8_t *state)
- void [motor_task](#) (uint8_t *state)
- int [main](#) (void)
The application entry point.
- void [Error_Handler](#) (void)
This function is executed in case of error occurrence.

Variables

- ADC_HandleTypeDef [hadc1](#)
- I2C_HandleTypeDef [hi2c2](#)
- TIM_HandleTypeDef [htim1](#)
- TIM_HandleTypeDef [htim3](#)
- TIM_HandleTypeDef [htim4](#)
- TIM_HandleTypeDef [htim5](#)
- TIM_HandleTypeDef [htim6](#)
- TIM_HandleTypeDef [htim8](#)
- UART_HandleTypeDef [huart2](#)
- uint8_t [t1state](#) = 0
- uint8_t [t2state](#) = 0
- uint8_t [Buffer](#) [50] = {0}
These are buffers used for displaying stuff via uart during debugging.
- uint8_t [Pos_Buffer](#) [50] = {0}
- uint8_t [Speed_Buffer](#) [50] = {0}
- uint8_t [Space](#) [] = " - "
- uint8_t [StartMSG](#) [] = "Starting I2C Scanning: \r\n"
- uint8_t [EndMSG](#) [] = "Done! \r\n\r\n"
- uint8_t [led_buff](#)
- uint8_t [Eff_Buffer](#) [50] = {0}

5.24.1 Detailed Description

: Main program body

!

Attention

Copyright (c) 2024 STMicroelectronics. All rights reserved.

This software is licensed under terms that can be found in the LICENSE file in the root directory of this software component. If no LICENSE file comes with this software, it is provided AS-IS.

Definition in file [main.c](#).

5.24.2 Function Documentation

5.24.2.1 display_task()

```
void display_task (
    uint8_t * state )
```

This function implements the [Display](#) and Note task finite state machine.

Parameters

<i>state</i>	The pointer to the current state of the task
--------------	--

Definition at line 233 of file [main.c](#).

```
00233                                     {
00234     switch(*state){
00235     case 0:
00236         pe.encoder->timer->Instance->CNT = ((htim5.Init.Period)+1)/2;
00237         *state = 1;
00238         break;
00239     case 1:
00240         ptch = get_pitch(&pe);
00241         display_note(&display,ptch);
00242         break;
00243     }
00244 }
```

5.24.2.2 Error_Handler()

```
void Error_Handler (
    void )
```

This function is executed in case of error occurrence.

Return values

<i>None</i>	
-------------	--

Definition at line 1054 of file [main.c](#).

```
01055 {
01056     /* USER CODE BEGIN Error_Handler_Debug */
01057     /* User can add his own implementation to report the HAL error return state */
01058     __disable_irq();
01059     while (1)
01060     {
01061     }
01062     /* USER CODE END Error_Handler_Debug */
01063 }
```

5.24.2.3 main()

```
int main (
    void )
```

The application entry point.

Return values

<i>int</i>	
------------	--

initialize I2C

start the timers in their various modes.

set the display driver enable pin and the motor driver enable pin

This code was all used to debug the i2c driver to no avail, I have left it here for transparency.

This is the game loop that runs forever to convert user inputted code into a PWM signal to drive the motors.

run the tasks in round-robin style.

Definition at line 273 of file [main.c](#).

```
00274 {
00275
00276     /* USER CODE BEGIN 1 */
00277
00278     /* USER CODE END 1 */
00279
00280     /* MCU Configuration-----*/
00281
00282     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
00283     HAL_Init();
00284
00285     /* USER CODE BEGIN Init */
00286
00287     /* USER CODE END Init */
00288
00289     /* Configure the system clock */
00290     SystemClock_Config();
00291
00292     /* USER CODE BEGIN SysInit */
00293
00294     /* USER CODE END SysInit */
00295
00296     /* Initialize all configured peripherals */
00297     MX_GPIO_Init();
00298     MX_USART2_UART_Init();
00299     MX_TIM1_Init();
00300     MX_TIM3_Init();
00301     MX_ADC1_Init();
00302     MX_I2C2_Init();
00303     MX_TIM4_Init();
00304     MX_TIM5_Init();
00305     MX_TIM8_Init();
00306     MX_TIM6_Init();
00307     /* USER CODE BEGIN 2 */
00309     MX_I2C2_Init();
```

```

00311 HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_ALL);
00312 HAL_TIM_Encoder_Start_IT(&htim5, TIM_CHANNEL_ALL);
00313 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
00314 HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
00315 HAL_TIM_Base_Start(&htim6);
00317 HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10,GPIO_PIN_SET);
00318 HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2,GPIO_PIN_SET);
00320 /*uint8_t aTxBuffer[2] = {};
00321 aTxBuffer[0] = 0x00;
00322 aTxBuffer[1] = 0x00;
00323 uint8_t lvl = 85;
00324 uint8_t addr = 0b11000000;
00325 HAL_I2C_Mem_Read(&hi2c2, addr, 0x00, 1, led_buff, 1, 500);
00326 sprintf(Buffer, "Reg 0x14x: %X \r\n",led_buff);
00327 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00328 HAL_Delay(500);
00329 ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x14, 1, lvl, 1, 500);
00330 /*HAL_Delay(500);
00331 if(ret!=HAL_OK){
00332     sprintf(Buffer, "did not work :(\n\r");
00333     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00334 }else if(ret == HAL_OK)
00335 {
00336     sprintf(Buffer, "worked!\n\r");
00337     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00338 }
00339 HAL_Delay(500);
00340 ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x15, 1, lvl, 1, 500);
00341 /*if(ret!=HAL_OK){
00342     sprintf(Buffer, "did not work :(\n\r");
00343     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00344 }else if(ret == HAL_OK)
00345 {
00346     sprintf(Buffer, "worked!\n\r");
00347     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00348 }
00349 HAL_Delay(500);
00350 ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x16, 1, lvl, 1, 500);
00351 /*if(ret!=HAL_OK){
00352     sprintf(Buffer, "did not work :(\n\r");
00353     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00354 }else if(ret == HAL_OK)
00355 {
00356     sprintf(Buffer, "worked!\n\r");
00357     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00358 }
00359 HAL_Delay(500);
00360 ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x17, 1, lvl, 1, 500);
00361 /*if(ret!=HAL_OK){
00362     sprintf(Buffer, "did not work :(\n\r");
00363     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00364 }else if(ret == HAL_OK)
00365 {
00366     sprintf(Buffer, "worked!\n\r");
00367     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00368 }
00369 HAL_I2C_Mem_Read(&hi2c2, addr, 0x14, 1, (uint8_t*)led_buff, 1, 500);
00370 sprintf(Buffer, "Reg 0x14x: %X \r\n",led_buff);
00371 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00372 HAL_I2C_Mem_Read(&hi2c2, addr, 0x15, 1, led_buff, 1, 500);
00373 sprintf(Buffer, "Reg 0x15x: %X \r\n",led_buff);
00374 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00375 HAL_I2C_Mem_Read(&hi2c2, addr, 0x16, 1, led_buff, 1, 500);
00376 sprintf(Buffer, "Reg 0x16x: %X \r\n",led_buff);
00377 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00378 HAL_I2C_Mem_Read(&hi2c2, addr, 0x17, 1, led_buff, 1, 500);
00379 sprintf(Buffer, "Reg 0x17x: %X \r\n",led_buff);
00380 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00381 /*if(ret!=HAL_OK){
00382     sprintf(Buffer, "did not work :(\n\r");
00383     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00384 }else if(ret == HAL_OK)
00385 {
00386     sprintf(Buffer, "worked!\n\r");
00387     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00388 }*/
00389 //HAL_Delay(500);
00390 //ret = HAL_I2C_Master_Transmit(&hi2c2, 0b11000001, ((uint8_t*)1),1,100);
00391 /*if(ret!=HAL_OK){
00392     sprintf(Buffer, "did not work :(\n\r");
00393     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00394 }else if(ret == HAL_OK)
00395 {
00396     sprintf(Buffer, "worked!\n\r");
00397     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00398 }
00399 uint32_t time1 = HAL_GetTick();

```

```

00400     uint32_t time2 = HAL_GetTick();
00401     int32_t sped = 0;
00402     //HAL_I2C_Master_Transmit(&hi2c2, 0b11000001, ((uint8_t*)0b1101100001111000),2,100);
00403     uint8_t prev_count = 1;
00404
00405     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x02, 1, (uint8_t*)lv1, 1, 100);
00406     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x03, 1, (uint8_t*)lv1, 1, 100);
00407     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x04, 1, (uint8_t*)lv1, 1, 100);
00408     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x05, 1, (uint8_t*)0xFF, 1, 100);
00409     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x06, 1, (uint8_t*)lv1, 1, 100);
00410     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x07, 1, (uint8_t*)lv1, 1, 100);
00411     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x08, 1, (uint8_t*)lv1, 1, 100);
00412     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x09, 1, (uint8_t*)0xFF, 1, 100);
00413     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0A, 1, (uint8_t*)lv1, 1, 100);
00414     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0B, 1, (uint8_t*)lv1, 1, 100);
00415     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0C, 1, (uint8_t*)lv1, 1, 100);
00416     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0D, 1, (uint8_t*)0xFF, 1, 100);
00417     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0E, 1, (uint8_t*)lv1, 1, 100);
00418     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0F, 1, (uint8_t*)lv1, 1, 100);
00419     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x10, 1, (uint8_t*)lv1, 1, 100);
00420     //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x11, 1, (uint8_t*)0xFF, 1, 100);
00421     // The folloqing code was taken from here:
    https://deepbluembedded.com/stm32-i2c-scanner-hal-code-example/
00422     uint8_t TLC59116_PWM0_AUTOINCR = 0x82;
00423     HAL_UART_Transmit(&huart2, StartMSG, sizeof(StartMSG), 10000);
00424     for(i=1; i<128; i++)
00425     {
00426         ret = HAL_I2C_IsDeviceReady(&hi2c2, (uint16_t)(i<<1), 3, 5);
00427         if (ret != HAL_OK)
00428         {
00429             HAL_UART_Transmit(&huart2, Space, sizeof(Space), 10000);
00430         }
00431         else if(ret == HAL_OK)
00432         {
00433             sprintf(Buffer, "0x%X", i);
00434             HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00435         }
00436     }
00437     HAL_UART_Transmit(&huart2, EndMSG, sizeof(EndMSG), 10000);
00438     uint8_t bruh[] = { TLC59116_PWM0_AUTOINCR, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
00439     while(HAL_I2C_Master_Transmit(&hi2c2, 0xC0, *bruh, sizeof(bruh), 100) != HAL_OK)
00440     {
00441         HAL_Delay(1);
00442         if (HAL_I2C_GetError(&hi2c2) != HAL_I2C_ERROR_AF)
00443         {
00444             Error_Handler();
00445         }
00446     }
00447     /* USER CODE END 2 */
00448
00449     /* Infinite loop */
00450     /* USER CODE BEGIN WHILE */
00451     while (1)
00452     {
00453         display_task(&t1state);
00454         motor_task(&t2state);
00455
00456         /* USER CODE END WHILE */
00457
00458         /* USER CODE BEGIN 3 */
00459     }
00460     /* USER CODE END 3 */
00461 }
00462
00463 }

```

5.24.2.4 motor_task()

```

void motor_task (
    uint8_t * state )

```

This function implements the [Motor](#) task finite state machine.

Parameters

<i>state</i>	The pointer to the current state of the task
--------------	--

Definition at line 250 of file [main.c](#).

```

00250                                     {
00251     switch(*state){
00252     case 0:
00253         *state = 1;
00254         timmy = HAL_GetTick();
00255         break;
00256     case 1:
00257         if(HAL_GetTick() >= timmy + 2){
00258             timmy = HAL_GetTick();
00259             m_con.setpoint = motor_speeds[ptch];
00260             encoder_read_curr_state(&mot_enc);
00261             eff = run(&m_con, mot_enc.speed);
00262             motor_set_duty_cycle(&m, eff);
00263             break;
00264         }
00265     }
00266 }

```

5.24.2.5 SystemClock_Config()

```

void SystemClock_Config (
    void )

```

System Clock Configuration.

Return values

None	
------	--

Configure the main internal regulator output voltage

Initializes the RCC Oscillators according to the specified parameters in the RCC_OscInitTypeDef structure.

Initializes the CPU, AHB and APB buses clocks

Definition at line 469 of file [main.c](#).

```

00470 {
00471     RCC_OscInitTypeDef RCC_OscInitStruct = {0};
00472     RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
00473
00475
00476     if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
00477     {
00478         Error_Handler();
00479     }
00480
00483
00484     RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
00485     RCC_OscInitStruct.HSIState = RCC_HSI_ON;
00486     RCC_OscInitStruct.HSICalibrationValue = RCC_HSICALIBRATION_DEFAULT;
00487     RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00488     RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
00489     RCC_OscInitStruct.PLL.PLLM = 1;
00490     RCC_OscInitStruct.PLL.PLLN = 10;
00491     RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
00492     RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
00493     RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV2;
00494     if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
00495     {
00496         Error_Handler();
00497     }
00498
00500
00501     RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
00502                                   |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
00503     RCC_ClkInitStruct.SYSClkSource = RCC_SYSCCLKSOURCE_PLLCLK;
00504     RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCCLK_DIV1;
00505     RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
00506     RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
00507
00508     if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_4) != HAL_OK)
00509     {
00510         Error_Handler();
00511     }
00512 }

```


5.24.3 Variable Documentation

5.24.3.1 Buffer

```
uint8_t Buffer[50] = {0}
```

These are buffers used for displaying stuff via uart during debugging.

Definition at line 176 of file [main.c](#).

```
00176 {0};
```

5.24.3.2 Eff_Buffer

```
uint8_t Eff_Buffer[50] = {0}
```

Definition at line 183 of file [main.c](#).

```
00183 {0};
```

5.24.3.3 EndMSG

```
uint8_t EndMSG[] = "Done!  \r\n\r\n"
```

Definition at line 181 of file [main.c](#).

5.24.3.4 hadc1

```
ADC_HandleTypeDef hadc1
```

Definition at line 50 of file [main.c](#).

5.24.3.5 hi2c2

```
I2C_HandleTypeDef hi2c2
```

Definition at line 52 of file [main.c](#).

5.24.3.6 htim1

```
TIM_HandleTypeDef htim1
```

Definition at line 54 of file [main.c](#).

5.24.3.7 htim3

```
TIM_HandleTypeDef htim3
```

Definition at line 55 of file [main.c](#).

5.24.3.8 htim4

```
TIM_HandleTypeDef htim4
```

Definition at line 56 of file [main.c](#).

5.24.3.9 htim5

```
TIM_HandleTypeDef htim5
```

Definition at line 57 of file [main.c](#).

5.24.3.10 htim6

```
TIM_HandleTypeDef htim6
```

Definition at line 58 of file [main.c](#).

5.24.3.11 htim8

```
TIM_HandleTypeDef htim8
```

Definition at line 59 of file [main.c](#).

5.24.3.12 huart2

```
UART_HandleTypeDef huart2
```

Definition at line 61 of file [main.c](#).

5.24.3.13 led_buff

```
uint8_t led_buff
```

Definition at line 182 of file [main.c](#).

5.24.3.14 Pos_Buffer

```
uint8_t Pos_Buffer[50] = {0}
```

Definition at line 177 of file [main.c](#).
00177 {0};

5.24.3.15 Space

```
uint8_t Space[] = " - "
```

Definition at line 179 of file [main.c](#).

5.24.3.16 Speed_Buffer

```
uint8_t Speed_Buffer[50] = {0}
```

Definition at line 178 of file [main.c](#).

```
00178 {0};
```

5.24.3.17 StartMSG

```
uint8_t StartMSG[] = "Starting I2C Scanning: \r\n"
```

Definition at line 180 of file [main.c](#).

5.24.3.18 t1state

```
t1state = 0
```

The current state of the [Display](#) and note task

Definition at line 167 of file [main.c](#).

5.24.3.19 t2state

```
t2state = 0
```

The current state of the motor task

Definition at line 173 of file [main.c](#).

5.25 main.c

[Go to the documentation of this file.](#)

```

00001 /* USER CODE BEGIN Header */
00018 /* USER CODE END Header */
00019 /* Includes -----*/
00020 #include "main.h"
00021
00022 /* Private includes -----*/
00023 /* USER CODE BEGIN Includes */
00024 #include "stm32l4xx_hal.h"
00025 #include "CLController.h"
00026 #include "controller_driver.h"
00027 #include "display_driver.h"
00028 #include "encoder_handler.h"
00029 #include "motor_driver.h"
00030 #include "pitch_encoder_handler.h"
00031 #include <stdio.h>
00032 /* USER CODE END Includes */
00033
00034 /* Private typedef -----*/
00035 /* USER CODE BEGIN PTD */
00036
00037 /* USER CODE END PTD */
00038
00039 /* Private define -----*/
00040 /* USER CODE BEGIN PD */
00041
00042 /* USER CODE END PD */
00043
00044 /* Private macro -----*/
00045 /* USER CODE BEGIN PM */
00046
00047 /* USER CODE END PM */
00048
00049 /* Private variables -----*/
00050 ADC_HandleTypeDef hadc1;
00051
00052 I2C_HandleTypeDef hi2c2;
00053
00054 TIM_HandleTypeDef htim1;
00055 TIM_HandleTypeDef htim3;
00056 TIM_HandleTypeDef htim4;
00057 TIM_HandleTypeDef htim5;
00058 TIM_HandleTypeDef htim6;
00059 TIM_HandleTypeDef htim8;
00060
00061 UART_HandleTypeDef huart2;
00062
00063 /* USER CODE BEGIN PV */
00064 static RC_Controller con = {.timer = &htim3,
00065     .timer = &htim3,
00066     .period1 = 1500,
00067     .period2 = 1500,
00068     .re1 = 0,
00069     .re2 = 0,
00070     .fe1 = 0,
00071     .fe2 = 0,
00072     .fe_flag1 = 0,
00073     .fe_flag2 = 0,
00074     };
00075
00076 static Motor m = {.timer = &htim1,
00077     .channels = 1,
00078     .duty_cycle = 0,
00079     .enable_flag = 1,
00080     };
00081
00082 static Encoder pitch = {.timer = &htim5,
00083     .timing_timer = &htim6,
00084     .prev_count = 0,
00085     .curr_count = 0,
00086     .prev_time = 0,
00087     .curr_time = 1,
00088     .pos = 0,
00089     .speed = 0,
00090     .dx = 0,
00091     .dt = 0,
00092     };
00093
00094 static Encoder mot_enc = {.timer = &htim4,
00095     .timing_timer = &htim6,
00096     .prev_count = 0,
00097     .curr_count = 0,
00098     .prev_time = 0,

```

```

00115         .curr_time = 1,
00116         .pos = 0,
00117         .speed = 0,
00118         .dx = 0,
00119         .dt = 0
00120     };
00121
00122     static CLController m_con = {
00123         .kp = 500,
00124         .ki = 0,
00125         .kd = 0,
00126         .kf = 706,
00127         .setpoint = 0,
00128         .eff = 0,
00129         .curr = 0,
00130         .err = 0,
00131         .err_acc = 0,
00132         .prev_err_index = 0,
00133         .initial_time = 0,
00134         .curr_time = 1,
00135         .slope = 0,
00136         .prev_err_list_length = 10,
00137         .prev_err_list = {0,0,0,0,0,0,0,0,0,0}
00138     };
00139
00140     static PitchEncoder pe = {.pitch = 0,
00141         .encoder = &pitch,
00142         .delta = 0
00143     };
00144
00145     static Display display = {.hi2c = &hi2c2,
00146         .curr_note = 0,
00147         .huart = &huart2
00148     };
00149
00150     uint8_t t1state = 0;
00151     uint8_t t2state = 0;
00152
00153     uint8_t Buffer[50] = {0};
00154     uint8_t Pos_Buffer[50] = {0};
00155     uint8_t Speed_Buffer[50] = {0};
00156     uint8_t Space[] = " - ";
00157     uint8_t StartMSG[] = "Starting I2C Scanning: \r\n";
00158     uint8_t EndMSG[] = "Done! \r\n\r\n";
00159     uint8_t led_buff;
00160     uint8_t Eff_Buffer[50] = {0};
00161
00162     static int32_t motor_speeds[12] =
00163         {56320, 59679, 63222, 66970, 70963, 75182, 79647, 84378, 89395, 94720, 100352, 106312};
00164
00165     static uint32_t ptch;
00166
00167     static uint32_t timmy = 0;
00168
00169     static int32_t eff = 0;
00170     /* USER CODE END PV */
00171
00172     /* Private function prototypes -----*/
00173     void SystemClock_Config(void);
00174     static void MX_GPIO_Init(void);
00175     static void MX_USART2_UART_Init(void);
00176     static void MX_TIM1_Init(void);
00177     static void MX_TIM3_Init(void);
00178     static void MX_ADC1_Init(void);
00179     static void MX_I2C2_Init(void);
00180     static void MX_TIM4_Init(void);
00181     static void MX_TIM5_Init(void);
00182     static void MX_TIM8_Init(void);
00183     static void MX_TIM6_Init(void);
00184     /* USER CODE BEGIN PFP */
00185     void display_task(uint8_t* state);
00186     void motor_task (uint8_t* state);
00187     /* USER CODE END PFP */
00188
00189     /* Private user code -----*/
00190     /* USER CODE BEGIN 0 */
00191     void display_task(uint8_t* state){
00192         switch(*state){
00193             case 0:
00194                 pe.encoder->timer->Instance->CNT = ((htim5.Init.Period)+1)/2;
00195                 *state = 1;
00196                 break;
00197             case 1:
00198                 ptch = get_pitch(&pe);
00199                 display_note(&display,ptch);
00200                 break;

```

```

00243     }
00244 }
00245
00250 void motor_task (uint8_t* state){
00251     switch(*state){
00252         case 0:
00253             *state = 1;
00254             timmy = HAL_GetTick();
00255             break;
00256         case 1:
00257             if(HAL_GetTick() >= timmy + 2){
00258                 timmy = HAL_GetTick();
00259                 m_con.setpoint = motor_speeds[ptch];
00260                 encoder_read_curr_state(&mot_enc);
00261                 eff = run(&m_con,mot_enc.speed);
00262                 motor_set_duty_cycle(&m, eff);
00263                 break;
00264             }
00265     }
00266 }
00267 /* USER CODE END 0 */
00268
00273 int main(void)
00274 {
00275
00276     /* USER CODE BEGIN 1 */
00277
00278     /* USER CODE END 1 */
00279
00280     /* MCU Configuration-----*/
00281
00282     /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
00283     HAL_Init();
00284
00285     /* USER CODE BEGIN Init */
00286
00287     /* USER CODE END Init */
00288
00289     /* Configure the system clock */
00290     SystemClock_Config();
00291
00292     /* USER CODE BEGIN SysInit */
00293
00294     /* USER CODE END SysInit */
00295
00296     /* Initialize all configured peripherals */
00297     MX_GPIO_Init();
00298     MX_USART2_UART_Init();
00299     MX_TIM1_Init();
00300     MX_TIM3_Init();
00301     MX_ADC1_Init();
00302     MX_I2C2_Init();
00303     MX_TIM4_Init();
00304     MX_TIM5_Init();
00305     MX_TIM8_Init();
00306     MX_TIM6_Init();
00307     /* USER CODE BEGIN 2 */
00309     MX_I2C2_Init();
00311     HAL_TIM_Encoder_Start(&htim4, TIM_CHANNEL_ALL);
00312     HAL_TIM_Encoder_Start_IT(&htim5, TIM_CHANNEL_ALL);
00313     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_1);
00314     HAL_TIM_PWM_Start(&htim1, TIM_CHANNEL_2);
00315     HAL_TIM_Base_Start(&htim6);
00317     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10,GPIO_PIN_SET);
00318     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2,GPIO_PIN_SET);
00320     /*uint8_t aTxBuffer[2] = {};
00321     aTxBuffer[0] = 0x00;
00322     aTxBuffer[1] = 0x00;
00323     uint8_t lvl = 85;
00324     uint8_t addr = 0b11000000;
00325     HAL_I2C_Mem_Read(&hi2c2, addr, 0x00, 1, led_buff, 1, 500);
00326     sprintf(Buffer, "Reg 0x14x: %X \r\n",led_buff);
00327     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00328     HAL_Delay(500);
00329     ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x14, 1, lvl, 1, 500);
00330     /*HAL_Delay(500);
00331     if(ret!=HAL_OK){
00332         sprintf(Buffer, "did not work :(\n\r");
00333         HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00334     }else if(ret == HAL_OK)
00335     {
00336         sprintf(Buffer, "worked!\n\r");
00337         HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00338     }
00339     HAL_Delay(500);
00340     ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x15, 1, lvl, 1, 500);
00341     /*if(ret!=HAL_OK){

```

```

00342     sprintf(Buffer, "did not work :(\n\r");
00343     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00344 }else if(ret == HAL_OK)
00345 {
00346     sprintf(Buffer, "worked!\n\r");
00347     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00348 }
00349 HAL_Delay(500);
00350 ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x16, 1, lvl, 1, 500);
00351 /*if(ret!=HAL_OK){
00352     sprintf(Buffer, "did not work :(\n\r");
00353     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00354 }else if(ret == HAL_OK)
00355 {
00356     sprintf(Buffer, "worked!\n\r");
00357     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00358 }
00359 HAL_Delay(500);
00360 ret = HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x17, 1, lvl, 1, 500);
00361 /*if(ret!=HAL_OK){
00362     sprintf(Buffer, "did not work :(\n\r");
00363     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00364 }else if(ret == HAL_OK)
00365 {
00366     sprintf(Buffer, "worked!\n\r");
00367     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00368 }
00369 HAL_I2C_Mem_Read(&hi2c2, addr, 0x14, 1, (uint8_t*)led_buff, 1, 500);
00370 sprintf(Buffer, "Reg 0x14x: %X \r\n",led_buff);
00371 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00372 HAL_I2C_Mem_Read(&hi2c2, addr, 0x15, 1, led_buff, 1, 500);
00373 sprintf(Buffer, "Reg 0x15x: %X \r\n",led_buff);
00374 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00375 HAL_I2C_Mem_Read(&hi2c2, addr, 0x16, 1, led_buff, 1, 500);
00376 sprintf(Buffer, "Reg 0x16x: %X \r\n",led_buff);
00377 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00378 HAL_I2C_Mem_Read(&hi2c2, addr, 0x17, 1, led_buff, 1, 500);
00379 sprintf(Buffer, "Reg 0x17x: %X \r\n",led_buff);
00380 HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00381 /*if(ret!=HAL_OK){
00382     sprintf(Buffer, "did not work :(\n\r");
00383     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00384 }else if(ret == HAL_OK)
00385 {
00386     sprintf(Buffer, "worked!\n\r");
00387     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00388 }*/
00389 //HAL_Delay(500);
00390 //ret = HAL_I2C_Master_Transmit(&hi2c2, 0b11000001, ((uint8_t*)1),1,100);
00391 /*if(ret!=HAL_OK){
00392     sprintf(Buffer, "did not work :(\n\r");
00393     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00394 }else if(ret == HAL_OK)
00395 {
00396     sprintf(Buffer, "worked!\n\r");
00397     HAL_UART_Transmit(&huart2, Buffer, sizeof(Buffer), 10000);
00398 }
00399 uint32_t time1 = HAL_GetTick();
00400 uint32_t time2 = HAL_GetTick();
00401 int32_t sped = 0;
00402 //HAL_I2C_Master_Transmit((&hi2c2), 0b11000001, ((uint8_t*)0b1101100001111000),2,100);
00403 uint8_t prev_count = 1;
00404
00405 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x02, 1, (uint8_t*)lvl, 1, 100);
00406 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x03, 1, (uint8_t*)lvl, 1, 100);
00407 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x04, 1, (uint8_t*)lvl, 1, 100);
00408 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x05, 1, (uint8_t*)0xFF, 1, 100);
00409 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x06, 1, (uint8_t*)lvl, 1, 100);
00410 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x07, 1, (uint8_t*)lvl, 1, 100);
00411 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x08, 1, (uint8_t*)lvl, 1, 100);
00412 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x09, 1, (uint8_t*)0xFF, 1, 100);
00413 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0A, 1, (uint8_t*)lvl, 1, 100);
00414 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0B, 1, (uint8_t*)lvl, 1, 100);
00415 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0C, 1, (uint8_t*)lvl, 1, 100);
00416 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0D, 1, (uint8_t*)0xFF, 1, 100);
00417 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0E, 1, (uint8_t*)lvl, 1, 100);
00418 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x0F, 1, (uint8_t*)lvl, 1, 100);
00419 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x10, 1, (uint8_t*)lvl, 1, 100);
00420 //HAL_I2C_Mem_Write(&hi2c2, 0b11000000, 0x11, 1, (uint8_t*)0xFF, 1, 100);
00421 // The folloqing code was taken from here:
00422 https://deepbluembedded.com/stm32-i2c-scanner-hal-code-example/
00423 uint8_t TLC59116_PWM0_AUTOINCR = 0x82;
00424 HAL_UART_Transmit(&huart2, StartMSG, sizeof(StartMSG), 10000);
00425 for(i=1; i<128; i++)
00426 {
00427     ret = HAL_I2C_IsDeviceReady(&hi2c2, (uint16_t)(i<1), 3, 5);
00428     if (ret != HAL_OK)

```



```

00535     hadc1.Instance = ADC1;
00536     hadc1.Init.ClockPrescaler = ADC_CLOCK_ASYNC_DIV1;
00537     hadc1.Init.Resolution = ADC_RESOLUTION_12B;
00538     hadc1.Init.DataAlign = ADC_DATAALIGN_RIGHT;
00539     hadc1.Init.ScanConvMode = ADC_SCAN_DISABLE;
00540     hadc1.Init.EOCSelection = ADC_EOC_SINGLE_CONV;
00541     hadc1.Init.LowPowerAutoWait = DISABLE;
00542     hadc1.Init.ContinuousConvMode = DISABLE;
00543     hadc1.Init.NbrOfConversion = 1;
00544     hadc1.Init.DiscontinuousConvMode = DISABLE;
00545     hadc1.Init.ExternalTrigConv = ADC_SOFTWARE_START;
00546     hadc1.Init.ExternalTrigConvEdge = ADC_EXTERNALTRIGCONVEDGE_NONE;
00547     hadc1.Init.DMAContinuousRequests = DISABLE;
00548     hadc1.Init.Overrun = ADC_OVR_DATA_PRESERVED;
00549     hadc1.Init.OversamplingMode = DISABLE;
00550     if (HAL_ADC_Init(&hadc1) != HAL_OK)
00551     {
00552         Error_Handler();
00553     }
00554
00557     multimode.Mode = ADC_MODE_INDEPENDENT;
00558     if (HAL_ADCEx_MultiModeConfigChannel(&hadc1, &multimode) != HAL_OK)
00559     {
00560         Error_Handler();
00561     }
00562
00565     sConfig.Channel = ADC_CHANNEL_14;
00566     sConfig.Rank = ADC_REGULAR_RANK_1;
00567     sConfig.SamplingTime = ADC_SAMPLETIME_2CYCLES_5;
00568     sConfig.SingleDiff = ADC_SINGLE_ENDED;
00569     sConfig.OffsetNumber = ADC_OFFSET_NONE;
00570     sConfig.Offset = 0;
00571     if (HAL_ADC_ConfigChannel(&hadc1, &sConfig) != HAL_OK)
00572     {
00573         Error_Handler();
00574     }
00575     /* USER CODE BEGIN ADC1_Init 2 */
00576
00577     /* USER CODE END ADC1_Init 2 */
00578 }
00579
00580
00586 static void MX_I2C2_Init(void)
00587 {
00588
00589     /* USER CODE BEGIN I2C2_Init 0 */
00590
00591     /* USER CODE END I2C2_Init 0 */
00592
00593     /* USER CODE BEGIN I2C2_Init 1 */
00594
00595     /* USER CODE END I2C2_Init 1 */
00596     hi2c2.Instance = I2C2;
00597     hi2c2.Init.Timing = 0x010F3FE;
00598     hi2c2.Init.OwnAddress1 = 0;
00599     hi2c2.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
00600     hi2c2.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
00601     hi2c2.Init.OwnAddress2 = 0;
00602     hi2c2.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
00603     hi2c2.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
00604     hi2c2.Init.NoStretchMode = I2C_NOSTRETCH_ENABLE;
00605     if (HAL_I2C_Init(&hi2c2) != HAL_OK)
00606     {
00607         Error_Handler();
00608     }
00609
00612     if (HAL_I2CEx_ConfigAnalogFilter(&hi2c2, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
00613     {
00614         Error_Handler();
00615     }
00616
00619     if (HAL_I2CEx_ConfigDigitalFilter(&hi2c2, 0) != HAL_OK)
00620     {
00621         Error_Handler();
00622     }
00623     /* USER CODE BEGIN I2C2_Init 2 */
00624
00625     /* USER CODE END I2C2_Init 2 */
00626 }
00627
00628
00634 static void MX_TIM1_Init(void)
00635 {
00636
00637     /* USER CODE BEGIN TIM1_Init 0 */
00638
00639     /* USER CODE END TIM1_Init 0 */

```

```

00640
00641 TIM_MasterConfigTypeDef sMasterConfig = {0};
00642 TIM_OC_InitTypeDef sConfigOC = {0};
00643 TIM_BreakDeadTimeConfigTypeDef sBreakDeadTimeConfig = {0};
00644
00645 /* USER CODE BEGIN TIM1_Init 1 */
00646
00647 /* USER CODE END TIM1_Init 1 */
00648 htim1.Instance = TIM1;
00649 htim1.Init.Prescaler = 0;
00650 htim1.Init.CounterMode = TIM_COUNTERMODE_UP;
00651 htim1.Init.Period = 999;
00652 htim1.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00653 htim1.Init.RepetitionCounter = 0;
00654 htim1.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00655 if (HAL_TIM_PWM_Init(&htim1) != HAL_OK)
00656 {
00657     Error_Handler();
00658 }
00659 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00660 sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
00661 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00662 if (HAL_TIMEx_MasterConfigSynchronization(&htim1, &sMasterConfig) != HAL_OK)
00663 {
00664     Error_Handler();
00665 }
00666 sConfigOC.OCMode = TIM_OCMODE_PWM1;
00667 sConfigOC.Pulse = 0;
00668 sConfigOC.OCpolarity = TIM_OCPOLARITY_HIGH;
00669 sConfigOC.OCNPolarity = TIM_OCNPOLARITY_HIGH;
00670 sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
00671 sConfigOC.OCIdleState = TIM_OCIDLESTATE_RESET;
00672 sConfigOC.OCNIdleState = TIM_OCNIDLESTATE_RESET;
00673 if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
00674 {
00675     Error_Handler();
00676 }
00677 if (HAL_TIM_PWM_ConfigChannel(&htim1, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
00678 {
00679     Error_Handler();
00680 }
00681 sBreakDeadTimeConfig.OffStateRunMode = TIM_OSSR_DISABLE;
00682 sBreakDeadTimeConfig.OffStateIDLEMode = TIM_OSSI_DISABLE;
00683 sBreakDeadTimeConfig.LockLevel = TIM_LOCKLEVEL_OFF;
00684 sBreakDeadTimeConfig.DeadTime = 0;
00685 sBreakDeadTimeConfig.BreakState = TIM_BREAK_DISABLE;
00686 sBreakDeadTimeConfig.BreakPolarity = TIM_BREAKPOLARITY_HIGH;
00687 sBreakDeadTimeConfig.BreakFilter = 0;
00688 sBreakDeadTimeConfig.Break2State = TIM_BREAK2_DISABLE;
00689 sBreakDeadTimeConfig.Break2Polarity = TIM_BREAK2POLARITY_HIGH;
00690 sBreakDeadTimeConfig.Break2Filter = 0;
00691 sBreakDeadTimeConfigAutomaticOutput = TIM_AUTOMATICOUTPUT_DISABLE;
00692 if (HAL_TIMEx_ConfigBreakDeadTime(&htim1, &sBreakDeadTimeConfig) != HAL_OK)
00693 {
00694     Error_Handler();
00695 }
00696 /* USER CODE BEGIN TIM1_Init 2 */
00697
00698 /* USER CODE END TIM1_Init 2 */
00699 HAL_TIM_MspPostInit(&htim1);
00700
00701 }
00702
00703 static void MX_TIM3_Init(void)
00704 {
00705
00706     /* USER CODE BEGIN TIM3_Init 0 */
00707
00708     /* USER CODE END TIM3_Init 0 */
00709
00710     TIM_MasterConfigTypeDef sMasterConfig = {0};
00711     TIM_IC_InitTypeDef sConfigIC = {0};
00712
00713     /* USER CODE BEGIN TIM3_Init 1 */
00714
00715     /* USER CODE END TIM3_Init 1 */
00716 htim3.Instance = TIM3;
00717 htim3.Init.Prescaler = 79;
00718 htim3.Init.CounterMode = TIM_COUNTERMODE_UP;
00719 htim3.Init.Period = 65535;
00720 htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00721 htim3.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00722 if (HAL_TIM_IC_Init(&htim3) != HAL_OK)
00723 {
00724     Error_Handler();
00725 }
00726
00727 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;

```

```

00732     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00733     if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
00734     {
00735         Error_Handler();
00736     }
00737     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
00738     sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
00739     sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
00740     sConfigIC.ICFilter = 0;
00741     if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
00742     {
00743         Error_Handler();
00744     }
00745     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
00746     sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
00747     if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
00748     {
00749         Error_Handler();
00750     }
00751     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
00752     if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_3) != HAL_OK)
00753     {
00754         Error_Handler();
00755     }
00756     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
00757     sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
00758     if (HAL_TIM_IC_ConfigChannel(&htim3, &sConfigIC, TIM_CHANNEL_4) != HAL_OK)
00759     {
00760         Error_Handler();
00761     }
00762     /* USER CODE BEGIN TIM3_Init 2 */
00763
00764     /* USER CODE END TIM3_Init 2 */
00765
00766 }
00767
00773 static void MX_TIM4_Init(void)
00774 {
00775
00776     /* USER CODE BEGIN TIM4_Init 0 */
00777
00778     /* USER CODE END TIM4_Init 0 */
00779
00780     TIM_Encoder_InitTypeDef sConfig = {0};
00781     TIM_MasterConfigTypeDef sMasterConfig = {0};
00782
00783     /* USER CODE BEGIN TIM4_Init 1 */
00784
00785     /* USER CODE END TIM4_Init 1 */
00786     htim4.Instance = TIM4;
00787     htim4.Init.Prescaler = 0;
00788     htim4.Init.CounterMode = TIM_COUNTERMODE_UP;
00789     htim4.Init.Period = 65535;
00790     htim4.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00791     htim4.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00792     sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
00793     sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
00794     sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
00795     sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
00796     sConfig.IC1Filter = 0;
00797     sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
00798     sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
00799     sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
00800     sConfig.IC2Filter = 0;
00801     if (HAL_TIM_Encoder_Init(&htim4, &sConfig) != HAL_OK)
00802     {
00803         Error_Handler();
00804     }
00805     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00806     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00807     if (HAL_TIMEx_MasterConfigSynchronization(&htim4, &sMasterConfig) != HAL_OK)
00808     {
00809         Error_Handler();
00810     }
00811     /* USER CODE BEGIN TIM4_Init 2 */
00812
00813     /* USER CODE END TIM4_Init 2 */
00814
00815 }
00816
00822 static void MX_TIM5_Init(void)
00823 {
00824
00825     /* USER CODE BEGIN TIM5_Init 0 */
00826
00827     /* USER CODE END TIM5_Init 0 */
00828

```

```

00829 TIM_Encoder_InitTypeDef sConfig = {0};
00830 TIM_MasterConfigTypeDef sMasterConfig = {0};
00831
00832 /* USER CODE BEGIN TIM5_Init 1 */
00833
00834 /* USER CODE END TIM5_Init 1 */
00835 htim5.Instance = TIM5;
00836 htim5.Init.Prescaler = 0;
00837 htim5.Init.CounterMode = TIM_COUNTERMODE_UP;
00838 htim5.Init.Period = 1073741823;
00839 htim5.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00840 htim5.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00841 sConfig.EncoderMode = TIM_ENCODERMODE_TI12;
00842 sConfig.IC1Polarity = TIM_ICPOLARITY_RISING;
00843 sConfig.IC1Selection = TIM_ICSELECTION_DIRECTTI;
00844 sConfig.IC1Prescaler = TIM_ICPSC_DIV1;
00845 sConfig.IC1Filter = 0;
00846 sConfig.IC2Polarity = TIM_ICPOLARITY_RISING;
00847 sConfig.IC2Selection = TIM_ICSELECTION_DIRECTTI;
00848 sConfig.IC2Prescaler = TIM_ICPSC_DIV1;
00849 sConfig.IC2Filter = 0;
00850 if (HAL_TIM_Encoder_Init(&htim5, &sConfig) != HAL_OK)
00851 {
00852     Error_Handler();
00853 }
00854 if (HAL_TIM_OnePulse_Init(&htim5, TIM_OPMODE_SINGLE) != HAL_OK)
00855 {
00856     Error_Handler();
00857 }
00858 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00859 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00860 if (HAL_TIMEx_MasterConfigSynchronization(&htim5, &sMasterConfig) != HAL_OK)
00861 {
00862     Error_Handler();
00863 }
00864 /* USER CODE BEGIN TIM5_Init 2 */
00865
00866 /* USER CODE END TIM5_Init 2 */
00867 }
00868
00869 static void MX_TIM6_Init(void)
00870 {
00871     /* USER CODE BEGIN TIM6_Init 0 */
00872
00873     /* USER CODE END TIM6_Init 0 */
00874
00875     TIM_MasterConfigTypeDef sMasterConfig = {0};
00876
00877     /* USER CODE BEGIN TIM6_Init 1 */
00878
00879     /* USER CODE END TIM6_Init 1 */
00880 htim6.Instance = TIM6;
00881 htim6.Init.Prescaler = 79;
00882 htim6.Init.CounterMode = TIM_COUNTERMODE_UP;
00883 htim6.Init.Period = 65535;
00884 htim6.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00885 if (HAL_TIM_Base_Init(&htim6) != HAL_OK)
00886 {
00887     Error_Handler();
00888 }
00889 sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00890 sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00891 if (HAL_TIMEx_MasterConfigSynchronization(&htim6, &sMasterConfig) != HAL_OK)
00892 {
00893     Error_Handler();
00894 }
00895 /* USER CODE BEGIN TIM6_Init 2 */
00896
00897 /* USER CODE END TIM6_Init 2 */
00898 }
00899
00900 static void MX_TIM8_Init(void)
00901 {
00902     /* USER CODE BEGIN TIM8_Init 0 */
00903
00904     /* USER CODE END TIM8_Init 0 */
00905
00906     TIM_MasterConfigTypeDef sMasterConfig = {0};
00907 TIM_IC_InitTypeDef sConfigIC = {0};
00908
00909     /* USER CODE BEGIN TIM8_Init 1 */
00910
00911     /* USER CODE END TIM8_Init 1 */

```

```

00926     htim8.Instance = TIM8;
00927     htim8.Init.Prescaler = 0;
00928     htim8.Init.CounterMode = TIM_COUNTERMODE_UP;
00929     htim8.Init.Period = 65535;
00930     htim8.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
00931     htim8.Init.RepetitionCounter = 0;
00932     htim8.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
00933     if (HAL_TIM_IC_Init(&htim8) != HAL_OK)
00934     {
00935         Error_Handler();
00936     }
00937     sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00938     sMasterConfig.MasterOutputTrigger2 = TIM_TRGO2_RESET;
00939     sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00940     if (HAL_TIMEx_MasterConfigSynchronization(&htim8, &sMasterConfig) != HAL_OK)
00941     {
00942         Error_Handler();
00943     }
00944     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_RISING;
00945     sConfigIC.ICSelection = TIM_ICSELECTION_DIRECTTI;
00946     sConfigIC.ICPrescaler = TIM_ICPSC_DIV1;
00947     sConfigIC.ICFilter = 0;
00948     if (HAL_TIM_IC_ConfigChannel(&htim8, &sConfigIC, TIM_CHANNEL_1) != HAL_OK)
00949     {
00950         Error_Handler();
00951     }
00952     sConfigIC.ICPolarity = TIM_INPUTCHANNELPOLARITY_FALLING;
00953     sConfigIC.ICSelection = TIM_ICSELECTION_INDIRECTTI;
00954     if (HAL_TIM_IC_ConfigChannel(&htim8, &sConfigIC, TIM_CHANNEL_2) != HAL_OK)
00955     {
00956         Error_Handler();
00957     }
00958     /* USER CODE BEGIN TIM8_Init 2 */
00959
00960     /* USER CODE END TIM8_Init 2 */
00961
00962 }
00963
00964 static void MX_USART2_UART_Init(void)
00965 {
00966     /* USER CODE BEGIN USART2_Init 0 */
00967
00968     /* USER CODE END USART2_Init 0 */
00969
00970     /* USER CODE BEGIN USART2_Init 1 */
00971
00972     /* USER CODE END USART2_Init 1 */
00973     huart2.Instance = USART2;
00974     huart2.Init.BaudRate = 115200;
00975     huart2.Init.WordLength = UART_WORDLENGTH_8B;
00976     huart2.Init.StopBits = UART_STOPBITS_1;
00977     huart2.Init.Parity = UART_PARITY_NONE;
00978     huart2.Init.Mode = UART_MODE_TX_RX;
00979     huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
00980     huart2.Init.OverSampling = UART_OVERSAMPLING_16;
00981     huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
00982     huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
00983     if (HAL_UART_Init(&huart2) != HAL_OK)
00984     {
00985         Error_Handler();
00986     }
00987     /* USER CODE BEGIN USART2_Init 2 */
00988
00989     /* USER CODE END USART2_Init 2 */
00990
00991 }
00992
01004 static void MX_GPIO_Init(void)
01005 {
01006     GPIO_InitTypeDef GPIO_InitStruct = {0};
01007     /* USER CODE BEGIN MX_GPIO_Init_1 */
01008     /* USER CODE END MX_GPIO_Init_1 */
01009
01010     /* GPIO Ports Clock Enable */
01011     __HAL_RCC_GPIOH_CLK_ENABLE();
01012     __HAL_RCC_GPIOA_CLK_ENABLE();
01013     __HAL_RCC_GPIOC_CLK_ENABLE();
01014     __HAL_RCC_GPIOB_CLK_ENABLE();
01015
01016     /*Configure GPIO pin Output Level */
01017     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_2, GPIO_PIN_RESET);
01018
01019     /*Configure GPIO pin Output Level */
01020     HAL_GPIO_WritePin(GPIOA, GPIO_PIN_10, GPIO_PIN_RESET);
01021
01022     /*Configure GPIO pin : PB2 */

```

```

01023     GPIO_InitStruct.Pin = GPIO_PIN_2;
01024     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
01025     GPIO_InitStruct.Pull = GPIO_NOPULL;
01026     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
01027     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
01028
01029     /*Configure GPIO pin : PA10 */
01030     GPIO_InitStruct.Pin = GPIO_PIN_10;
01031     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
01032     GPIO_InitStruct.Pull = GPIO_NOPULL;
01033     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
01034     HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
01035
01036     /*Configure GPIO pin : PC10 */
01037     GPIO_InitStruct.Pin = GPIO_PIN_10;
01038     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
01039     GPIO_InitStruct.Pull = GPIO_NOPULL;
01040     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
01041
01042     /* USER CODE BEGIN MX_GPIO_Init_2 */
01043     /* USER CODE END MX_GPIO_Init_2 */
01044 }
01045
01046 /* USER CODE BEGIN 4 */
01047
01048 /* USER CODE END 4 */
01049
01054 void Error_Handler(void)
01055 {
01056     /* USER CODE BEGIN Error_Handler_Debug */
01057     /* User can add his own implementation to report the HAL error return state */
01058     __disable_irq();
01059     while (1)
01060     {
01061     }
01062     /* USER CODE END Error_Handler_Debug */
01063 }
01064
01065 #ifdef USE_FULL_ASSERT
01073 void assert_failed(uint8_t *file, uint32_t line)
01074 {
01075     /* USER CODE BEGIN 6 */
01076     /* User can add his own implementation to report the file name and line number,
01077     ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
01078     /* USER CODE END 6 */
01079 }
01080 #endif /* USE_FULL_ASSERT */

```

5.26 doxy_core/Src/motor_driver.c File Reference

this file implements the [Motor](#) struct to allow for pseudo object oriented programming motor control.

```
#include "motor_driver.h"
```

Functions

- void [motor_set_duty_cycle](#) ([Motor](#) *motor, int32_t doot)
- void [motor_enable_disable](#) ([Motor](#) *motor, uint8_t enable)

5.26.1 Detailed Description

this file implements the [Motor](#) struct to allow for pseudo object oriented programming motor control.

Date

Apr 25, 2024

Author

Jared Sinasohn

Definition in file [motor_driver.c](#).

5.26.2 Function Documentation

5.26.2.1 motor_enable_disable()

```
void motor_enable_disable (
    Motor * motor,
    uint8_t enable )
```

Enables or disables motor based on user input

Parameters

<i>motor,the</i>	Motor struct to act upon
<i>enable,the</i>	boolean of whether to enable or disable the motor with 1 being to enable and 0 being to disable.

Definition at line 87 of file [motor_driver.c](#).

```
00087                                     {
00088     // if user wants to enable motor
00089     if(enable == 1){
00090         motor->enable_flag = 1;
00091         // First retrieve ARR to set motor to brake mode
00092         uint32_t ARR = (uint32_t)(motor->timer->Init.Period + 1);
00093
00094         // Now set the correct motor pair to brake mode.
00095         if(motor->channels == 1){
00096             motor->timer->Instance->CCR1 = ARR;
00097             motor->timer->Instance->CCR2 = ARR;
00098         } else if(motor->channels == 2){
00099             motor->timer->Instance->CCR3 = ARR;
00100             motor->timer->Instance->CCR4 = ARR;
00101         }else{
00102             return;
00103         }
00104
00105         // set the motor's enable flag to 1
00106         motor->enable_flag = 1;
00107
00108     // if user wants to disable motor
00109     } else if(enable == 0){
00110         motor->enable_flag = 0;
00111         if(motor->channels == 1){
00112             motor->timer->Instance->CCR1 = 0;
00113             motor->timer->Instance->CCR2 = 0;
00114         } else if(motor->channels == 2){
00115             motor->timer->Instance->CCR3 = 0;
00116             motor->timer->Instance->CCR4 = 0;
00117         }else{
00118             return;
00119         }
00120
00121         // set the motor's enable flag to 0
00122         motor->enable_flag = 0;
00123     }
00124 }
```

5.26.2.2 motor_set_duty_cycle()

```
void motor_set_duty_cycle (
    Motor * motor,
    int32_t doot )
```

This function implements the duty cycle setting of the motor. It takes in the motor struct and a duty cycle from -100 to 100 (though the function saturates values above and below these values).

Parameters

<i>motor,the</i>	Motor struct to be operated on.
<i>doot,the</i>	duty cycle to be set to.

Definition at line 21 of file [motor_driver.c](#).

```

00021                                     {
00022     motor->duty_cycle = doot;
00023     // First, check if the motor is disabled
00024     if(motor->enable_flag != 1){
00025         // if the enable flag isn't set exit the function and do nothing.
00026         // we are also using != 1 so if there is a stray value in memory,
00027         // the motor doesn't accidentally enable.
00028         return;
00029     }
00030
00031     // Next, saturate the duty cycle just in case.
00032     if(doot < -100){
00033         doot = -100;
00034     }
00035     if(doot > 100){
00036         doot = 100;
00037     }
00038
00039     // We need to get the auto reload value for the timer we are using
00040     // signed value so we don't run into sign issues later
00041     int32_t ARR = (int32_t)(motor->timer->Init.Period + 1);
00042
00043     // Now calculate the duty cycle in terms of the CCR value
00044     doot = doot*ARR/100; // multiply first so we don't lose data
00045
00046     // now we need to set the motors to the correct duty cycles
00047     // Forwards will be channels 1 and 3 for motors 1 and 2 respectively
00048     // Backwards will be channels 2 and 4 for motors 1 and 2 respectively
00049
00050
00051     // the below CCR's are based on the logic table of the toshiba, setting motor.
00052     // to brake mode
00053     // if duty cycle is <0
00054     if (doot < 0){
00055         // check if it is the first or second motor.
00056         if(motor->channels == 1){
00057             motor->timer->Instance->CCR1 = ARR;
00058             motor->timer->Instance->CCR2 = ARR + doot;
00059         } else if(motor->channels == 2){
00060             motor->timer->Instance->CCR3 = ARR;
00061             motor->timer->Instance->CCR4 = ARR + doot;
00062         }else{
00063             // if neither return
00064             return;
00065         }
00066     } // if duty cycle >=0
00067     else{
00068         if(motor->channels == 1){
00069             motor->timer->Instance->CCR1 = ARR - doot;
00070             motor->timer->Instance->CCR2 = ARR;
00071         } else if(motor->channels == 2){
00072             motor->timer->Instance->CCR3 = ARR - doot;
00073             motor->timer->Instance->CCR4 = ARR;
00074         }else{
00075             return;
00076         }
00077     }
00078 }
```

5.27 motor_driver.c

[Go to the documentation of this file.](#)

```

00001
00011 #include "motor_driver.h"
00012
00021 void motor_set_duty_cycle(Motor* motor, int32_t doot){
00022     motor->duty_cycle = doot;
00023     // First, check if the motor is disabled
00024     if(motor->enable_flag != 1){
00025         // if the enable flag isn't set exit the function and do nothing.
00026         // we are also using != 1 so if there is a stray value in memory,
00027         // the motor doesn't accidentally enable.
00028         return;
00029     }
00030
00031     // Next, saturate the duty cycle just in case.
00032     if(doot < -100){
00033         doot = -100;
00034     }
00035     if(doot > 100){
```



```

00036     doot = 100;
00037 }
00038
00039 // We need to get the auto reload value for the timer we are using
00040 // signed value so we don't run into sign issues later
00041 int32_t ARR = (int32_t)(motor->timer->Init.Period + 1);
00042
00043 // Now calculate the duty cycle in terms of the CCR value
00044 doot = doot*ARR/100; // multiply first so we don't lose data
00045
00046 // now we need to set the motors to the correct duty cycles
00047 // Forwards will be channels 1 and 3 for motors 1 and 2 respectively
00048 // Backwards will be channels 2 and 4 for motors 1 and 2 respectively
00049
00050
00051 // the below CCR's are based on the logic table of the toshiba, setting motor.
00052 // to brake mode
00053 // if duty cycle is <0
00054 if (doot < 0){
00055     // check if it is the first or second motor.
00056     if(motor->channels == 1){
00057         motor->timer->Instance->CCR1 = ARR;
00058         motor->timer->Instance->CCR2 = ARR + doot;
00059     } else if(motor->channels == 2){
00060         motor->timer->Instance->CCR3 = ARR;
00061         motor->timer->Instance->CCR4 = ARR + doot;
00062     }else{
00063         // if neither return
00064         return;
00065     }
00066 // if duty cycle >=0
00067 } else{
00068     if(motor->channels == 1){
00069         motor->timer->Instance->CCR1 = ARR - doot;
00070         motor->timer->Instance->CCR2 = ARR;
00071     } else if(motor->channels == 2){
00072         motor->timer->Instance->CCR3 = ARR - doot;
00073         motor->timer->Instance->CCR4 = ARR;
00074     }else{
00075         return;
00076     }
00077 }
00078 }
00079
00080 void motor_enable_disable(Motor* motor, uint8_t enable){
00081     // if user wants to enable motor
00082     if(enable == 1){
00083         motor->enable_flag = 1;
00084         // First retrieve ARR to set motor to brake mode
00085         uint32_t ARR = (uint32_t)(motor->timer->Init.Period + 1);
00086
00087         // Now set the correct motor pair to brake mode.
00088         if(motor->channels == 1){
00089             motor->timer->Instance->CCR1 = ARR;
00090             motor->timer->Instance->CCR2 = ARR;
00091         } else if(motor->channels == 2){
00092             motor->timer->Instance->CCR3 = ARR;
00093             motor->timer->Instance->CCR4 = ARR;
00094         }else{
00095             return;
00096         }
00097
00098         // set the motor's enable flag to 1
00099         motor->enable_flag = 1;
00100
00101         // if user wants to disable motor
00102     } else if(enable == 0){
00103         motor->enable_flag = 0;
00104         if(motor->channels == 1){
00105             motor->timer->Instance->CCR1 = 0;
00106             motor->timer->Instance->CCR2 = 0;
00107         } else if(motor->channels == 2){
00108             motor->timer->Instance->CCR3 = 0;
00109             motor->timer->Instance->CCR4 = 0;
00110         }else{
00111             return;
00112         }
00113
00114         // set the motor's enable flag to 0
00115         motor->enable_flag = 0;
00116     }
00117 }
00118
00119 }
00120
00121 }
00122
00123 }
00124 }

```

5.28 doxy_core/Src/pitch_encoder_handler.c File Reference

Implements the methods of the [PitchEncoder](#) class.

```
#include "pitch_encoder_handler.h"
```

Functions

- `uint32_t get_pitch (PitchEncoder *p_enc)`

5.28.1 Detailed Description

Implements the methods of the [PitchEncoder](#) class.

Date

May 30, 2024

Author

Jared Sinasohn

Definition in file [pitch_encoder_handler.c](#).

5.28.2 Function Documentation

5.28.2.1 get_pitch()

```
uint32_t get_pitch (
    PitchEncoder * p_enc )
```

This function gets the current pitch based on the pitch selection knob

Parameters

<i>p_enc</i>	The pitch encoder object to read from
--------------	---------------------------------------

Returns

the current pitch, a number 0-11 mapped through the chromatic notes from A-Ab

read the current state of the encoder

store the delta of the encoder

add the delta to the pitch

if the pitch hasn't changed, just return the pitch

we can treat the pitch as a number between 0 and 11, which can underflow and overflow. We can run a similar algorithm to the `delta()` function in the encoder class to correct for this.

Definition at line 15 of file `pitch_encoder_handler.c`.

```
00015 {
00017     encoder_read_curr_state((p_enc->encoder));
00019     p_enc->delta = (int16_t)(p_enc->encoder->dx);
00021     p_enc->pitch += p_enc->delta;
00023     if(p_enc->delta == 0){
00024         return p_enc->pitch;
00025     }
00027     if(p_enc->pitch < 0){
00028         p_enc->pitch += 12;
00029     }
00030     if(p_enc->pitch >= 12){
00031         p_enc->pitch -= 12;
00032     }
00033     return p_enc->pitch;
00034 }
```

5.29 pitch_encoder_handler.c

[Go to the documentation of this file.](#)

```
00001
00008 #include "pitch_encoder_handler.h"
00009
00015 uint32_t get_pitch(PitchEncoder* p_enc){
00017     encoder_read_curr_state((p_enc->encoder));
00019     p_enc->delta = (int16_t)(p_enc->encoder->dx);
00021     p_enc->pitch += p_enc->delta;
00023     if(p_enc->delta == 0){
00024         return p_enc->pitch;
00025     }
00027     if(p_enc->pitch < 0){
00028         p_enc->pitch += 12;
00029     }
00030     if(p_enc->pitch >= 12){
00031         p_enc->pitch -= 12;
00032     }
00033     return p_enc->pitch;
00034 }
00035
```


Index

Buffer
 main.c, [57](#)

channels
 Motor, [16](#)

CLController, [9](#)
 curr, [10](#)
 curr_time, [10](#)
 eff, [10](#)
 err, [10](#)
 err_acc, [11](#)
 initial_time, [11](#)
 kd, [11](#)
 kf, [11](#)
 ki, [11](#)
 kp, [11](#)
 prev_err_index, [11](#)
 prev_err_list, [11](#)
 prev_err_list_length, [12](#)
 setpoint, [12](#)
 slope, [12](#)

CLController.c
 reset_controller, [40](#)
 run, [41](#)

CLController.h
 reset_controller, [21](#)
 run, [22](#)

controller_driver.c
 controller_driver_calc_per1, [43](#)
 controller_driver_calc_per2, [43](#)

controller_driver.h
 controller_driver_calc_per1, [24](#)
 controller_driver_calc_per2, [25](#)

controller_driver_calc_per1
 controller_driver.c, [43](#)
 controller_driver.h, [24](#)

controller_driver_calc_per2
 controller_driver.c, [43](#)
 controller_driver.h, [25](#)

curr
 CLController, [10](#)

curr_count
 Encoder, [14](#)

curr_note
 Display, [13](#)

curr_time
 CLController, [10](#)
 Encoder, [14](#)

delta
 encoder_handler.c, [47](#)
 encoder_handler.h, [29](#)
 PitchEncoder, [18](#)

disp_addr
 display_driver.c, [45](#)

Display, [12](#)
 curr_note, [13](#)
 hi2c, [13](#)
 huart, [13](#)

display_driver.c
 disp_addr, [45](#)
 display_note, [44](#)
 note_addresses, [45](#)
 Pitch_Buffer, [46](#)
 Pitch_Message, [46](#)

display_driver.h
 display_note, [26](#)

display_note
 display_driver.c, [44](#)
 display_driver.h, [26](#)

display_task
 main.c, [52](#)

doxy_core/Inc/CLController.h, [21](#), [23](#)
doxy_core/Inc/controller_driver.h, [23](#), [25](#)
doxy_core/Inc/display_driver.h, [26](#), [28](#)
doxy_core/Inc/encoder_handler.h, [28](#), [32](#)
doxy_core/Inc/main.h, [32](#), [35](#)
doxy_core/Inc/motor_driver.h, [35](#), [38](#)
doxy_core/Inc/pitch_encoder_handler.h, [38](#), [40](#)
doxy_core/README.md, [40](#)
doxy_core/Src/CLController.c, [40](#), [42](#)
doxy_core/Src/controller_driver.c, [42](#), [44](#)
doxy_core/Src/display_driver.c, [44](#), [46](#)
doxy_core/Src/encoder_handler.c, [47](#), [50](#)
doxy_core/Src/main.c, [50](#), [60](#)
doxy_core/Src/motor_driver.c, [70](#), [72](#)
doxy_core/Src/pitch_encoder_handler.c, [74](#), [75](#)

dt
 Encoder, [15](#)

duty_cycle
 Motor, [17](#)

dx
 Encoder, [15](#)

eff
 CLController, [10](#)

Eff_Buffer
 main.c, [57](#)

enable_flag
 Motor, [17](#)

- Encoder, 13
 - curr_count, 14
 - curr_time, 14
 - dt, 15
 - dx, 15
 - pos, 15
 - prev_count, 15
 - prev_time, 15
 - speed, 15
 - timer, 15
 - timing_timer, 15
- encoder
 - PitchEncoder, 18
- encoder_calc_speed
 - encoder_handler.c, 48
 - encoder_handler.h, 30
- encoder_handler.c
 - delta, 47
 - encoder_calc_speed, 48
 - encoder_read_curr_state, 49
 - zero, 49
- encoder_handler.h
 - delta, 29
 - encoder_calc_speed, 30
 - encoder_read_curr_state, 31
 - zero, 31
- encoder_read_curr_state
 - encoder_handler.c, 49
 - encoder_handler.h, 31
- EndMSG
 - main.c, 57
- err
 - CLController, 10
- err_acc
 - CLController, 11
- Error_Handler
 - main.c, 52
 - main.h, 34
- fe1
 - RC_Controller, 19
- fe2
 - RC_Controller, 19
- fe_flag1
 - RC_Controller, 19
- fe_flag2
 - RC_Controller, 19
- get_pitch
 - pitch_encoder_handler.c, 74
 - pitch_encoder_handler.h, 39
- hadc1
 - main.c, 57
- HAL_TIM_MspPostInit
 - main.h, 34
- hi2c
 - Display, 13
- hi2c2
 - main.c, 57
- htim1
 - main.c, 57
- htim3
 - main.c, 57
- htim4
 - main.c, 57
- htim5
 - main.c, 58
- htim6
 - main.c, 58
- htim8
 - main.c, 58
- huart
 - Display, 13
- huart2
 - main.c, 58
- initial_time
 - CLController, 11
- kd
 - CLController, 11
- kf
 - CLController, 11
- ki
 - CLController, 11
- kp
 - CLController, 11
- led_buff
 - main.c, 58
- main
 - main.c, 53
- main.c
 - Buffer, 57
 - display_task, 52
 - Eff_Buffer, 57
 - EndMSG, 57
 - Error_Handler, 52
 - hadc1, 57
 - hi2c2, 57
 - htim1, 57
 - htim3, 57
 - htim4, 57
 - htim5, 58
 - htim6, 58
 - htim8, 58
 - huart2, 58
 - led_buff, 58
 - main, 53
 - motor_task, 55
 - Pos_Buffer, 58
 - Space, 58
 - Speed_Buffer, 59
 - StartMSG, 59
 - SystemClock_Config, 56
 - t1state, 59

- t2state, [59](#)
- main.h
 - Error_Handler, [34](#)
 - HAL_TIM_MspPostInit, [34](#)
 - TCK_GPIO_Port, [33](#)
 - TCK_Pin, [33](#)
 - TMS_GPIO_Port, [33](#)
 - TMS_Pin, [33](#)
 - USART_RX_GPIO_Port, [33](#)
 - USART_RX_Pin, [34](#)
 - USART_TX_GPIO_Port, [34](#)
 - USART_TX_Pin, [34](#)
- ME 507 Strobe Tuner Documentation, [1](#)
- Motor, [16](#)
 - channels, [16](#)
 - duty_cycle, [17](#)
 - enable_flag, [17](#)
 - timer, [17](#)
- motor_driver.c
 - motor_enable_disable, [71](#)
 - motor_set_duty_cycle, [71](#)
- motor_driver.h
 - motor_enable_disable, [36](#)
 - motor_set_duty_cycle, [37](#)
- motor_enable_disable
 - motor_driver.c, [71](#)
 - motor_driver.h, [36](#)
- motor_set_duty_cycle
 - motor_driver.c, [71](#)
 - motor_driver.h, [37](#)
- motor_task
 - main.c, [55](#)
- note_addresses
 - display_driver.c, [45](#)
- period1
 - RC_Controller, [19](#)
- period2
 - RC_Controller, [20](#)
- pitch
 - PitchEncoder, [18](#)
- Pitch_Buffer
 - display_driver.c, [46](#)
- pitch_encoder_handler.c
 - get_pitch, [74](#)
- pitch_encoder_handler.h
 - get_pitch, [39](#)
- Pitch_Message
 - display_driver.c, [46](#)
- PitchEncoder, [17](#)
 - delta, [18](#)
 - encoder, [18](#)
 - pitch, [18](#)
- pos
 - Encoder, [15](#)
- Pos_Buffer
 - main.c, [58](#)
- prev_count
 - Encoder, [15](#)
- prev_err_index
 - CLController, [11](#)
- prev_err_list
 - CLController, [11](#)
- prev_err_list_length
 - CLController, [12](#)
- prev_time
 - Encoder, [15](#)
- RC_Controller, [18](#)
 - fe1, [19](#)
 - fe2, [19](#)
 - fe_flag1, [19](#)
 - fe_flag2, [19](#)
 - period1, [19](#)
 - period2, [20](#)
 - re1, [20](#)
 - re2, [20](#)
 - timer, [20](#)
- re1
 - RC_Controller, [20](#)
- re2
 - RC_Controller, [20](#)
- reset_controller
 - CLController.c, [40](#)
 - CLController.h, [21](#)
- run
 - CLController.c, [41](#)
 - CLController.h, [22](#)
- setpoint
 - CLController, [12](#)
- slope
 - CLController, [12](#)
- Space
 - main.c, [58](#)
- speed
 - Encoder, [15](#)
- Speed_Buffer
 - main.c, [59](#)
- StartMSG
 - main.c, [59](#)
- SystemClock_Config
 - main.c, [56](#)
- t1state
 - main.c, [59](#)
- t2state
 - main.c, [59](#)
- TCK_GPIO_Port
 - main.h, [33](#)
- TCK_Pin
 - main.h, [33](#)
- timer
 - Encoder, [15](#)
 - Motor, [17](#)
 - RC_Controller, [20](#)
- timing_timer

- Encoder, [15](#)
- TMS_GPIO_Port
 - main.h, [33](#)
- TMS_Pin
 - main.h, [33](#)
- USART_RX_GPIO_Port
 - main.h, [33](#)
- USART_RX_Pin
 - main.h, [34](#)
- USART_TX_GPIO_Port
 - main.h, [34](#)
- USART_TX_Pin
 - main.h, [34](#)
- zero
 - encoder_handler.c, [49](#)
 - encoder_handler.h, [31](#)